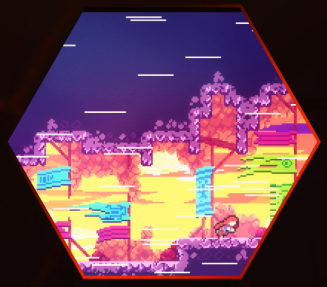
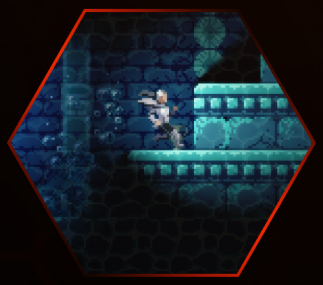



MONOGAME

One framework for creating powerful **cross-platform games**

Complete Reference Guide

Table of Contents

- Articles
  - Introduction ---- 17
  - Getting Started ---- 18
    - Setting up MacOS ---- 20
    - Setting up Linux
      - Setting up Arch Linux ---- 26
      - Setting up Ubuntu ---- 31
    - Setting up Windows ---- 34
    - Using Visual Studio ---- 39
    - Using Rider ---- 49
    - Using Visual Studio Code ---- 54
  - Getting to know MonoGame ---- 64
    - What is ---- 65
      - Audio ---- 67
      - Content Pipeline ---- 70
      - Graphics ---- 72
      - Input ---- 75
      - The Game Loop ---- 76
      - Vector / Matrix / Quaternions ---- 82
      - MonoGame Class Library ---- 85
    - How to ---- 88
      - Audio ---- 91
      - Content Pipeline ---- 92
      - Graphics ---- 93
      - Input ---- 98
  - Advanced Topics
    - Packaging ---- 99
    - Preparing for consoles ---- 106
    - Using Development Nuget Packages ---- 110
    - Tools
      - Introduction ---- 116
      - MGCB ---- 117
      - MGCB Editor ---- 124
      - MGFXC ---- 126
    - Content Pipeline
      - Introduction ---- 128

Why use the Content Pipeline ---- 129
Using the Content Builder Project ---- 133
Automating the Content Builder ---- 162
Using MGCB Editor ---- 178
Custom Effects ---- 196
TrueType fonts ---- 199
Localization ---- 201
Migration
Migrating from XNA ---- 205
Migrating from 3.7 ---- 207
Upgrading 3.8.x to latest ---- 209
Updating Versions ---- 216
Tutorials ---- 220
Building 2D Games ---- 226
01: What Is MonoGame? ---- 234
02: Getting Started ---- 238
03: The Game1 File ---- 252
04: Creating a Class Library ---- 262
05: Content Pipeline ---- 276
06: Working with Textures ---- 295
07: Optimizing Texture Rendering ---- 333
08: The Sprite Class ---- 358
09: The AnimatedSprite Class ---- 368
10: Handling Input ---- 388
11: Input Management ---- 418
12: Collision Detection ---- 453
13: Working With Tilemaps ---- 487
14: Sound Effects and Music ---- 514
15: Audio Controller ---- 534
16: Working with SpriteFonts ---- 560
17: Scenes ---- 583
18: Texture Sampling ---- 621
19: User Interface Fundamentals ---- 639
20: Implementing UI with Gum ---- 649
21: Customizing Gum UI ---- 689
22: Snake Game Mechanics ---- 728
23: Completing the Game ---- 761
24: Shaders ---- 799
25: Packaging Your Game for Distribution ---- 825
26: Publishing Your Game to itch.io ---- 851

27: Conclusion and Next Steps 883
Console Access 888
Help and Support 891
API Reference
Microsoft.Xna.Framework 892
BoundingBox 897
BoundingFrustum 917
BoundingSphere 932
Color 948
ContainmentType 1009
Curve 1010
CurveContinuity 1015
CurveKey 1016
CurveKeyCollection 1024
CurveLoopType 1031
CurveTangent 1032
DisplayOrientation 1033
DrawableGameComponent 1034
ExitingEventArgs 1040
FileDropEventArgs 1041
FrameworkDispatcher 1043
Game 1044
GameComponent 1057
GameComponentCollection 1062
GameComponentCollectionEventArgs 1066
GameRunBehavior 1068
GameServiceContainer 1069
GameTime 1074
GameWindow 1077
GraphicsDeviceInformation 1086
GraphicsDeviceManager 1088
ICurveEvaluator<T> 1099
IDrawable 1100
IGameComponent 1102
IGraphicsDeviceManager 1103
IPlatformBackButton 1105
IUpdateable 1106
InputKeyEventArgs 1108
LaunchParameters 1110
MathHelper 1111

MathHelper.Random 1125
Matrix 1130
Plane 1189
PlaneIntersectionType 1205
PlayerIndex 1206
Point 1208
PreparingDeviceSettingsEventArgs 1216
Quaternion 1218
Ray 1247
Rectangle 1256
TextInputEventArgs 1274
TitleContainer 1276
Vector2 1278
Vector3 1328
Vector4 1379
Microsoft.Xna.Framework.Audio 1429
AudioCategory 1431
AudioChannels 1436
AudioEmitter 1437
AudioEngine 1440
AudioListener 1445
AudioStopOptions 1448
Cue 1449
DynamicSoundEffectInstance 1457
InstancePlayLimitException 1463
Microphone 1464
MicrophoneState 1470
NoAudioHardwareException 1471
NoMicrophoneConnectedException 1473
SoundBank 1475
SoundEffect 1479
SoundEffectInstance 1490
SoundState 1497
WaveBank 1498
Microsoft.Xna.Framework.Content 1502
ArrayReader<T> 1505
ContentLoadException 1507
ContentManager 1509
ContentReader 1519
ContentReaderExtensions 1535

ContentSerializerAttribute 1537
ContentSerializerCollectionItemNameAttribute 1541
ContentSerializerIgnoreAttribute 1543
ContentSerializerRuntimeTypeAttribute 1545
ContentSerializerTypeVersionAttribute 1547
ContentTypeReader 1549
ContentTypeReaderManager 1552
ContentTypeReader<T> 1555
DictionaryReader<TKey, TValue> 1557
EnumReader<T> 1560
ListReader<T> 1562
MultiArrayReader<T> 1565
NullableReader<T> 1567
ReflectiveReader<T> 1569
ResourceContentManager 1571
Microsoft.Xna.Framework.Content.Pipeline 1573
ChildCollection<TParent, TChild> 1577
ContentBuildLogger 1581
ContentIdentity 1587
ContentImporterAttribute 1590
ContentImporterContext 1594
ContentImporter<T> 1597
ContentItem 1600
ContentProcessorAttribute 1602
ContentProcessorContext 1604
ContentProcessor<TInput, TOutput> 1618
ContentStats 1621
ContentStatsCollection 1624
EffectImporter 1629
ExternalReference<T> 1631
FbxImporter 1634
FontDescriptionImporter 1636
FontImporter 1638
H264Importer 1642
IContentImporter 1644
IContentProcessor 1645
InvalidContentException 1647
LoadedTypeCollection<T> 1651
LogLevel 1653
Mp3Importer 1654

NamedValueDictionary<T> 1656
OggImporter 1663
OpaqueDataDictionary 1665
OpenAssetImporter 1667
PipelineComponentScanner 1669
PipelineException 1674
ProcessorParameter 1677
ProcessorParameterCollection 1680
TargetPlatform 1681
TextureImporter 1683
VideoContent 1685
WavImporter 1689
WmaImporter 1691
WmvImporter 1693
XImporter 1695
XmlImporter 1697
Microsoft.Xna.Framework.Content.Pipeline.Audio 1699
AudioContent 1700
AudioFileType 1706
AudioFormat 1707
AudioProfile 1710
ConversionFormat 1714
ConversionQuality 1716
Microsoft.Xna.Framework.Content.Pipeline.Builder.Convertors 1717
StringToColorConverter 1718
Microsoft.Xna.Framework.Content.Pipeline.Graphics 1723
ABCFloat 1728
AlphaTestMaterialContent 1730
AnimationChannel 1735
AnimationChannelDictionary 1740
AnimationContent 1742
AnimationContentDictionary 1744
AnimationKeyframe 1746
AstcBitmapContent 1749
AtcBitmapContent 1753
AtcExplicitBitmapContent 1757
AtcInterpolatedBitmapContent 1760
BasicMaterialContent 1763
BitmapContent 1768
BoneContent 1775

BoneWeight 1776
BoneWeightCollection 1778
CharacterRegion 1780
CharacterRegionTypeConverter 1784
DualTextureMaterialContent 1787
Dxt1BitmapContent 1791
Dxt3BitmapContent 1793
Dxt5BitmapContent 1795
DxtBitmapContent 1797
EffectContent 1801
EffectMaterialContent 1803
EnvironmentMapMaterialContent 1805
Etc1BitmapContent 1811
Etc2BitmapContent 1816
FontDescription 1821
FontDescriptionStyle 1826
GeometryContent 1827
GeometryContentCollection 1829
GraphicsUtil 1831
GrowRule 1835
IndexCollection 1836
IndirectPositionCollection 1838
LocalizedFontDescription 1842
MaterialContent 1844
MaxRectsBin 1849
MaxRectsHeuristic 1857
MeshBuilder 1858
MeshContent 1865
MeshHelper 1867
MipmapChain 1874
MipmapChainCollection 1876
NodeContent 1879
NodeContentCollection 1882
PixelBitmapContent<T> 1885
PositionCollection 1891
PvrtcBitmapContent 1892
PvrtcRgb2BitmapContent 1896
PvrtcRgb4BitmapContent 1898
PvrtcRgba2BitmapContent 1900
PvrtcRgba4BitmapContent 1902

SkinnedMaterialContent 1904
SpriteFontContent 1909
Texture2DContent 1913
Texture3DContent 1915
TextureContent 1917
TextureCubeContent 1920
TextureProfile 1922
TextureReferenceDictionary 1925
VertexChannel 1927
VertexChannelCollection 1932
VertexChannelNames 1943
VertexChannel<T> 1950
VertexContent 1954
Microsoft.Xna.Framework.Content.Pipeline.Processors 1960
CompiledEffectContent 1963
EffectProcessor 1965
EffectProcessorDebugMode 1968
FontDescriptionProcessor 1969
FontTextureProcessor 1971
LocalizedFontProcessor 1974
MaterialProcessor 1977
MaterialProcessorDefaultEffect 1984
ModelBoneContent 1985
ModelBoneContentCollection 1987
ModelContent 1988
ModelMeshContent 1990
ModelMeshContentCollection 1993
ModelMeshPartContent 1994
ModelMeshPartContentCollection 1997
ModelProcessor 1998
PassThroughProcessor 2005
SongContent 2007
SongProcessor 2008
SoundEffectContent 2010
SoundEffectProcessor 2011
TextureProcessor 2013
TextureProcessorOutputFormat 2017
VertexBufferContent 2019
VertexDeclarationContent 2024
VideoProcessor 2026

Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler 2028
ContentCompiler 2029
ContentTypeWriter 2032
ContentTypeWriterAttribute 2037
ContentTypeWriter<T> 2039
ContentWriter 2042
SpriteFontContentWriter 2050
Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate 2053
ContentTypeSerializer 2054
ContentTypeSerializer.ChildCallback 2058
ContentTypeSerializerAttribute 2059
ContentTypeSerializer<T> 2061
IntermediateReader 2066
IntermediateSerializer 2076
IntermediateWriter 2079
Microsoft.Xna.Framework.Design 2085
Byte4TypeConverter 2086
Vector2TypeConverter 2091
Vector3TypeConverter 2096
Vector4TypeConverter 2101
Microsoft.Xna.Framework.Graphics 2106
AlphaTestEffect 2115
BasicEffect 2121
Blend 2130
BlendFunction 2132
BlendState 2133
BufferUsage 2142
ClearOptions 2143
ColorWriteChannels 2144
CompareFunction 2145
CubeMapFace 2147
CullMode 2148
DefaultColorProcessors 2149
DepthFormat 2151
DepthStencilState 2152
DeviceLostException 2164
DeviceNotResetException 2166
DirectionalLight 2168
DisplayMode 2171
DisplayModeCollection 2176

DualTextureEffect 2178
DynamicIndexBuffer 2184
DynamicVertexBuffer 2190
Effect 2196
EffectAnnotation 2201
EffectAnnotationCollection 2204
EffectMaterial 2207
EffectParameter 2209
EffectParameterClass 2229
EffectParameterCollection 2230
EffectParameterType 2233
EffectPass 2235
EffectPassCollection 2237
EffectPassCollection.Enumerator 2240
EffectTechnique 2242
EffectTechniqueCollection 2245
EnvironmentMapEffect 2248
FillMode 2256
GraphicsAdapter 2257
GraphicsAdapter.DriverType 2263
GraphicsDebug 2264
GraphicsDebugMessage 2266
GraphicsDevice 2268
GraphicsDeviceStatus 2299
GraphicsMetrics 2300
GraphicsProfile 2304
GraphicsResource 2305
IEffectBones 2309
IEffectFog 2310
IEffectLights 2312
IEffectMatrices 2315
IGraphicsDeviceService 2317
IVertexType 2319
IndexBuffer 2320
IndexElementSize 2332
Model 2333
ModelBone 2338
ModelBoneCollection 2342
ModelBoneCollection.Enumerator 2345
ModelEffectCollection 2347

ModelEffectCollection.Enumerator 2348
ModelMesh 2350
ModelMeshCollection 2353
ModelMeshCollection.Enumerator 2356
ModelMeshPart 2358
ModelMeshPartCollection 2362
NoSuitableGraphicsDeviceException 2364
OcclusionQuery 2366
PresentInterval 2369
PresentationParameters 2370
PrimitiveType 2376
RasterizerState 2377
RenderTarget2D 2383
RenderTargetBinding 2394
RenderTargetCube 2397
RenderTargetUsage 2402
ResourceCreatedEventArgs 2403
ResourceDestroyedEventArgs 2404
SamplerState 2406
SamplerStateCollection 2414
SetDataOptions 2415
ShaderCompilerException 2416
ShaderStage 2420
SkinnedEffect 2421
SpriteBatch 2430
SpriteEffect 2447
SpriteEffects 2450
SpriteFont 2451
SpriteFont.Glyph 2456
SpriteSortMode 2460
StencilOperation 2461
SurfaceFormat 2463
TargetBlendState 2467
Texture 2472
Texture2D 2474
Texture2D.SurfaceType 2494
Texture3D 2495
TextureAddressMode 2506
TextureCollection 2507
TextureCube 2508

TextureFilter 2517
TextureFilterMode 2519
VertexBuffer 2520
VertexBufferBinding 2530
VertexDeclaration 2534
VertexElement 2540
VertexElementFormat 2546
VertexElementUsage 2548
VertexPosition 2550
VertexPositionColor 2554
VertexPositionColorNormal 2559
VertexPositionColorNormalTexture 2564
VertexPositionColorTexture 2570
VertexPositionNormalTexture 2575
VertexPositionTexture 2580
Viewport 2585
Microsoft.Xna.Framework.Graphics.PackedVector 2592
Alpha8 2594
Bgr565 2599
Bgra4444 2605
Bgra5551 2610
Byte4 2615
HalfSingle 2620
HalfVector2 2625
HalfVector4 2630
IPackedVector 2635
IPackedVector<TPacked> 2637
NormalizedByte2 2638
NormalizedByte4 2643
NormalizedShort2 2648
NormalizedShort4 2653
Rg32 2658
Rgba1010102 2663
Rgba64 2668
Short2 2673
Short4 2678
Microsoft.Xna.Framework.Input 2683
ButtonState 2685
Buttons 2686
GamePad 2689

GamePadButtons 2698
GamePadCapabilities 2705
GamePadDPad 2718
GamePadDeadZone 2723
GamePadState 2724
GamePadThumbSticks 2733
GamePadTriggers 2737
GamePadType 2741
Joystick 2743
JoystickCapabilities 2746
JoystickHat 2752
JoystickState 2757
KeyState 2762
Keyboard 2763
KeyboardInput 2765
KeyboardState 2768
Keys 2775
MessageBox 2789
Mouse 2792
MouseCursor 2795
MouseState 2801
Microsoft.Xna.Framework.Input.Touch 2810
GestureSample 2811
GestureType 2814
TouchCollection 2816
TouchCollection.Enumerator 2822
TouchLocation 2824
TouchLocationState 2831
TouchPanel 2832
TouchPanelCapabilities 2837
TouchPanelState 2839
Microsoft.Xna.Framework.Media 2844
Album 2846
AlbumCollection 2851
Artist 2854
Genre 2858
MediaLibrary 2862
MediaPlayer 2866
MediaQueue 2873
MediaSource 2875

MediaSourceType 2877
MediaState 2878
Playlist 2879
PlaylistCollection 2881
Song 2886
SongCollection 2894
Video 2900
VideoPlayer 2903
VideoSoundtrackType 2908
MonoGame.Effect 2909
CommandLineParser 2910
CommandLineParser.NameAttribute 2912
CommandLineParser.ProfileNameAttribute 2915
CommandLineParser.RequiredAttribute 2917
IEffectCompilerOutput 2919
Options 2921
PassInfo 2923
Preprocessor 2931
SamplerStateInfo 2933
ShaderCompilerException 2938
ShaderInfo 2939
ShaderProfile 2940
ShaderResult 2944
TechniqueInfo 2948
TextureFilterType 2950
MonoGame.Framework.Content 2951
IContentProvider 2952
MonoGame.Framework.Content.Pipeline.Builder 2954
ContentBuilder 2956
ContentBuilderMode 2961
ContentBuilderParams 2962
ContentCollection 2968
ContentInfo 2975
ContentRule 2979
FileHelper 2981
IContentCache 2984
IContentCollection 2988
IContentFileCache 2989
PathHelper 2992
PipelineBuildEvent 2995

PipelineBuildEvent.Pair ---- 3002
PipelineBuildLogger ---- 3004
PipelineImporterContext ---- 3007
PipelineManager ---- 3010
PipelineProcessorContext ---- 3025
RegexRule ---- 3037
TypeExtensions ---- 3038
WildcardRule ---- 3041
XmlColor ---- 3042
MonoGame.Framework.Content.Pipeline.Builder.Server ---- 3046
ContentRequestedArgs ---- 3047
ContentServer ---- 3049
ContentServerParameterAttribute ---- 3052
NetworkContentServer ---- 3055
MonoGame.Framework.Devices.Sensors ---- 3057
Accelerometer ---- 3059
AccelerometerFailedException ---- 3062
AccelerometerReading ---- 3063
CalibrationEventArgs ---- 3065
Compass ---- 3066
CompassReading ---- 3069
ISensorReading ---- 3071
SensorBase<TSensorReading> ---- 3072
SensorFailedException ---- 3076
SensorReadingEventArgs<T> ---- 3077
SensorState ---- 3079
MonoGame.Framework.Utilities ---- 3080
GraphicsBackend ---- 3081
MonoGamePlatform ---- 3082
PlatformInfo ---- 3084

Thank you for downloading the MonoGame Complete Reference Guide. We hope it helps you on your game development journey.

If you would like to help to support the continued development and maintenance of MonoGame please consider donating.

© 2009-2026 MonoGame Foundation, Inc. All rights reserved.

This document is provided for personal and educational use only. Redistribution, republishing, or commercial distribution of this PDF, in whole or in part, is strictly prohibited without prior written permission from the MonoGame Foundation.

For the latest documentation, please visit https://docs.monogame.net

Getting Started

This section walks you through the basics of MonoGame, and helps you to create your first game.

First, select the operating system and toolset you will be working with to create your first MonoGame project, then continue reading to understand the basic layout of a MonoGame project.

Supported platforms

MonoGame supports the following platforms by default:

- Windows
- macOS
- Linux
- iOS
- iPadOS
- Android
- Xbox*
- PlayStation 4*
- PlayStation 5*
- Nintendo Switch*

> **IMPORTANT**
>
> Platforms marked with a "*" require developer access from their host vendor platforms.

See Supported Platforms for more information.

Setting up and creating your first MonoGame Project

By the end of this tutorial set, you will have a working project to build for your target platform and will be ready to tackle your next steps.

1. Setting up your OS for development

- Windows
- macOS
- Ubuntu

2. Choosing your IDE for development

- Visual Studio for Windows
- VS Code
- Rider

3. Packaging Games

- Packaging Games for Distribution

4. Advanced Topics

- Preparing for Consoles
- Using Development Nuget Packages

Setting up your OS for development on MacOS

This section provides a step-by-step guide for setting up your development environment on Mac.

To develop with MonoGame in C#, you will need to install the .NET SDK. As of MonoGame 3.8.4.1 the recommended version is .NET 9, .NET 10 is also supported.

> **TIP**
>
> You can still continue to use .NET 8 if you wish, you will just need to downgrade your client project .NET version in the `csproj` setup for your project (if using the default templates)

Install .NET 9 SDK

1. Navigate to https://dotnet.microsoft.com/en-us/download/dotnet/9.0

2. Download the `.NET SDK` Installer for your current platform of `macOS`

3. Once the **.pkg** file finishes downloading, run it and follow the prompts to install the .NET SDK

Download .NET 9.0

9.0.10 Security patch

Release notes **Latest release date** October 14, 2025

Build apps - SDK

SDK 9.0.306

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

Visual Studio support
Visual Studio 2022 (v17.14)

Included in
Visual Studio 17.14.17

Included runtimes
.NET Runtime 9.0.10
ASP.NET Core Runtime 9.0.10
.NET Desktop Runtime 9.0.10

Language support
C# 13.0
F# 9.0
Visual Basic 17.13

SDK 9.0.111

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

Visual Studio support
Visual Studio 2022 (v17.12)

Included in
Visual Studio 17.12.13

Included runtimes
.NET Runtime 9.0.10
ASP.NET Core Runtime 9.0.10
.NET Desktop Runtime 9.0.10

Language support
C# 13.0
F# 9.0
Visual Basic 17.13

Run apps - Runtime

ASP.NET Core Runtime 9.0.10

The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommend installing the Hosting Bundle, which includes the .NET Runtime and IIS support.**

IIS runtime support (ASP.NET Core Module v2)
19.0.25269.10

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS		Arm64 | x64
Windows	x64 | x86 | Arm64 | Hosting Bundle | winget instructions	x64 | x86 | Arm64

.NET Desktop Runtime 9.0.10

The .NET Desktop Runtime enables you to run existing Windows desktop applications. **This release includes the .NET Runtime; you don't need to install it separately.**

OS	Installers	Binaries
Windows	x64 | x86 | Arm64 | winget instructions	

.NET Runtime 9.0.10

The .NET Runtime contains just the components needed to run a console app. Typically, you'd also install either the ASP.NET Core Runtime or .NET Desktop Runtime.

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

Install additional workloads

If you intend to also work with platforms such as `Android` or `iOS`, you will need to install the additional .NET workload templates for those platforms which include additional features and simulators to support those platforms. Just run the following commands from the terminal for the platforms below you wish to install.

> **IMPORTANT**
>
> For mobile development with iOS and Android, you must also install the MAUI workload even though MonoGame does not use MAUI. The MAUI workload contains the debugging tools required to run and debug mobile .NET applications. Without it, you will not be able to properly debug your MonoGame mobile projects.

```
dotnet workload install android
```

```
dotnet workload install ios
```

```
dotnet workload install maui
```

```
dotnet workload install android ios maui
```

> **NOTE**
>
> You can use `dotnet workload search` to detect any other available workloads you wish to use.

Setup Wine For Effect Compilation

Effect (shader) compilation requires access to DirectX. This means it will not work natively on macOS and Linux systems, but it can be used through Wine.

MonoGame provides a setup script that can be executed to setup the Wine environment for Effect (shader) compilation.

1. Install brew.sh

2. Enter the following command:

```
brew install wget p7zip curl && brew install --cask wine-stable && xattr -dr com.apple.quarantine "/Applications/Wine Stable.app"
```

3. Now that the prerequisites are installed, download the mgfxc_wine_setup.sh script and execute it by entering the following command in the terminal:

```
wget -qO- https://monogame.net/downloads/net9_mgfxc_wine_setup.sh | bash
```

This will create new directory called `.winemonogame` in your home directory. If you ever wish to undo the setup this script performed, just simply delete that directory.

Next Steps

Choose from one of the two IDE options on macOS:

- Setting up VSCode
- Setting up Rider

Setting up your OS for development on Arch Linux

This section provides a step-by-step guide for setting up your development environment on Arch Linux.

> **TIP**
>
> Arch Linux is a rolling release distribution. This guide was tested in October 2025 with kernel `version 6.15.9-arch1-1`.

To develop with MonoGame in C#, you will need to install the .NET 9 SDK. As of MonoGame 3.8.4.1 the recommended version is .NET 9, .NET 10 is also supported.

> **TIP**
>
> You can still continue to use .NET 8 if you wish, you will just need to downgrade your client project .NET version in the `csproj` setup for your project (if using the default templates)

Install .NET 9 SDK

1. Open a new **Terminal** window.
2. Enter the following command in the terminal to install the latest .NET 9 SDK:

```
sudo pacman -Syu
sudo pacman -S dotnet-sdk-9.0
```

Install additional workloads

If you intend to also work with platforms such as `Android` or `iOS`, you will need to install the additional .NET workload templates for those platforms which include additional features and simulators to support those platforms. Just run the following commands from the terminal for the platforms below you wish to install.

> **IMPORTANT**
>
> For mobile development with iOS and Android, you must also install the MAUI workload even though MonoGame does not use MAUI. The MAUI workload contains the debugging tools required to run and debug mobile .NET applications. Without it, you will not be able to properly debug your MonoGame mobile projects.

```
dotnet workload install android
```

```
dotnet workload install ios
```

```
dotnet workload install maui
```

```
dotnet workload install android ios maui
```

> **NOTE**
>
> You can use `dotnet workload search` to detect any other available workloads you wish to use.

Setup Wine For Effect Compilation

Effect (shader) compilation requires access to DirectX. This means it will not work natively on macOS and Linux systems, but it can be used through Wine.

MonoGame provides a setup script that can be executed to setup the Wine environment for Effect (shader) compilation.

1. Open a terminal window
2. Install Wine and required dependencies

```
sudo pacman -S wget curl 7zip wine
```

> **IMPORTANT**
>
> Arch Linux recently transitioned the Wine package to a pure wow64 build to align with upstream Wine development. Because of this, when you install Wine using pacman, you will not get a separate `wine64` executable that the MonoGame tools will expect.
>
> Reference: https://archlinux.org/news/transition-to-the-new-wow64-wine-and-wine-staging/
>
> This means you will need to create a symlink for MonoGame to reference with the following command
>
> ```
> sudo ln -s /usr/bin/wine /usr/local/bin/wine64
> ```

3. Now that the prerequisites are installed, download the mgfxc_wine_setup.sh script and execute it by entering the following command in the terminal:

```
wget -qO- https://monogame.net/downloads/net9_mgfxc_wine_setup.sh | bash
```

> **TIP**
>
> When running the Wine setup script, you may see a popup asking to install **Wine Mono** for .NET support. You can cancel/close this popup. It's not needed because the MonoGame script installs the actual Windows .NET SDK instead.

> **NOTE**
>
> This script will create a new directory called `.winemonogame` in your home directory. If you ever wish to undo the setup this script performed, just simply delete that directory.

4. After the script completes, you must add the `MGFXC_WINE_PATH` environment variable to your **system environment** for it to be accessible to all applications:

```
echo 'MGFXC_WINE_PATH="'$HOME'/.winemonogame"' | sudo tee -a /etc/environment
```

> **IMPORTANT**
>
> Unlike other Linux distributions, such as Ubuntu, Arch Linux may not automatically load environment variables from your user profile at `~/.profile` for all applications. Adding `MGFXC_WINE_PATH` to `/etc/environment` ensures it is available system-wide, including for GUI applications (such as the MGCB Editor) and IDEs

5. For the environment variable to take effect, you can either log out and back in (recommended) or reboot your system.

6. After logging back in, verify the environment variable is set:

```
echo $MGFXC_WINE_PATH
```

> **NOTE**
>
> When running the above command you should see an output similar to the following
>
> ```
> /home/yourusername/.winemonogame
> ```

Next Steps

Choose from one of the two IDE options on Arch Linux:

- Setting up VSCode
- Setting up Rider

Troubleshooting

Issue: "wine64 not found error"

If the setup script fails with `wine64 not found`, ensure you created the symlink from step 2 above

```
sudo ln -s /usr/bin/wine /usr/local/bin/wine64
```

Issue: Shader compilation fails with "MGFXC effect compiler requires a valid wine installation"

This means the `MGFXC_WINE_PATH` environment variable is not set. Ensure you:

1. Added it to `/etc/environment` as shown above in step 4
2. Logged out and logged back in
3. Can see it with `echo $MGFXC_WINE_PATH`

Issue: Wine creates a `.wine-mgfxc` directory instead of using `.winemonogame`

This is another symptom of the environment variable not being set. Follow the steps in the previous issue to resolve it.

Issue: Wine version too old

The MonoGame setup script requires Wine 8.0 or later. Check your version:

```
wine --version
```

If it's too old, update your system:

```
sudo pacman -Syu
```

Setting up your OS for development on Ubuntu

This section provides a step-by-step guide for setting up your development environment on Ubuntu.

> **TIP**
>
> The minimum version of Ubuntu that is supported by MonoGame is `20.04`.

To develop with MonoGame in C#, you will need to install the .NET SDK. As of MonoGame 3.8.4.1 the minimum supported version is .NET 9.

Install .NET 9 SDK

1. Open a new **Terminal** window.
2. Enter the following command in the terminal to install the latest .NET 9 SDK:

```
sudo apt-get update && sudo apt-get install -y dotnet-sdk-9.0
```

Install additional workloads

If you intend to also work with platforms such as `Android` or `iOS`, you will need to install the additional .NET workload templates for those platforms which include additional features and simulators to support those platforms. Just run the following commands from the terminal for the platforms below you wish to install.

> **IMPORTANT**
>
> For mobile development with iOS and Android, you must also install the MAUI workload even though MonoGame does not use MAUI. The MAUI workload contains the debugging tools required to run and debug mobile .NET applications. Without it, you will not be able to properly debug your MonoGame mobile projects.

```
dotnet workload install android
```

```
dotnet workload install ios
```

```
dotnet workload install maui
```

```
dotnet workload install android ios maui
```

> **NOTE**
>
> You can use `dotnet workload search` to detect any other available workloads you wish to use.

Setup Wine For Effect Compilation

Effect (shader) compilation requires access to DirectX. This means it will not work natively on macOS and Linux systems, but it can be used through Wine.

MonoGame provides a setup script that can be executed to setup the Wine environment for Effect (shader) compilation.

1. Download the latest wine for your version of Ubuntu from: https://gitlab.winehq.org/wine/wine/-/wikis/Debian-Ubuntu

2. Open a terminal window and enter the following command

```
sudo apt install wget curl p7zip-full
```

> **TIP**
> If you receive an error stating that either of the packages do not have an install candidate, you may need to enable the universe apt repository. To do this, enter the following commands in the terminal
>
> ```
> sudo add-apt-repository universe
> sudo apt update
> ```
>
> Then try installing the packages again.

3. Now that the prerequisites are installed, download the mgfxc_wine_setup.sh script and execute it by entering the following command in the terminal:

```
wget -qO- https://monogame.net/downloads/net9_mgfxc_wine_setup.sh | bash
```

This will create new directory called `.winemonogame` in your home directory. If you ever wish to undo the setup this script performed, just simply delete that directory.

Next Steps

Choose from one of the two IDE options on Ubuntu:

- Setting up VSCode
- Setting up Rider

Setting up your OS for development on Windows

This section provides a step-by-step guide for setting up your development environment on Windows.

To develop with MonoGame in C#, you must install the .NET SDK. As of MonoGame 3.8.4.1 the minimum supported version is .NET 9, .NET 10 is also supported.

> **IMPORTANT**
>
> If your chosen IDE for development is Visual Studio then you can skip this step as .NET 8 is included with the Visual Studio Installer.

Install .NET 9 SDK

You can follow the instructions below based on your operating system to install the .NET 9.0 SDK

1. Navigate to https://dotnet.microsoft.com/en-us/download/dotnet/9.0

Download .NET 9.0

9.0.10 Security patch

Release notes **Latest release date** October 14, 2025

Build apps - SDK

SDK 9.0.306

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

Visual Studio support
Visual Studio 2022 (v17.14)

Included in
Visual Studio 17.14.17

Included runtimes
.NET Runtime 9.0.10
ASP.NET Core Runtime 9.0.10
.NET Desktop Runtime 9.0.10

Language support
C# 13.0
F# 9.0
Visual Basic 17.13

SDK 9.0.111

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

Visual Studio support
Visual Studio 2022 (v17.12)

Included in
Visual Studio 17.12.13

Included runtimes
.NET Runtime 9.0.10
ASP.NET Core Runtime 9.0.10
.NET Desktop Runtime 9.0.10

Language support
C# 13.0
F# 9.0
Visual Basic 17.13

Run apps - Runtime

ASP.NET Core Runtime 9.0.10

The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommend installing the Hosting Bundle, which includes the .NET Runtime and IIS support.**

IIS runtime support (ASP.NET Core Module v2)
19.0.25269.10

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS		Arm64 | x64
Windows	x64 | x86 | Arm64 | Hosting Bundle | winget instructions	x64 | x86 | Arm64

.NET Desktop Runtime 9.0.10

The .NET Desktop Runtime enables you to run existing Windows desktop applications. **This release includes the .NET Runtime; you don't need to install it separately.**

OS	Installers	Binaries
Windows	x64 | x86 | Arm64 | winget instructions	

.NET Runtime 9.0.10

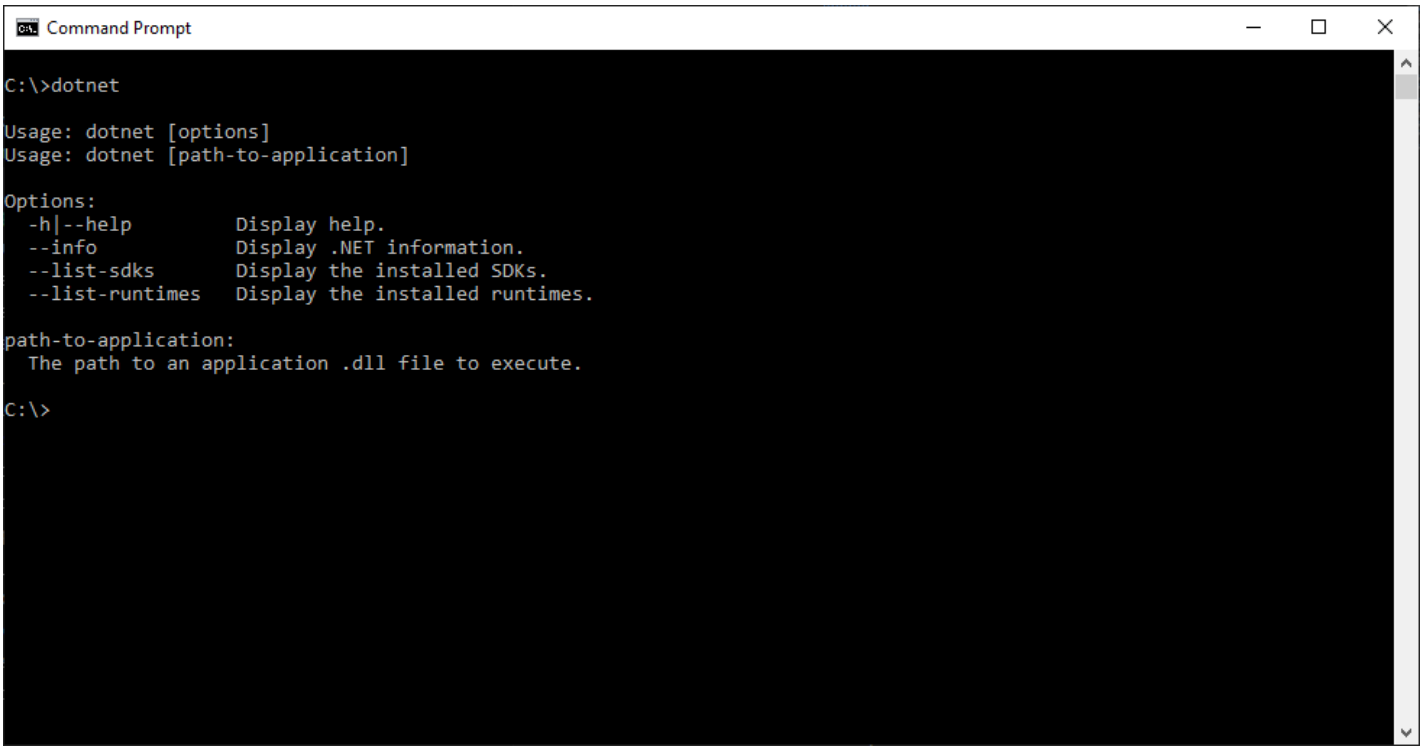
The .NET Runtime contains just the components needed to run a console app. Typically, you'd also install either the ASP.NET Core Runtime or .NET Desktop Runtime.

OS	Installers	Binaries
Linux	Package manager instructions	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	x64 | x86 | Arm64 | winget instructions	x64 | x86 | Arm64
All	dotnet-install scripts	

2. Click the **.NET SDK x64** download button to download. This will take you to the download page where the **dotnet-sdk-9.x.yyy-win-x64.exe** will download.

3. When the download completes, run the **dotnet-sdk-9.x.yyy-win-x64.exe** installer and complete the steps to install .NET on your machine.

4. When the installation completes, open a new **Command Prompt** window and run the command `dotnet` to verify the installation was successful.



Install additional workloads

If you intend to also work with platforms such as `Android` or `iOS`, you will need to install the additional .NET workload templates for those platforms which include additional features and simulators to support those platforms. Just run the command-line command for the platforms below you wish to install.

> **IMPORTANT**
>
> For mobile development with iOS and Android, you must also install the MAUI workload even though MonoGame does not use MAUI. The MAUI workload contains the debugging tools required to run and debug mobile .NET applications. Without it, you will not be able to properly debug your MonoGame mobile projects.

```
dotnet workload install android
```

```
dotnet workload install ios
```

```
dotnet workload install maui
```

```
dotnet workload install android ios maui
```

> **NOTE**
>
> You can use `dotnet workload search` to detect any other available workloads you wish to use.

Next Steps

Choose from one of the three IDE options on Windows:

- Setting up Visual Studio
- Setting up VSCode
- Setting up Rider

Setting up Visual Studio on Windows

This section provides a step-by-step guide for setting up your development IDE on Windows for Visual Studio.

MonoGame can work with most .NET compatible tools, but we recommend Visual Studio 2022

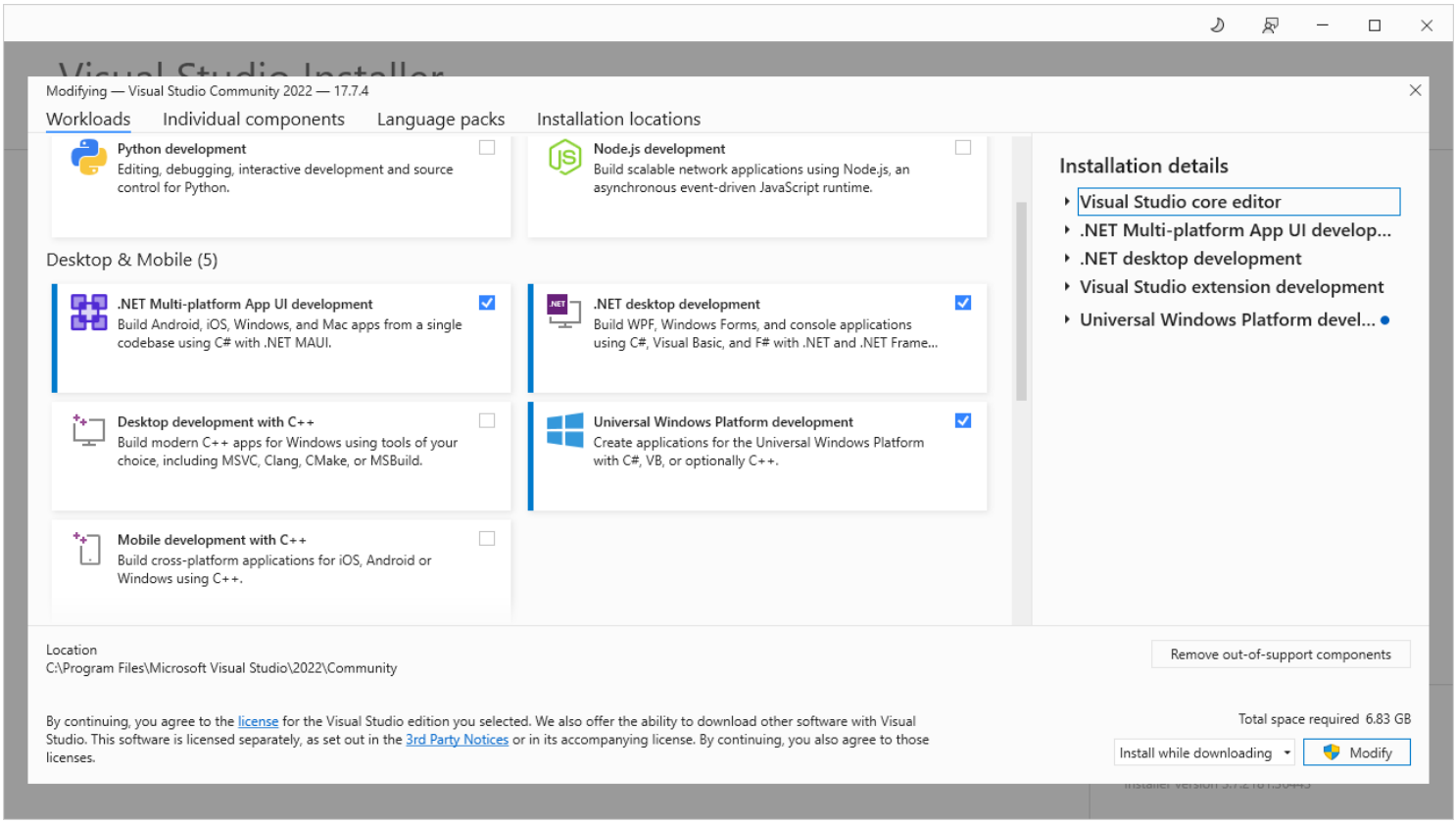
> **NOTE**
>
> Alternatively, you can use JetBrains Rider or Visual Studio Code.
>
> Check out the guides for Setting up VSCode / Setting up Rider using these links.

Install Visual Studio 2022

Before using MonoGame with Visual Studio you need to ensure you have installed the latest Visual Studio 2022 along with the required components.

When installing Visual Studio, the following workloads are required depending on your desired target platform(s):

- Mandatory for all platforms:
  - **.Net desktop development**
- Optional
  - **.Net Multi-platform App UI Development** if you wish to target Android, iOS, or iPadOS.
  - **Universal Windows Platform development** if you wish to build for Windows store or Xbox.



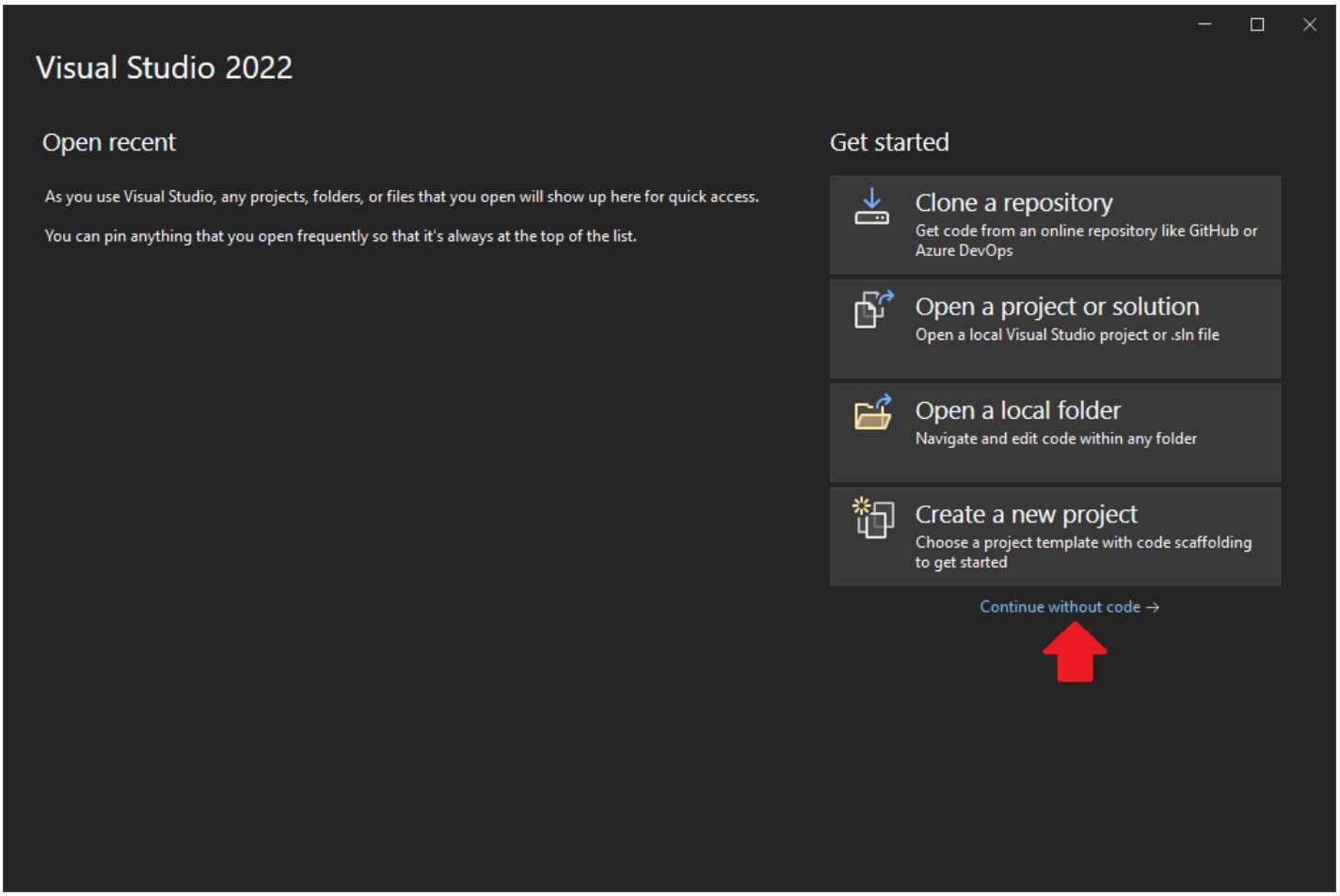
> **WARNING**
>
> **Targeting Windows**
>
> If you are targeting the standard Windows DirectX backend, you will also need the DirectX June 2010 runtime for audio and gamepads to work properly.
>
> Work is underway however to update to support DirectX 12 on Windows and Xbox very soon.

Install MonoGame extension for Visual Studio 2022

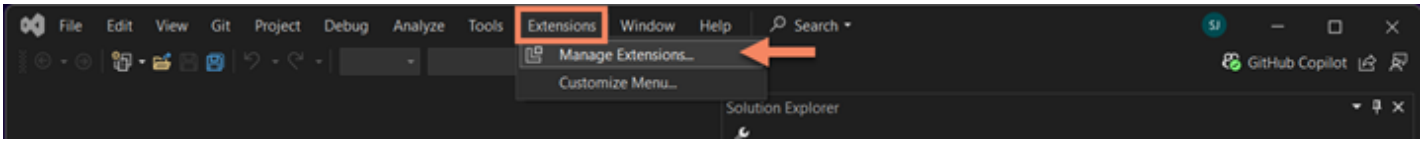
To create new MonoGame projects from within Visual Studio 2022, you will need to install the **MonoGame Framework C# project templates** extension. The following steps demonstrate how to install the extension.

1. Launch Visual Studio 2022

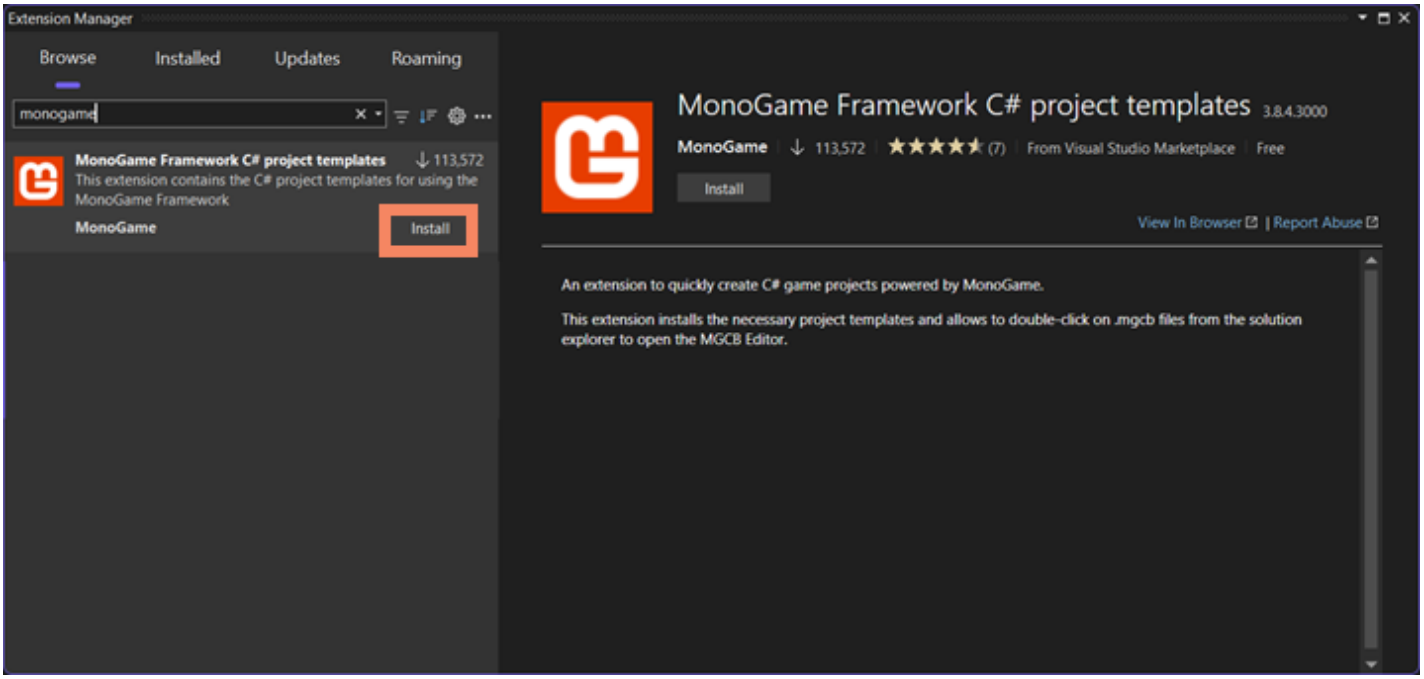
2. Select **Continue without code**. This will launch Visual Studio without any project or solution opened.



3. Click "*Extensions -> Manage Extensions* in the Visual Studio 2022 menu bar. This will open the Manage Extensions dialog window.

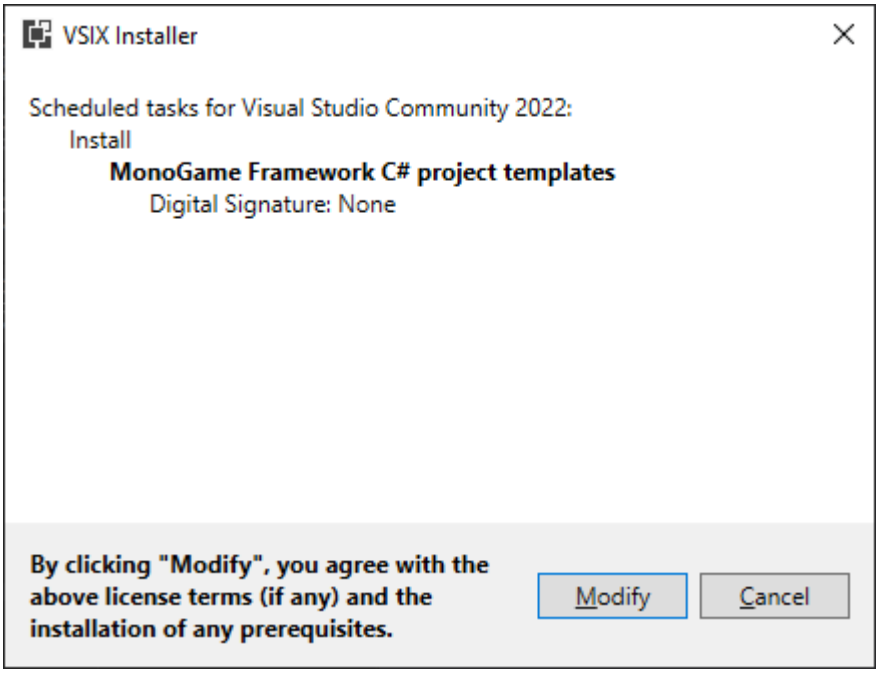


4. Use the search box in the top-right corner of the Manage Extensions dialog window to search for **MonoGame**, then click the **MonoGame Framework C# project templates** extension as shown below and download it to install it.



5. After it is downloaded, an alert will appear at the bottom of the Manage Extensions window that states "Your changes will be scheduled. The modifications will begin when all Microsoft Visual Studio windows are closed." Click the **Close** button, then close Visual Studio 2022.

6. After closing Visual Studio 2022, a VSIX Installer window will open confirming that you want to install the **MonoGame Framework C# project templates** extension. Click the **Modify** button to accept the install.



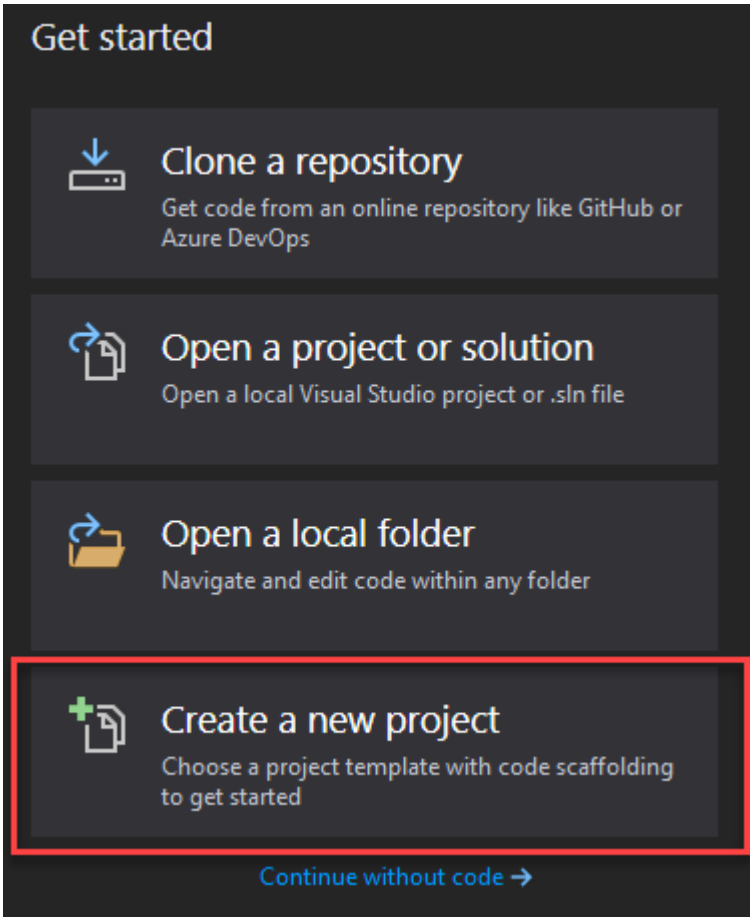
You now have the MonoGame templates installed and are ready to create new projects.

> **TIP**
>
> Alternatively, consider using the Preview Packages provided by MonoGame to get access to the latest developments.
>
> - How to install MonoGame Preview packages

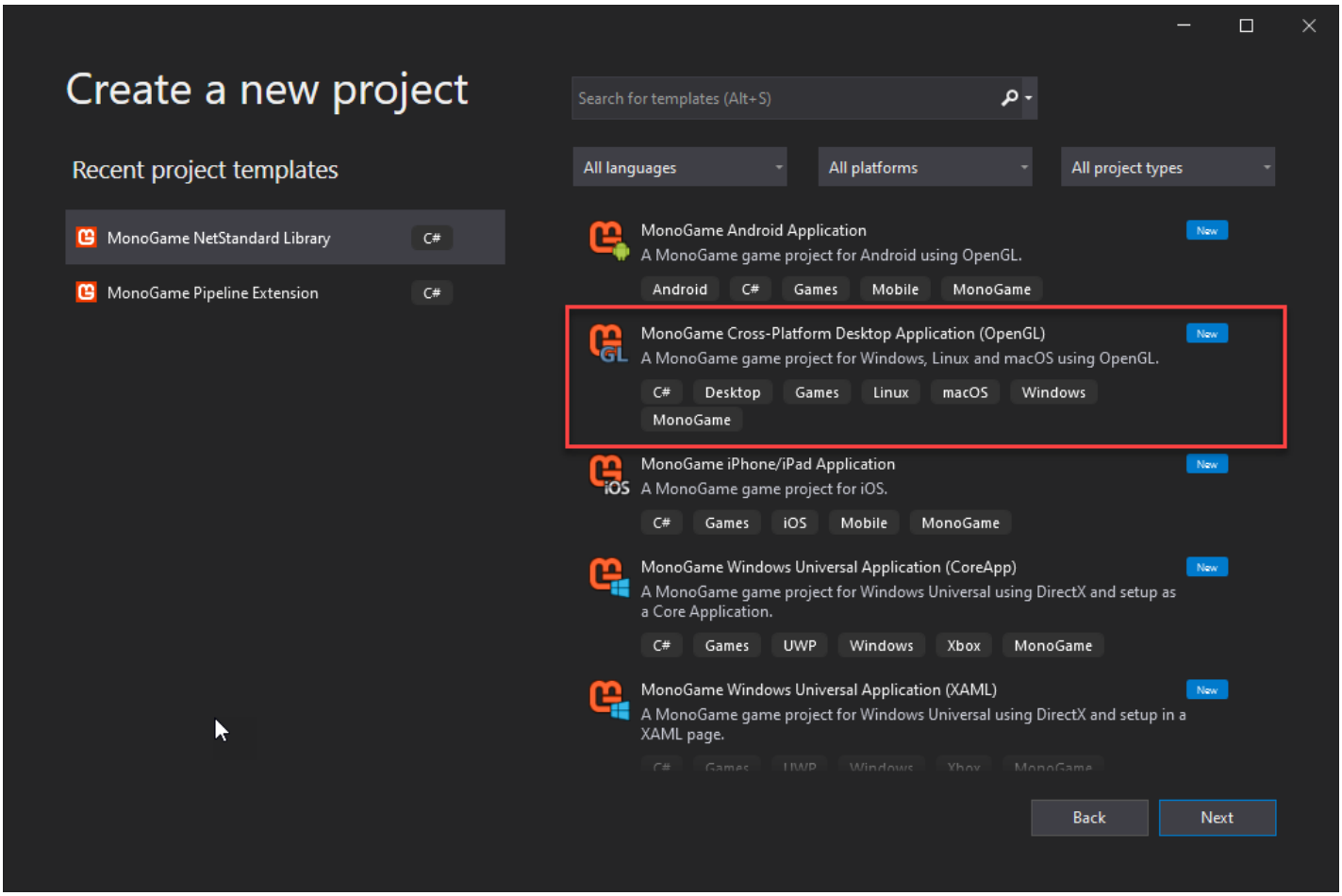
Creating a new MonoGame project

To get you started with Visual Studio, here are the steps for setting up a new MonoGame project.

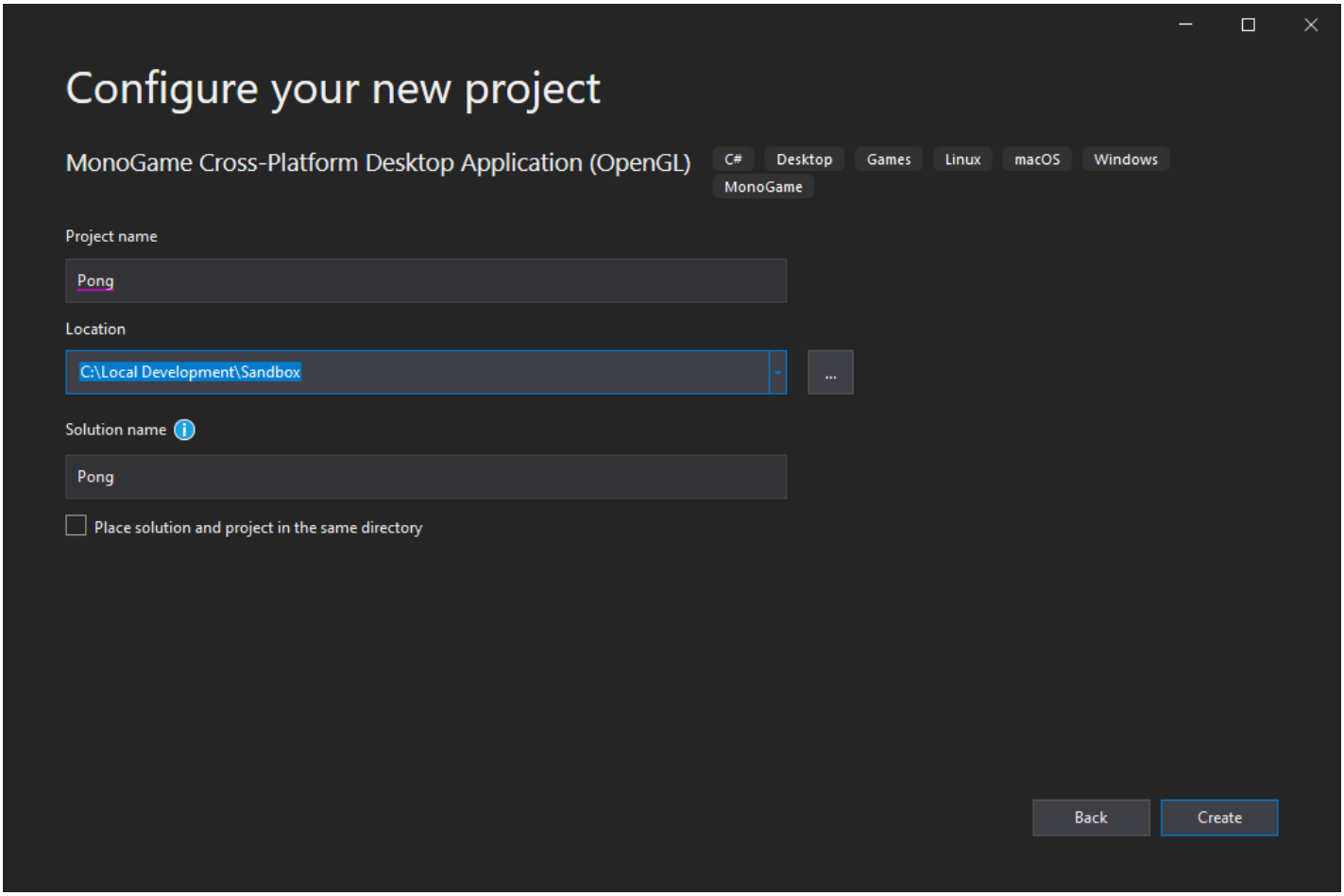
1. Start Visual Studio 2022 and select **New Project...** in the upper left corner.



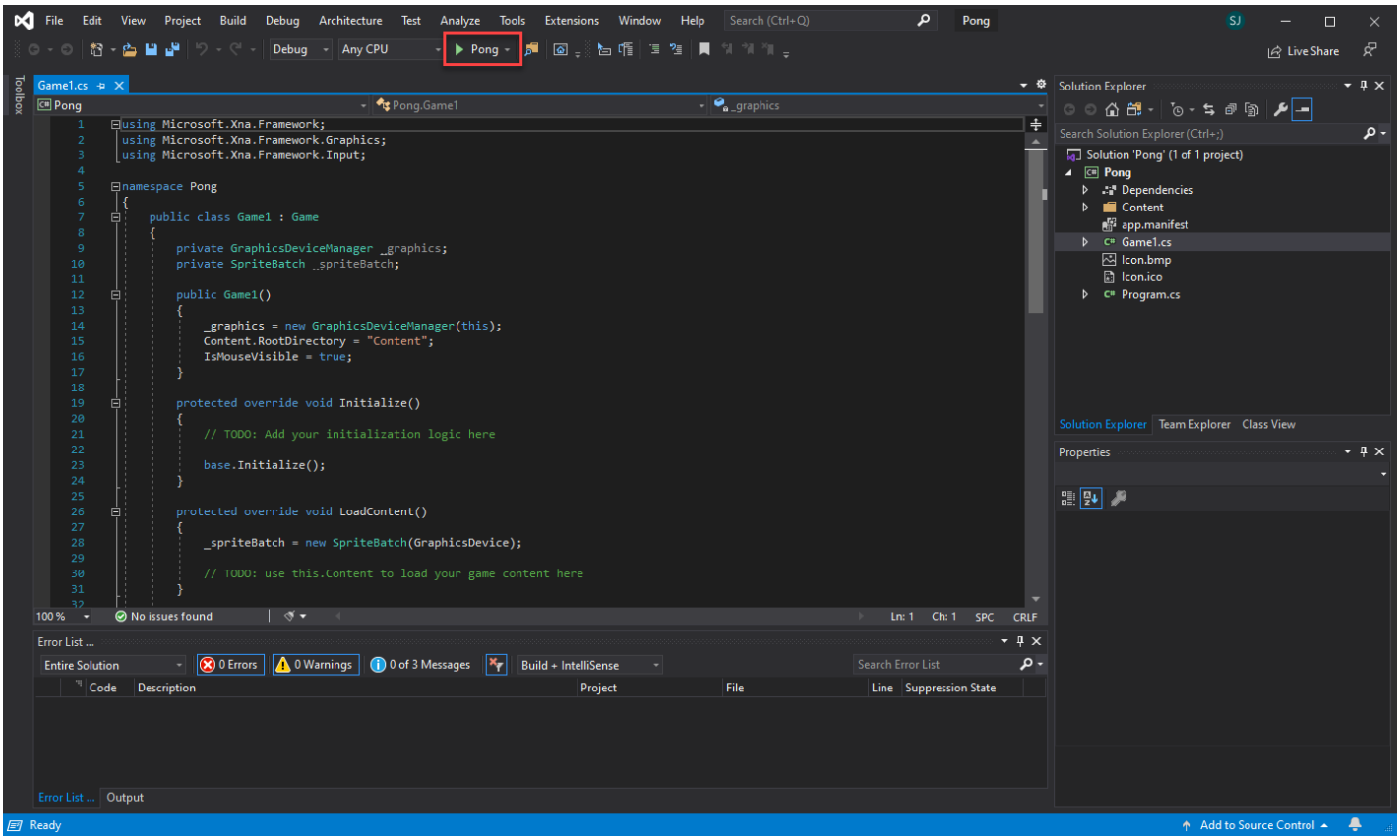
2. You should see the "Create a new project" dialog pop up. From here, select the **Templates > Visual C# > MonoGame** category, and then select **MonoGame Cross Platform Desktop Project**.



3. Next, type in a name for your project. For this tutorial, **Pong** will be used (note: project names cannot contain spaces). After you have entered the name, click on the **...** button next to the Location text field and choose the folder you want to save the project in. Finally, click **OK** to create the project.



4. If everything went correctly, you should see a project named **Pong** open up like in the picture below. To run your game, simply press the big **Play Button** in the toolbar, or press **F5**.



5. You should now see your game window running.

Setting up your development environment for Rider

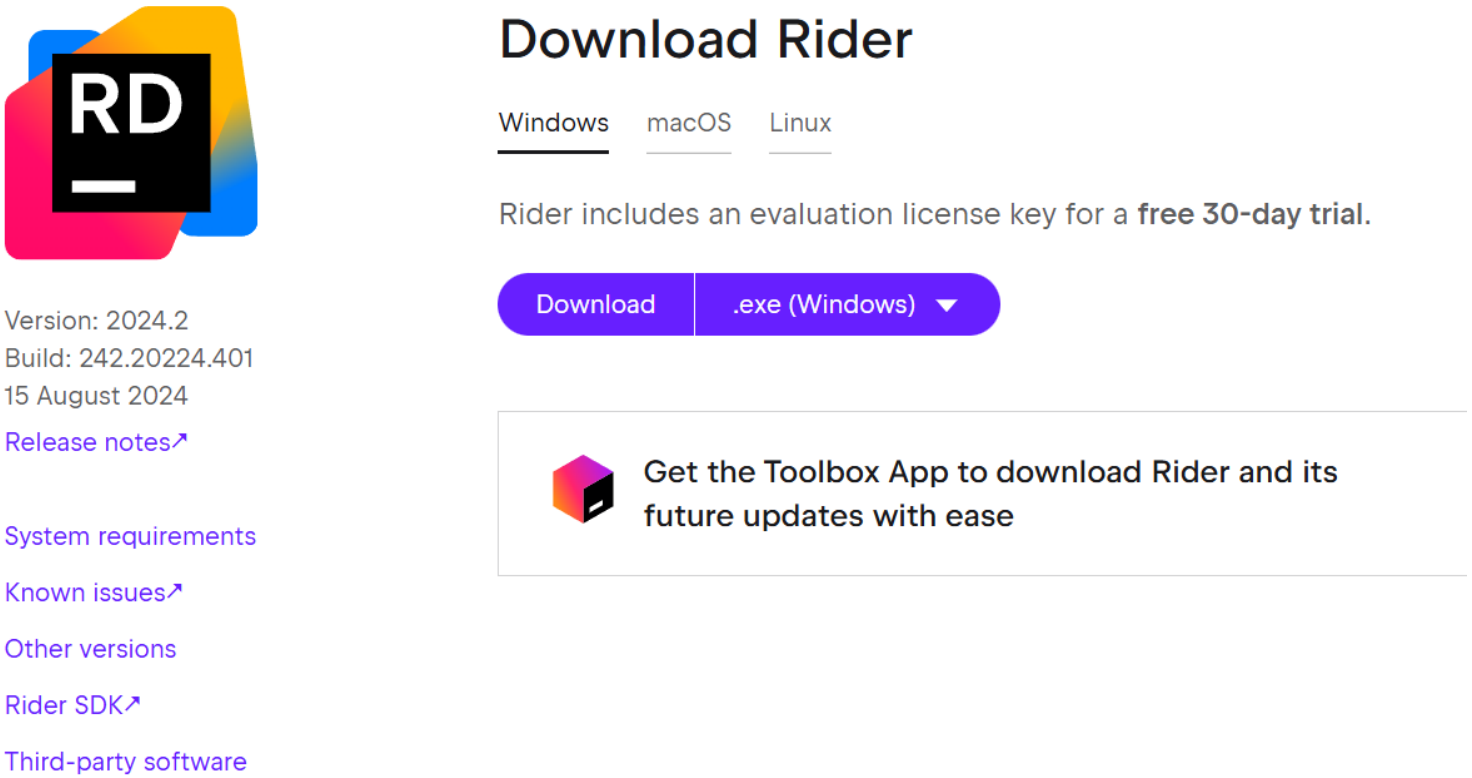
A step-by-step guide for setting up your development IDE for Rider

This is a guide on setting up your development environment for creating games with MonoGame using Rider by JetBrains. By following this guide, you will learn how to install the tools for developing C# applications and configure Rider with recommended extensions for C# development.

By the end, you will be fully equipped to start creating games with MonoGame using Rider.

Install Rider

You can download and install Rider from: https://www.jetbrains.com/rider/download



> **NOTE**
>
> Rider is free for personal use, you do have to register and create an account, but otherwise all features will be unlocked.

Setting up Rider for development with MonoGame

1. Open up terminal (or Powershell on Windows)
2. Run the following command to install templates:

```
dotnet new install MonoGame.Templates.CSharp
```

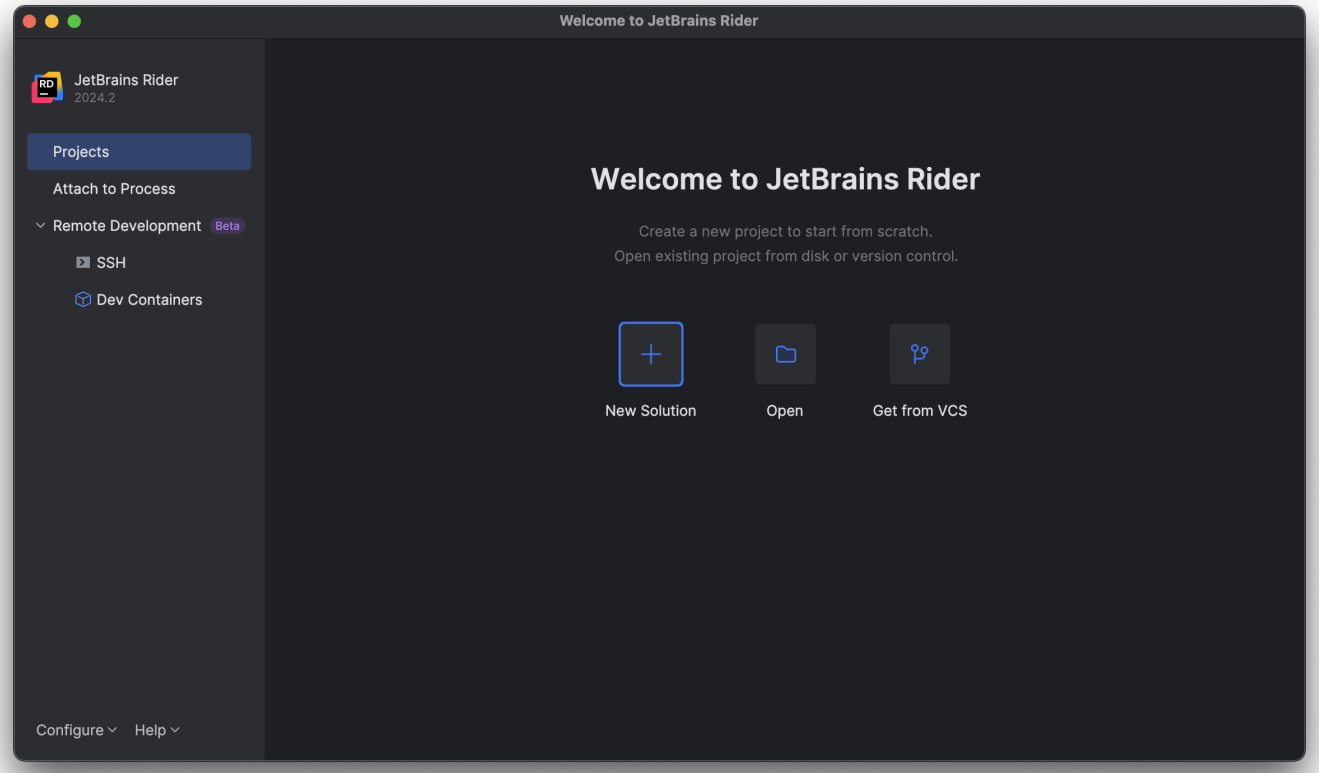
> **TIP**
>
> Alternatively, consider using the Preview Packages provided by MonoGame to get access to the latest developments.
>
> - How to install MonoGame Preview packages

Creating a new MonoGame project

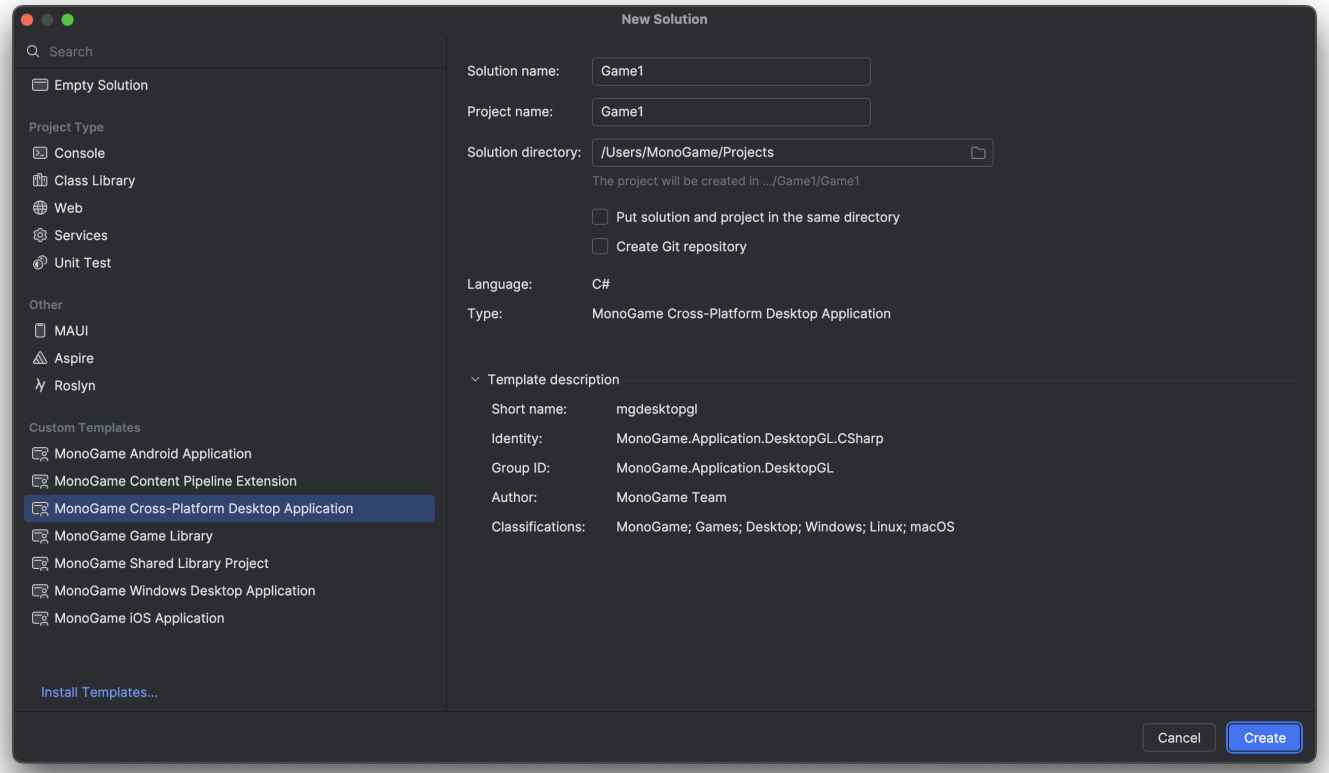
To get you started with Rider, here are the steps for setting up a new Rider MonoGame project.

1. Open up Rider

2. Click on the "New Solution" button



3. Select "MonoGame Cross-Platform Desktop Application" on the list on the left



4. Press "Create"

5. You can now press `F5` to compile and debug your game, happy coding :)

> **NOTE**
>
> If you are experiencing issues with compiling effects under Linux for JetBrains Rider, you might need to add the `MGFXC_WINE_PATH` environment variable to `/etc/environment` for it to be picked up. See 5777151094034-Rider-cannot-see-all-Environmental-Variables for details.

Update Project Tool references

The MonoGame Content Editor (MGCB) it a tool delivered through NuGet for your project using the tools configuration held in your `dotnet-tools.json` file (located in the `.config` folder of your project).

Once you have created your project you should run the following terminal/command-line command to ensure the tool (and the pipeline) is setup and read for your project:

```
dotnet tool restore
```

> **NOTE**
>
> If you ever change the version of the tools or want to upgrade them by editing the `dotnet-tools.json` configuration, you MUST run this command again to update the tools.

Setting up your development environment for VSCode

A step-by-step guide for setting up your development IDE using VSCode

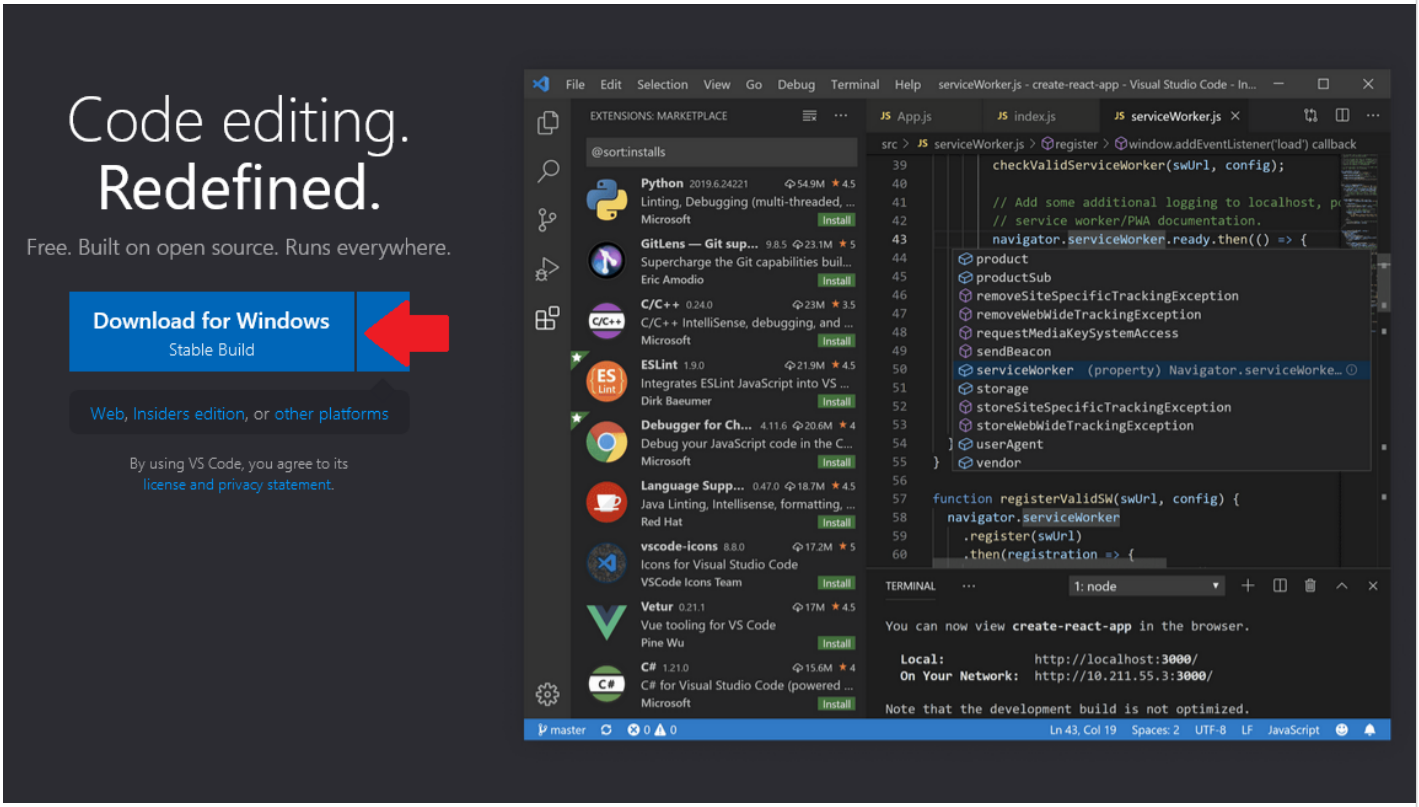
This is a guide on setting up your development IDE for creating games with MonoGame using Visual Studio Code. By following this guide, you will learn how to install the necessary tools for developing C# applications and configure Visual Studio Code with recommended extensions for C# development.

By the end, you will be fully equipped to start creating games with MonoGame using Visual Studio Code.

Install Visual Studio Code

Windows

1. Open your web browser and navigate to https://code.visualstudio.com/.

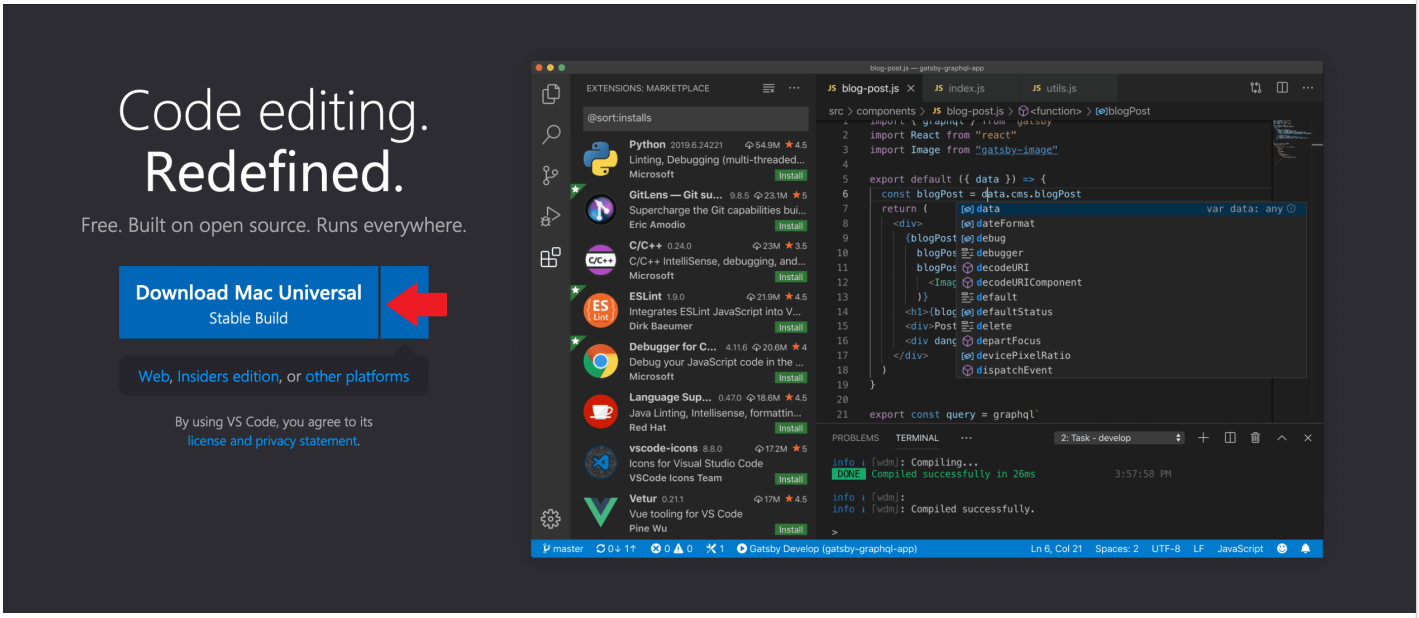


2. Click the **Download for Windows** button. This will redirect you to the download page where the installer will automatically download.

3. When the download completes, run the installer and complete the steps to install Visual Studio Code.

macOS

1. Open your web browser and navigate to https://code.visualstudio.com/.



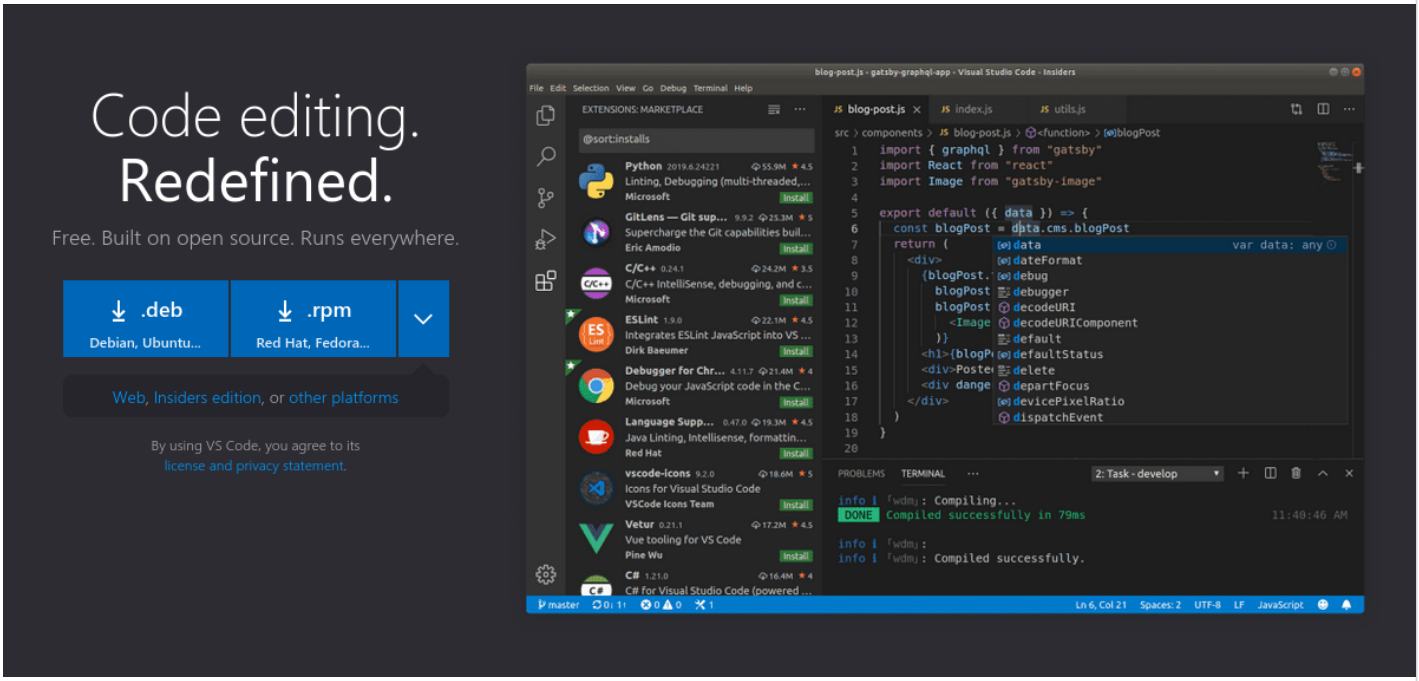
2. Click the **Download Mac Universal** button. This will redirect you to the page where the application archive (.zip) file will begin downloading.

3. Extract the contents of the VSCode archive that downloaded by double-clicking it inside a Finder window. This will extract the **Visual Studio Code.app** file.

4. Drag the **Visual Studio Code.app** file into the **Applications** folder, making it available in the macOS Launchpad.

Ubuntu

1. Open your web browser and navigate to https://code.visualstudio.com/.



2. Click the Download `.deb` button.

3. Double click the `.deb` file and press the `Install` button.

Arch Linux

On Arch Linux, you have two options for installing Visual Studio Code:

Option 1: Code OSS (Open Source Version)

Code OSS is the fully open-source version available under the MIT License. It's available in the official Arch repositories and can be installed with a single command:

```
sudo pacman -S code
```

> **IMPORTANT**
>
> Code OSS **cannot** use the proprietary Microsoft **C# Dev Kit** extension. However, it can use the base **C#** extension (formerly OmniSharp, now LSP-based), which provides IntelliSense, syntax highlighting, debugging, and code navigation, sufficient for MonoGame development.

Option 2: Visual Studio Code (Official Microsoft Version) - Recommended

The official Microsoft version includes proprietary features like settings sync, Microsoft account integration, and **support for the C# Dev Kit extension**. It's available through the AUR (Arch User Repository).

1. First, ensure you have an AUR helper installed. If you don't have one, install `yay`:

```
sudo pacman -S git base-devel
git clone https://aur.archlinux.org/yay.git
cd yay
makepkg -si
```

2. Install Visual Studio Code using yay:

```
yay -S visual-studio-code-bin
```

> **TIP**
>
> For the best MonoGame development experience with full C# Dev Kit support (enhanced project management, integrated testing, better solution support), install the official Visual Studio Code. Code OSS works but has limitations with Microsoft extensions.

Setting up VS Code for development with MonoGame

1. Open Visual Studio Code
2. Open up its terminal window (`Ctrl` + `` ` ``)
3. Run the following command to install MonoGame Templates

```
dotnet new install MonoGame.Templates.CSharp
```

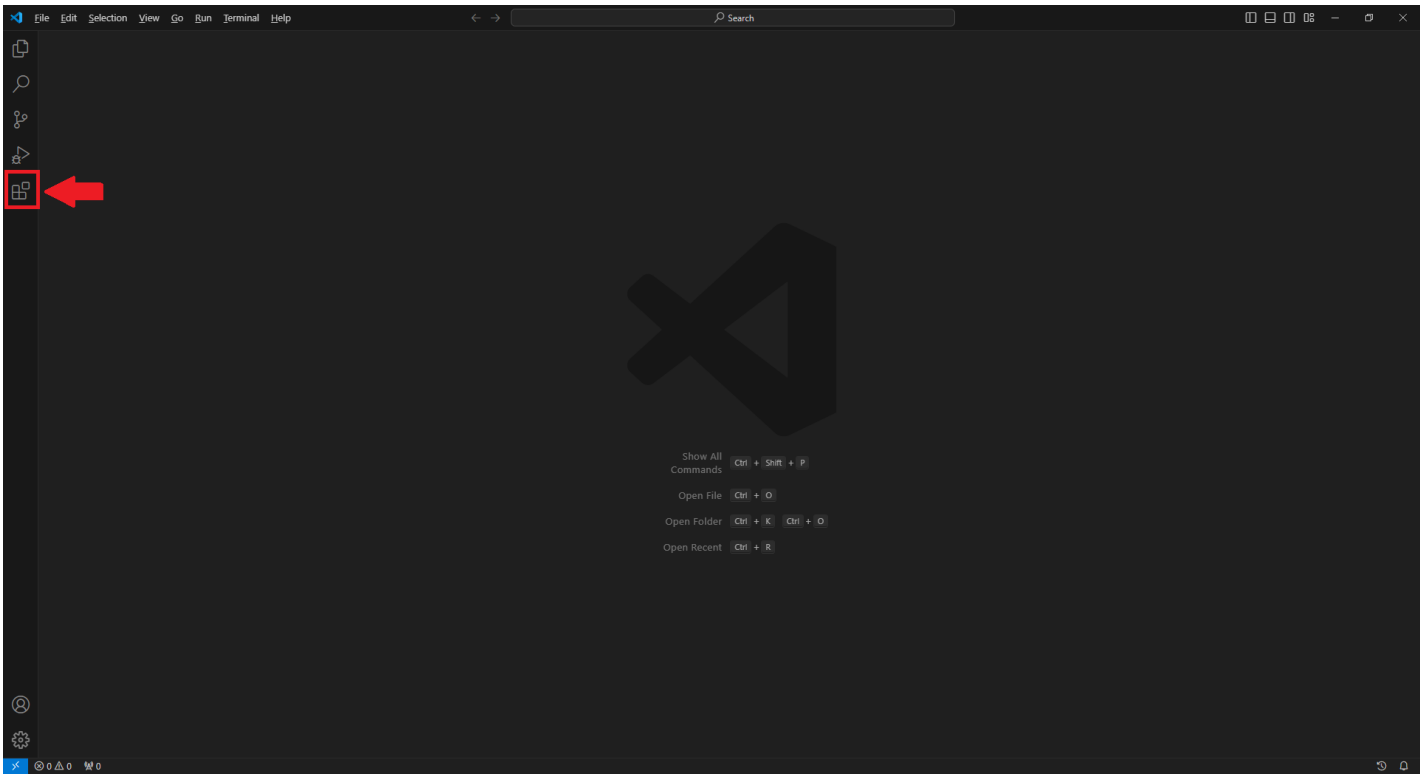
> **TIP**
>
> Alternatively, consider using the Preview Packages provided by MonoGame to get access to the latest developments.
>
> - How to install MonoGame Preview packages

Install Visual Studio Code C# Extensions

To transform Visual Studio Code from a simple text editor into a powerful development environment for C# projects, you must install the Visual Studio Code C# extension. This extension enhances the editor by providing syntax highlighting, code analysis, IntelliSense, and other features that significantly improve the development experience and productivity when working with C#.

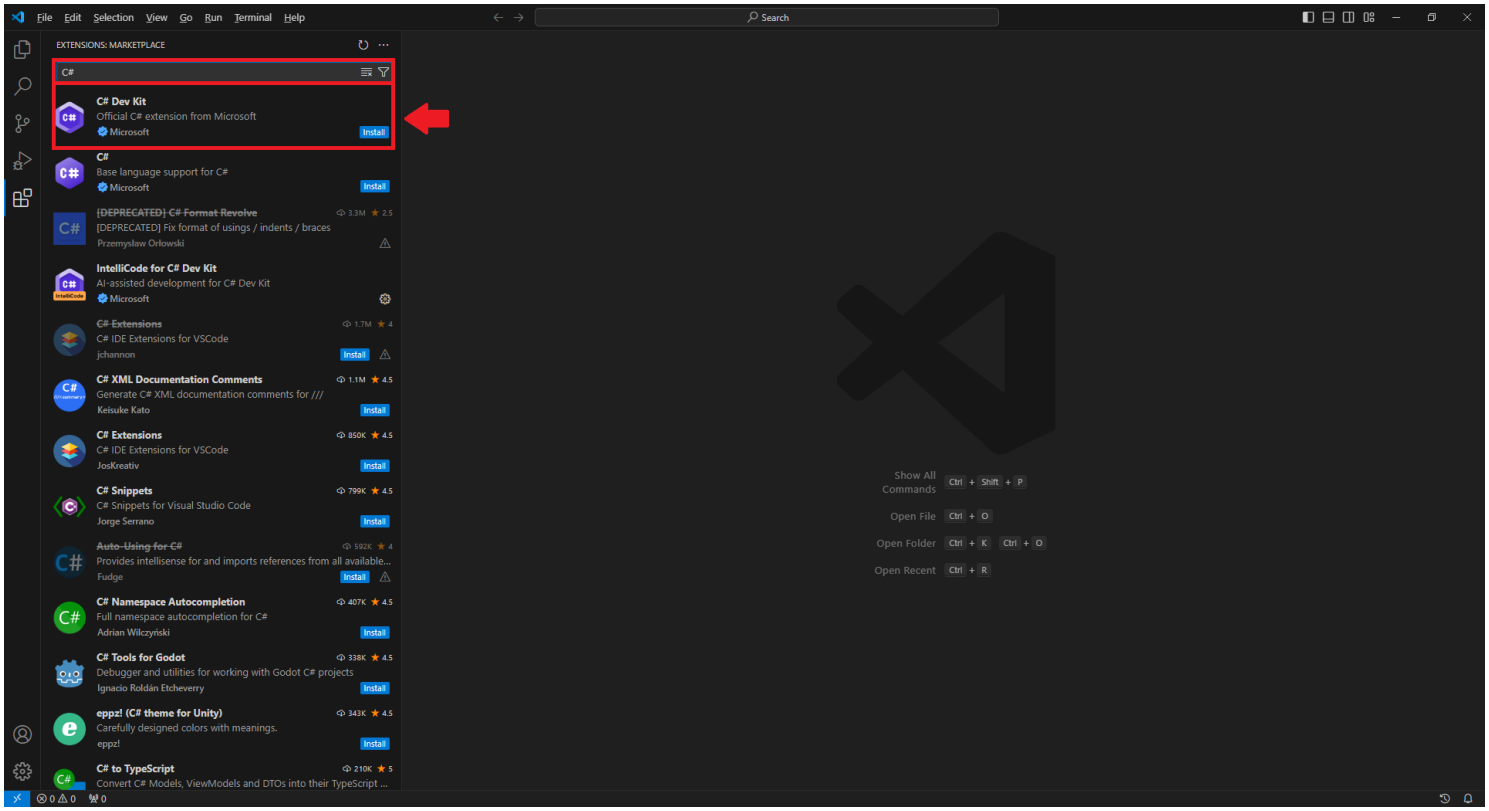
1. Open Visual Studio Code.

2. Click the **Extensions** icon in the **Activity Bar** on the left.



3. In the **Search Box** type `C#`.

4. Click **Install** for the **C# Dev Kit** extension. Installing this will also install the base **C#** extension.



> **NOTE**
>
> **For Arch Linux Code OSS users:** The C# Dev Kit extension is not compatible with Code OSS. Instead, install only the base **C#** extension (by Microsoft), which provides all the core functionality needed for MonoGame development including IntelliSense, debugging, and code navigation.

(Optional) Install the "MonoGame for VSCode" extension

A community member has built a VSCode extension for VS code to enable:

- Automatically install MonoGame templates
- Create new projects from the file menu or command palette
- Open the MGCB editor with the click of a button (very useful)

You can find this extension by following the steps above and searching for "MonoGame for VS Code" by r88.

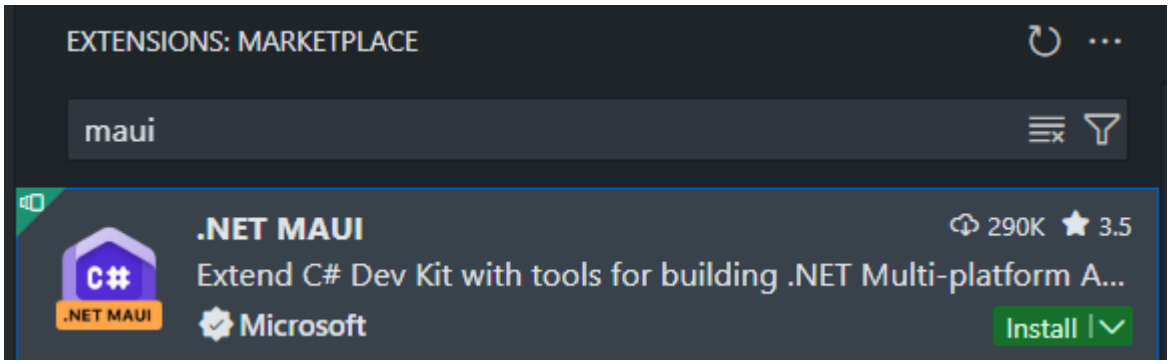
> **NOTE**
>
> While not built by the MonoGame team, we regularly use this extension ourselves and fully support it.

(Optional) Install the ".NET MAUI" extension to help with mobile development

This extension is provided by Microsoft and is built on top of the C# and C# Dev Kit extensions. They will be installed as dependencies if they were not already installed. While primarily designed for MAUI development, it provides essential mobile debugging capabilities that MonoGame mobile projects can utilize.

The .NET MAUI extension adds features for building mobile apps, including:

- F5 debugging support - Debug your MonoGame mobile apps on emulators, simulators, and physical devices
- Easy target switching - Change debug/deploy targets between different mobile platforms
- Enhanced C# Dev Kit integration - Leverages Solution Explorer, Test Explorer, and advanced language features



> **NOTE**
>
> This extension is recommended if you wish to do iOS or Android development. It is required for debugging support when working on mobile games.

> **WARNING**
>
> **For Arch Linux Code OSS users:** The .NET MAUI extension requires C# Dev Kit and therefore is not compatible with Code OSS. You will need to use the official Visual Studio Code for mobile development support.

Creating a new MonoGame project

1. Open up an empty folder in Visual Studio Code
2. Open up its terminal window (`Ctrl` + `` ` ``)
3. Run the following command to create an empty project for desktop platforms:

```
dotnet new mgdesktopgl
```

4. Once the files are created, open up the `Game1.cs` file and wait a second for the C# extension to load
5. You can now press F5, select C# and then your projects name if Visual Studio Code asks you, and it should start up your brand new game!

Update Project Tool references

The MonoGame Content Editor (MGCB) it a tool delivered through NuGet for your project using the tools configuration held in your `dotnet-tools.json` file (located in the `.config` folder of your project).

Once you have created your project you should run the following terminal/command-line command to ensure the tool (and the pipeline) is setup and read for your project:

```
dotnet tool restore
```

> **NOTE**
>
> If you ever change the version of the tools or want to upgrade them by editing the `dotnet-tools.json` configuration, you MUST run this command again to update te tools.

Welcome to the MonoGame documentation hub

The one stop shop for answers to those questions raised by the community about MonoGame

This new and improved documentation area aims to bridge the gaps in the understanding of the MonoGame Framework, covering all aspects of the features of MonoGame with some support from the original XNA documentation graciously donated by Microsoft.

Please check back regularly for updates and more details on the evolution of the MonoGame Framework and its API.

In This Section

What Is articles

Provides simple definitions for most of the central concepts maintained by the MonoGame Framework.

The aim with this documentation section is to answer those most basic of questions, what is this thing and what is it for? Ultimately, these guides are here to help peel back the curtains as to what each feature of the MonoGame framework is for.

How To articles

Short guides for everything you wanted to know about how to do game development with MonoGame, divided up in to simple articles for each activity.

These guides are here to give shortcuts to specific questions, like how to draw an image or how do I get input from a device. They are not a replacement for the full featured tutorials that MonoGame is investing in, but there to get to to an answer for how to do specific things quicker.

Online Resources

Check out the full API documentation here

What Is Articles for MonoGame

A series of articles to answer common questions related to MonoGame operation!

In This Section

Ever wondered why a specific feature exists and what it is supposed to do? The following articles go into detail as to what each of the major components of the MonoGame Framework do and how best to make use of them.

Where applicable, these articles also cross-reference with the "How-To" articles for the use of these features.

> **NOTE**
> These guides are not meant to be a replacement for "full on" tutorials, they are intended as short cuts to relevant information only.

This section is broken down into 3 distinct categories:

- Detailed sections
- Game Operations
- References

Detailed sections

Due the complexity of the Content and Graphics pipelines, the information is broken up into smaller pieces for easier understanding:

- What Is the Content Pipeline?

  An overview of how the MonoGame Framework provides content management and loading for games.

- What Is the Graphics Pipeline?

  An overview of how the MonoGame Framework provides graphical support for games.

Game Operations

This section details some general aspects of the operation of a MonoGame Framework game.

- What Is Audio?

  An overview of how the MonoGame Framework provides audio playback through several core audio classes.

- What Is Input?

  An overview of how the MonoGame Framework provides input management for games.

- What Is the GameLoop?

  A viewport is a 2D rectangle that defines the size of the rendering surface onto which a 3D scene is projected.

- What are Vectors, Matrices, and Quaternions?

  Presents an overview of the math-related functionality provided by the MonoGame Framework.

See Also

References

- What Is MonoGame Class Library

  An overview of the MonoGame Class Library reference, containing all the API calls available to the MonoGame Framework.

© 2012 Microsoft Corporation. All rights reserved.

Sounds Overview

An overview of how the MonoGame Framework provides audio playback through several core audio classes.

If your game is to use a few sound files, then the SoundEffect, SoundEffectInstance, and DynamicSoundEffectInstance classes will provide everything you need to play and stream audio during gameplay.

Simple Audio Playback

The simplest way to play sounds for background music or sound effects is to use SoundEffect and SoundEffectInstance. Source audio files are added like any other game asset to the project. For example code, see Playing a Sound, Looping a Sound, and Adjusting Pitch and Volume. For background music, see Playing a Song.

Accessing the Audio Buffer

Developers can use DynamicSoundEffectInstance for direct access to an audio buffer. By accessing the audio buffer, developers can manipulate sound, break up large sound files into smaller data chunks, and stream sound. For example code, see Streaming Data from a WAV File.

3D Audio

The SoundEffect class provides the ability to place audio in a 3D space. By creating AudioEmitter and AudioListener objects, the API can position a sound in 3D, and can change the 3D position of a sound during playback. Once you create and initialize AudioEmitter and AudioListener, call SoundEffectInstance.Apply3D.

Audio Constraints

When working with multiple platforms using MonoGame, there are a few constraints around audio that you will need to keep in mind and cater for, namely:

- Mobile platforms have a maximum of 32 sounds playing simultaneously.
- Desktop platforms have a maximum of 256 sounds playing simultaneously.
- Consoles and other platforms have their own constraints, please look at the console sdk documentation for more information,

IMPORTANT

An InstancePlayLimitException exception is thrown if this limit is exceeded.

Audio Buffer Format

The `byte[]` buffer format used as a parameter for the SoundEffect constructor, Microphone.GetData method, and DynamicSoundEffectInstance.SubmitBuffer method is PCM wave data. Additionally, the PCM format is interleaved and in little-endian.

The audio format has the following constraints:

- The audio channels can be mono (1) or stereo (2).
- The PCM wave file must have 16-bits per sample.
- The sample rate must be between 8,000 Hz and 48,000 Hz.
- The interleaving for stereo data is left channel to right channel.

Songs as Background Music

Access to the media library, combined with the ability to use playlists, allows games to create interesting background scores that can change with gameplay. Songs can be played directly from the media library, or can be imported by using the Content Pipeline. For more information, see Playing a Song.

Concepts

IMPORTANT

How to articles to follow.

Reference

- SoundEffect Class

  Provides a loaded sound resource.

- SoundEffectInstance Class

  Provides a single playing, paused, or stopped instance of a SoundEffect sound.

- DynamicSoundEffectInstance Class

  Provides properties, methods, and events for play back of the audio buffer.

- Song Class

  Provides access to a song in the song library.

© 2012 Microsoft Corporation. All rights reserved.

What is the Content pipeline?

The topics in this section describe how the content pipeline works and thr requirements for extending it to meet your games needs.

In This Section

- What Is Content?

  Describes the purpose of the MonoGame Content Pipeline and how it helps you add art and data assets to your game.

- Loading Content

  Demonstrates how to load content such as models, textures, sounds, and effects.

- What is the Content Pipeline Architecture?

  The Content Pipeline is designed to be extensible, so that it can easily support new input file formats and new types of conversion.

- Standard Content Importers and Content Processors

  Describes the standard Content Pipeline Content Importers and Content Processors of MonoGame that support various common art asset file formats.

- What is a Custom Importer

  MonoGame provides standard importers and processors for a number of common file formats used to store such basic game assets as models, materials effects, sprites, textures, and so on. For a list of file formats supported by these standard importers and processors.

- Parameterized Content Processors

  Describes how parameterized Content Processors work in MonoGame. Many of the standard Content Pipeline Content Processors shipped with MonoGame support parameter usage.

- What are Tips for Developing Custom Importers and Processors?

  The information provided here should help when you develop Content Pipeline extensions.

References

- What is MonoGame Content Pipeline Class Library?

  An overview of the MonoGame Content Pipeline Class Library reference, containing all the API calls available to the MonoGame Framework Content Framework.

- What is the Content Pipeline Document Object Model?

  The MonoGame Content Document Object Model (DOM) represents the set of built-in classes that can be consumed by standard content processors.

- What is Sprite Font XML Schema Reference?

  Describes the valid tags and values for Sprite-Font (.spritefont) XML files used by the Content Pipeline to create SpriteFont textures.

- What are the XML Elements for XMLImporter?

  The base elements that are recognized by XmlImporter Class.

© 2012 Microsoft Corporation. All rights reserved.

What are Graphics?

The basics of the graphics architecture for MonoGame!

Graphical basics 101

The topics in this section describe the graphical conventions used by MonoGame in rendering and managing content on a screen.

2D Concepts

- What Is Sprite?

  Sprites are 2D bitmaps that are drawn directly to a render target without using the pipeline for transformations, lighting or effects. Sprites are commonly used to display information such as health bars, number of lives, or text such as scores. Some games, especially older games, are composed entirely of sprites.

3D concepts

- What Is 3D Rendering?

  The 3D graphics pipeline uses a graphics device to load resources and render a 3D scene using an effect.

- What Is a Configurable Effect?

  MonoGame ships with a number of built in effects. These effects have properties so you can use to change behaviour or switch features on or off.

- What Is a Depth Buffer?

  A depth buffer contains per-pixel floating-point data for the z depth of each pixel rendered. A depth buffer may also contain stencil data which can be used to do more complex rendering such as simple shadows or outlines.

- What Is Blend State?

  Blend state controls how color and alpha values are blended when combining rendered data with existing render target data.

- What Is Depth Stencil State?

  Depth stencil state controls how the depth buffer and the stencil buffer are used.

- What Is Rasterizer State?

  Rasterizer state determines how to render 3D data such as position, color, and texture onto a 2D surface.

- What Is Sampler State?

  Sampler state determines how texture data is sampled using texture addressing modes, filtering, and level of detail.

- What Is Color Blending?

  Color blending mixes two colors together to produce a third color.

- What Is a Stencil Buffer?

  A stencil buffer contains per-pixel integer data which is used to add more control over which pixels are rendered. A stencil buffer can also be used in combination with a depth buffer to do more complex rendering such as simple shadows or outlines.

- What Is a Model Bone?

  A model bone is a matrix that represents the position of a mesh as it relates to other meshes in a 3D model.

Other Graphical concepts

- What Is a Graphics Profile?

  Provides conceptual information about the MonoGame Framework concept of graphics profiles, including explanations of the Reach and HiDef profiles.

- What Is a Back Buffer?

  A back buffer is a render target whose contents will be sent to the device when GraphicsDevice.Present is called.

- What Is a Render Target?

  A render target is a memory buffer for rendering pixels. One common use for a render target is offscreen rendering.

- What Is a View Frustum?

  A view frustum is a 3D volume that defines how models are projected from camera space to projection space. Objects must be positioned within the 3D volume to be visible.

- What Is a Viewport?

  A viewport is a 2D rectangle that defines the size of the rendering surface onto which a 3D scene is projected.

- What Is Antialiasing?

  Antialiasing is a technique for softening or blurring sharp edges so they appear less jagged when rendered.

© 2012 Microsoft Corporation. All rights reserved.

Overview of User Input and Input Devices

Input is a general term referring to the process of receiving actions from the user.

Input is a general term referring to the process of receiving actions from the user. In MonoGame, the Microsoft.Xna.Framework.Input namespace provides support for most input devices.

Methods related to input devices unavailable on the platform your game is running on are always available to your code. For example, you can access all GamePad methods on Mobile, but they will not return valid information (unless a GamePad is connected). Although using these methods will not cause exceptions or build errors in your code, they will silently fail when run.

Physical keyboards may or may not be present on mobile devices; you should not rely on the presence of a physical keyboard. For text input, you should use a software input panel (SIP), which will work on all devices, including those with physical keyboards.

If the mobile device does have access to a physical keyboard, the same methods used for keyboards on desktop can be used, given a few caveats.

For multi-touch devices, you can use the raw touch data provided by the TouchPanel class, but you can also use MonoGame's support for predefined gestures if your input fits one of the supported gesture types. For information about working with raw multi-touch input, see Working with Touch Input. For information about gesture support, see Detecting Gestures on a Multi-touch Screen.

The microphone on mobile can be used to capture audio that can be used in your game. For more information, see Recording Sounds with Microphones.

See Also

- How To input articles

© 2012 Microsoft Corporation. All rights reserved.

What is the Game Loop

There is a specific order of operations related to how a MonoGame project runs.

The MonoGame Framework Game class implements a game loop, which provides not only the window which displays your game, but also provides overloadable methods that your game implements to facilitate communication between your game and the operating system. This topic provides an overview of the basic functionality provided by the game loop.

- Making a New Game
- Game Loop Timing
- Game Components
- Game Services
- Game Components Consuming Game Services

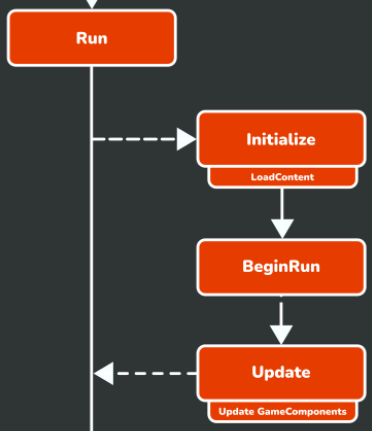
MonoGame Execution Order of Events Overview

High-Level View

Constructor

The Game constructor is called at the start of the application. The base Game constructor is executed first, then any logic you define in your Game1 constructor is executed. The base constructor will instantiate the GameServicesContainer, the GameComponentCollection, the ContentManager, and perform any platform specific constructions needed.

Run



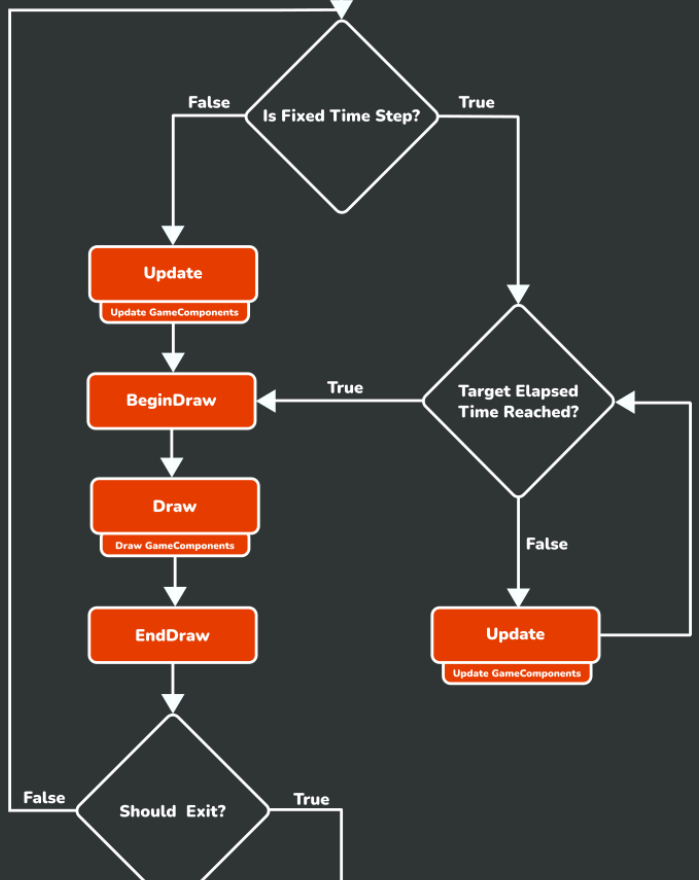
Run is only called once, after the constructor. This is where the framework performs initializations and prepares the game instance before it enters the main game loop.

Initialize is called during the Run call. If this is overridden in your Game1 class, your initialization logic will be run. Here the LoadContent method is called from within the base Initialize method.

Immediately after initializations are finished, BeginRun is called. Internally this performs no logic in MonoGame, but can be overridden in your Game1 class to perform any actions needed at this time.

Finally, a single call to Update is made. This call is here as part of XNA compatibility only.

Game Loop



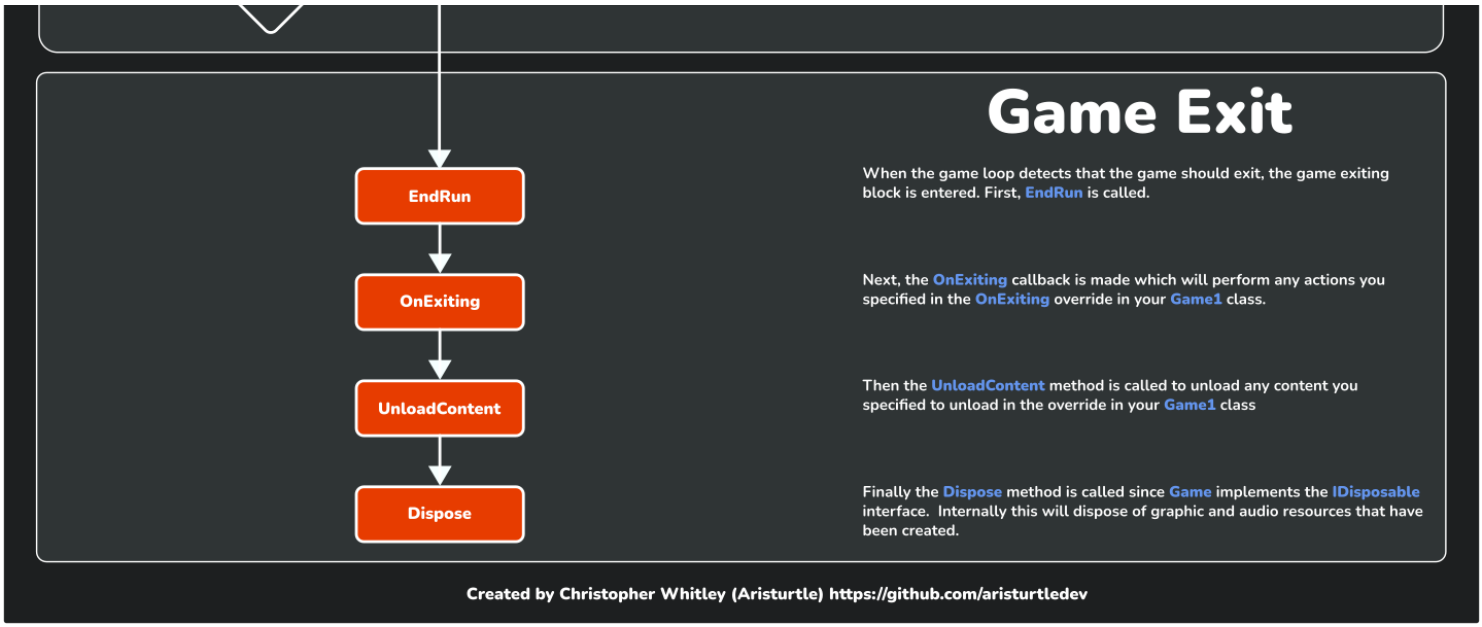
At the start of each iteration of the game loop, a check is made to see if a fixed time step is being used. If a fixed time step is being used, then Update is called continuously until the amount of time that has elapsed is greater than or equal to the target elapsed time; otherwise, Update is only called once in the loop iteration.

When Update is called, the base Update method will loop through any objects added to the GameComponents container that implement IUpdatable and perform the Update method on them.

After Update(s) complete, the rendering steps begin, first with the BeginDraw call, then Draw, then EndDraw.

When Draw is called, the base Draw method will loop through any objects added to the GameComponents container that implement IDrawable and perform the draw method on them.

Finally, a check is made to see if the game has been told to exit, if not then the game loop is run again.



Making a New Game

The first step in creating a new game is to make a class that derives from Game. The new class needs to override the following methods:

- Initialize - which is the method is responsible for game setup before the first frame of the game.
- Update - which is the method is responsible for handling game logic.
- Draw - which is the method responsible for drawing content to the screen.

Game Loop Timing

A Game is either fixed step or variable step, defaulting to fixed step. The type of step determines how often Update will be called and affects how you need to represent time-based procedures such as movement and animation.

Fixed-Step Game Loops

A fixed-step Game tries to call its **Update** method on the fixed interval specified in **TargetElapsedTime**. Setting **Game.IsFixedTimeStep** to **true** causes a Game to use a fixed-step game loop. A new MonoGame project uses a fixed-step game loop with a default **TargetElapsedTime** of 1/60th of a second.

In a fixed-step game loop, Game calls **Update** once the **TargetElapsedTime** has elapsed. After **Update** is called, if it is not time to call **Update** again, the Game calls **Draw**. After **Draw** is called, if it is not time to call **Update** again, Game idles until it is time to call **UpdateS**.

If **Update** takes too long to process, Game sets **IsRunningSlowly** to **true** and calls **Update** again, without calling **Draw** in between. When an update runs longer than the **TargetElapsedTime**, Game responds by calling **Update** extra times and dropping the frames associated with those updates to catch up. This ensures that **Update** will have been called the expected number of times when the game loop catches up from a slowdown. You can check the value of **IsRunningSlowly** in your **Update** if you want to detect dropped frames and shorten your **Update** processing to compensate. You can reset the elapsed times by calling **ResetElapsedTime**.

When your game pauses in the debugger, Game will not make extra calls to **Update** when the game resumes.

Variable-Step Game Loops

A variable-step game calls its **Update** and **Draw** methods in a continuous loop without regard to the TargetElapsedTime. Setting Game.IsFixedTimeStep to **false** causes a Game to use a variable-step game loop.

Animation and Timing

For operations that require precise timing, such as animation, the type of game loop your game uses (fixed-step or variable-step) is important.

Using a fixed step allows game logic to use the **TargetElapsedTime** as its basic unit of time and assume that **Update** will be called at that interval. Using a variable step requires the game logic and animation code to be based on **ElapsedGameTime** to ensure smooth gameplay. Because the **Update** method is called immediately after the previous frame is drawn, the time between calls to **Update** can vary. Without taking the time between calls into account, the game would seem to speed up and slow down. The time elapsed between calls to the **Update** method is available in the **Update** method's *gameTime* parameter. You can reset the elapsed times by calling **ResetElapsedTime**.

When using a variable-step game loop, you should express rates—such as the distance a sprite moves—in game units per millisecond (ms). The amount a sprite moves in any given update can then be calculated as the rate of the sprite times the elapsed time. Using this approach to calculate the distance the sprite moved ensures that the sprite will move consistently if the speed of the game or computer varies.

Game Components

Game components provide a modular way of adding functionality to a game. You create a game component by deriving the new component either from the

- GameComponent class, or
- DrawableGameComponent class if the component loads and draws graphics content.

You then add game logic and rendering code to the game component by overriding the following methods:

- **GameComponent.Update**
- **GameComponent.Initialize**

Additionally, if you derive from the `DrawableGameComponent` class, you will additionally be able to override:

- **DrawableGameComponent.Draw**

A game component is registered with a game by passing the component to `Game.Components.Add`. A registered component will have its `initialize`, `update`, and `draw` methods called from the `Game.Initialize`, `Game.Update`, and `Game.Draw` methods, respectively.

Game Services

Game services are a mechanism for maintaining loose coupling between objects that need to interact with each other. Services work through a mediator—in this case, Game.Services. Service providers register with Game.Services, and service consumers request services from Game.Services. This arrangement allows an object that requires a service to request the service without knowing the name of the service provider.

Game services are defined by a unique interface. A class specifies the services it provides by implementing interfaces and registering the services with Game.Services. A service is registered by calling **Game.Services.AddService** specifying the type of service being implemented and a reference to the object providing the service. For example, to register an object that provides a service represented by the interface IMyService, you would use the following code.

```
    Services.AddService( typeof( IMyService ), myobject );
```

Once a service is registered, the object providing the service can be retrieved by **Game.Services.GetService** and specifying the desired service. For example, to retrieve IGraphics DeviceService, you would use the following code.

```
    IGraphicsDeviceService graphicsservice =
(IGraphicsDeviceService)Services.GetService( typeof(IGraphicsDeviceService) );
```

Game Components Consuming Game Services

The GameComponent class provides the Game property so a GameComponent can determine what Game it is attached to. With the Game property, a GameComponent can call Game.Services.GetService to find a provider of a particular service. For example, a GameComponent would find the IGraphics DeviceService provider by using the following code.

```
    IGraphicsDeviceService graphicsservice =
(IGraphicsDeviceService)Game.Services.GetService( typeof( IGraphicsDeviceService
) );
```

Related articles

IMPORTANT

How to articles to follow.

© 2012 Microsoft Corporation. All rights reserved.

What are Vectors, Matrices, and Quaternions?

What are What are Vectors, Matrices, and Quaternions definitions for MonoGame!

The MonoGame Framework Math Libraries are in the Microsoft.Xna.Framework namespace alongside a number of additional types that deal with The MonoGame Framework Application model.

- Coordinate system
- Mathematical Constants and Scalar Manipulation
- Basic Geometric Types
- Precision and Performance

Coordinate system

The MonoGame Framework uses a right-handed coordinate system, with the positive z-axis pointing toward the observer when the positive x-axis is pointing to the right, and the positive y-axis is pointing up.

Mathematical Constants and Scalar Manipulation

The MonoGame Framework provides the MathHelper Members class for manipulating scalar values and retrieving some common mathematical constants. This includes methods such as the `ToDegrees` and `ToRadians` utility methods for converting between degrees and radians.

Basic Geometric Types

The MonoGame Framework Math library has multiple basic geometric types for manipulating objects in 2D or 3D space. Each geometric type has a number of mathematical operations that are supported for the type.

Vectors

The MonoGame Framework provides the Vector2, Vector3, and Vector4 classes for representing and manipulating vectors. A vector typically is used to represent a direction and magnitude. In The MonoGame Framework, however, it also could be used to store a coordinate or other data type with the same storage requirements.

Each vector class has methods for performing standard vector operations such as:

- Dot product
- Cross product
- Normalization
- Transformation
- Linear, Cubic, Catmull-Rom, or Hermite spline interpolation.

Matrices

The MonoGame Framework provides the Matrix class for transformation of geometry. The Matrix class uses row major order to address matrices, which means that the row is specified before the column when describing an element of a two-dimensional matrix. The Matrix class provides methods for performing standard matrix operations such as calculating the `determinate` or `inverse` of a matrix. There also are helper methods for creating scale, rotation, and translation matrices.

Quaternions

The MonoGame Framework provides the Quaternion structure to calculate the efficient rotation of a vector by a specified angle.

Curves

The Curve class represents a Hermite curve for interpolating varying positions at different times without having to explicitly define each position. The curve is defined by a collection of CurveKey points representing each varying position at different times. This class can be used not only for spatial motion, but also to represent any response that changes over time.

Bounding Volumes

The MonoGame Framework provides the BoundingBox, BoundingFrustum, BoundingSphere, Plane, and Ray classes for representing simplified versions of geometry for the purpose of efficient collision and hit testing. These classes have methods for checking for intersection and containment with each other.

Precision and Performance

The MonoGame Framework Math libraries are single-precision. This means that the primitives and operations contained in this library use 32-bit floating-point numbers to achieve a balance between precision and efficiency when performing large numbers of calculations.

A 32-bit floating-point number ranges from `-3.402823e38` to `+3.402823e38`. The 32 bits store the sign, mantissa, and exponent of the number that yields seven digits of floating-point precision.

[!NOTE]

> Some numbers—for example π, 1/3, or the square root of two—can be approximated only with seven digits of precision, so be aware of rounding errors when using a binary representation of a floating-point number.

For more information about single-precision numbers, see the documentation for the Single data type.

© 2012 Microsoft Corporation. All rights reserved.

MonoGame Framework Class Library

MonoGame Framework Class Library Reference

The MonoGame Framework class library is a library of classes, interfaces, and value types that are included in MonoGame Game Studio. This library provides access to MonoGame Framework functionality and is designed to be the foundation on which MonoGame Game Studio applications, components, and controls are built.

Namespaces

- Microsoft.Xna.Framework

  Provides commonly needed game classes such as timers and game loops.

- Microsoft.Xna.Framework.Audio

  Contains low-level application programming interface (API) methods that can load and manipulate content files to play audio.

- Microsoft.Xna.Framework.Content

  Contains the run-time components of the Content Pipeline.

- Microsoft.Xna.Framework.Design

  Provides a unified way of converting types of values to other types.

- Microsoft.Xna.Framework.Graphics

  Contains low-level application programming interface (API) methods that take advantage of hardware acceleration capabilities to display 3D objects.

- Microsoft.Xna.Framework.Graphics.PackedVector

  Represents data types with components that are not multiples of 8 bits.

- Microsoft.Xna.Framework.Input

  Contains classes to receive input from keyboard, mouse, and GamePad devices.

- Microsoft.Xna.Framework.Input.Touch

  Contains classes that enable access to touch-based input on devices that support it.

- Microsoft.Xna.Framework.Media

  Contains classes to enumerate, play, and view songs, albums, playlists, and pictures.

- Microsoft.Xna.Framework.Utilities

  Contains classes that provide additional support for working with MonoGame.

- Microsoft.Xna.Framework.Utilities.Deflate

  Contains classes that provide additional support for compression with MonoGame.

© 2012 Microsoft Corporation. All rights reserved.

How To Articles for MonoGame

These articles provide a brief introduction into performing some basic tasks in MonoGame.

In This Section

Ever wondered how to do a specific thing quickly, or just try to recall how an operation or function is meant to be implemented, these "How To" guides are quick and short tutorials on how to use each of the features of the MonoGame Framework.

> **NOTE**
>
> These guides are not meant to be a replacement for "full on" tutorials, they are intended as short cuts to relevant information only.

This section is broken down into 3 distinct categories:

- Core concepts
- Game Operations
- Guidance

Core concepts

This section details the core concepts to understand how MonoGame projects fit together.

- Audio

  A set of articles detailing how to use or implement specific audio features of MonoGame.

- Content Pipeline

  Guides on how to use the Content Pipeline and the MGCB tool for getting content into your MonoGame projects.

- Graphics

  Detailed tutorials on specific graphics features and operations.

- Input

  Tutorials on how to get the most out of specific inputs from MonoGame while running on platforms.

Game Operations

This section details some of the operational aspects of running a MonoGame game.

- How to manage automatic rotation and scaling

  Demonstrates how to manage rotation and scaling based on device orientation.

- How to exit a Game Immediately

  Demonstrates how to exit a game in response to user input.

- How to exit a Game After a Time Out

  Demonstrates how to exit a game after a period of time (such as inactivity) has passed.

- How to update a game with Variable or Fixed Timing

  Demonstrates how to set up the runtime to call your Update method using variable or fixed timing.

- How to handle resizing of a Game

  Demonstrates how to handle the resizing of the active game window.

Guidance

This section contains articles to help make your MonoGame project successful.

- How to work with Asynchronous Methods in MonoGame

  This topic describes how you can work with asynchronous methods in MonoGame.

- How to apply Best Practices for MonoGame Games

  The practices discussed here will help you have the most success possible with your MonoGame game.

- How to install MonoGame Preview packages

  This topic describes how to use a preview release of MonoGame in a new or existing project.

© 2012 Microsoft Corporation. All rights reserved.

How To tasks for Audio

A series of articles to answer common questions related to audio operation!

In This Section

No game is truly complete without sound, whether is is a melodious background track, one off sound effects or even recapturing the players voice through a microphone. Without sound games lack depth (although also remember for accessibility, some hear sounds through audio description!, make sure to be inclusive.)

The following articles showcase some of the best ways to get sound out of your game:

- Playing a Sound

  Demonstrates how to play a simple sound by using SoundEffect.

- Looping a Sound

  Demonstrates how to loop a sound.

- Adjusting Pitch and Volume

  Demonstrates how to change pitch and volume of a playing sound.

- Playing a Song

  Demonstrates how to play a song from a media file.

- Streaming Data from a WAV File

  Demonstrates how to stream audio from a wave (.wav) file.

- Accessing Microphone devices

  Topics that provide guidance on writing code to support access to microphones for MonoGame.

- Recording Sound with a Microphone

  Demonstrates the basics of recording audio using a microphone attached to a device.

© 2012 Microsoft Corporation. All rights reserved.

How To articles for the Content Pipeline

A series of articles to answer common questions related to Content Pipeline operation!

In This Section

- How to add content

  Demonstrates how to add content, such as images or sounds, to your game.

- How to use custom XML

  Demonstrates how to use a custom XML File to Specify Content.

- How to Load content from a library

  Demonstrates how to create A localized game using alternative sets of text that are appropriate to the language and culture of the gamer.

- How to extend a content processor

  Describes how MonoGame lets you modify or extend the behavior of any standard Content Pipeline processor.

- How to Extend the Font Description Processor to Support Additional Characters

  Describes the process of developing a custom content processor needed to add additional characters to a FontDescription object based on the text that is required by the game.

- How to Package Textures on Android

  Describes how to support multiple texture compression formats on Android.

© 2012 Microsoft Corporation. All rights reserved.

How to articles for the Graphics pipeline

These articles provide a details on how to use the Graphics API in MonoGame.

In This Section

This section demonstrates several graphical concepts divided into the following categories:

- 2D Sprite Rendering
- Cameras
- 3D Rendering
- 3D Collisions

2D Sprite Rendering

This section walks through several core concepts related to sprite rendering, including drawing text to the screen.

- How To Draw A Sprite

  Demonstrates how to draw a sprite by using the SpriteBatch class.

- How To Draw A Sprite Background

  Demonstrates how to draw a foreground and background sprite using the SpriteBatch class, where only part of the foreground sprite masks the background.

- How To Tint A Sprite

  Demonstrates how to tint a sprite using a Color value.

- How To Rotate A Sprite

  Demonstrates how to rotate a sprite around its center.

- How To Rotate A Sprite Group

  Demonstrates how to rotate a group of sprites around a single point.

- How To Scale A Sprite

  Demonstrates how to scale a sprite using a uniform scale.

- How To Scale A Sprite using A Matrix

  Demonstrates how to scale sprites using a matrix that is created based on the viewport width.

- How To Tile Sprites

  Demonstrates how to draw a sprite repeatedly in the x and y directions in one Draw call.

- How To Draw Text

  Demonstrates how to import a SpriteFont into a project and to draw text using the DrawString method.

- How To Animate A Sprite

  Demonstrates how to animate a sprite from a texture using a custom class.

- How To Make A Scrolling Background

  Demonstrates how to draw a scrolling background sprite using the SpriteBatch class.

Cameras

- Rotating and Moving the Camera

  Demonstrates how to rotate and move a camera in a 3D environment. You can rotate the camera about its y-axis, and move it forward and backward. You control the camera's position and orientation by using the directional keys on your keyboard or by using the D-pad of your gamepad.

- How to position the Camera

  Demonstrates how to position the camera so that all objects in a scene are within the view frustum while maintaining the camera's original orientation.

- How to create a Render Target

  Demonstrates how to create a render target using the RenderTarget2D class.

- How to create a Full-Screen Game

  Demonstrates how to start a game in full-screen mode.

- How to restrict Aspect Ratio on a Graphics Device

  Demonstrates how to create a custom GraphicsDeviceManager that only selects graphics devices with widescreen aspect ratios in full-screen mode.

- How to display Multiple Screens with Viewports

  Demonstrates how to use viewports to display different scenes simultaneously using two cameras.

3D Rendering

This section walks through several core concepts related 3D rendering and math practices with MonoGame.

- How to transform a Point

  This example demonstrates how to use the Vector3 and Matrix classes to transform a point. A matrix transform can include scaling, rotating, and translating information.

- How to create a Basic Effect

  Demonstrates how to create and initialize an instance of the BasicEffect class and use it to draw simple geometry.

- Using a Basic Effect with Texturing

  Demonstrates how to create and draw a simple quad—two triangles that form a rectangle or square—using **DrawUserIndexedPrimitives**.

- How to create a State Object

  Demonstrates how to create a state object using any of the state object classes: BlendState, DepthStencilState, RasterizerState, or SamplerState.

- Drawing 3D Primitives using Lists or Strips

  Demonstrates how to draw 3D primitives using lines and triangles arranged as strips or lists.

- How to render a Model using a Basic Effect

  Demonstrates how to load and render a model using the MonoGame Content Pipeline. It is assumed that an existing Windows game project is loaded in MonoGame.

- How to enable Anti-aliasing

  Demonstrates how to enable anti-aliasing for your game.

- How to create a Custom Vertex Declaration

  Demonstrates how to create a custom vertex declaration and use it to render a 3D object.

- How to Dynamically Update Vertex Data

  Geometry in a 3D game is defined by vertex data. Sometimes, a game needs to modify vertex data or even generate new vertex data dynamically (at run time). Here are some solutions for dynamically updating vertex data.

3D Collisions

- Bounding Volumes and Collisions

  Collision detection determines whether objects in a game world overlap each other.

- Selecting an Object with a Mouse

  Demonstrates how to check whether the mouse is positioned over a 3D object by creating a ray starting at the camera's near clipping plane and ending at its far clipping plane.

- Testing for Collisions

  Demonstrates how to use the BoundingSphere class to check whether two models are colliding.

© 2012 Microsoft Corporation. All rights reserved.

Working with Input

This section provides topics that demonstrate how to write code that managed input in your MonoGame project for keyboard, mouse, gamepad and touch.

In This Section

A game without input is basically just a movie, the user is not interacting and just watching what happens on screen. If the images are not moving, then it is basically just a digital painting. Games are all about interactivity and how we turn the sometimes random presses of buttons, keys, and waving of gyroscopes into interesting reactions on screen.

The following articles showcase some of the best ways to get input into your game:

- Overview of User Input and Input Devices

  Describes the various types of input devices that can be used with MonoGame.

- Detecting a Key Press

  Demonstrates how to detect if a user pressed or released a key on the keyboard.

- Detecting a Button Press

  Demonstrates how to detect if a user performed input on a GamePad.

- Working with Touch Input

  Demonstrates how to detect and use multi-touch input in a MonoGame game.

- Detecting Gestures on a Multi-touch Screen

  Demonstrates how to detect and use multi-touch gestures in a MonoGame game.

- How to select an Object in 3D Space

  Demonstrates how to check whether the mouse (or touch location) is positioned over a 3D object by creating a ray starting at the camera's near clipping plane and ending at its far clipping plane.

© 2012 Microsoft Corporation. All rights reserved.

Package games for distribution

How to package your game for distribution.

Desktop games

To publish desktop games, it is recommended that you build your project as a self-contained .NET application. As such, your game will require absolutely no external dependencies and should run out-of-the-box as-is.

Building and packaging

Windows

From the .NET CLI:

```
dotnet publish -c Release -r win-x64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

You can then zip the content of the publish folder and distribute the archive as-is.

If you are targeting WindowsDX, note that players will need the DirectX June 2010 runtime to be installed on their machine for audio and gamepads to work properly.

macOS

We recommend that you distribute your game as an application bundle. Application bundles are directories with the following file structure:

```
YourGame.app                    (this is your root folder)
    - Contents
        - Resources
            - Content           (this is where all your content and XNB's should go)
            - YourGame.icns     (this is your app icon, in ICNS format)
        - MacOS
            - YourGame          (the main executable for your game)
        - Info.plist            (the metadata of your app, see below for contents)
```

So first lets create our directory structure.

```
mkdir -p bin/Release/YourGame.app/Contents/MacOS/
mkdir -p bin/Release/YourGame.app/Contents/Resources/Content
```

Next we need to publish our application for both `arm64` (Apple Silicon) and `x64` (Intel). From the .NET CLI:

```
dotnet publish -c Release -r osx-x64 -p:PublishReadyToRun=false -p:TieredCompilation=fal
dotnet publish -c Release -r osx-arm64 -p:PublishReadyToRun=false -p:TieredCompilation=f
```

Next we need to combine the two binaries into one Universal Binary which will work on both arm64 and x64 machines. We can do this using the `xcode` utility `lipo`.

```
lipo -create bin/Release/net8.0/osx-arm64/publish/YourGame bin/Release/net8.0/osx-x64/pu
```

The above command will combine the two output executables into one. It assumes you are using the standard `Output` path for your application. If you are using a custom `Output` folder, you will need to make adjustments to the above command.

Copy over your content

```
cp -R bin/Release/net8.0/Content bin/Release/YourGame.app/Contents/Resources/Content
```

The `Info.plist` file is a standard macOS file containing metadata about your game. Here is an example file with required and recommended values set:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>YourGame</string>
    <key>CFBundleIconFile</key>
    <string>YourGame</string>
    <key>CFBundleIdentifier</key>
    <string>com.your-domain.YourGame</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>YourGame</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>FONV</string>
    <key>CFBundleVersion</key>
    <string>1</string>
    <key>LSApplicationCategoryType</key>
    <string>public.app-category.games</string>
    <key>LSMinimumSystemVersion</key>
    <string>10.15</string>
    <key>NSHumanReadableCopyright</key>
    <string>Copyright © 2022</string>
    <key>NSPrincipalClass</key>
    <string>NSApplication</string>
    <key>LSRequiresNativeExecution</key>
    <true/>
    <key>LSArchitecturePriority</key>
    <array>
        <string>arm64</string>
    </array>
</dict>
</plist>
```

> **NOTE**
>
> For more information about `Info.plist` files, see the Apple documentation.

After completing these steps, your `.app` folder should appear as an executable application on macOS. However it does need an icon. So we need to create an `.icns` file. We can use online tools to do this or you can use the following:

```
mkdir -p bin/Release/YourGame.iconset
sips -z 16 16 Icon.png --out bin/Release/YourGame.iconset/icon_16x16.png
sips -z 32 32 Icon.png --out bin/Release/YourGame.iconset/icon_16x16@2x.png
sips -z 32 32 Icon.png --out bin/Release/YourGame.iconset/icon_32x32.png
sips -z 64 64 Icon.png --out bin/Release/YourGame.iconset/icon_32x32@2x.png
sips -z 128 128 Icon.png --out bin/Release/YourGame.iconset/icon_128x128.png
sips -z 256 256 Icon.png --out bin/Release/YourGame.iconset/icon_128x128@2x.png
sips -z 256 256 Icon.png --out bin/Release/YourGame.iconset/icon_256x256.png
sips -z 512 512 Icon.png --out bin/Release/YourGame.iconset/icon_256x256@2x.png
sips -z 512 512 Icon.png --out bin/Release/YourGame.iconset/icon_512x512.png
sips -z 1024 1024 Icon.png  bin/Release/YourGame.iconset/icon_512x512@2x.png
iconutil -c icns bin/Release/YourGame.iconset --output bin/Release/YourGame.app/Contents
```

> **NOTE**
>
> This code is expecting an `Icon.png` file to be in the same directory. This file should be `1024` x `1024` pixels.

For archiving, we recommend using the `.tar.gz` format to preserve the execution permissions (you will likely run into permission issues if you use `.zip` at any point).

Ubuntu

From the .NET CLI:

```
dotnet publish -c Release -r linux-x64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

You can then archive the content of the publish folder and distribute the archive as-is.

We recommend using the `.tar.gz` archiving format to preserve the execution permissions.

Special notes about .NET parameters

.NET proposes several parameters when publishing apps that may sound helpful, but have many issues when it comes to games (because they were never meant for games in the first place, but for small lightweight applications).

PublishAot and PublishTrimmed

> **IMPORTANT**
>
> The WindowsDX target is not compatible with `PublishAot` or `PublishTrimmed` because it uses Windows Forms, which is (as of .NET 9) not compatible with these options. If you need trimming or AOT compilation for desktop platforms, please consider using the DesktopGL target instead of WindowsDX.

The `PublishAot` option optimises your game code "Ahead of Time" for performance. It allows you to ship your game without the need to JIT (Just In Time compile), and will basically natively compile your game.

`PublishAot` binaries are much faster, which is typically desired for games. It however comes with limitations, like the inability to use runtime reflection and runtime code generation (IL emition).

`PublishAot` makes use of `PublishTrimmed`, which is another option that strip binaries of unused code to make much lighter executables and assemblies. Trimming can be aggressive and might remove types if the compiler can't detect if they are used (e.g. if you are using reflection or generics).

MonoGame is mostly compatible with `PublishAot` and `PublishTrimmed`, and will just work in most cases. It may however crash at runtime if you are using custom content importers that use generic collections. If you are using `PublishAot` and you are running into runtime exceptions occuring when loading content saying that a type is missing, the solution is to call `ContentTypeReaderManager.AddTypeCreator()` on that type before trying to load your content. This will tell the AOT compiler to include that type.

Besides MonoGame itself, it may happen that the third party libraries that you are using are not compatible with AOT or trimming. In that case, you should refer to those libraries maintainers for workarounds, or replace them with compatible libraries.

Overall, AOT and trimming have similar limitations you need to watchout for:

1. Using `XmlSerializer` in your game will probably cause issues. Since it uses reflection it will be difficult for the Trimmer to figure out what needs to be kept. It is recommended that, instead of using the `Deserialize` method, you write your own custom deserializer using `XDocument` or `XmlReader`. Alternatively you can use the Content Pipeline and create a custom `Processor` and `Reader` to convert the Xml into a binary format that can be loaded via the usual `Content.Load<T>` method.
2. Dynamically loading assemblies via `Assembly.LoadFile`.
3. No run-time code generation, for example, System.Reflection.Emit.

You can also refer to the Preparing for consoles documentation, which leverage AOT and has the same limitations. If your game runs with `PublishAot`, you'll be well ahead into porting your game to consoles.

For more information, please see Native AOT deployment and Trim self-contained deployments.

ReadyToRun (R2R)

ReadyToRun is advertised as improving application startup time, but slightly increasing binary size. We recommend not using it for games because it produces micro stutters when your game is running.

ReadyToRun code is of low quality and makes the Just-In-Time compiler (JIT) trigger regularly to promote the code to a higher quality. Whenever the JIT runs, it produces potentially very visible stutters.

Disabling ReadyToRun solves this issue (at the cost of a slightly longer startup time, but typically very negligible).

ReadyToRun is disabled by default. You can configure it by setting the `PublishReadyToRun` property in your `.csproj` file.

MonoGame templates for .NET projects explicitly set this to `false`.

Tiered compilation

Tiered compilation is a companion system to ReadyToRun and works on the same principle to enhance startup time. We suggest disabling it to avoid any stutter while your game is running.

Tiered compilation is **enabled by default**. To disable it, set the `TieredCompilation` property to `false` in your `.csproj`. MonoGame templates for .NET projects explicitly set this to `false`.

PublishSingleFile

PublishSingleFile packages your game into a single executable file with all dependencies and content integrated.

While it sounds very convenient, be aware that it's not magical and is in fact a hidden self-extracting zip archive. As such, it may make app startup take **a lot** longer if your game is large, and may fail to launch on systems where user permissions don't allow extracting files (or if there is not enough storage space available).

We highly recommend not using it for better compatibility across systems.

If you need to reduce the footprint of your game, please refer to `PublishAot` and `PublishTrimmed` instead.

Mobile games

Please refer to the Xamarin documentation:

- Android
- iOS

Preparing for consoles

If you would like to port your game to consoles, there are some best practices to follow if you want to avoid running into issues while porting.

MonoGame for gaming consoles uses a dedicated .NET runtime that is not maintained by Microsoft. Therefore, a 100% accuracy and reliability is not guaranteed. Moreover, the console runtime makes use of `ahead-of-time` native compilation (AOT), which means that some .NET features will not, and cannot work on consoles.

This article explains the most common pitfalls and suggested guidelines to optimize your chances of having a smoother porting experience.

> **NOTE**
>
> *Gaming consoles are restricted to registered developers and are not publicly available nor publicly documented. To get access to those platforms, please contact your console account manager(s). MonoGame documentation for closed platforms is available in their respective repositories.*

No use of runtime reflection

The main show stopper when it comes to porting a .NET game to consoles, is the use of runtime reflection.

Reflection is, at large, unsupported in an AOT context. Which means that anywhere you or a third party library uses reflection, your game will crash.

In order to make sure that your game abides to that rule, you can try to publish an AOT'd version of it for desktop computers and verify how it fares.

To publish with AOT:

- Add `<PublishAot>true</PublishAot>` to your `.csproj`.
- Then run `dotnet publish` from the command-line/terminal.

This will natively compile your game in a fashion similar to consoles. The output executable will be in your output folder, within a sub-folder named `publish`.

From there, you can try running this executable. If it does not start or crashes later on, you likely are using reflection or another unsupported feature in a AOT runtime.

Native executables can debugged by:

- Starting an empty Visual Studio with no code.
- Opening the compiled exe.
- Then hitting "Start debugging".

It should show you on which C# lines it crashed.

It is important to note that you should test every aspect of your game, and not just if it starts. Run through all the menus/screens, scene transitions and gameplay.

Another way to make sure that everything is safe is to enable `<EnableTrimAnalyzer>true</EnableTrimAnalyzer>` in your `.csproj`, and then **rebuild** (not just build) your game and check the build output for AOT warnings. Those warnings will tell you which parts of your code might trigger crashes or unexpected results when running on AOT compilation. You should seek to resolve all of them.

No runtime compilation / IL emit

Generating code at runtime is a scenario that is also not supported in AOT contexts, it is also forbidden by console manufacturers.

Like reflection, trying to make a `PublishAot` build is a good way to verify that your game is compliant because any use of IL emit will crash.

No use of dynamic assembly loading

Loading assemblies at runtime with `Assembly.Load()` is not supported.

Third party libraries

Many third party libraries heavily rely on using reflection or IL emit, this is a common practice for JSON or XML parsers for example (and they are the most common source of third party incompatibilities).

It is advised to choose very carefully the libraries that you are using when porting to consoles. If you do not select them with this in mind, you might run into a situation in which you will have to rewrite entire chunks of data handling.

The best way to make sure if they will work, is to search if they are **"AOT-compatible"**, or try to compile with the `<EnableTrimAnalyzer>true</EnableTrimAnalyzer>` setting in your `.csproj` and check if there are any warnings related to those libraries.

For example, here are some parsing libraries known for their compliance with AOT compilation and good handling of memory:

- TurboXML
- TinyJSON

On the contrary, Newtonsoft JSON is known to be unsupported (there are modified forks around, but we overall recommend to not use it for games).

Native libraries

If you are using native libraries, make sure that they provide builds for consoles, or make sure that you can compile and run them on consoles yourself.

Even though a library might be open-source, it is unlikely they will just compile and run on consoles.

It is suggested to only use native libraries that have been proven to run on consoles.

LINQ

While LINQ mostly works on consoles, it should be noted that due to AOT compilation, LINQ queries can not be optimized and performance can be very slow.

Moreover, LINQ is very garbage-prone and if your game has stuttering issues you might want to reduce your usage of LINQ.

Avoiding garbage generation

Even though your game has good performance on PC and does not show stutters, you might want to be very cautious about how your game handles memory.

The garbage collector is slower on consoles and if your game generates a lot of garbage memory, there will be visible stutters.

To verify that your game is not too garbage-prone, you can run Visual Studio's Performance Profiler (`Debug/Performance Profiler...`) and check the **".NET Object Allocation Tracking"** tool.

From there, you can check which parts of your code generate garbage and you can pinpoint where to optimize.

In order to avoid garbage, here are some best practices:

- Only use strings as const. Do not use strings dynamically (e.g. string concatenation, patterns...), this is the most common source of garbage.
- Avoid allocating anything with the `new` keyword during your game loop, e.g. pre-allocate everything ahead of using them during gameplay.
- Pool your dynamic objects, e.g. do not destroy your projectiles or particles, instead place them into another "unused" list and reuse them instead of creating new instances when needed.
- Avoid using LINQ.
- If you are using collections, initialize them with a large enough capacity to avoid their internal data structure being silently recreated.
- Mind your foreach loops, depending on the data you are looping on, the loop might create garbage when duplicating an item. Or better yet, use a for loop for tigher control.

Do not rely on system calls

If your game calls directly to system functions, like kernel, win32 or Unix commands, you might want to get rid of them.

Consider I/O to be asynchronous

Saving player data/settings, or unlocking achievements are operations that should be considered asynchronous.

Most, if not all consoles, consider system accesses to be asynchronous. Even though it is not on PC, you should prepare your game to handle asynchronous operation (e.g. consider saving game data in a thread which will not block the game).

If you consider all your I/O and system operations as asynchronous, you will likely be spared some headache.

Suggestions

If you have other tips or suggestions when building for consoles, then let the MonoGame team know by raising an issue and we will improve this article even further over time.

Using the Development Nuget Packages

How to use the latest development NuGet packages to use the cutting edge of MonoGame development in your project.

Overview

When the MonoGame develop branch builds, it publishes development NuGet packages to the MonoGame NuGet Feed on GitHub. If you want to test a new feature or just be on the very latest code you can use this Feed to do that.

> **TIP**
>
> Alternatively, consider using the Preview Packages provided by MonoGame to get access to the latest preview release rather than the raw development code.
>
> - How to install MonoGame Preview packages

Adding a NuGet Source

1. Create a `NuGet.config` in the root or top level directory of your project.

> **NOTE**
>
> NuGet will automatically walk up the directory tree to find `NuGet.config` files.

2. Add the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <packageSources>
        <add key="MonoGameGitHub"
value="https://nuget.pkg.github.com/MonoGame/index.json" />
    </packageSources>
</configuration>
```

3. Next (in the root or top level directory), create a `Directory.Build.props` file and add the following content.

```
<Project>
    <PropertyGroup>
        <MonoGamePackageVersion>1.0.0.1233-develop</MonoGamePackageVersion>
    </PropertyGroup>
</Project>
```

`Directory.Build.props` is an MSBuild file which will be imported by all projects in your game. It is like a file that contains global variables. In this case the version of MonoGame we want to use.

> **NOTE**
>
> To find out the latest version number, you can look at one of the packages at https://github.com/orgs/MonoGame/packages?repo_name=MonoGame. Or to get the information from the GitHub feed, you can run the following command.
>
> ```
> nuget search "MonoGame.Framework" -PreRelease -Source MonoGameGitHub
> ```

This will give you the following output

```
====================
Source: MonoGameGitHub
--------------------
> MonoGame.Framework.Android | 1.0.0.1278-develop | Downloads: 164
  The MonoGame runtime for Android.
--------------------
> MonoGame.Framework.Content.Pipeline | 1.0.0.1278-develop | Downloads: 69
  The Monogame Content Pipeline for Windows, Mac and Linux is used to compile
raw content to xnb files...
--------------------
> MonoGame.Framework.DesktopGL | 1.0.0.1278-develop | Downloads: 337
  The MonoGame runtime supporting Windows, Linux and macOS using SDL2
and OpenGL.
--------------------
> MonoGame.Framework.Native | 1.0.0.1278-develop | Downloads: 1
  The MonoGame Native platform.
--------------------
> MonoGame.Framework.WindowsDX | 1.0.0.1278-develop | Downloads: 76
  The MonoGame runtime for Windows using DirectX API's.
--------------------
> MonoGame.Framework.WindowsUniversal | 3.8.1.1128-develop | Downloads: 54
  The MonoGame runtime for UWP (Universal Windows Platform) which supports
Windows 10 and Xbox One.
--------------------
> MonoGame.Framework.iOS | 1.0.0.1278-develop | Downloads: 153
  The MonoGame runtime for iOS amd iPadOS.
--------------------
```

The version number you want to use is listed in the output.

> **NOTE**
>
> As packages are published, the version number will always change. Unfortunately, due to limitations in the way NuGet works, we cannot use a wildcard with a pre-release package (so you cannot do `1.0.0.*-develop`). So this is the best way to find the latest version you want to use.

4. Next update all the `PackageReference` entries in your `csproj`'s which use MonoGame to use `$(MonoGamePackageVersion)` MSBuild property. For example:

```
<ItemGroup>
    <PackageReference Include="MonoGame.Framework.DesktopGL"
Version="$(MonoGamePackageVersion)" />
    <PackageReference Include="MonoGame.Content.Builder.Task"
Version="$(MonoGamePackageVersion)" />
</ItemGroup>
```

If you try to build now you will get an error. This is because the NuGet feeds on GitHub are not public. You need to be a valid GitHub user to use them.

Authentication

You need to create a **Personal Access Token** (PAT) on your GitHub account in order to use the NuGet feed. See the following documentation on how to create your PAT.

managing-your-personal-access-tokens

> **NOTE**
>
> You need to create a "PAT (Classic)" token in order for it to work with the Nuget feed. See creating-a-personal-access-token-classic for details.

Once you have your **PAT**, you can create a new `NuGet.config` file in the directory ABOVE your game project directory. To be clear, this file should NOT be in your source tree. It should be outside of any directory which is under source control.

```
Projects
     NuGet.config <-- THIS IS WHERE YOU PUT THE FILE.
     MyGame
       .git
       Directory.Build.props
       NuGet.config
       MyGame.DesktopGL
         MyGame.DesktopGL.csproj
```

> **IMPORTANT**
>
> Do Not... **DO NOT** place a `NuGet.config` file with valid `packageSourceCredentials` in your source control.

The contents of the file are as follows, replace `%GITHUB_USER%` with your GitHub username and the `%GITHUB_TOKEN%` with your token.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <packageSourceCredentials>
        <MonoGameGitHub>
            <add key="Username" value="%GITHUB_USER%" />
            <add key="ClearTextPassword" value="%GITHUB_TOKEN%" />
        </MonoGameGitHub>
    </packageSourceCredentials>
</configuration>
```

The really good thing about placing these credentials outside of source control is that they are safe. But also any new projects you create under that folder can also make use of these credentials. So it is a good idea to keep them in one place.

> **NOTE**
>
> For more information, you can read consuming-packages-authenticated-feeds.

Tools

Distributed tools to help mange and build content for your game.

MonoGame distributes tooling to help manage and build content for your game.

These tools are available as .NET Tools and are automatically installed if you are using the MonoGame templates.

Usage instructions are on the following pages:

- MonoGame Content Builder (MGCB): used to build content pipeline content.
- MGCB Editor: The MonoGame Content Builder (MGCB) Editor is a graphical front end to edit mgcb files.
- MonoGame Effect Compiler (MGFXC): used to compile stand alone effects (shaders).

MonoGame Content Builder (MGCB)

Learn about the MonoGame Content Builder (MGCB), the command line tool for building XNB content on Windows, Mac, and Linux desktop.

The MonoGame Content Builder is a command line tool for building XNB content on Windows, Mac, and Linux desktop systems.

Typically, it is executed by the MGCB Editor when editing content or by `MonoGame.Content.Builder.Task` during the build process of a MonoGame project. Alternatively you can use it yourself from the command line for specialized build pipelines or for debugging content processing.

Installation

MGCB can be installed as a .NET tool. Make sure you have the .NET SDK installed. You can download it here.

In a terminal run `dotnet tool install -g dotnet-mgcb` to install MGCB. Then you can execute MGCB by simply running `mgcb`.

Command Line Options

The options are processed “left to right”. When an option is repeated, it is overwritten.

Output Directory

```
/outputDir:<directory_path>
```

Specifies the directory where all content is written. Defaults to the current working directory.

Intermediate Directory

```
/intermediateDir:<directory_path>
```

Specifies the directory where all intermediate files are written. Defaults to the current working directory.

Rebuild Content

```
/rebuild
```

Force a full rebuild of all content.

Clean Content

```
/clean
```

Delete all previously built content and intermediate files. Only the `/intermediateDir` and `/outputDir` need to be defined for clean to do its job.

Incremental Build

```
/incremental
```

Only build content that changed since the last build.

Assembly Reference

```
/reference:<assembly_path>
```

An optional parameter which adds an assembly reference which contains importers, processors, or writers needed during content building.

Target Platform

```
/platform:<target_Platform>
```

Set the target platform for this build. It must be a member of the TargetPlatform enum:

- Windows
- iOS
- Android
- DesktopGL
- WindowsStoreApp
- PlayStation4
- PlayStation5
- XboxOne
- Switch

If not set, it will default to Windows.

NOTE: PlayStation 4, PlayStation 5, Xbox One, and Switch support is only available to licensed console developers.

Target Graphics Profile

```
/profile:<graphics_Profile>
```

Set the target graphics profile for this build. It must be a member of the GraphicsProfile enum:

- HiDef
- Reach

If not set, it will default to HiDef.

Target Build Configuration

```
/config:<build_config>
```

The optional build configuration name from the build system. This is sometimes used as a hint in content processors.

Content Compression

```
/compress
```

Uses LZ4 compression to compress the contents of the XNB files. Content build times will increase with this option enabled. Compression is not recommended for Android as the app package is already compressed. This is not compatible with LZX compression used in XNA content.

Content Importer Name

```
/importer:<class_name>
```

An optional parameter which defines the class name of the content importer for reading source content. If the option is omitted or used without a class name the default content importer for the source type is used.

Content Processor Name

```
/processor:<class_name>
```

An optional parameter which defines the class name of the content processor for processing imported content. If the option is omitted used without a class name the default content processor for the imported content is used.

Note that when you change the processor all previously defined `/processorParam` are cleared.

Content Processor Parameter

```
/processorParam:<name>=<value>
```

An optional parameter which defines a parameter name and value to set on a content processor.

Note all defined processor parameters are cleared when the `/processor` is set.

Build Content File

```
/build:<content_filepath>
/build:<content_filepath>;<destination_filepath>
```

Instructs the content builder to build the specified content file using the previously set switches and options. Optional destination path may be specified if you want to change the output file path.

Launch Debugger

```
/launchdebugger
```

Allows a debugger to attach to the MGCB executable before content is built.

Response File

```
/@:<response_filepath>
```

This defines a text response file (sometimes called a command file) that contains the same options and switches you would normally find on the command line.

Each switch is specified on a new line. Comment lines are prefixed with #. These lines are ignored. You can specify multiple response files or mix normal command line switches with response files.

An example response file could look like this:

```
# Directories
/outputDir:bin/foo
/intermediateDir:obj/foo

/rebuild

# Build a texture
/importer:TextureImporter
/processor:TextureProcessor
/processorParam:ColorKeyEnabled=false
/build:Textures\wood.png
/build:Textures\metal.png
/build:Textures\plastic.png
```

Preprocessor Macros

Response files support preprocessor macros to allow conditionals within a response file.

```
$if <name>=<value>
$endif
```

Preprocessor symbols can be defined from the command line with the `define` option or in a response file with the `$set` directive.

```
<example command line>
MGCB.exe /define:BuildEffects=No /@:example.mgcb

<example.mgcb file>
$if BuildEffects=Yes
   /importer:EffectImporter
   /processor:EffectProcessor
   /build:Effects\custom.fx
   # all other effects here....
$endif


$set BuildEffects=Yes

$if BuildEffects=Yes
    # ...
    # This is executed
$endif
```

For booleans you can omit a value to set a symbol and to check if it is set:

```
$set BuildEffects

$if BuildEffects
    # ...
    # This is executed
$endif
```

Customizing your Build Process

When building content from your project with `MonoGame.Content.Builder.Task`, there are a few ways to hook into the build process. `MonoGame.Content.Builder.Task` runs a target called `RunContentBuilder` just before your project builds. If you want to do any processing before or after this process you can use the `BeforeTargets` and `AfterTargets` mechanism provided by `msbuild` to run your own targets.

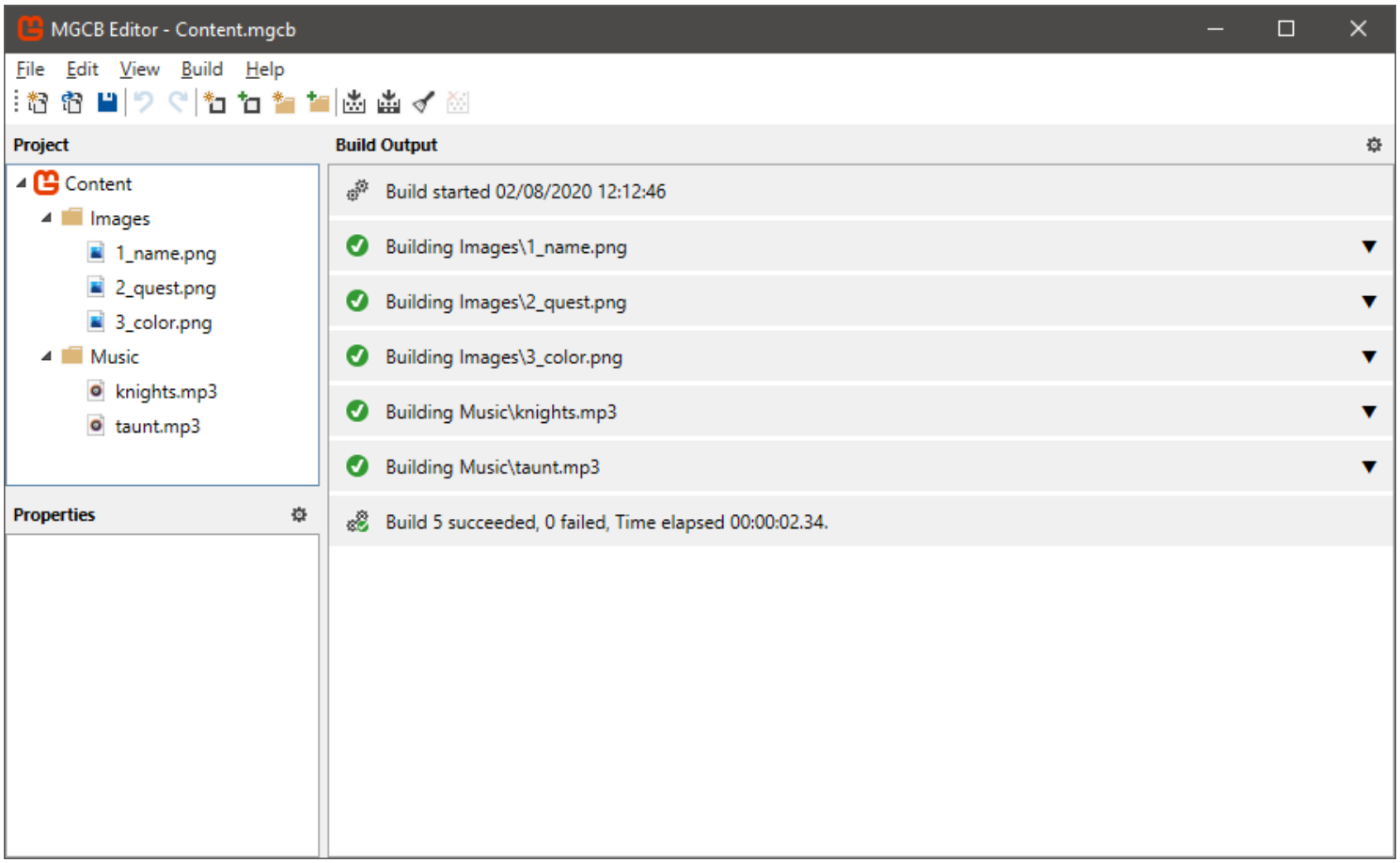
```
<Target Name="MyBeforeTarget" BeforeTargets="RunContentBuilder">
   <Message Text="MyBeforeTarget Ran"/>
</Target>
<Target Name="MyAfterTarget" AfterTargets="RunContentBuilder">
   <Message Text="MyAfterTarget Ran"/>
</Target>
```

If you want to customize the arguments sent to the `MGCB.exe` as part of the build process you can use the `<MonoGameMGCBAdditionalArguments>` property to define those. For example to pass in the current project configuration you could include the following code in a PropertyGroup in your .csproj file.

```
<MonoGameMGCBAdditionalArguments>-config:$(Configuration)
</MonoGameMGCBAdditionalArguments>
```

MGCB Editor

Learn about the MonoGame Content Builder (MGCB) Editor, the front-end GUI editor for MonoGame content builder projects.



The MGCB Editor has the following features:

- Create, open, and save MGCB projects.
- Import existing XNA .contentproj.
- Tree view showing content of project.
- Property grid for editing content settings.
- Full undo/redo support.
- Build, rebuild, and clean the project.
- Rebuild selected items.
- Create new content like fonts and xml.
- Support for custom importers/processors/writers.
- Template format for adding new custom content types.

Installation Instructions

The MGCB Editor is automatically installed (if you are using MonoGame's templates) and accessible by double-clicking an .mgcb file from Visual Studio 2022 if you have the extension installed, or right-clicking and selecting "Open in MGCB editor" in VSCode if you have the `MonoGame Content Builder (editor)` extension installed.

Alternatively, you can open the MGCB Editor from the .NET command line. This will only work if you are using the MonoGame templates and executing the command from the root directory of your project:

```
dotnet mgcb-editor
```

> **NOTE**
>
> You will need to buid the project at least once in order for the .NET system to download and register the tool with your project utilizing the `dotnet-tools.json` configuration file located in the `.config` folder, or use the `dotnet tool restore` command shown below.
>
> ```
> dotnet tool restore
> ```

See Using MGCB Editor for more information.

MonoGame Effects Compiler (MGFXC)

Learn about the MonoGame Effects Compiler (MGFXC), the tool used to compile shaders for usage in MonoGame.

The MGFXC tool is used to compile DirectX Effect files (shaders) for usage with MonoGame. It will compile shaders into mgfxo files ready to be consumed by the `Effect` class.

The MGCB Editor uses MGFXC to compile effects and wrap them into an xnb file, so they can be loaded using the `ContentManager`.

If you compile effects directly with MGFXC you can load effects using the `Microsoft.Framework.Xna.Graphics.Effect` constructor that takes a byte array with the effect code.

Effects compiled directly are not content files and can not be loaded by the `ContentManager`.

Installation

MGFXC can be installed as a .NET tool. Make sure you have the .NET SDK installed. You can download it here.

In a terminal run `dotnet tool install -g dotnet-mgfxc` to install MGFXC.

Command Line

The command line options are:

```
mgfxc <SourceFile> <OutputFile> [/Debug] [/Profile:<DirectX_11,OpenGL,PlayStation4>]
```

Source File

The input effect file in typical FX format with samplers, techniques, and passes defined. This parameter is required.

Output File

The path to write the compiled effect to. This parameter is required.

NOTE: The generated file is not an XNB file for use with the ContentManager.

If the `/Debug` flag is passed the resulting compiled effect file will contain extra debug information and the fewest possible optimizations.

Platform Profile

The `/Profile` option defines the platform we're targeting with this effect file. It can be one of the following:

- DirectX_11
- OpenGL
- PlayStation4
- XboxOne
- Switch

NOTE: PlayStation 4, Xbox One, and Switch support is only available to licensed console developers.

Help

If you use `/?`, `/help`, or simply pass no parameters to MGFXC you will get information about these command-line options.

Runtime Use

The resulting compiled effect file can be used from your game code like so:

```
byte[] bytecode = File.ReadAllBytes("mycompiled.mgfx");
var effect = new Effect(graphicsDevice, bytecode);
```

This is how the stock effects (BasicEffect, DualTextureEffect, etc) are compiled and loaded.

Adding Content

Learn how to add content to your game using the Content Pipeline.

A big part of your game is your content. This includes standard files like textures, sound effects, music, videos, and custom effects as well as custom content like level and enemy files.

MonoGame implements its own content pipeline for transforming your unoptimized assets into platform optimized content. This is critical in building a game which runs as fast as possible under tight resource constraints.

This section will cover the following topics:

- Why use the Content Pipeline
- Using MGCB Editor
- TrueType Fonts
- Custom Effects
- Localization

Why use the Content Pipeline

Why would you use the Content Pipeline when MonoGame also supports loading assets natively, like .png, .mp3, .wav? Well, it all boils down to a couple of words, performance and efficiency.

The MonoGame team continues to put a lot of effort into a cross-platform content pipeline, but why would you use the Content Pipeline when MonoGame also supports loading assets natively, like .png, .mp3, .wav? Well, it all boils down to a couple of words, performance and efficiency.

Textures

Most image formats (like PNGs) are generally not optimized for games, most GPUs have specific hardware supported compressed formats for textures. PNGs or JPEGs just get uncompressed when passed to the GPU. By using the hardware compression you often get 4x to 8x more textures space for your game as well as faster load times and smaller packages.

When we load the .png from storage at runtime, the texture is then loaded into memory and decompressed/unpacked from its compressed png format into raw bytes.

> Note that 262 KB is quite a bit bigger than the compressed size.

A new texture is then created for that data because your device cannot decompress on the fly (yet) so it has to use that data as is. Creating the texture uses 262kb of graphics memory on the GPU. That does not sound too bad, but if you are using larger textures say 1024×1024, then you are using 4 MB or more of GPU memory for that one texture. Multiply that over the number of textures in your game and you soon run out of texture memory on the GPU. If this happens then the GPU has to swap that data out into system memory (if it supports that) or throw an error when you try to create textures that will not fit into available memory.

> So to sum up:
>
> using **.pngs** = smaller package size & higher memory usage & less textures

If you pre-process the texture using the content pipeline, because we know that we are targeting iOS and we know the GPU on those devices support using PVRTC texture compression, we can compress the textures ready for the GPU to use. So we take our sample .png and compress it at build time using PVRTC and we end up with a 32kb file (size depends on the texture, alpha channel, etc). Hmm, that is a lot bigger than the .png on disk, but that is not the whole story. The difference here is that there is no need to unpack/decompress it at runtime which saves on load time, also, we can create a texture from that data directly so we only use 32kb of texture memory on the GPU and not 262kb. That is a massive saving.

> Summing up:
>
> **compressed textures** = larger package size (maybe) & lower runtime memory usage & more textures

This applies to all platforms as most desktop GPUs support DXT texture compression, so the content pipeline will produce DXT compressed textures which can be loaded and used directly. The Android platform currently does not have consistent support for compressed textures at the moment so MonoGame has to decompress DXT textures on the device and use it directly, when this changes MonoGame will adapt.

In the Content Pipeline tool, MonoGame will automatically pick the correct texture format to use, so for opaque textures, it will use ETC1 (which is supported on all android devices but does not support alpha channels) but for textures with an alpha channel, it will use RGBA4444 (dithered). It will also allow the user to override this default and enable picking from a wide variety of compression options manually such as PVRTC, ATITC, DXT/S3TC, ETC1, and RGBA4444. This will give the developer the choice of what to use/support.

Audio

All platforms support different audio formats, if you are handling this yourself you will need to manually convert all your files and include the right formats for each platform. A better option would be to keep one source file (be it .mp3, .wmv, etc) and then convert it to a supported format for the target platform at build time. This creates longer build times, but at least we know the music will work. MonoGame uses ffmpeg to do the heavy lifting when converting between formats as it can pretty much convert any type to any other type which is really cool.

> The build times are only for the first time the asset is processed, the Content Pipeline will preserve the generated content for subsequent builds until the source is changed

Most platforms have audio processing that is optimized to certain compressed formats. By not using them you loose performance and system memory. If you save the sound effects as ADPCM, these run pretty optimally on Windows systems. The problem however, is that once you try to take your game to another platform where ADPCM is not optimal. You will need to manually re-export all your game content into the new optimal format for that platform (assuming you kept all your original content uncompressed).

Shaders

There are a number of shading languages that you can use depending on the platform you are targeting:

- For OpenGL based systems that is GLSL
- For DirectX based systems it is HLSL
- There is also CG from Nvidia.

The Effect system in XNA/MonoGame was designed around the HLSL language as it is based around the .fx format, which allows a developer to write both vertex and pixel shaders in one place. Historically both GLSL and HLSL have separate vertex and pixel shaders. HLSL until recently compiled and linked these at build time, however, GLSL does this at runtime.

Without a content pipeline or some form of tooling a developer would need to write two shaders, one for HLSL and one for GLSL. The good news is the MonoGame MGFX.exe tool can create a shader in .fx format and also enable it to work in GLSL. It does this by using an open-source library called libmojoshader, which does some funky HLSL to GLSL instruction conversion to create an OpenGL-based shader, but rather than doing that at runtime, we do it at build time so we do not need to deploy mojoshader with the OpenGL based games. All this saves you the hassle of having to write and maintain two shaders.

> From MonoGame 3.8, the MGFX tool can build content on any platform. For MonoGame 3.7 or earlier, MGFX only runs on a windows box, this is because it needs the DirectX shader tooling to compile the HLSL before passing it to libmojoshader for conversion (for OpenGL platform targets).

Models

MonoGame, thanks to the excellent assimp project, supports a much wider range of models including .x, .fbx and .3ds. However, some of these formats might produce some weirdness at render time as only .fbx has been fully tested. Also note that assimp does not support the older version .fbx format which ships with most of the original XNA samples, so you will need to convert those to the new format manually.

> A nice trick to upgrade the old .fbx is to open them in Visual Studio 2012+ and then save it again under a new name. This article walks through some of the quick, easy, and FREE methods for updating older .fbx files.

When the MonoGame content pipeline processes a Model file, it is converted into an optimized internal format that will contain the Vertices, Texture Coordinates, and Normals. The pipeline will also pull out the textures used in the model and put those through the pipeline too, so you get optimized textures automatically.

Summary

This is why using the content pipeline and XNB's is superior. You feed uncompressed source content to the MonoGame content pipeline, tell it what platform you are building for and it takes care of converting it to the most optimal format for that platform. When you decide to ship to another platform it is a one-click change to rebuild your content optimized for that platform.

Hopefully, you have a good idea of why you should use the content pipeline in your games. Using the raw assets is ok when you are putting together a simple demo or proof of concept but sooner or later you will need to start optimizing your content. My advice would be to use the Pipeline tooling from the outset so you get used to it.

Information on the Pipeline tool can be found here.

Working with new Content Builder Projects

Learn how to use the newest approach to building content in MonoGame without the legacy MGCB-Editor.

> **NOTE**
>
> These instructions **REQUIRE** you to use the latest `MonoGame Develop` release (currently `3.8.5-develop.13`)
>
> See the instructions here for how to install the preview project templates and update your project to use `3.8.5-develop` (preview) releases.
>
> How to install MonoGame Preview packages

The MonoGame Content Pipeline is a powerful tool for packaging and managing all the complex needs for working with and distributing assets in your MonoGame projects, solving all the complexities surrounding the use of assets on all the platforms that MonoGame supports.

From MonoGame `3.8.5` we are making a new `Console Project Style` solution to more effectively manage your content in source to reduce complexity (and installation demands) and put control of your source assets fully within your control.

> **IMPORTANT**
>
> **This documentation is a work in progress and subject to change.**
>
> While every effort will be made during the `3.8.5` preview to keep this documentation up to date, check back often for reference and log issues for corrections to assist the team in its development.

Table of Contents

1. What is the ContentBuilder?
2. Getting Started
3. Adding the Content Builder Project to your Game
4. Understanding ContentBuilderParams
5. Creating Your ContentCollection
6. Including and Excluding Content
7. Working with Importers and Processors
8. Advanced Scenarios

If you prefer, the MonoGame Foundation recorded a short video demonstrating how to get setup with the new Content Builder Project.

What is the ContentBuilder?

The `ContentBuilder` is a powerful tool that allows you to build game content (like images, sounds, fonts, and models) programmatically using C# code instead of manually configuring content files. Think of it as your own personal content pipeline that you control entirely through code!

Instead of an installed dotnet tool (which historically generated issues), the new approach uses a simple `Console` application you manage, which is simply executed to build and generate your content. By default, the project template pulls all the content from the designated "**Assets**" folder and compiles it using the default importers used by MonoGame. To customize this, you only need to specify additional commands to override this default behaviour with your customizations.

> **TIP**
>
> When designing the builder for your content, try to focus on building patterns for your content to reduce the need to rebuild the `Builder` itself.
>
> In an ideal scenario, you build the `Builder` once, then just keep adding assets and running the builder to produce the runtime content, identifying "early on" any issues with your assets and simplifying production.

In addition, due to content now being built through your own console application, it is easily distributable and can even be used in automation pipelines in source control.

Features provided with the new Content Builder project

- **Automation**: Build content as part of your build process.
- **Flexibility**: Apply complex rules to determine which files get processed.
- **Control**: Fine-tune how each asset is imported and processed.
- **Integration**: Perfect for tools, editors, and custom-built pipelines.
- **Customization**: Inject additional capabilities into your asset management as required.

> **TIP**
>
> Yes, it is completely possible, through automation, to have your Content and Game EXE/deliverables built separately, and then simply combined later when packaging the solution.
>
> We will look to provide more guides in the future to showcase this capability.

Getting Started

To start using the ContentBuilder, you will need to create a console project using the MonoGame ContentBuilder template. Here are instructions to create your own Content Builder Project:

Visual Studio Code

To add the class library using the `MonoGame Content Builder` project template in Visual Studio Code, perform the following:

1. In the *Solution Explorer* panel, right-click the solution.
2. Choose *New Project* from the context menu.
3. Enter "MonoGame Content Builder" and select it as the template to use.
4. Name the project "MyContentBuilder".
5. When prompted for a location, use the default option, which will put the new project in a folder next to your game project.
6. Select "Create Project".

To add the class library using the `MonoGame Content Builder` project template in Visual Studio 2022, perform the following:

1. Right-click the solution in the Solution Explorer panel.
2. Choose Add > New Project from the context menu.
3. Enter "MonoGame Content Builder" in the search box, select that template, then click Next.
4. Name the project "MyContentBuilder".
5. The default location will place the new project in a folder next to your game project; you do not need to adjust this setting.
6. Click "Create".

To create a Content Builder project using the `MonoGame Content Builder` project template with the dotnet CLI, perform the following:

1. Open a new Command Prompt or Terminal window in the same folder as your Game solution file. (or in an empty folder if it is separate from your Game)
2. Enter the command `dotnet new mgcb -o MyContentBuilder` to create the project, placing it in a folder next to your game project.
3. (optional) Enter the command `dotnet sln <your solution>.sln add ./MyContentBuilder/MyContentBuilder.csproj` to add the newly created Content Builder project to your game's solution file.

This creates a project with the following structure:

```
MyContentBuilder/
├── Builder/
│   └── Builder.cs          # Your custom builder class
├── Assets/
│   └── readme.txt          # Placeholder text file to be replaced with your content
└── MyContentBuilder.csproj  # Project file with required packages
```

This is the default (recommended) layout for a Content Builder project, but you can customize its use how you wish:

- There is a single "entry point" (command line launch point) contained in the `builder.cs` which contains all the instructions to build content.
- A default `Assets` folder to host your content, but your content can be in any location you need it to be, simply update the parameters used to run the console application to point to the source directory and folder for your content.

> **TIP**
>
> While it is possible to include the builder as part of your game code, rather than a separate project, at this time we do not recommend this. Based on experience, it is always better to build your content separate from your runtime game to avoid collisions or issues arising in game production/deployment.

Adding the Content Builder Project to your Game

Once you have created your `Builder` project, you have two options:

1. After building your game, you need to copy the build output from the Builder to the output of the Game, so the `exe` and the `content` folder are in the same output folder.
2. Add a `Target` definition to your `csproj` to automatically copy the Builder output to the game output when the project is built (like how the current MGCB solution works)

This section will focus on Option 2 to replicate what happens in `3.8.4` projects.

1. Clean up redundant files

As the Content Builder project is **NOT** a DotNet tool, you no longer need the `dotnet-tools.json` configuration.

1. Remove any `.config` folders from your solution that contain `dotnet-tools.json` files.

2. Remove legacy MGCB references from your `csproj`

As you would expect, we no longer need any old `Content` references or tools for building that content, aka, the `MonoGame.Content.Builder.Task` reference.

Per platform/game project:

1. Remove any `MonoGameContentReference` elements from your game's `csproj`.
2. Remove references to `MonoGame.Content.Builder.Task`.
3. Update the MonoGame version to match the Builder (not essential, but recommended)

3. Add the Builder task

To complete the process, add the following `Target` section to your `csproj` (before the final `</Project>` entry):

```
  <Target Name="BuildContent" BeforeTargets="Build">
    <PropertyGroup>
      <ContentOutput>$(ProjectDir)$(OutputPath)</ContentOutput>
      <ContentTemp>$(ProjectDir)$(IntermediateOutputPath)</ContentTemp>
      <ContentArgs>build -p $(MonoGamePlatform) -s Assets -o $(ContentOutput) -i
$(ContentTemp)</ContentArgs>
    </PropertyGroup>
    <MSBuild Projects="..\Content\Content.csproj" Targets="Build;Run"
      Properties="RunArguments=$(ContentArgs);" />
  </Target>
```

Of note, you should take care with the following elements:

- `-s Assets` - this assumes you are using the default "Assets" source folder in the builder project. If your source content is elsewhere you will need to provide the path to it.
- `-i $(ContentTemp)` - this uses the default `obj` folder for building the content, if you are short on space, you can provide an alternate path for temp storage.
- `Projects="..\Content\Content.csproj"` - This should **MATCH** the folder and project name of your Builder. If you called your Builder something else or renamed the folder/csproj, make sure it matches.

You should not need to touch anything else.

> **NOTE**
>
> Additionally, if you want to control HOW and WHEN your content is built/rebuilt, then you can change the `Targets="Build;Run"` options to only execute the last on Build OR Run, or a different action.
>
> See the MS Documentation on Built targets for reference.

Platform specific additions - Android / iOS

For platforms where the content is required to be packaged inside the output bundle, there are some additional steps required to ensure the built content is included during the Build/Publish step.

Each platform has specific requirements and requires the built content to be available prior to processing the project itself.

> **NOTE**
>
> Also check out the dedicated Packaging Games advice published by the MonoGame team.

Android

For Android, a specific `Include` section is needed in the project's `csproj` project definition, as follows:

> **NOTE**
>
> Android uses a special `AndroidAsset` tag to identify content to include.

```
<ItemGroup>
    <AndroidAsset Include="$(OutputPath)Content\**\*">
        <Link>Content\%(RecursiveDir)%(Filename)%(Extension)</Link>
    </AndroidAsset>
</ItemGroup>
```

> **NOTE**
>
> If you change the location where the Content Builder project outputs content to, the paths above will need to be updated to the new location to find the content to include.

At build time, the "BuildContent" task will run and execute the Content Builder, then with the output from the builder, the Android solution will automatically identify and package the content as per Google's packaging requirements, generating an `APK` for sideloading and an `AAB` for publishing to the Google Play Store.

> **NOTE**
>
> If your content is too large to fit within the maximum size of the base application, you will need to use Asset Bundles to package your content, as demonstrated here:

```
<ItemGroup>
   <AndroidAsset Include="$(OutputPath)Content\**\*">
       <Link>Content\%(RecursiveDir)%(Filename)%(Extension)</Link>
   </AndroidAsset>
   <AndroidAsset Update="Content\Path\SomeLargeFile.xnb" AssetPack="foo" />
</ItemGroup>

However, your game project will also need to be aware and update its strategy
for loading the content to take the asset packs into account.

For more details, see the [Microsoft Documentation]
(https://devblogs.microsoft.com/dotnet/android-asset-packs-in-dotnet-android/).
```

iOS

For iOS, a specific `Include` section is needed in the project's `csproj` project definition, as follows:

> **NOTE**
>
> iOS uses a special `BundleResource` tag to identify content to include.

```
<ItemGroup>
    <BundleResource  Include="$(OutputPath)Content\**\*">
        <Link>Content\%(RecursiveDir)%(Filename)%(Extension)</Link>
    </BundleResource >
</ItemGroup>
```

> **NOTE**
>
> If you change the location where the Content Builder project outputs content to, the paths above will need to be updated to the new location to find the content to include.

At build time, the "BuildContent" task will run and execute the Content Builder, then with the output from the builder, the Android solution will automatically identify and package the content within the App as per Apple's packaging standards.

Basic Builder Project Structure

The default `Builder.cs` file looks like this, which contains everything needed to build all content in the designated Assets/Content folder using the default Importers and Processors (how MonoGame compiles content) into processed `.XNB` files for consumption by your runtime project:

```
using Microsoft.Xna.Framework.Content.Pipeline;
using MonoGame.Framework.Content.Pipeline.Builder;

// Console program entry point, make a new builder and run using
commandline arguments
var builder = new Builder();
builder.Run(args);
return builder.FailedToBuild > 0 ? -1 : 0;


public class Builder : ContentBuilder
{
    public override IContentCollection GetContentCollection()
    {
        var contentCollection = new ContentCollection();

        contentCollection.Include<WildcardRule>("*");

        // Add your content rules here

        return contentCollection;
    }
}
```

> **NOTE**
>
> The example above has been simplified for reference. The default template includes options to run the Content Builder project without parameters using default configuration to aid testing.

The default `Include` rule simply builds a list of content to process from the target folder when the program is executed.

> **IMPORTANT**
>
> We recommend keeping this default structure of getting all files and then specifying overrides. You can alter this if you so wish to only use specific includes, but that it would require more maintenance as your content grows. Try to be generic, design rules based on the structure of your content rather than being too specific.
>
> Although check the details later about providing parameters, while samples are currently "hard coding" importer/processor values, this would be improved through your own configuration.

Understanding ContentBuilderParams

`ContentBuilderParams` defines the operational configuration that tells the ContentBuilder where to find your content, where to output it, and how to process it. Think of it as the "settings panel" for your content building operation.

Core Properties

Here are the main properties available to work with:

Property	Description	Default Value
`WorkingDirectory`	The root directory for all operations	Current directory
`SourceDirectory`	Where your source assets are located	`"Content"`
`OutputDirectory`	Where built content will be saved	`"bin/Content"`
`IntermediateDirectory`	Where temporary build files go	`"obj/Content"`
`Platform`	Target platform for the content	`TargetPlatform.DesktopGL`
`GraphicsProfile`	Graphics capability level	`GraphicsProfile.HiDef`
`CompressContent`	Whether to compress output files	`false`
`LogLevel`	How much information to display	`LogLevel.Info`
`Rebuild`	Force rebuild all content (ignore cache)	`false`

Creating ContentBuilderParams

Option 1: Direct Initialization

```
var contentParams = new ContentBuilderParams()
{
    Mode = ContentBuilderMode.Builder,
    WorkingDirectory = $"{AppContext.BaseDirectory}../../../",
    SourceDirectory = "Assets",
    OutputDirectory = "bin/DesktopGL/Content",
    Platform = TargetPlatform.DesktopGL,
    GraphicsProfile = GraphicsProfile.HiDef,
    CompressContent = false,
    LogLevel = LogLevel.Info
};
```

Option 2: From Command Line Arguments

```
var contentParams = ContentBuilderParams.Parse(args);
```

This parses command line arguments like:

```
mybuilder.exe --src "MyAssets" --output "bin/Content" --platform DesktopGL -
-compress
```

Understanding Directory Paths

The directory paths work together to organize your content:

> **IMPORTANT**
>
> The `WorkingDirectory` should always be fixed path or directly resolvable from the application's execution path, which CAN be different based on whether the console application is run in `release` or `debug` mode. The default path provided uses a `relative` path from the point of execution. If in doubt, test with an `absolute` path and work back from there.

```
WorkingDirectory/           # Your project root
├── SourceDirectory/        # Input: Your raw assets (images, sounds, etc.)
│   ├── Textures/
│   └── Audio/
├── OutputDirectory/        # Output: Built .xnb files ready for your game
│   ├── Textures/
│   └── Audio/
└── IntermediateDirectory/  # Temporary: Cache and build information
    └── cache/
```

Important Note: Paths can be absolute or relative to the `WorkingDirectory`.

Creating Your ContentCollection

The `ContentCollection` is where you define the rules for how your content should be handled. It is like creating a recipe book that tells the ContentBuilder what to do with each file.

Basic ContentCollection Setup

```
public override IContentCollection GetContentCollection()
{
    var contentCollection = new ContentCollection();

    contentCollection.Include<WildcardRule>("*");

    // Your rules go here

    return contentCollection;
}
```

The default process queries your target assets folder and builds a list of all content, this ensures that **EVERYTHING** contained within that folder is available for processing, making it easy to simply add new content files to the structure without having to rebuild the `ContentBuilder`. Only when you need to change the rules or specify different processing logic for various asset types or folders should you need to change this.

Including and Excluding Content

The ContentCollection provides flexible ways to specify which files should be processed and how, by default in the template, "All content" is included and processed using the defaults. Any files that cannot be compiled will be reported in the console output.

Any additional `ContentCollection` commands provided after the global include simply update the "processing" list, as all content is only built ONCE (per folder).

> **TIP**
>
> The recommended approach is to leave the default "include" rule to gather all assets, and then provide overrides to the assets collected below. You "Can" remove the default include and just write specific inclusion commands if you wish instead, the choice is completely up to you and how you want to manage your content.
>
> Just make sure to use a path that reduces the need to rebuild the "Builder" application using policies or patterns for your content to minimise friction when building content!

Understanding Rule Order

Important: Rules are evaluated in order, and the **last matching rule wins**!

> **NOTE**
>
> Rules simply update the processing order for each file. Whatever the last rule applied to affect the file/folder definition is, is the final state in which a file will be processed. If you are not getting the output you expect, you can output the collection to file and examine it for reference, then check WHY a file was in its final state.

For example, the following rule order shows that all `png` files are included for processing initially, but then a later rule updates this and specifically excludes `png` files from processing. The result, no `png` files are processed and they are ignored:

```
// Include all PNG files
contentCollection.Include<WildcardRule>("*.png");

// But exclude debug files
contentCollection.Exclude<WildcardRule>("*debug*.png");

// Result: All PNG files EXCEPT those with "debug" in the name
```

Simple File Inclusion

If you want to do it the hard way, you can specify individual rules for including / updating content to be built, specifying specifically how the file should be processed, to which folder/file and how it should be processed.

> **NOTE**
>
> Note the "Copy" rule, there are situations in content where you simply DO NOT want the file to be processed, only copied to the output folder, normally where there is additional processing required at runtime.
>
> MonoGame does recommend using custom Importers and Processors to handle custom content, especially `XML`, `TXT`, or `JSON` files, as issues with content can be identified earlier when content is built (and not when your game runs, and then invariably crashes)

Include a Single File:

```
// Build a specific texture
contentCollection.Include("Textures/hero.png");

// Copy a file without processing
contentCollection.IncludeCopy("Data/config.json", "config.json");
```

With Custom Output Path:

```
// Save to a different location
contentCollection.Include("Sprites/player.png", "Characters/Player.xnb");
```

Pattern-Based Inclusion

The recommended pattern to use is to group your content either by folder or type and then use Wildcards to collectively refer to how content should be processed, e.g. all images in this folder should be processed in this way, rather than file by file.

Using Wildcards

Wildcards are perfect for matching multiple files with simple patterns:

```
// Include ALL files
contentCollection.Include<WildcardRule>("*");

// Include all PNG files
contentCollection.Include<WildcardRule>("*.png");

// Include all files in Textures folder
contentCollection.Include<WildcardRule>("Textures/*");

// Include all PNG files in any subfolder
contentCollection.Include<WildcardRule>("*/*.png");

// Include specific file types
contentCollection.Include<WildcardRule>("*.{png,jpg,bmp}");
```

Wildcard Pattern Reference:

Pattern	Matches
`*`	Any sequence of characters
`?`	Any single character
`[abc]`	Any character in the set
`[!abc]`	Any character NOT in the set
`{a,b,c}`	Any of the alternatives

Using Regular Expressions

For more complex patterns, use regular expressions:

> **NOTE**
>
> Be wary, RexEx expressions are a dark art and may behave differently depending on the language used. Use tools like regex101 (an online regex tester tool) to verify and check your search results, but note they have been known to operate differently based on OS/Language at runtime.

```
// Match files ending with _diffuse.png
contentCollection.Include<RegexRule>(@".*_diffuse\.png$");

// Match numbered files
contentCollection.Include<RegexRule>(@"level_\d+\.tmx");

// Match files in specific folders
contentCollection.Include<RegexRule>(@"^(Textures|Sprites)/.*\.png$");
```

Excluding Content

Sometimes you need to exclude certain files from processing:

> **NOTE**
>
> This prevents the Content Builder from doing anything with a file and effectively ignores it, useful for associated files (like `.ttf` font files usually included alongside a `.spriteFont` descriptor) that do not need processing as they are consumed by another content process.

Exclude Specific Files:

```
// Exclude a specific file
contentCollection.Exclude("Textures/debug_texture.png");

// Exclude multiple files
contentCollection.Exclude("Textures/test_01.png");
contentCollection.Exclude("Textures/test_02.png");
```

Exclude Using Patterns:

```
// Exclude all files starting with "test_"
contentCollection.Exclude<WildcardRule>("test_*");

// Exclude all files in Debug folder
contentCollection.Exclude<WildcardRule>("Debug/*");

// Exclude backup files
contentCollection.Exclude<RegexRule>(@".*\.bak$");

// Exclude temporary files
contentCollection.Exclude<RegexRule>(@".*~$");
```

Advanced Pattern Examples

As stated, it is recommended to define patterns for your content, defining rules for processing your content (and preventing changing/rebuilding the Content Builder exe), the following are examples of such patterns, grouping specific content into specific folders.

> **NOTE**
>
> There are no predefined rules for how you organize your content, it is completely up to you and the needs of your project to determine how it is best organized. Just make sure to include decisions on how to ease the burden of managing the importing content in those decisions so that you do not need to frequently keep updating your Builder.
>
> Set it once, and then you have strict rules for how content is processed and errors should be easily identifiable.

Organize by File Type:

```
var contentCollection = new ContentCollection();

// Images
contentCollection.SetContentRoot("Graphics");
contentCollection.Include<WildcardRule>("*.{png,jpg,bmp}");

// Audio
contentCollection.SetContentRoot("Audio");
contentCollection.Include<WildcardRule>("*.{wav,mp3,ogg}");

// Fonts
contentCollection.SetContentRoot("Fonts");
contentCollection.Include<WildcardRule>("*.spritefont");

// Everything else (copy as-is)
contentCollection.SetContentRoot("Data");
contentCollection.IncludeCopy<WildcardRule>("*.{xml,json,txt}");
```

Custom Output Paths with Functions

If you wish, you can also transform file paths dynamically:

```
// Flatten directory structure
contentCollection.Include<WildcardRule>(
    "Textures/**/*.png",
    outputPath: filePath => Path.GetFileName(filePath)
);

// Add prefixes
contentCollection.Include<WildcardRule>(
    "*.png",
    outputPath: filePath => $"processed_{filePath}"
);

// Reorganize structure
contentCollection.Include<WildcardRule>(
    "Source/*/icon.png",
    outputPath: filePath => {
        var folderName = Path.GetDirectoryName(filePath);
        return $"Icons/{folderName}_icon.xnb";
    }
);
```

Working with Importers and Processors

Importers and processors are the heart of the content pipeline. They determine how your raw assets are transformed into game-ready content and built to meet the needs for each target platform.

> **IMPORTANT**
>
> Different platforms have different requirements for content and while you can manage it yourself by packaging and loading it manually, this usually leads to complicated scenarios, especially when targeting consoles.

What are Importers and Processors?

- **Importer**: Reads your source file (e.g., PNG, WAV) and converts it into an intermediate format
- **Processor**: Takes the imported data and processes it into the final game format, sometimes including additional content or textures identified by the Importer. (Such as the skinned model importer)

Default Behavior

By default, MonoGame automatically selects the appropriate importer and processor based on file extension:

```
// MonoGame will automatically use:
// - TextureImporter for .png files
// - TextureProcessor to create the final texture
contentCollection.Include("hero.png");
```

Specifying Custom Importers and Processors

Sometimes you need more control. Here is how to specify which importer and processor to use and override the default behaviour:

Using Built-in Processors:

```
// Texture with specific processor
var textureImporter = new TextureImporter();
var textureProcessor = new TextureProcessor();
contentCollection.Include("hero.png", textureImporter, textureProcessor);

// Audio with specific processor
var audioImporter = new WavImporter();
var audioProcessor = new SoundEffectProcessor();
contentCollection.Include("explosion.wav", audioImporter, audioProcessor);
```

> **NOTE**
>
> As can be seen in the updated examples using the new Content Builder Project system, this can also be simplified to just:
>
> `contentCollection.Include("hero.png", new TextureImporter(), new TextureProcessor());`
>
> It just depends on which pattern you are more comfortable with.

Pattern-Based with Custom Processors:

This can also be used with pattern-based logic, applying the specified Importer/Processor to a group of assets:

```
// Apply texture processor to all PNG files
var textureImporter = new TextureImporter();
var textureProcessor = new TextureProcessor();
contentCollection.Include<WildcardRule>(
    "*.png",
    contentImporter: textureImporter,
    contentProcessor: textureProcessor
);

// Apply font processor to all sprite fonts
var fontImporter = new FontDescriptionImporter();
var fontProcessor = new FontDescriptionProcessor();
contentCollection.Include<WildcardRule>(
    "*.spritefont",
    contentImporter: fontImporter,
    contentProcessor: fontProcessor
);
```

Handling Custom Importer / Processor caching

The caching validation process that determines whether the Content Builder re-processes content by default is decided by:

- The comparison between the last state of the built asset and the source
- It will also take into account the "Version" of the Content Importer and Processor that was used at the time the content was built.

The cache is only invalidated when either of these conditions are met, else it will skip over processing of the asset for performance as it has not detected a change.

> **NOTE**
>
> MonoGame manages the versions of the built-in Importers and Processors, so unless MonoGame makes changes to these elements, the cache will not be invalidated unless you Force a rebuild.

If you want content to be rebuilt when a Custom Importer or Processor has changed (content that uses the custom entities in its builder configuration), then the version of the Custom Content Pipeline Extension has to change, to do this, simply update the "Version" property on the respective Importer or Processor as follows:

```
public override string Version { get; set; } = "new Version";
```

> **NOTE**
>
> The version can be anything you like, so long as it is a recognisable and readable string. Preferably something you can increment.

Configuring Processor Parameters

Many processors have configurable parameters that control how they process content. If you need to override these parameters with specific values, here is how to customize them:

Example: Texture Processing Options

```
var textureImporter = new TextureImporter();
var textureProcessor = new TextureProcessor
{
    ColorKeyColor = Color.Magenta,           // Transparent color
    ColorKeyEnabled = true,                   // Enable color keying
    GenerateMipmaps = true,                   // Create mipmaps
    PremultiplyAlpha = true,                  // Premultiply alpha
    ResizeToPowerOfTwo = false,               // Keep original size
    TextureFormat = TextureProcessorOutputFormat.Color
};

contentCollection.Include<WildcardRule>(
    "Sprites/*.png",
    contentImporter: textureImporter,
    contentProcessor: textureProcessor
);
```

Example: Audio Processing Options

```
var audioImporter = new WavImporter();
var soundProcessor = new SoundEffectProcessor
{
    Quality = ConversionQuality.Best
};

contentCollection.Include<WildcardRule>(
    "SoundEffects/*.wav",
    contentImporter: audioImporter,
    contentProcessor: soundProcessor
);
```

Example: Font Processing Options

```
var fontImporter = new FontDescriptionImporter();
var fontProcessor = new FontDescriptionProcessor
{
    PremultiplyAlpha = true,
    TextureFormat = TextureProcessorOutputFormat.Color
};

contentCollection.Include<WildcardRule>(
    "Fonts/*.spritefont",
    contentImporter: fontImporter,
    contentProcessor: fontProcessor
);
```

Advanced Scenarios

The following scenarios are of a more advanced nature for scenarios where you need specific customisation to suit your delivery needs.

Scenario 1: Platform-Specific Content

Build different content for different platforms:

> **NOTE**
>
> By default, content will be formatted specifically for the platform it is built for. This example is only where you might swap content based on the platform, e.g. for different quality sizes, store requirements or controller references. All to make the runtime project simpler to manage.

```
public override IContentCollection GetContentCollection()
{
    var contentCollection = new ContentCollection();

    if (Parameters.Platform == TargetPlatform.DesktopGL)
    {
        // High-quality textures for desktop
        var processor = new TextureProcessor
        {
            GenerateMipmaps = true,
            TextureFormat = TextureProcessorOutputFormat.Color
        };
        contentCollection.Include<WildcardRule>("*.png",
            contentProcessor: processor);
    }
    else if (Parameters.Platform == TargetPlatform.Android)
    {
        // Compressed textures for mobile
        var processor = new TextureProcessor
        {
            GenerateMipmaps = true,
            TextureFormat = TextureProcessorOutputFormat.DxtCompressed
        };
        contentCollection.Include<WildcardRule>("*.png",
            contentProcessor: processor);
    }

    return contentCollection;
}
```

Scenario 2: Multi-Language Content

Organize content by language:

> **NOTE**
>
> MonoGame recommends using a custom localized importer for language-specific content, this example serves only as an alternative strategy for handling localization.

```
public override IContentCollection GetContentCollection()
{
    var contentCollection = new ContentCollection();
    string language = "en"; // Get from configuration

    // Default content
    contentCollection.Include<WildcardRule>("Common/*");

    // Language-specific content
    contentCollection.SetContentRoot("Localized");
    contentCollection.Include<WildcardRule>($"Languages/{language}/*");

    return contentCollection;
}
```

Scenario 3: Dynamic Content from External Sources

Process content from multiple sources:

```
public override IContentCollection GetContentCollection()
{
    var contentCollection = new ContentCollection();

    // Base game content
    contentCollection.Include<WildcardRule>("BaseGame/*");

    // DLC content (if available)
    var dlcPath = Path.Combine(Parameters.RootedSourceDirectory, "DLC");
    if (Directory.Exists(dlcPath))
    {
        contentCollection.SetContentRoot("DLC");
        contentCollection.Include<WildcardRule>("DLC/*");
    }

    // Mod content (if available)
    var modPath = Path.Combine(Parameters.RootedSourceDirectory, "Mods");
    if (Directory.Exists(modPath))
    {
        contentCollection.SetContentRoot("Mods");
        contentCollection.Include<WildcardRule>("Mods/*");
    }

    return contentCollection;
}
```

Debugging

At times you might need to debug your Content Builder, and while you can just run it manually in debug mode and attach your IDE to it, sometimes it is also valuable to FORCE the application to pause and let you step through importing and processing, especially when working with Content Pipeline Extensions.

Alternatively, you may simply want to inject additional assets into your project for debugging purposes (such as overlays, etc) which are only consumed in a `Debug` build of your project (using `#if DEBUG` in your runtime too).

> **NOTE**
>
> This is a rather advanced area which for most situations you do not need to go to these lengths.

The following scenarios outline these options as they pertain to the Content Builder solution:

Scenario 1: Debugging the Builder

If you wish to inject a `Debugger` breakpoint to determine the cause of any issues when building content, you can add the following before the `Builder` runs in your `ContentBuilder` project, this will cause DotNet to request to launch the default debugger and automatically attach it to the project when it is run:

```
#if DEBUG
using System.Diagnostics;
#endif

// If you need to debug the content build process you can enable
// this, build the game, then attach the debugger when prompted.
#if DEBUG
Debugger.Launch();
#endif

// Launch point
builder.Run(args);
```

Scenario 2: Conditional Debug Content

Useful when you need to include extra content in debug only builds for testing, e.g. texture overlays or additional fonts:

```
public override IContentCollection GetContentCollection()
{
    var contentCollection = new ContentCollection();

    // Always include game content
    contentCollection.Include<WildcardRule>("GameAssets/*");

    #if DEBUG
    // Include debug visualizations only in debug builds
    contentCollection.SetContentRoot("Debug");
    contentCollection.Include<WildcardRule>("DebugAssets/*");
    #endif

    return contentCollection;
}
```

Summary

The MonoGame ContentBuilder provides a powerful, flexible way to build your game content programmatically. Here are the key takeaways:

1. **ContentBuilderParams** configures where and how content is built.
2. **ContentCollection** defines rules for which files to process.
3. **Include/Exclude** methods with patterns give you precise control.
4. **Importers and Processors** can be customized for your needs.
5. **Rule order matters** - the last matching rule wins!

With these tools, you can create sophisticated content build pipelines that adapt to different platforms, languages, build configurations, and project structures.

Happy content building! 🎮✨

Additional Resources

- MonoGame Documentation
- Content Pipeline Documentation
- Custom Content Processors

Automating Content Building with GitHub Actions

Learn how to automate MonoGame content building using the Build-Content GitHub Action in your CI/CD workflows.

> **NOTE**
>
> These instructions **REQUIRE** you to use the latest `MonoGame Develop` release (currently `3.8.5-develop.13`)
>
> See the instructions here for how to install the preview project templates and update your project to use `3.8.5-develop` (preview) releases.
>
> How to install MonoGame Preview packages

Overview

The **Build-Content** composite GitHub Action provides a powerful way to automate the building of MonoGame content as part of your continuous integration and deployment (CI/CD) workflows. This action streamlines the content pipeline process, enabling you to build game assets for multiple platforms automatically whenever you push changes to your repository. Alternatively, it can be useful for offloading the building of content to "agents" without having to alter your environment (such as having a Windows automated host to build shaders without Wine for Linux/Mac).

Why Use GitHub Actions for Content Building?

Automating your content building process offers several key advantages:

- **Consistency**: Ensures content is built the same way every time, eliminating "works on my machine" issues.
- **Multi-platform Support**: Build content for iOS, Android, DesktopGL, Windows, and other platforms in parallel.
- **Version Control Integration**: Automatically rebuild content when assets change.
- **Artifact Management**: Store and distribute pre-built content as GitHub artifacts.
- **Team Collaboration**: Enable team members to trigger builds without needing local MonoGame setup.
- **Repository Separation**: You can optionally maintain content builders, assets and even the Game project in separate repositories for better organization.
- **Time Savings**: Reduce manual build steps and eliminate repetitive tasks.

Automation Architecture

There are several paths open when using automation to manage content building, each with their own benefits:

- A standard game project, with the builder and assets all in the same repository - easy to maintain and self-contained. The project simply needs to be built and the content will be generated automatically.
- A Game project, with the builder and assets all in the same repository, however the content is built first before the game project. Preventing accidental failures and identifying asset issues in advance. Although requires the build/runtime project to have content injected manually.
- Separate repositories for the Game and Builder/Assets. Allows content to be worked on independently and repeatedly built without impact to the main game. The Asset output and game output are then combined with an automated process and shipped.
- Separate repositories for all elements, Game, Builder, and assets. Allows for greater separation of concerns, in some instances, the builder generation pushes the compiled EXE to a package for consumption by the asset build so that the builder can be worked on independently while assets continue to build with the previous generation.

There is no one single path, only that which fits the development cycle of the project and the timelines needed to support different pathways.

MonoGame Custom Actions

MonoGame hosts several GitHub composite actions for use in automated workflows, including:

- Automated Wine installation - "install-wine"
- Font installation automation - "install-font"
- Install Android dependencies - "install-android-dependencies"
- Itchio publishing helper - "publish-itchio"
- Content Builder integration for running Content Builds - "build-content"

For the new Content Builder solution, the "**Build-Content**" action provides a comprehensive set of functionality to aid in MonoGame content automation, specifically:

- Build MonoGame content for any platform (iOS, Android, DesktopGL, Windows, MacOSX, Linux, etc.)
- Support for local or external content builder repositories. (Have the Content Builder bundled with the Game project or in a separate repository)
- Support for local or external asset repositories. (Have the source Game Assets bundled with the Game project or in a separate repository)
- Comprehensive logging with automatic artifact uploads.
- Optional content output uploads to the Source host for distribution.
- Subfolder targeting for complex repository structures.

Basic Usage Example

Here's a simple example of using the Build-Content action in a workflow where all components (game, content builder, and assets) are in the same repository:

```yaml
name: Build Game with Content

on:
  workflow_dispatch:

jobs:
  build:
    runs-on: windows-latest
    steps:
      - uses: actions/checkout@v5

      - name: Setup .NET
        uses: actions/setup-dotnet@v5
        with:
          dotnet-version: '9.0.x'

      - name: Process content
        uses: MonoGame/monogame-actions/build-content@v1
        with:
          content-builder-path: './Content/Builder'
          assets-path: './Content/Assets'
          monogame-platform: 'DesktopGL'
          output-folder: './MyGame/bin/Release/Content'
          configuration: Release

      - name: Build game
        run: dotnet build -c Release MyGame/MyGame.csproj -r win-x64
```

For detailed examples including external repositories, multi-platform builds, and advanced configurations, see the Build-Content Action Documentation.

Example setup demonstrations

MonoGame has provided several public demonstration repositories to showcase how the "Build-Content" action can be used to automate content production:

Complete game sample in a single repository:

- MonoGame-CBPlatform-Test

Contains a multi-platformer project (2D Startkit) updated and outfitted with two automated paths:

- .github/workflows/build.yml Demonstrates a matrix solution that builds a project for DesktopGL, Android and iOS. Once built, it is available as an Artifact from the build output.
- .github/workflows/build-remote Demonstrates a single project build solution that uses an external "Builder" from a separate repository and "Assets" from another repository.

> **NOTE**
>
> At this time, the iOS project only builds and does not "Publish", the reason for this is that you need signing details or an associated "Team" in an iOS development environment to do so.
>
> See the Packaging Games guide for more details on how to setup iOS publishing.

Dedicated builder repository - no assets:

- MonoGame-CBPlatform-BuilderTest

A dedicated project for the sole purpose of maintaining the Content Builder project separate from the main Game and Assets. Useful in scenarios where you want to generate a dedicated builder deliverable for use by artists. Also includes a workflow to automatically build Assets from a remote repository that can be downloaded from the last run.

Dedicated Assets only repository:

- MonoGame-CBPlatform-TestAssets

A dedicated repository containing the source assets for a game project, not affected by changes in the main game project. This can be used remotely by the projects automation, or consumed by a Builder project. Also contains a workflow that utilizes the above Builder project to demonstrate that the "Build-Content" automation can be run from either the builder or from the assets themselves.

Note, that while the assets are in the root of this repository, they do not have to be, and you can even maintain variations of the assets in different folders for different targets and use the automation to select the correct source folder at build time.

Complete MonoGame Build Automation

The Build-Content action can be integrated into a complete automated build pipeline for your MonoGame projects. Here's how to set up end-to-end automation for building your game across multiple platforms.

Multi-Platform Build Strategy

A complete workflow should handle:

1. **Environment Setup**: Configure .NET SDK and platform-specific workloads
2. **Content Building**: Process game assets for each target platform
3. **Project Building**: Compile the game for each platform
4. **Artifact Creation**: Package and upload build outputs
5. **Publishing**: Create distribution-ready packages

Key Components of a Complete Workflow

1. Platform Matrix Configuration

Use GitHub Actions matrix strategy to build for multiple platforms in parallel:

```
strategy:
  matrix:
    include:
      - platform: iOS
        os: macos-latest
        runtime: ios-arm64
        workload: ios
        tfm: net9.0-ios
      - platform: Android
        os: windows-latest
        runtime: android-arm64
        workload: android
        tfm: net9.0-android
      - platform: DesktopGL
        os: windows-latest
        runtime: win-x64
        workload: ''
        tfm: net9.0
```

2. Dynamic Path Resolution

Different platforms require different output paths. Use dynamic path calculation:

```
  - name: Set paths
    id: paths
    shell: bash
    run: |
      PROJECT_DIR="${{ matrix.project }}"
      PROJECT_DIR="${PROJECT_DIR%/*}"

      if [[ "${{ matrix.platform }}" == "iOS" ]]; then
        RID_PATH="bin/${{ env.Configuration }}/${{ matrix.tfm }}/iossimulator-x64"
      else
        RID_PATH="bin/${{ env.Configuration }}/${{ matrix.tfm }}/${{
  matrix.runtime }}"
      fi

      echo "content_output=./MyGame/$PROJECT_DIR/$RID_PATH/Content" >> $GITHUB_OUTPUT
```

3. Platform-Specific Workload Installation

Some platforms require additional .NET workloads:

```
  - name: Install workload
    if: ${{ matrix.workload != '' }}
    run: dotnet workload install ${{ matrix.workload }}
```

4. Content Processing Integration

Integrate the Build-Content action to process assets:

```
  - name: Process content
    uses: MonoGame/monogame-actions/build-content@v1
    with:
      content-builder-path: "./Content/Builder"
      assets-path: "./Content/Assets"
      monogame-platform: ${{ matrix.platform }}
      output-folder: ${{ steps.paths.outputs.content_output }}
      configuration: ${{ env.Configuration }}
```

5. Build and Publish

Execute the build and publish steps:

```
- name: Restore
  run: dotnet restore ${{ matrix.project }} -r ${{ matrix.runtime }}

- name: Build
  run: dotnet build -c ${{ env.Configuration }} ${{ matrix.project }} -r ${{
matrix.runtime }}

- name: Publish
  if: ${{ matrix.platform != 'iOS' }}
  run: dotnet publish ${{ matrix.project }} -c ${{ env.Configuration }} -r ${{
matrix.runtime }} --self-contained
```

Important Considerations

.NET SDK Versions

> **IMPORTANT**
>
> Ensure consistency between your local development environment and GitHub Actions:
>
> - Use a `global.json` file to pin the .NET SDK version
> - Specify the exact version in `actions/setup-dotnet@v5`
> - MonoGame 3.8.5+ requires .NET 9.0 or later

Example `global.json`:

```
{
  "sdk": {
    "version": "9.0.307"
  }
}
```

Platform-Specific Requirements

iOS Builds

- **Runner**: Must use `macos-26` or specific macOS version
- **Workload**: Requires `dotnet workload install ios`
- **Runtime**: Use `ios-arm64` for devices, `iossimulator-x64` for simulator
- **Special Handling**: iOS builds have different output paths and require additional parameters
- **Publishing**: iOS builds typically skip the publish step in CI

Android Builds

- **Runner**: Works on `windows-latest` or `ubuntu-latest`
- **Workload**: Requires `dotnet workload install android`
- **Runtime**: Common runtimes include `android-arm64`, `android-x64`
- **Signing**: Consider adding keystore signing for release builds

Desktop Platforms

- **Windows**: Use `windows-latest` runner, `win-x64` or `win-x86` runtime
- **Linux**: Use `ubuntu-latest` runner, `linux-x64` runtime
- **DesktopGL**: Cross-platform, can build on any runner

CSProj updates

If you are using automation alongside normal project building, then the project templates need to be updated to work both locally (run through Visual Studio/VSCode) and through automation with a switch. This is critical in terms of bundled platforms such as iOS and Android, which need their built content available in the right place even before the build begins.

> **NOTE**
>
> See the Build workflow in the main sample repository which demonstrates the complete flow from Content Building, copying, project building and packaging.

An example update for android is as follows:

```
  <ItemGroup>
    <!-- For CI/CD with RuntimeIdentifier -->
    <AndroidAsset
Include="$(ProjectDir)bin\$(Configuration)\$(TargetFramework)\$(RuntimeIdentifier)\C
ontent\**\*" Condition="'$(RuntimeIdentifier)' != '' AND
Exists('$(ProjectDir)bin\$(Configuration)\$(TargetFramework)\$(RuntimeIdentifier)\Co
ntent')">
      <Link>Content\%(RecursiveDir)%(Filename)%(Extension)</Link>
    </AndroidAsset>
    <!-- For local builds without RuntimeIdentifier -->
    <AndroidAsset Include="$(ProjectDir)$(OutputPath)Content\**\*"
Condition="'$(RuntimeIdentifier)' == '' AND
Exists('$(ProjectDir)$(OutputPath)Content')">
      <Link>Content\%(RecursiveDir)%(Filename)%(Extension)</Link>
    </AndroidAsset>
  </ItemGroup>
```

The extension based on the default path for Android and iOS breaks down two fold:

- One AndroidAsset/BundleResource entry for content in the location the final build will output, populated by the CI/CD pipeline.
- One entry set using the local build output when executed by the normal Game project build.

Additionally, the Local project build is prevented from running with the following amendment:

```
  <Target Name="BuildContent" BeforeTargets="Build" Condition="'$(WorkflowMode)'
!= 'true'">
```

Which states that if the project build is run with a `-p:WorkflowMode=true` argument, the default Content build is skipped as this has already been pre-processed by the automation.

> **NOTE**
>
> The Full example of this flow can be observed in the MonoGame-CBPlatform-Test project sample, notably the build.yml, Android csproj and iOS csproj files.

Configuration Management

Use workflow inputs or environment variables for flexible configuration:

```yaml
on:
  workflow_dispatch:
    inputs:
      configuration:
        description: 'Build configuration'
        required: false
        default: 'Release'
        type: choice
        options:
          - Debug
          - Release

env:
  Configuration: ${{ github.event.inputs.configuration || 'Release' }}
```

Artifact Management

Upload build outputs for distribution or debugging:

```yaml
- name: Upload build artifact
  uses: actions/upload-artifact@v5
  with:
    name: ${{ matrix.platform }}-build
    path: MyGame/${{ steps.paths.outputs.project_dir }}/bin/${{
env.Configuration }}/

- name: Upload publish artifact
  uses: actions/upload-artifact@v5
  with:
    name: ${{ matrix.platform }}-publish
    path: MyGame/${{ steps.paths.outputs.project_dir }}/bin/${{ env.Configuration
}}/${{ matrix.tfm }}/${{ matrix.runtime }}/publish/
```

Content Build Logs

The Build-Content action automatically uploads comprehensive logs:

- **Name**: `content-build-logs-<run_id>`
- **Contents**: restore.log, build.log, content-pipeline.log
- **Retention**: 30 days
- **Always Available**: Uploaded even on failure for debugging

Release Patterns

Consider different triggers for your workflows:

```
on:
  # Manual trigger
  workflow_dispatch:

  # On push to main branch
  push:
    branches:
      - main

  # On pull request
  pull_request:
    branches:
      - main

  # When content changes
  push:
    paths:
      - 'Content/Assets/**'
      - '**/*.csproj'
```

Common Workflow Patterns

Pattern 1: Separate Content Building

Build content once and reuse across multiple platform builds:

```yaml
jobs:
  build-content:
    runs-on: windows-latest
    strategy:
      matrix:
        platform: [DesktopGL, iOS, Android]
    steps:
      - name: Process content
        uses: MonoGame/monogame-actions/build-content@v1
        with:
          monogame-platform: ${{ matrix.platform }}
          upload-output: 'true'

  build-game:
    needs: build-content
    runs-on: ${{ matrix.os }}
    steps:
      - name: Download content
        uses: actions/download-artifact@v4
        with:
          name: content-output-${{ matrix.platform }}-${{ github.run_id }}
```

Pattern 2: Integrated Build

Build content as part of each platform build:

```yaml
jobs:
  build:
    runs-on: ${{ matrix.os }}
    steps:
      - name: Process content
        uses: MonoGame/monogame-actions/build-content@v1

      - name: Build game
        run: dotnet build
```

Troubleshooting

Path Issues

- **Problem**: Content builder or assets not found
- **Solution**: Ensure paths are relative to repository root and use correct forward slashes
- **Tip**: Use `./` prefix for clarity (e.g., `./Content` not just `Content`)

Platform-Specific Failures

- **Problem**: Build fails on specific platform
- **Solution**: Check workload installation and runtime compatibility
- **Tip**: Review platform-specific build logs in artifacts

SDK Version Mismatches

- **Problem**: Build works locally but fails in Actions
- **Solution**: Add `global.json` to lock SDK version across environments
- **Tip**: Use `dotnet --list-sdks` locally to verify your version

Content Pipeline Errors

- **Problem**: Assets fail to process
- **Solution**: Download `content-build-logs` artifact and review `content-pipeline.log`
- **Tip**: Test content building locally with the same configuration

Additional Resources

Official Documentation

- Build-Content Action Repository
- MonoGame Content Pipeline Documentation
- GitHub Actions Documentation

.NET Publishing and Packaging

Cross-Platform Publishing

- .NET Application Publishing Overview
- Self-Contained Deployment
- Runtime Identifier (RID) Catalog

Windows Packaging

- Create MSIX Packages in GitHub Actions
- Windows Application Packaging
- Code Signing for Windows

Android Packaging

- Publishing .NET Android Apps
- Android App Signing
- Publishing to Google Play
- GitHub Actions for Android Publishing

iOS Packaging

- Publishing .NET iOS Apps
- iOS Code Signing
- TestFlight Distribution
- App Store Submission
- iOS CI/CD with GitHub Actions

macOS Packaging

- Publishing .NET macOS Apps
- macOS App Notarization

Linux Packaging

- .NET Linux Distribution
- Creating AppImage for Linux
- Snap Packages
- Flatpak Applications

GitHub Actions Resources

- GitHub Actions Marketplace
- Matrix Strategy Documentation
- Workflow Syntax Reference
- Artifact Upload/Download Actions

MonoGame Community Resources

- MonoGame Packaging guide
- MonoGame Discord Community
- MonoGame Forums
- MonoGame GitHub Repository

Summary

Automating MonoGame content building with GitHub Actions provides a robust foundation for continuous integration and deployment. By leveraging the Build-Content composite action, you can:

- Streamline content processing across multiple platforms
- Ensure consistent builds in your team environment
- Reduce manual intervention in the build process
- Create reproducible, distributable game packages

Start with a simple single-platform workflow and gradually expand to multi-platform builds as your project grows. The flexibility of the Build-Content action supports everything from straightforward local builds to complex multi-repository architectures.

Using MGCB Editor

Learn how to use the MGCB Editor to organize and build content for use with MonoGame.

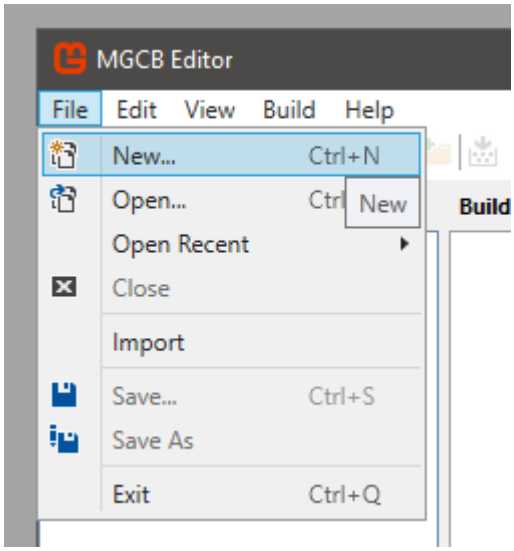
The MGCB Editor is used to organize and build content for use with MonoGame. It is accessed by either:

- Double-clicking a "**.mgcb**" file from within the Visual Studio solution explorer (providing that you installed MonoGame's Visual Studio extension).
- Right-clicking a "**.mgcb**" file from within the Visual Studio code and selecting "Open in MGCB Editor" (provided you have installed the VSCode extension)

Alternatively, you can build it directly from source if needed.

Create A Project

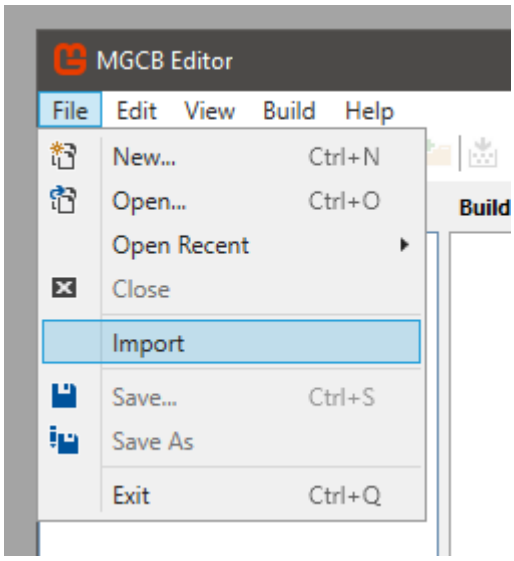
To start a new project just select “New...” from the “File” menu.



This will give you a new empty project to add content to.

Importing an XNA (.contentproj) content project

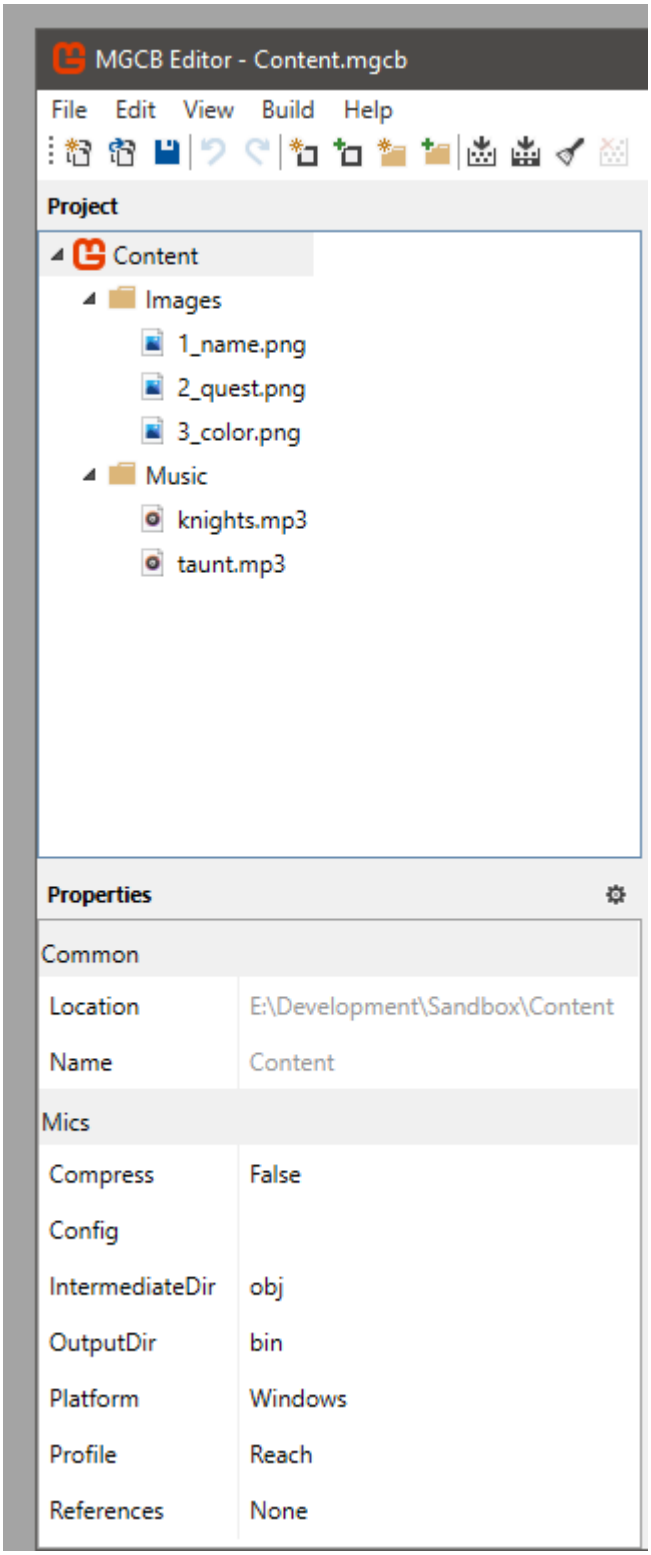
If you are starting from an existing XNA project, the MGCB Editor supports importing your existing **.contentproj**. Again you can access this from the "**File**" menu:



This creates a new project, adding all your content and content settings from the XNA project. If you happened to be using custom processors, you may need to edit the assembly references to link to the correct paths which we discuss next.

Project Settings

You can edit the content project settings directly from the property grid editor after selecting the project node in the tree view:

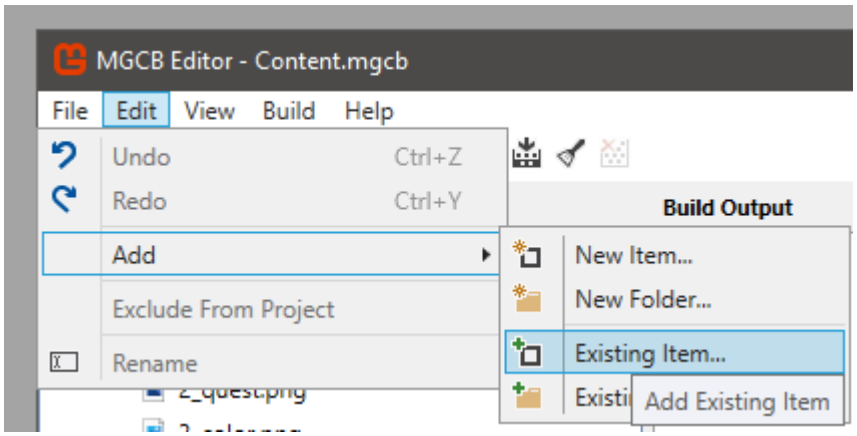


This is where you set up the folders for output, the platform to target, the assembly references for custom processors, etc.

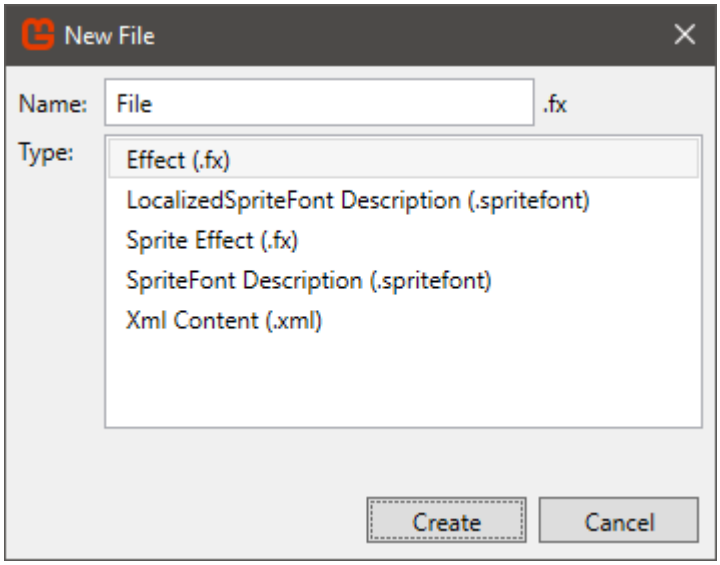
> Note that currently, the MGCB Editor is not set up to support multiple target platforms. This means you may need to manage multiple content projects or manually change the target platform between builds. If you are using `MonoGame.Content.Builder.Task` to build content as part of building your project, the correct platform will automatically be targeted.

Adding Content Items

Once you have a project set up you can add content to it for building. You can do this from the "**Edit**" menu:



Selecting "**New Item...**" will bring up the New Item dialog which displays a list of new items that can be created:



When you select “**Existing Item...**”, you can select an existing item from disk to add to the content project.

Built-in Content Importers and Processors

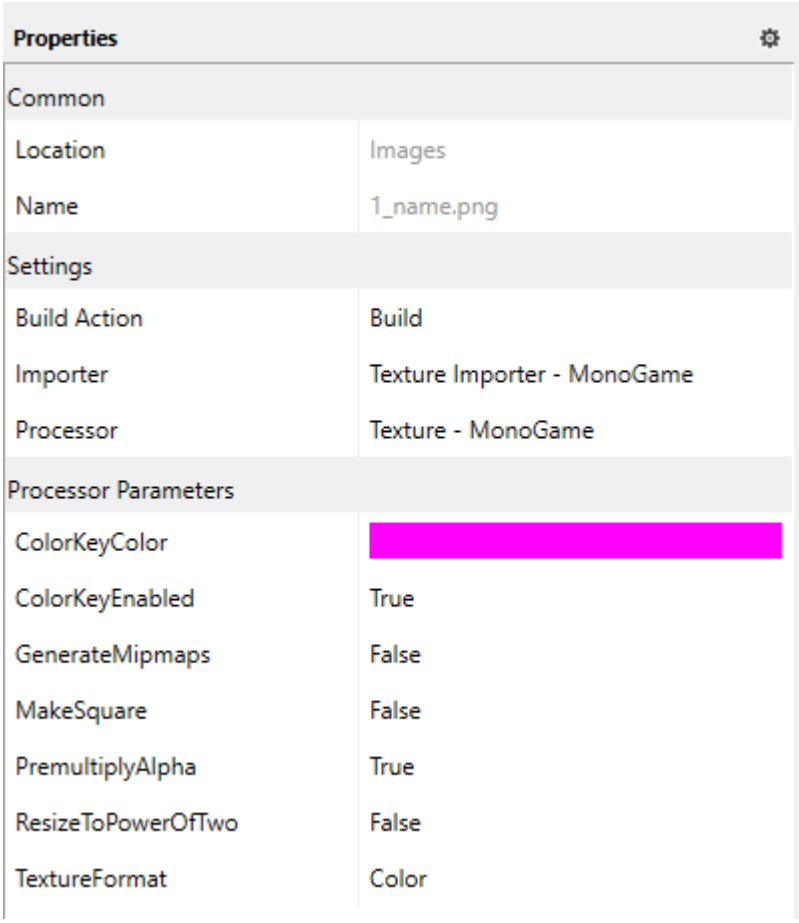
MonoGame provides default **Importers** (File Read) and **Processors** (File Transform) methods for handling almost all standard content which is automatically selected for known content types. In most cases, you do not need to alter these settings when adding content. The lists below highlight the default configuration options for each file type.

> If you wish to create your own Importers and Processors for custom file types, please see the "Custom Content Processors" section.

File Types

The MonoGame Content Builder (MGCB) pipeline supports many different file types within a content project, along with several default processing options for those files:

Images

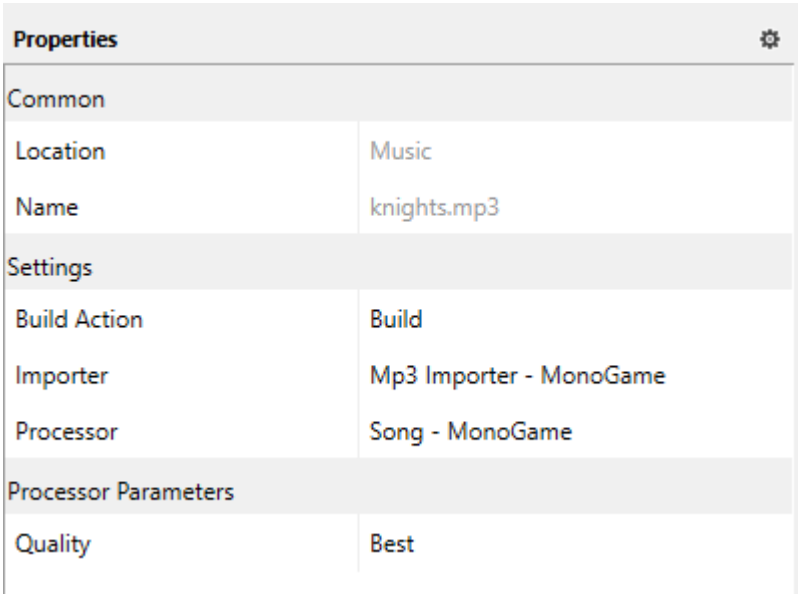


Images come in a variety of formats such as png or jpg, most of which are natively supported by the MonoGame content pipeline.

Additionally, the following options are configurable per image asset when imported:

Property	Description
ColorKeyColor	Background color used for transparency
ColorKeyEnabled	If enabled, any colors identified using the Color Key will be marked as transparent (Alpha 0). The default value is **false**
GenerateMipmaps	If enabled, this will generate MipMaps for the selected texture. The default value is **false**
MakeSquare	if enabled this will make best efforts to expand the image to a square. The default value is **false**
PremultiplyAlpha	If enabled this will treat the selected texture as an Alpha Mask. The default value is **false**
ResizeToPowerOfTwo	If enabled this will resize the image dimensions to the next largest power of 2 size, e.g. 128x128 or 512x512. The default value is **false**
TextureFormat	Selected format of the Texture to process, e.g. raw Color, DXT. The default value is **color**

Audio



MonoGame supports several audio formats to cope with a wide variety of audio needs, from Wav files for effects to MP3's for songs/background music. The type of audio you use will depend on the audio needs for your project.

Audio can be configured to use either the **Sound Effect** processor which imports audio as a Sound Effect which is generally used for short and quick audio sounds. Alternatively, by selecting the **Song** processor, the audio will be imported as a **Song** which is used for long-running audio using the Media Player class.

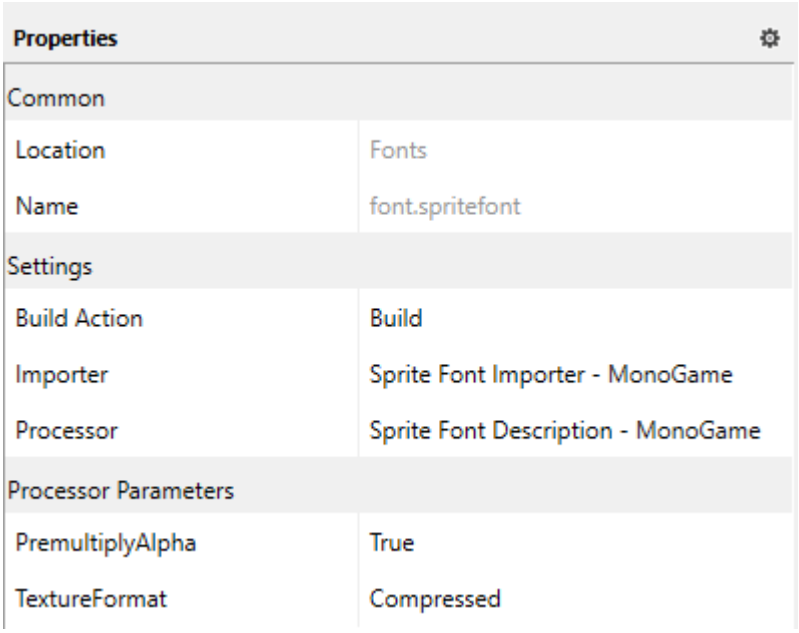
Property	Description
DebugMode	Depending on the setting, this will either leave the audio uncompressed (for speed) when debugging or optimized for a production build. The default value is **false**
Quality	The quality of compression to be used when compiling the asset. The default value is **best**

Models

Properties	
Common	
Location	Models
Name	model.fbx
Settings	
Build Action	Build
Importer	Fbx Importer - MonoGame
Processor	Model - MonoGame
Processor Parameters	
ColorKeyColor	
ColorKeyEnabled	False
DefaultEffect	BasicEffect
GenerateMipmaps	True
GenerateTangentFrames	False
PremultiplyTextureAlpha	True
PremultiplyVertexColors	True
ResizeTexturesToPowerOfTwo	False
RotationX	0.00
RotationY	0.00
RotationZ	0.00
Scale	1.00
SwapWindingOrder	False
TextureFormat	Compressed

Property	Description
ColorKeyColor	Background color used for transparency
ColorKeyEnabled	If enabled, any colors identified using the Color Key will be marked as transparent (Alpha 0). The default value is **false**
DefaultEffect	The default effect class to apply to this model. The default value is **BasicEffect**
GenerateMipmaps	If enabled, this will generate MipMaps for the selected texture. The default value is **false**
GenerateTangentFrames	Generate model tangents for use in Normal Mapping. The default value is **false**
PremultiplyTextureAlpha	If enabled this will treat the selected texture as an Alpha Mask. The default value is **false**
PremultiplyVertexColors	If enabled this will treat the selected texture as a Vertex Color Mask. The default value is **false**
ResizeTexturesToPowerOfTwo	If enabled this will resize the models texture(s) dimensions to the next largest power of 2 size, e.g. 128x128 or 512x512
RotationX	Default model X Axis Rotation–Number, in degrees of rotation. The default value is **0**
RotationY	Default model Y Axis Rotation–Number, in degrees of rotation. The default value is **0**
RotationZ	Default model Z Axis Rotation–Number, in degrees of rotation. The default value is **0**
Scale	Default model scale. The default value is **1.0f**
SwapWindingOrder	Boolean value indicating if the winding order is swapped. This is useful for models that appear to be drawn inside out. The default value is **false**.
TextureFormat	Selected format of the Texture to process, e.g. raw Color, DXT. The default value is **compressed**

Fonts



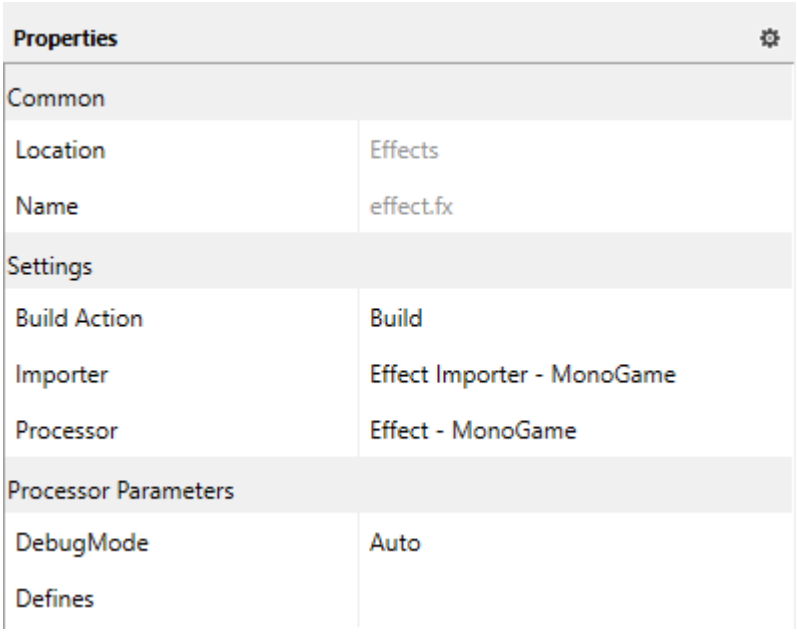
MonoGame uses a **Font descriptor** file to describe the type of font to import, the font in question needs to be installed in the machines Font library for it to enable the project to build, else this will result in a Font importer error.

> It is recommended you keep a copy of the Font file in the content folder for reference to avoid issues with missing Fonts, that way any developer on the project can re-add the Font to their machine when needed.

Property	Description
PremultiplyAlpha	If enabled this will treat the font texture that is generated as an Alpha Mask. The default value is **false**
TextureFormat	Selected format of the font Texture to process, e.g. raw Color, DXT. The default value is **compressed**

See the TrueType fonts section for more detail on working with Fonts. Additionally, check out the Localization section for details on working with managing translated text within your project using resx files.

Effects

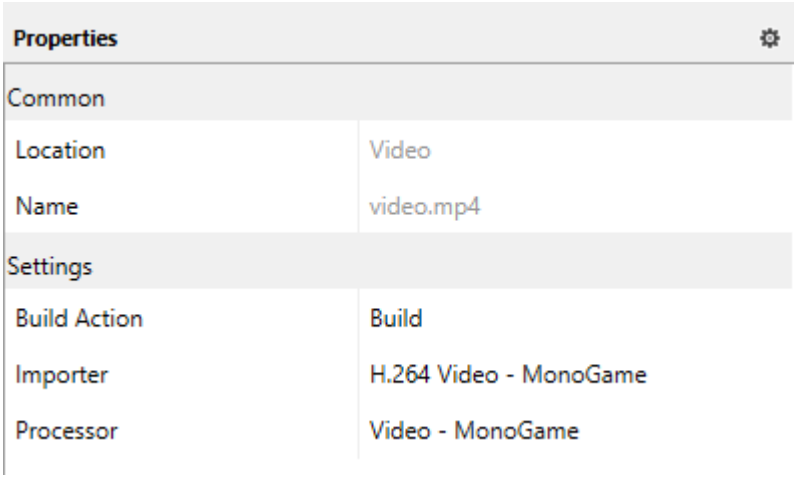


Effects in MonoGame are shaders that have been built with a supported shader language (HLSL / GLSL) which will then be pre-processed into a common shader language / effect used by MonoGame.

Property	Description
DebugMode	Depending on the setting, this will either leave the audio uncompressed (for speed) when debugging, or optimise for a production build. The default value is **false**
Defines	Custom compiler defines to enable specific code paths (#if) in the shader file, if appropriate. The default value is **empty**

See the Effects section for more detail on the built-in effects with MonoGame and how to generate your own custom effects.

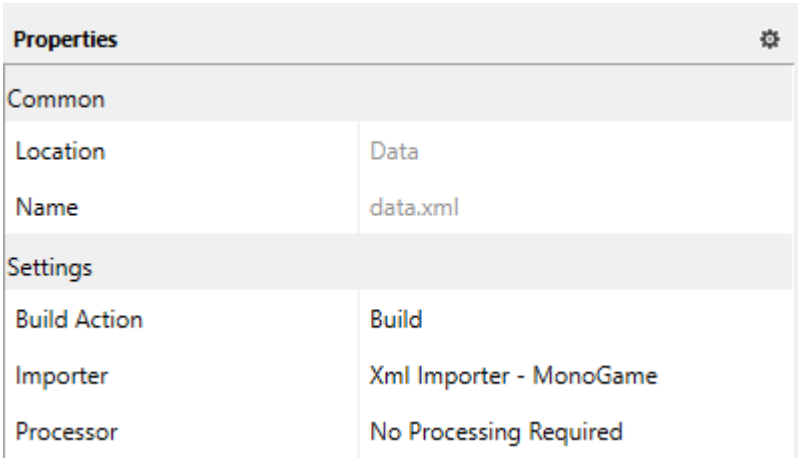
Video



MonoGame uses a custom video player which converts video files into a format that can be used and rendered by MonoGame using a series of VideoTextures.

There are no configurable properties for the Video Player currently.

Text (e.g. XML)



One of the most powerful processors in MonoGame is the text processor, ideally used for XML files used to manage Game Class content in text files rather than writing it all in code.

So long as the XML file conforms to the MonoGame (XNA) standard, it will generate data within the game for use in the solution.

For an example on how to use XML in your MonoGame project, check out this article.

Custom Content Processors

Just like XNA, the MonoGame content pipeline supports custom content processors. To use them you need to rebuild them correctly to work against MonoGame.

The first step is removing all `Microsoft.Xna.Framework.XXX` references and replacing them with references to `MonoGame.Framework` and `MonoGame.Framework.Content.Pipeline`. This is required as you will no longer be building against Microsoft XNA.

Once your references are working, you then need to change your assembly target platform. MonoGame does not support x86 (aka 32bit) assemblies in the content pipeline. This is mainly to allow of processing really big content as well as to simplify the number of configurations and native code dependencies. For this reason, you should try to target “Any CPU” with your custom content assembly.

After you have done these fixes, you should be able to add these new processors to the content project “References”.

Building Content

The MGCB Editor has 3 actions related to building content: Build, Rebuild and Clean:

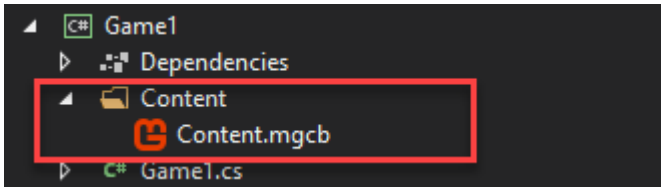
- **Build** - This will build all content that needs to be built and put it in the output directory (bin by default). Content will be skipped if it has not changed since the last build. The time source content was last edited is saved in the intermediate directory (obj by default) to determine if content changed since the last build.
- **Rebuild** - Cleans the project first and then Builds it again.
- **Clean** - Cleaning the project will empty the output and intermediate directories.

Linking Content To Your Game

Once you have built your content, you have a few different ways to add it to your game project. They all have the same goal, to get the built XNB's in your project output folder so a ContentManager can easily find and load them.

MonoGameContentReference

The simplest method is to set up your game project from one of the templates that come with the SDK. When you create a new project, it will include a Content.mgcb file with its Build Action set to **MonoGameContentReference** and a reference to `MonoGame.Content.Builder.Task` which processes *MonoGameContentReference* items.



> This build action is defined in the .targets file here.

The **MonoGameContentReference** is set up so that when the project is built, the MGCB will automatically build any new/modified content and copy the resulting XNB's to the project output directory, so they can be used in the project. This way you do not even have to manually build the content with the MGCB Editor.

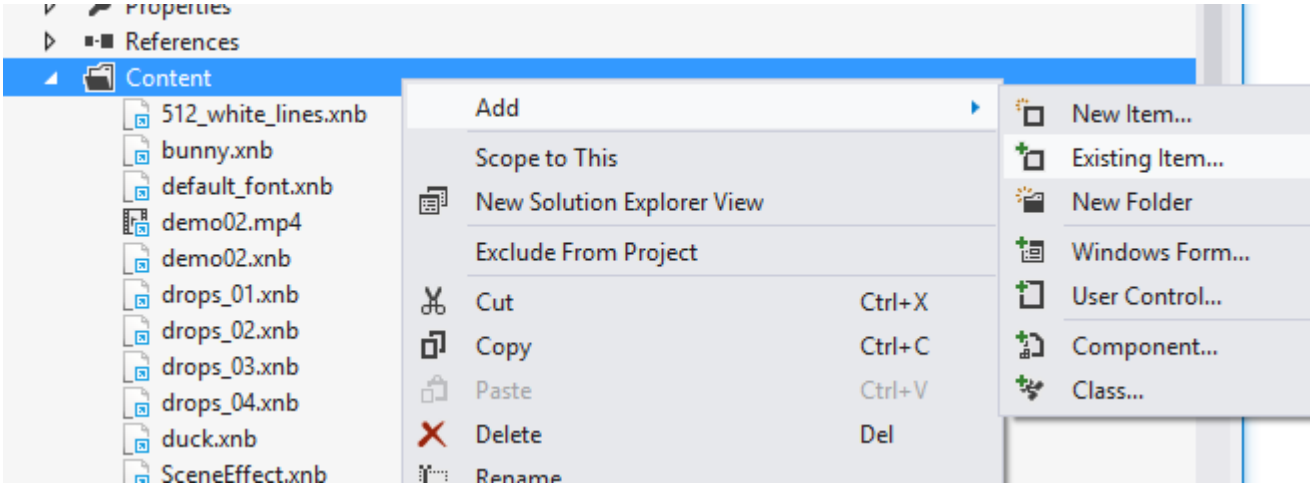
Just add your content to the .mgcb with the MGCB Editor and the rest will happen when you build your project. The content files do not need to be added to your project.

Manual Copy

If you do not want to use the automated process, you can build the content project with the MGCB Editor and copy the XNB's to the output folder of your project manually.

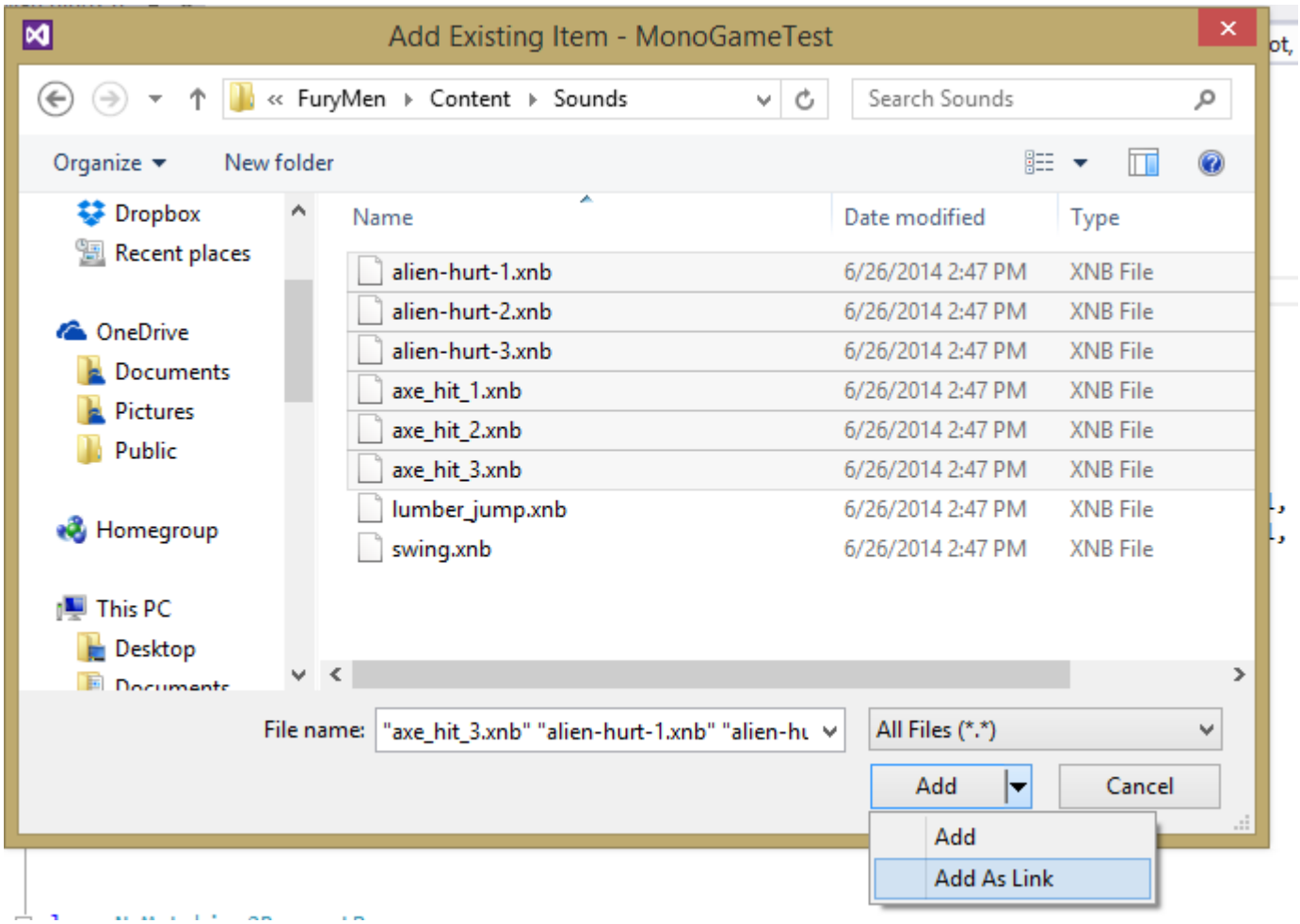
Add As Content

If you are using Visual Studio, you can bypass the content pipeline all together and simply the files to your C# game project directly (or as links). Create a folder in the project called Content then right-click on the folder and select Add > Existing Item.



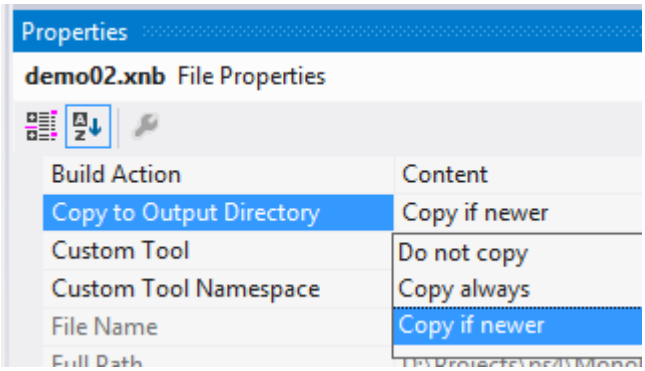
You will now see a file dialog from which you can add your content files.

Note that if you do not want Visual Studio to make a local copy of the files. Make sure to use "**Add As Link**" when "Linking" files.



Once the files are added you will need to select them all and change their properties in the properties window to:

- "Build Action" to **Content**
- "Copy to Output Directory" to **Copy if newer**



Add With Wildcard

The more automatic option is to edit your game .csproj manually and have it include you content automatically using wildcards.

To do this just open the .csproj with any text editor, then add the following after any other `**<ItemGroup>**`:

```
<ItemGroup>
  <Content Include="Content\**\*.xnb">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </Content>
</ItemGroup>
```

Then any files you put in a Content folder within your game project will automatically be included in the build.

Reporting Bugs

If you run into any problems with MGCB or the MGCB Editor, please ask for help on the community site or submit a bug report on GitHub.

Custom Effects

Learn how to create and use custom effects for rendering in MonoGame.

A core element of Microsoft XNA is the effect system which is used for all rendering.

For MonoGame we have the burden of supporting stock and custom effects for desktop GLSL, mobile GLSL, DirectX HLSL, and custom formats like that of the PlayStation Mobile. There currently is no effect system or shader language that supports all the platforms we require, forcing us to build a new custom effect system.

MGFX

MGFX is MonoGame's own "FX" runtime and tools which with the following core goals:

- Support a similar technique, passes, shaders structure as Microsoft FX files.
- Have a textual format for ease of editing.
- Have a compiled and optimized binary format for runtime use.
- Be cross-platform and support multiple shader languages and bytecodes.
- Easy to extend for future platforms and features.

Stock Effects

MonoGame has the following effects built-in and fully supported on current platforms:

- SpriteEffect
- BasicEffect
- AlphaTestEffect
- DualTextureEffect
- EnvironmentMapEffect
- SkinnedEffect

Under the hood these effects use the same system and tools as one would for a custom Effect. The source and pre-compiled versions of these effects can be found in the 'MonoGame.Framework\Platform\Graphics\Effect\Resources' folder.

> If your game requires an extra little bit of performance you can easily hand edit the existing effects to remove unnecessary features or optimize for specific hardware and rebuild them with the MGFX tool.

Custom Effects

To use a custom effect with MonoGame you must do one of the following:

- Run the effect file through the MonoGame Effect content processor for loading via the Content Manager (Recommended).
- Process your effect file with the MGFXC tool and load them yourself at runtime.

Effect Writing Tips

These are some tips for writing or converting effects for use with MonoGame.

The supported shader models when targeting DX are the following:
`vs_4_0_level_9_1` and `ps_4_0_level_9_1`
`vs_4_0_level_9_3` and `ps_4_0_level_9_3`
`vs_4_0` and `ps_4_0` (requires `HiDef` `GraphicsProfile` at runtime)
`vs_4_1` and `ps_4_1` (requires `HiDef` `GraphicsProfile` at runtime)
`vs_5_0` and `ps_5_0` (requires `HiDef` `GraphicsProfile` at runtime)

When targeting GL platforms we automatically translate FX files to GLSL using a library called MojoShader. The supported feature levels are the following:
`vs_2_0` and `ps_2_0`
`vs_3_0` and `ps_3_0`

You can use preprocessor checks to add conditional code or compilation depending on defined symbols. MonoGame defines the following symbols when compiling effects:
`2MGFX`
`HLSL` and `SM4` for DirectX
`OpenGL` and `GLSL` for OpenGL

As an example, you can conditionally set shader models depending on the platform with the following code:

```
#if OPENGL
    #define VS_SHADERMODEL vs_3_0
    #define PS_SHADERMODEL ps_3_0
#else
    #define VS_SHADERMODEL vs_4_0_level_9_1
    #define PS_SHADERMODEL ps_4_0_level_9_1
#endif

technique
{
    pass
    {
        VertexShader = compile VS_SHADERMODEL MainVS();
        PixelShader = compile PS_SHADERMODEL MainPS();
    }
};
```

Custom symbols can be defined from the MGCB Editor or via MGFXC.

- Make sure the pixel shaders inputs **exactly match** the vertex shader outputs so the parameters are passed in the correct registers. The parameters need to have the same size and order. Omitting parameters might not break compilation, but can cause unexpected results.
- Note that on GL platforms default values on Effect parameters do not work. Either set the parameter from code or use a real constant like a #define.
- The effect compiler is aggressive about removing unused parameters, be sure the parameters you are setting are actually used.
- Preshaders are not supported.
- If you think you have found a bug porting a shader, please let us know.

Adding TTF Fonts

Learn how to use TrueType fonts in MonoGame.

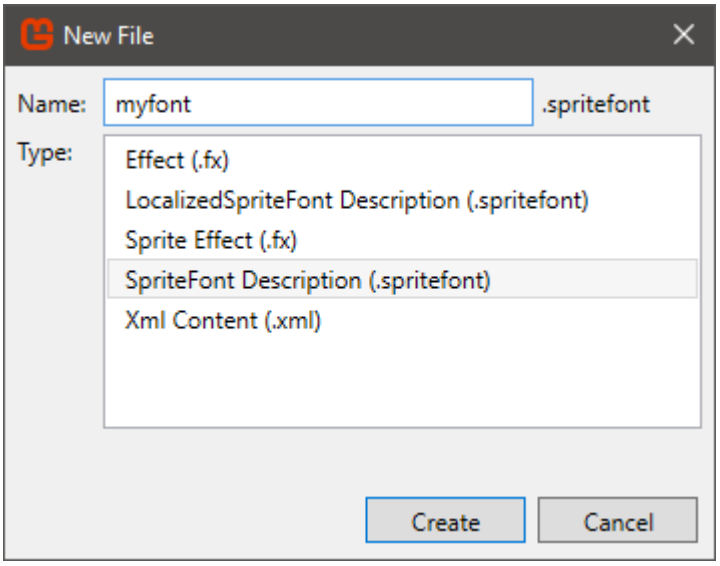
MonoGame supports more than one method of using fonts, the following is an explanation of how to use TrueType fonts.

Using TrueType Fonts with MonoGame

To be able to use a TrueType font, MonoGame requires the **TrueType font file** and a **.spritefont** file.

> TrueType fonts may be installed on the system, or added manually in to the same directory as the .spritefont file.

1. Create the .spritefont file by selecting "Edit -> Add -> New Item" from the MGCB Editor menu, then select **SpriteFont Description** from the list and click **Create**.



2. Open the newly created .spritefont file in your text editor of choice, find this line and change it to your selected .ttf font. If the font is installed on the system, just type the name of the font.

```
<FontName>Arial</FontName>
```

Usage Example

Make a class variable of type SpriteFont

```
SpriteFont font;
```

Load the font with ContentManager

```
font = myGame.Content.Load<SpriteFont>("Fonts/myFont")
```

Draw text with SpriteBatch

```
spriteBatch.Begin();
// Finds the center of the string in coordinates inside the text rectangle
Vector2 textMiddlePoint = font.MeasureString(text) / 2;
// Places text in center of the screen
Vector2 position = new Vector2(myGame.Window.ClientBounds.Width / 2,
myGame.Window.ClientBounds.Height / 2);
spriteBatch.DrawString(font, "MonoGame Font Test", position, Color.White, 0,
textMiddlePoint, 1.0f, SpriteEffects.None, 0.5f)
spriteBatch.End();
```

Localization

Learn how to add localization to your game to support multiple regions.

Localization is an important part of any game. While it can be possible to design a game that is region independent, it is quite hard. At some point you will need to produce localized text and graphics.

MonoGame has a simple localization system built in. If you want to develop your own system you are still able to do so. But the default system should be good enough for most use cases.

Creating resx files

MonoGame runs on .net/Mono on most platforms. Localization is handled by those platforms via the use of resx files. There are walkthroughs on MSDN which walk you through the process. A simplified version is presented here.

Create a .resx file in the IDE e.g Foo.resx and add it to your game project. Note this needs to be added to the main app projects. The Foo.resx file should have an Action of EmbeddedResouce and a Generator value of ResXFileCodeGenerator.

Here is a snippet from the .csproj:

```
<EmbeddedResource Include="Foo.resx">
  <Generator>ResXFileCodeGenerator</Generator>
  <LastGenOutput>Foo.Designer.cs</LastGenOutput>
</EmbeddedResource>
```

Add any string resources to that file. These are in the form of a Key/Value pair. You can use the built-in editor or manually edit the .resx file by hand. It is an xml file so you can view the contents easily.

```
<data name="Wall_Style" xml:space="preserve">
  <value>Wall Style : {0}</value>
</data>
```

When the resx is processed by the generator, it produces a **Foo.Designer.cs** file which is then included in your project. You can then access the "string" value by using code as follows:

```
var s = MyProject.Foo.Wall_Style;
```

> Note in the example we have a place holder ({0}) for additional text. You can still use te property of Foo.Wall_Style with things like string.Format.

```
int i = 1;
var s = string.Format (MyProject.Foo.Wall_Style, i);
```

All this means you do not need to hardcode the string directly. When accessing the **MyProject.Foo.Wall_Style** property, the code will look up the value from the embedded resx file automatically.

You can add support for a new language by adding a new resx file which uses the language/region code as a suffix, e.g Foo.**de-DE**.resx. This new file will contain the translations for that language/region. In the above example we are targetting German.

Universal Windows Platform (UWP) considerations

Unfortunately UWP does not support resx files any more. They have a new file called a **resw**. The format is similar but incompatible to resx. As a result, you will need to duplicate the data into a set of additional resw files to get the translation to work on UWP. The process is like the standard resx process.

Upgrading your SpriteFont files

By default, the SpriteFont processor uses a limited set of characters to generate the font. While this is fine for English languages it will probably not include special characters needed for other languages (French, Arabic, Korean etc).

As a result MonoGame has a **LocalizedFontProcessor** which does something slightly different. The process looks at the resx files you provide it with and generates an optimized **spritefont** which only contains the characters your game uses.

To make use of this functionality you need to tell the spritefont which resx files to use. Open the **.spritefont** with a xml/text editor and add lines like this inside the Asset node:

```
<ResourceFiles>
  <Resx>..\Foo.resx</Resx>
  <Resx>..\Foo.de-DE.resx</Resx>
</ResourceFiles>
```

> Note the paths are relative to the .spritefont directory. In the example above the resx files are in the directory above the .spritefont.

You should end up with a .spritefont file like this

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:LocalizedFontDescription">
    <FontName>Verdana</FontName>
    <Size>14</Size>
    <Spacing>1</Spacing>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#32;</End>
      </CharacterRegion>
    </CharacterRegions>
    <ResourceFiles>
      <Resx>..\Foo.resx</Resx>
      <Resx>..\Foo.de-DE.resx</Resx>
    </ResourceFiles>
  </Asset>
</XnaContent>
```

Once that is done you then need to change the **.mgcb file** so that the **SpriteFontProcessor** is replaced with the **LocalizedFontProcessor**. This can be done by editing the .mgcb file or using the MGCB Editor. After that you can just compile your content as normal. If the processor has any trouble resolving or reading the resx files you will get an error.

Loading the Font

Loading the font can be done in the normal way. The end result of the process is an .xnb file containing a normal SpriteFont.

```
var font = Content.Load<SpriteFont>("Foo");
```

Other Localized assets

Not all localized assets will be fonts. In certain situations you might need to swap out an entire textures or spritesheets. For these cases a new method has been added to the ContentManager, **LoadLocalized**. The idea behind this method is that it will look for localized files **BEFORE** loading the default one.

So for example say you have an asset, MyCharacter. You have a MyCharacter.xnb file which contains the data for that item. You can also has a MyCharacter.de-DE.xnb file which contains the German version of that asset. This asset could be a Texture, Audio or any other game asset. You can then use LoadLocalized to load the localized version of the asset.

```
var myCharacter = Content.LoadLocalized<Texture2D>("MyCharacter");
```

The decision on which localized asset to load is made by looking for a file with the following patterns

```
<AssetName>.<CurrentCulture.Name>
<AssetName>.<CurrentCulture.TwoLetterISOLanguageName>
```

These values are retrieved from

```
CultureInfo.CurrentCulture.Name                         // eg. "en-US"
CultureInfo.CurrentCulture.TwoLetterISOLanguageName     // eg. "en"
```

which are part of the **System.Globalization namespace**.

> On a side note you can also use the `LoadLocalized` to load language specific SpriteFonts. They just need to be named in the same way as we have described above.

Migrating from XNA

A guide on migrating an XNA project to the current version of MonoGame.

MonoGame is API compatible with XNA 4.0 even down to the namespaces. That means you do not have to change much of your game code to port from XNA to MonoGame. There are however some exceptions and some things to keep in mind.

> If your game targets XNA 3.1, you might want to use this archived migration cheatsheet to upgrade to 4.0:
>
> http://www.nelxon.com/blog/xna-3-1-to-xna-4-0-cheatsheet/

Missing/removed API

- The Storage namespace was removed due to portability issues (short discussion here).
- GamerServices was removed. This part of MonoGame was badly maintained due to low usage and difficulties in providing the GamerServices API for different platforms.
- The Net namespace was removed due to its cost of maintenance.

Effects

MonoGame does not use the legacy fxc compiler for effects that XNA used. Instead, MonoGame uses the DX11 compiler. The way MonoGame handles shaders imposes some restrictions and causes some caveats in what is and is not supported.

This is all documented in the custom effects documentation page.

Half pixel offset

XNA uses the DirectX 9 graphics API. MonoGame uses the newer Direct X 11 API for DirectX platforms.

DirectX 9 interprets UV coordinates differently from other graphics API's. This is typically referred to as the half-pixel offset.

MonoGame supports replicating XNA's behavior (currently only on OpenGL platforms) by setting the `PreferHalfPixelOffset` flag in GraphicsDeviceManager to `true`. This flag is set to `false` by default to encourage users to use the modern style of pixel addressing.

DirectX platforms will ignore setting the `PreferHalfPixelOffset` flag and will always render with a half pixel offset compared to XNA. This is usually not noticeable.

This value is passed to `UseHalfPixelOffset` in GraphicsDevice. If `UseHalfPixelOffset` is `true`, you have to add half-pixel offset to a Projection matrix.

SpriteBatch rendering is not affected by the flag.

Regardless of what value the flag has, `SpriteBatch` will render things exactly the same as in XNA.

If you migrated your game from XNA and some things seem blurred out or very slightly offset, you may want to try to enable the `PreferHalfPixelOffset` flag.

Migrating from 3.7

A guide on migrating a MonoGame v3.7 project to the current version of MonoGame.

Previously MonoGame installed on your machine through an installer, but from 3.8 onwards everything is installed through NuGet packages and Visual Studio Extensions.

> **Note** MonoGame 3.8 project templates are not compatible with earlier versions of MonoGame. If you wish to work on or build older MonoGame projects, then you will still need to install MonoGame 3.7.1 or earlier to open them.

WindowsDX and DesktopGL

WindowsDX and DesktopGL templates now use SDK-style projects. To migrate old projects we recommend creating a new project with the 3.8+ templates and copying the csproj to your project folder. Make sure you back up your old project.

> For more information about SDK-style projects see the documentation.

Other Platforms

To migrate open up your project file in a text editor. The reference to the MonoGame assembly looks like this:

```
</ItemGroup>
    <Reference Include="MonoGame.Framework">
        <HintPath>$(MonoGameInstallDirectory)\MonoGame\v3.0\Assemblies\
{Platform}\MonoGame.Framework.dll</HintPath>
    </Reference>
</ItemGroup>
```

The task to build your content is imported at the end of the project file like this:

```
<Import
Project="$(MSBuildExtensionsPath)\MonoGame\v3.0\MonoGame.Content.Builder.targets" />
```

You can remove these references and add a reference to the MonoGame NuGet packages instead.

```
<ItemGroup>
    <PackageReference Include="MonoGame.Framework.{Platform}" Version="3.8.1" />
    <PackageReference Include="MonoGame.Content.Builder.Task" Version="3.8.1" />
</ItemGroup>
```

Tooling

MonoGame tools (MGCB, 2MGFX, and the Pipeline Tool) are now distributed as .NET Tools automatically when using any of the MonoGame 3.8.1 templates.

The templates also build your .mgcb files automatically thanks to the `MonoGame.Content.Builder.Task` NuGet package.

The Pipeline Tool has been renamed MonoGame Content Builder Editor (MGCB Editor) and does not require an installation anymore (providing that you are using the MonoGame 3.8.1 templates). The Visual Studio 2022 extensions make .mgcb files within your solution clickable and will open the MGCB Editor on them.

Upgrading MonoGame from 3.8.x to latest

A guide on upgrading a MonoGame v3.8 project to the current 3.8.4.1+ releases of MonoGame.

Upgrading existing projects from earlier 3.8 releases should be straightforward for most platforms.

> **NOTE**
>
> If you are migrating/upgrading from XNA or MonoGame 3.7 or earlier then check the dedicated guides for those actions before returning here.

- The critical difference from pre 3.8.2 builds, is that the MGCB Editor is no longer a global .NET tool and the MGCB editor is now included as part of the specific project through the use of the dotnet tooling configuration `dotnet-tools.json` file located in the `.config` folder in your solution/project.
- The major difference from 3.8.4 onwards is that we recommend using .NET 9 in your client project, but it is not mandatory, we are also simplifying the `csproj` configuration to reduce management for developers from 3.8.4.1.

  > For iOS/Android however, DotNet 9 at a minimum is Mandatory, see details here.

You can follow the environment setup tutorial to make sure that you are not missing any components.

> **NOTE**
>
> If you are using Visual Studio 2022, we recommend that you use the MonoGame extension which helps with accessing the MGCB editor without the need of CLI commands.

The process of updating your project should be fairly quick and painless without having to change your code or your content project.

Contents

- Updating DotNet Target Framework
- Update MonoGame references
- Add/Update `dotnet-tools.json` Configuration
- Remove `RestoreDotNetTools` section from csproj
- iOS/iPadOS, and Android Considerations

Updating the DotNet Target Framework

Modern MonoGame projects now use the DotNet framework (older projects used to rely on the NetCore/NetFramework libraries). As MonoGame has used DotNet project templates since 3.8.x, you only need to update the `TargetFramework` element of any `csproj` files referencing MonoGame libraries, namely:

- Replace any `<TargetFramework>` entries with the following:

```
<TargetFramework>net9.0</TargetFramework>
```

> **NOTE**
>
> Using `DotNet 9` is only a **recommendation**, you can use any version of DotNet (from 8.0 and above) so long as it supports the MonoGame DotNet 8 dependency. This includes the upcoming DotNet 10 LTS.

Make sure to update any and all projects in your solution, especially you have a multi-project solution similar to those in the MonoGame.Samples

Update MonoGame references

Make sure you update the MonoGame references to the latest version, this can be achieved by either:

- Editing the `csproj` files that reference MonoGame to the latest version.

> **NOTE**
>
> The MonoGame templates set the Version number as `Version="3.8.*"`, this means that it will use the LATEST public version of MonoGame (not including preview releases) that is available.
>
> However, this does mean your `tools` configuration can become out of sync with your project and potentially cause issue.

- Use the Updating NuGet package dependencies documentation as part of the "Preview Release installation instructions", which states you should run the following commands (the example is for `DesktopGL`, use other platforms accordingly):

```
dotnet add package MonoGame.Framework.DesktopGL -v 3.8.4.1
dotnet add package MonoGame.Content.Builder.Task -v 3.8.4.1
```

> **NOTE**
>
> The `MonoGame.Content.Builder.Task` is only needed for client projects and not libraries.

This will ensure your project is using the intended version of MonoGame.

> **IMPORTANT**
>
> Always ensure your `dotnet-tools.json` version matches any updates to the version of MonoGame you are using, as detailed in the next section.

Add/Update `dotnet-tools.json` Configuration

MonoGame DotNet projects currently **Require** DotNet tools configuration to be able to locate and run the `MGCB` editor which is installed locally per project (preventing issues when working with multiple projects using different versions of MonoGame).

IMPORTANT

The MGCB Editor is no longer a .NET global tool, and does not need to be installed or registered separately. When migrating from 3.8.0, it is recommended that you **uninstall** the global versions of the tools. You can accomplish that with these commands:

```
dotnet tool uninstall dotnet-mgcb -g
dotnet tool uninstall dotnet-2mgfx -g
dotnet tool uninstall dotnet-mgcb-editor -g
```

You can either copy the `config` folder with the configuration from a new project template (e.g. `dotnet new mgdesktopgl -o <project name>`), or alternatively you can do the following:

- Create a new folder in the root of your project named `.config`.
- Add a new file called `dotnet-tools.json` and replace its contents with the following:

```
{
  "version": 1,
  "isRoot": true,
  "tools": {
    "dotnet-mgcb": {
      "version": "3.8.4.1",
      "commands": [
        "mgcb"
      ]
    },
    "dotnet-mgcb-editor": {
      "version": "3.8.4.1",
      "commands": [
        "mgcb-editor"
      ]
    },
    "dotnet-mgcb-editor-linux": {
      "version": "3.8.4.1",
      "commands": [
        "mgcb-editor-linux"
      ]
    },
    "dotnet-mgcb-editor-windows": {
      "version": "3.8.4.1",
      "commands": [
        "mgcb-editor-windows"
      ]
    },
    "dotnet-mgcb-editor-mac": {
      "version": "3.8.4.1",
      "commands": [
        "mgcb-editor-mac"
      ]
    }
  }
}
```

> **NOTE**
>
> Please note that you cannot use the `3.8.*` wildcard in the `dotnet-tools.json` file (tool versions have to be fully qualified). We strongly recommend that the versions match the MonoGame version referenced in your `.csproj` (if you are using the `*` wildcard, make sure that they do not end up mismatching if the NuGet packages are updated without you noticing).

The file is the same regardless of which platform / target you are intending to use. If you have a multi-project solution, you only need a **SINGLE** configuration at the root of the project for all client projects.

Remove `RestoreDotNetTools` section from csproj

From `3.8.4.1` and above, the `RestoreDotNetTools` section is no longer required in client project `csproj` files, as the processing is now handled within the MonoGame deliverables.

> **NOTE**
>
> Earlier versions of MonoGame, e.g. 3.8.0 do not have this configuration in the project template, if your `csproj` does not have a `RestoreDotNetTools` element, you can safely ignore this section.

```
<Target Name="RestoreDotNetTools" BeforeTargets="Restore">
  <Message Text="Restoring dotnet tools" Importance="High" />
  <Exec Command="dotnet tool restore" />
</Target>
```

> **TIP**
>
> The XML has changed over versions with various messages and layout, but the section to remove is always titled `Name="RestoreDotNetTools"`

Simply remove this section safely from any and all `csproj` files located in your solution that are dependent on MonoGame.

iOS/iPadOS, and Android Considerations

DotNet 9 is MANDATORY for iOS and Android due to platform requirements, as well as the following configurations:

- MonoGame 3.8.4.1 is **REQUIRED** to comply with the Google Policies on 16kb pages and other affordances.
- The Android `targetSdkVersion` in the `AndroidManifest.xml` **MUST** be a minimum "35" to comply with the latest Google policies. The templates set the minimum to `21` which is safe, only the `TARGET` SDK is critical.
- iOS **MUST** use a minimum `SupportedOSPlatformVersion` of `12.2` in both the `csproj` and in the `info.plist` configuration for the project.

Updating Versions

A guide on updating MonoGame when a new release is pushed.

- Update Existing Projects
  - Update Target Framework
  - Update NuGet References
    - Visual Studio 2022
    - Manually Editing .csproj File (VSCode/Rider)
  - Update MGCB Editor
- Updating Environment for New Projects
  - Update dotnet Templates
  - Visual Studio 2022
  - dotnet CLI (VSCode/Rider)

When a new release of MonoGame is pushed, developers will need to manually update their development environment and/or existing project to make use of the new version. This guide is intended to walk you through the steps of performing these updates.

Update Existing Projects

Developers wishing to update their current project to use a new release version only need to update the *target framework*, *NuGet package versions*, and *dotnet tools* version. Doing this depends on if you are using Visual Studio 2022 or another development environment.

Update Target Framework

Developers will need to ensure the project is updated to target the .NET version used by MonoGame at minimum (currently `net8.0`). To do this, open your project's *.csproj* file and find the `<TargetFramework>` element and change the `net-X.Y` version to `net-8.0`, then save the file.

> **NOTE**
>
> Only change the .NET version number. some project types have platform specifiers such as `net6.0-windows`. The only thing that should change here is the version number.

Update NuGet References

The following sections cover updating your NuGet packages for existing projects based on your development environment.

Visual Studio 2022

Open your existing project in Visual Studio 2022 and perform the following:

1. Right-click on the project in the *Solution Explorer* panel and select *Manage NuGet Packages...".
2. In the NuGet Packages Manager window, click the *Updates* tab.
3. Select the MonoGame Framework packages in the list on the left, then click the *Update* button for each one on the right.

Manually Editing .csproj File (VSCode/Rider)

Alternatively, regardless of the development environment, developers can manually edit the *.csproj* file for their project to update to the newest MonoGame release.

To do this:

1. Open your project's *.csproj* file
2. Locate the `<PacakgeReference>` elements for any MonoGame referenced packages.
3. Change the `Version=` attribute for each one to the current version of MonoGame (currently this is `3.8.2.1105`)
4. Save the *.csproj* file.

Once these changes are made, open a command prompt or terminal at the root of the project directory and enter the following commands

```
dotnet clean
dotnet restore
```

Update MGCB Editor

Regardless of the development environment, users will need to update the dotnet tools manifest file manually to use the newest version of the *MonoGame Content Builder*. To do this:

1. Open the *.config/dotnet-tools.json* manifest file located in the project root directory
2. Update the version specified for each tool to the current version of MonoGame (currently this is `3.8.4`).
3. Save the changes

Once these changes are made, open a command prompt or terminal at the root of the project directory and enter the following commands

```
dotnet tool restore
```

Updating Environment for New Projects

The following sections will cover updating your development environment for new projects.

Update dotnet Templates

Developers will need to update the MonoGame C# Templates used to create new projects. Doing this depends on if you are using Visual Studio 2022 or through the dotnet cli.

> **NOTE**
>
> Updating the templates will not affect existing projects. if developers wish to update existing project, see the info in the Update Existing Projects section.

Visual Studio 2022 templates

Developers using Visual Studio 2022 should be using the MonoGame C# Project Templates extension. This provides not only the templates but also the functionality to open the *MonoGame Content Builder Editor* (MGCB Editor) within Visual Studio. You can update the extension by performing the following

1. Open Visual Studio 2022
2. In the launch window, choose the *Continue without code* option at the bottom on the right.
3. From the top menu choose, *Extensions* > *Manage Extensions* to open the *Extension Manager* panel.
4. Click the *Updates* tab and choose *MonoGame Framework C# project templates* and update it to the latest version.

If prompted to close Visual Studio to finish the update, do so now to continue installing the update.

dotnet CLI (VSCode/Rider) templates

Developers using the dotnet CLI with environments such as Visual Studio Code, JetBrains Rider, or other editors, can execute the following command in a command prompt/terminal to update the templates

```
dotnet new install MonoGame.Templates.CSharp
```

> **TIP**
>
> If you receive a an error or warning stating there are naming conflicts in the templates, you may need to uninstall the templates first with the following command and then install them:
>
> ```
> dotnet new uninstall MonoGame.Templates.CSharp
> ```

Alternatively, you can perform `dotnet new update` which will update all templates installed to their most current version available. However, this may affect other templates you have installed that you may not wish to update, it's an all or nothing command.

Tutorials

Tutorials provided by MonoGame community members.

> **NOTE**
>
> The **MonoGame Foundation** are actively working with the community to update tutorials and documentation, while the existing content available is still valid and working, it is felt it does need modernizing.
>
> Have a tutorial suggestion, article or series that really helped you? Then please log an issue to get it included.

2D

A selection of 2D focused content and tutorials.

2D Getting started tutorial

The official MonoGame 2D getting started guide for MonoGame, covering everything from getting setup to publishing your first game on Itchio.

Click to view 2D tutorial

2D Mobile Deployment - coming soon

An extension tutorial following on from the Beginner 2D above demonstrating how to take a desktop project to Mobile devices.

Coming Soon

2D Advanced Shaders - coming soon

An extension tutorial following on from the Beginner 2D above demonstrating advanced shader techniques for 2D projects.

See what is coming

2D Networking - coming soon

An extension tutorial following on from the Beginner 2D above demonstrating connecting multiple clients together and implementing leaderboards.

See what is coming

3D

A selection of 3D focused content and tutorials.

3D Getting started - coming soon

The official MonoGame 3D getting started guide is currently in production, check back often or look at the Discord announcements.

Coming Soon

MonoGame Foundation

Links to other related resources and information.

Roadmap

Check out the MonoGame roadmap for more incoming educational content.

See what is coming

Blog

The official MonoGame Foundation blog for notes, minutes and announcements.

View the MonoGame blog

Bounties

The Foundation has a number of paid contribution efforts for the Framework, ranging from Tutorials, Framework extensions and samples.

Check out the bounties

Other community recommended tutorials

Check out the awesome work of the larger MonoGame community with their own tutorials, blogs and videos.

Microsoft

- XNA Game Studio educational resources archive
- Archived XNA Game Studio documentation

RB Whitaker's MonoGame Tutorials

- 1 - C# Crash Course
- 2 - MonoGame Getting started tutorials
- 3 - MonoGame 2D tutorials
- 4 - 3D tutorials
- Extra - XNA tutorials

Neil Danson's F# series

- Part 1 - MacOS
- Part 2 - Android
- Part 3 - iOS
- Part 4 - Content Pipeline

Shaders

- MonoGame-Pixel-Planets - a port of Pixel Planets made by DeepFold with a lot of GLSL shaders converted to HLSL
- MonoGame-Shader-Samples

Darkside of MonoGame video series

- Getting Started with MonoGame using Visual Studio 2019
- Getting Started with MonoGame using Visual Studio for Mac
- Getting Started with MonoGame using the command-line
- Walk-through setting up your Mac for MonoGame CLI
- Getting Started with MonoGame using 2D
- Getting Started with MonoGame using XML
- Getting the most out of your assets–The MonoGame Content Pipeline
- MonoGame - Building multi-platform solutions
- Building apps & games for Xbox One using UWP

Video Tutorials

- CodingMadeEasy RPG Tutorial
- Psuedo Games Tutorials
- Batholith Entertainment Game dev tutorials
- Let's Code: The T-Rex Runner Game with MonoGame

Others

- Dark Genesis Blog MonoGame content
- awesome-monogame - A large list of MonoGame libraries and more by aloisdeniel
- A collection of tutorials, libraries and more, many of which are MonoGame related
- How to create animations and sprite sheets for MonoGame
- Making a platformer in F# with MonoGame
- XNA 4.0 Shader Programming / HLSL
- Using Spine with MonoGame - by Randolph Burt (Randeroo)
- Porting a Windows Phone 7 Game to Android
- MonoGame "Hello World" on Mac OS X and Xamarin Studio
- Solving Resolution Independent Rendering And 2D Camera Using Monogame
- XNA is Dead; Long Live the New XNA, MonoGame
- Running MonoGame on Android Wear
- Text rendering in MonoGame
- Randomchaos-MonoGame-Samples

Other Languages

A collection of tutorials in other languages:

Portuguese

- Desenvolvimento de jogos para Windows 8 com XNA - Alexandre Chohfi (Portuguese)

We Need Your Help

MonoGame is an open-source project maintained by its community. Great open source projects require high-quality documentation. This is a call for volunteers to continue to help us make the MonoGame documentation truly great. If you can create tutorials, feature guides, code snippets, reference docs, video walkthroughs, or make any improvement to the current documentation, we could use your help!

Check out the README on GitHub or talk with us on discord to learn how to help!

Building 2D Games with MonoGame

A beginner tutorial for building 2D games with MonoGame.

This tutorial covers game development concepts using the MonoGame framework as our tool. Throughout each chapter, we will explore game development with MonoGame, introducing new concepts to build upon previous ones as we go. We will create a Snake game, which will serve as the vehicle to apply the concepts learned throughout the tutorial. The goal of this tutorial is to give you a solid foundation in 2D game development with MonoGame and provide you with reusable modules that you can leverage to jump start future projects.

What We Will Build

Throughout this tutorial series, we will build a complete game called **Dungeon Slime**; a snake-like game with a dungeon theme. By the end of this series, you will have created a polished game with:

- Smooth animated sprites
- Responsive controls across keyboard and gamepad
- Sound effects and background music
- UI menus with customized styling
- A complete game loop with scoring and game over states

You can see the finished game here: Dungeon Slime on itch.io

Figure 0-1: The title screen of Dungeon Slime

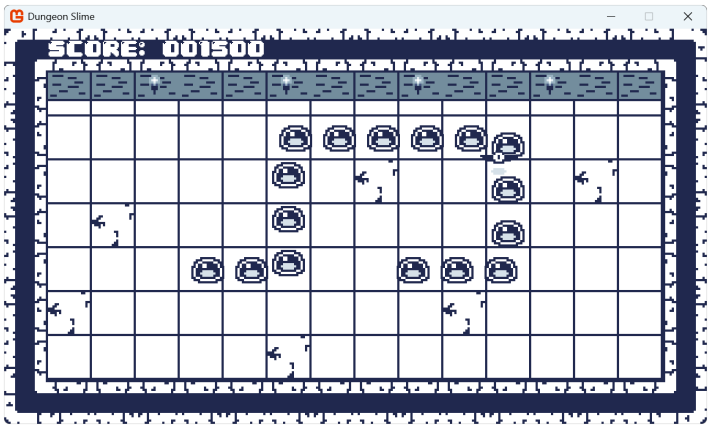


Figure 0-2: Gameplay of Dungeon Slime showing the player controlling a chain of slime segments (in blue) while navigating through the dungeon to collect bats for points.

Who This Is For

This documentation is meant to be an introduction to game development and MonoGame. Readers should have a foundational understanding of C# and be comfortable with concepts such as classes and objects.

> **NOTE**
>
> If you are just getting started with C# for the first time, I would recommend following the official Learn C# tutorials provided by Microsoft. These free tutorials will teach you programming concepts as well as the C# language. Once you feel you have a good foundation, come back and continue here.

How This Documentation Is Organized

This documentation will introduce game development concepts using the MonoGame framework while walking the reader through the development of a Snake clone. The documentation is organized such that each chapter should be read sequentially, with each introducing new concepts and building off of the previous chapters.

Chapter	Summary	Source Files	Video
Chapter 01: What Is MonoGame	Learn about the history of MonoGame and explore the features it provides to developers when creating games.	01-What-Is-MonoGame	Playlist]
Chapter 02: Getting Started	Setup your development environment for .NET development and MonoGame using Visual Studio Code as your IDE.	02-Getting-Started	Link
Chapter 03: The Game1 File	Explore the contents of the Game1 file generated when creating a new MonoGame project.	03-The-Game1-File	Link
Chapter 04: Creating a Class Library	Learn how to create and structure a reusable MonoGame class library to organize game components and share code between projects.	04-Creating-A-Class-Library	Link
Chapter 05: Content Pipeline	Learn the advantages of using the **Content Pipeline** to load assets and go through the processes of loading your first asset	05-Content-Pipeline	Link
Chapter 06: Working with Textures	Learn how to load and render textures using the MonoGame content pipeline and **SpriteBatch**.	06-Working-With-Textures	Link
Chapter 07: Optimizing Texture Rendering	Explore optimization techniques when rendering textures using a texture atlas.	07-Optimize-Texture-Rendering	Link
Chapter 08: The Sprite Class	Explore creating a reusable Sprite class to efficiently manage sprites and their rendering properties, including position, rotation, scale, and more.	08-The-Sprite-Class	Link
Chapter 09: The AnimatedSprite Class	Create an AnimatedSprite class that builds upon our Sprite class to support frame-based animations.	09-The-AnimatedSprite-Class	Link
Chapter 10: Handling Input	Learn how to handle keyboard, mouse, and gamepad input in MonoGame.	10-Handling-Input	Link

Chapter	Summary	Source Files	Video
Chapter 11: Input Management	Learn how to create an input management system to handle keyboard, mouse, and gamepad input, including state tracking between frames and creating a reusable framework for handling player input.	11-Input-Management	Coming Soon
Chapter 12: Collision Detection	Learn how to implement collision detection between game objects and handle collision responses like blocking, triggering events, and bouncing.	12-Collision-Detection	Coming Soon
Chapter 13: Collision Detection	This chapter explores tile-based game environments using tilemaps. You will learn how to efficiently create game worlds by arranging tiles from a tileset.	13-Tilemap	Coming Soon
Chapter 14: Sound Effects and Music	Learn how to load and play sound effects and background music in MonoGame, including managing audio volume, looping, and handling multiple simultaneous sound effects.	14-SoundEffects-And-Music	Coming Soon
Chapter 15: Audio Controller	Learn how to create a reusable audio controller class to manage sound effects and music, including volume control, muting/unmuting, and proper resource cleanup.	15-Audio-Controller	Coming Soon
Chapter 16: Working with SpriteFonts	Learn how to create and use SpriteFonts to render text in your MonoGame project, including loading custom fonts and controlling text appearance.	16-Working-With-SpriteFonts	Coming Soon
Chapter 17: Scenes	Learn how to implement scene management to handle different game screens like menus, gameplay, and transitions between scenes.	17-Scenes	Coming Soon
Chapter 18: Texture Sampling	Learn how to use texture sampling states and add a scrolling background effect to the game.	18-Texture-Wrapping	Coming Soon

Chapter	Summary	Source Files	Video
Chapter 19: User Interface Fundamentals	Learn the core principles of game user interface design.	19-User-Interface-Fundamentals	Coming Soon
Chapter 20: Implementing UI with Gum	Learn how to integrate and use the Gum UI framework to create functional menus, buttons, and sliders for your MonoGame projects.	20-Implementing-UI-With-Gum	Coming Soon
Chapter 21: Customizing Gum UI	Learn how to create custom UI components with animations and visual styling in Gum.	21-Customizing-Gum-UI	Coming Soon
Chapter 22: Snake Game Mechanics	Learn how to implement classic snake-like game mechanics and organize game objects into reusable components.	22-Snake-Game-Mechanics	Coming Soon
Chapter 23: Completing the Game	Finalize game mechanics by updating our current demo into a snake-like inspired game.	23-Completing-The-Game	Coming Soon
Chapter 24: Shaders	Learn how to create custom visual effects using shaders in MonoGame.	24-Completing-The-Game	Coming Soon
Chapter 25: Packaging Your Game for Distribution	Learn how to package your game for distribution across Windows, macOS, and Linux platforms.		Coming Soon
Chapter 26: Publishing Your Game to itch.io	Learn how to deploy your MonoGame project to itch.io and configure it for players across different platforms.		Coming Soon
Chapter 27: Conclusion and Next Steps	Review key MonoGame concepts learned, discover community resources, and get practical advice for beginning your own game development projects.		Coming Soon

Conventions Used in This Documentation

The following conventions are used in this documentation.

Italics

Italics are used for emphasis and technical terms.

Highlights

`Highlights` are used for paths such as file paths, including filenames, extensions and other critical information in the application of steps in a tutorial. These are similar to Inline code blocks as they stand out more in Markdown and require emphasis.

Inline Code Blocks

`Inline code` blocks are used for methods, functions, and variable names when they are discussed within the paragraphs of text. For example, when referring to a method such as `Foo` in a sentence,it will appear with this formatting.

Code Blocks

```
// Example Code Block
public void Foo() { }
```

Code blocks are used to show code examples with syntax highlight. These appear as separated blocks rather than inline with text, allowing for multi-line examples and better readability of complete code snippets.

MonoGame

If you ever have questions about MonoGame or would like to talk with other developers to share ideas or just hang out with us, you can find us in the various MonoGame communities below:

- Discord
- GitHub Discussions Forum
- Community Forums (deprecated)
- Reddit
- Facebook

Note From Author

> I have been using MonoGame for many years (since 2017), it was a time in my game development journey where I was looking for something that I had more control over. I did not want to spend the time to write a full game engine, but I also wanted to have more control than what the current engines at the time (i.e. Unity) offered. At that time, there was a vast amount of resources available for getting started, but none of them felt like they fit a good beginner series. Even now, the resources available still seem this way. They either require the reader to have a great understanding of game development and programming, or they assume the reader has none and instead focuses on teaching programming more than teaching MonoGame. Even still, some relied too heavily on third party libraries, while others were simply very bare bones, asking the reader to just copy and paste code without explaining the *what*/*why* of it all.
>
> Since then, I have written various small self contained tutorials on different topics for MonoGame to try and give back to the community for those getting started. I also participate regularly in the community discussion channels, answering questions and offering technical advice, so I am very familiar with the topics and pain points that users get hung up on when first starting out.
>
> With this documentation, I hope to take the lessons I have learned and provide a tutorial series that I wish was available when I first started, and to present using MonoGame in a straight forward way, introducing concepts and building off of them as we go along in a way that makes sense and is easy to follow.
>
> - Christopher Whitley (Aristurtle)

Acknowledgements

This documentation would not have been possible without the support and contributions from numerous individuals within the MonoGame community.

First and foremost, I want to express my gratitude to the MonoGame Foundation, its maintainers, and the countless others contributors, who have preserved and evolved this framework, ensuring developers like myself have the tools needed to bring our creative visions to life. Their dedication to keeping the spirit of XNA alive has been invaluable to indie game developers worldwide.

I am particularly grateful to the members of the MonoGame Discord community who reviewed early drafts of this content, providing feedback that helped shape these chapters into more a accessible and comprehensive learning resource.

To the many developers of games, such as Celeste, thank you for demonstrating what is possible with MonoGame and inspiring newcomers to explore this framework.

Finally, I would like to thank all the creators and contributors to open-source libraries and tools for MonoGame, including the the creators of Gum, Nez, and MonoGame.Extended and the many other libraries that have helped make game development in MonoGame more accessible.

This documentation stands on the shoulders of a vibrant community that continues to share knowledge, and it is my hope that these tutorials will contribute meaningfully to that ongoing exchange.

Get Started

- Chapter 1 - What is MonoGame

Chapter 01: What is MonoGame

Learn about the history of MonoGame and explore the features it provides to developers when creating games.

A Brief History

In 2006, Microsoft released a game development framework named *XNA Game Studio* to facilitate game development for Windows PC and the Xbox 360 console. It revolutionized game development for indie creators by bringing a simplified approach to building games and offering a set of tools that lowered the entry barrier for aspiring game developers. Out of XNA Game Studio came critically acclaimed titles such as Bastion and Terraria. In 2008, XNA was expanded to support development for both the Zune and Windows Phone.

> **NOTE**
>
> Fun fact, provided by community member stromkos, The release of XNA 3.0 in 2008, which added the support for Windows Phone, is also the release that specified the default window resolution of 800x480 for new projects as this was the preferred resolution on Windows Phone. It is still the default resolution used in MonoGame projects today.

As XNA became more popular, the need for cross-platform development started to grow. In 2009, José Antonio Leal de Farias introduced *XNA Touch*, an open-source project that aimed to make games with XNA playable on iOS devices. This marked the beginning of what would later become MonoGame. Dominique Louis came on board in 2009 and soon took over as full-time project lead, driving its initial development and expansion. The project attracted other developers, such as Tom Spilman, who were interested in expanding the scope of the project as well as its reach.

The official first release of MonoGame occurred in 2011, as an open-source version of XNA. While it still had the same familiar API as XNA, the cross-platform support was expanded to include Windows, macOS, Linux, iOS, Android, Xbox, and PlayStation. Despite Microsoft discontinuing XNA in 2013, MonoGame continued to grow and develop. Maintenance of the project was given to Steve Williams and Tom Spilman in 2014. In order to direct its future development and undertaking, the MonoGame Foundation was formed on September 29th, 2023.

Today, MonoGame is a mature cross-platform framework, that is built with the spirit of preserving XNA while adopting modern game development practices. Some popular titles created using MonoGame include Celeste, Stardew Valley, and Streets of Rage 4.

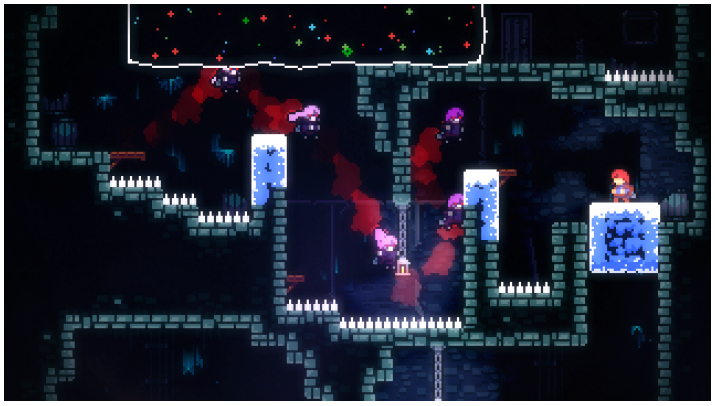


Figure 1-1 Celeste.

Figure 1-2: Stardew Valley

Figure 1-3: Streets of Rage 4

Figure 1-4: Bastion

> **NOTE**
>
> For more details about MonoGame's history, check the About on the official MonoGame website.

Features

MonoGame, following in the footsteps of XNA, is a "bring your own tools" framework. It provides developers the basic blocks to design the game, engines, and/or tools. As a code-first approach to game development, MonoGame does not include any pre-built editors or interfaces; instead, it gives developers the freedom to create their own working environment.

API

At its core, MonoGame offers a set of libraries and APIs to handle common game development tasks. These include:

1. **Graphics Rendering**: 2D and 3D rendering are supported through the graphics API offered by MonoGame. This API provides sprite batching for 2D graphics, a flexible 3D pipeline, and shaders for custom visuals and effects.
2. **Input Handling**: Input from keyboard, mouse, gamepads, and touchscreens is supported, allowing for development of games for any platform and different styles of play.
3. **Audio**: A comprehensive audio system that can be used to create sound effects as well as play music with included support for many audio formats.
4. **Content Pipeline**: An out-of-the-box workflow for importing and processing game assets such as textures, models, and audio, and compiling them to a format that is optimal for the game's target platform.
5. **Math Library**: A math library specifically optimized for game development, providing essential mathematical functions and operations.

Cross Platform

One of the main advantages of MonoGame is its cross-platform support. Games built with MonoGame are compatible with a variety of platforms, including:

- **Desktop**: Windows, macOS, and Linux.
- **Mobile**: iOS and Android.
- **Consoles** (with appropriate license): Xbox, PlayStation, and Nintendo Switch.

By providing cross-platform support, developers can target multiple platforms from a single code base, significantly reducing development time and resources needed for porting.

Programming Language Support

MonoGame is designed and built in C#. It is the official programming language supported in documentation, samples, and community discussion. However, MonoGame is not exclusively tied to C#. As a .NET library, MonoGame can be used with any .NET-compatible language, including Visual Basic and F#.

> **CAUTION**
>
> While the alternative .NET languages can be used, community support may be limited outside the scope of C#.

Regardless of which .NET language is used, developers should have a foundational understanding of the language and programming concepts such as:

- Object-oriented programming.
- Data types and structures.
- Control flow and loops.
- Error handling and debugging.

See Also

- About MonoGame | MonoGame

Test Your Knowledge

1. Name one of the advantages of using the MonoGame framework to develop games.

   Any of the following are advantages of using the MonoGame framework.

   1. It provides cross-platform support, allowing developers to target multiple platforms from a single code base.
   2. It offers a set of libraries and APIs common for game development tasks, such as graphics rendering, input handling, audio, and content management.
   3. It is a "bring your own tools" framework, giving developers flexibility in their working environment.

2. What programming languages can be used when creating a game with MonoGame?

   The primary language used is C#, which is the same language that the MonoGame framework is developed in. However, any .NET language can be used, such as F# or Visual Basic.

Chapter 02: Getting Started

Setup your development environment for .NET development and MonoGame using Visual Studio Code as your IDE.

Unlike game engines (such as Unity, Unreal or Godot), MonoGame is a *framework*. This means it does not come as a standalone program that you download and install, and does not include a graphical user interface used to create games. Instead, MonoGame integrates into the standard .NET development workflow, offering a code-first approach to game development. This approach offers several advantages:

- **Flexibility**: Developers are not locked into using a specific editor or interface, allowing them to use their preferred development tools.
- **Integration**: As a .NET library itself, MonoGame can easily integrate with other .NET libraries and tools.
- **Cross-platform Development**: Since C# is cross-platform, and MonoGame is cross-platform, developers can develop MonoGame projects on Windows, macOS, or Linux, with only slight differences in the setup process for each operating system.
- **Version Control Friendly**: The code-first approach makes it easier to use version control systems like Git for your game projects.

While the environment setup process is similar to the standard setup process for C# development, there are some MonoGame specific steps. These can vary slightly depending on your operating system and the *Integrated Development Environment* (IDE).

Chapter Video

Live Recording

Installing the .NET SDK

The first thing we need to do is install the .NET *Software Development Kit* (SDK). To install it, follow the instructions based on your operating system below:

> **IMPORTANT**
>
> As of MonoGame 3.8.4, the recommended minimum version of the .NET SDK is .NET 9, .NET 10 is also supported.

Windows

1. Open a web browser and navigate to https://dotnet.microsoft.com/en-us/download.
2. Choose the version of the .NET SDK to install and click the **Download .NET SDK x64** button to start the download.

   > **NOTE**
   >
   > The minimum recommended version is a minimum of .NET 9, .NET 10 is also supported.

3. Once the download finishes, run the installer.

macOS

1. Open a web browser and navigate to https://dotnet.microsoft.com/en-us/download.
2. Choose the version of the .NET SDK to install and click the *Download .NET SDK x64 (Intel)* button start the download of the .NET SDK Installer.

   > **NOTE**
   >
   > The minimum recommended version is a minimum of .NET 9, .NET 10 is also supported.

3. Once the download finishes, run the installer.

> **NOTE**
>
> For the time being, MonoGame requires that you install the **Intel** version even if you are using an Apple Silicon (M1/M2) Mac. For Apple Silicon Macs, it also requires that Rosetta be enabled.

Linux

1. Open a new *Terminal* window
2. Enter the following command to install the .NET SDK

```
sudo apt-get update && sudo apt-get install -y dotnet-sdk-8.0
```

> **NOTE**
>
> The minimum recommended version is a minimum of .NET 9, .NET 10 is also supported.

Install Additional Workloads (Optional)

After installing the .NET SDK, if you intend to target mobile devices such as Android or iOS, you will also need to install the corresponding mobile workloads. To do this, open a *Command Prompt* or *Terminal* window and enter the following commands.

```
dotnet workload install ios
dotnet workload install android
```

Install MonoGame Project Templates

MonoGame provides project templates that can be installed to create new projects that are pre-configured to target the current version of MonoGame as a base to begin creating games. To install the MonoGame templates, open a *Command Prompt* or *Terminal* window and enter the following command

```
dotnet new install MonoGame.Templates.CSharp
```

Installing Visual Studio Code

Visual Studio Code (VSCode) is a free, lightweight editor. Depending on the programming language you are using, it is just a matter of installing the appropriate extension to support that particular language. VSCode is also cross-platform, meaning you can use it for development on Windows, macOS, and Linux. To ensure that all readers can follow this tutorial regardless of the operating system used, we will be using VSCode as our IDE.

To install VSCode, follow the instructions for your operating system below:

Windows

1. Open a browser and navigate to https://code.visualstudio.com/.
2. Click the *Download for Windows* button to start the download of the installer.
3. Once the download finishes, run the installer.

macOS

1. Open a web browser and navigate to https://code.visualstudio.com/.
2. Click the *Download for macOS* button to start the download of the *.zip* archive.
3. Once the download finishes, double click the *.zip* archive to extract the *Visual Studio Code.app* application package
4. Drag-and-drop the *Visual Studio Code.app* application package into your *Application* folder to make it available in the macOS *LaunchPad*.

Linux

1. Open a web browser and navigate to https://code.visualstudio.com/.
2. Click the *.deb* download button to download the package for Debian based Linux distributions, or the *.rpm* download button for Red Hat based Linux distributions.
3. Once the download finishes, open the package downloaded to install.

Install the C# Dev Kit Extension

For C# development using VSCode, it is recommended to use the official *C# Dev Kit* extension provided by Microsoft. Installing this extension will add additional features to VSCode such as a project system and *Solution Explorer* for C# projects. It also provides code editing features such as syntax highlighting, code completion, code navigation, refactoring, NuGet package management, and debugging tools.

> **NOTE**
>
> The *Solution Explorer* panel is a hierarchical view provided by the C# Dev Kit extension that displays your solution structure similar to Visual Studio's Solution Explorer.
>
> When you open a workspace in Visual Studio Code containing a .NET solution file (*.sln*), the *Solution Explorer* panel automatically appears and loads your solution. From this panel, you can perform common operations like:
>
> - Adding new files.
> - Managing project references.
> - Viewing dependencies.
> - Executing build commands.
>
> For more information about the *Solution Explorer* panel in Visual Studio Code offered through the C# Dev Kit extension, you can view the official documentation in the Project Management documentation.

To install the C# Dev Kit extension, perform the following:

1. Launch the *Visual Studio Code* application.
2. Open the *Extensions Panel* by clicking the icon in the *Activity Bar* on the left or choosing *View* > *Extensions* from the top menu.
3. Enter `C#` in the *Search Box*
4. Click install for the *C# Dev Kit* extension.

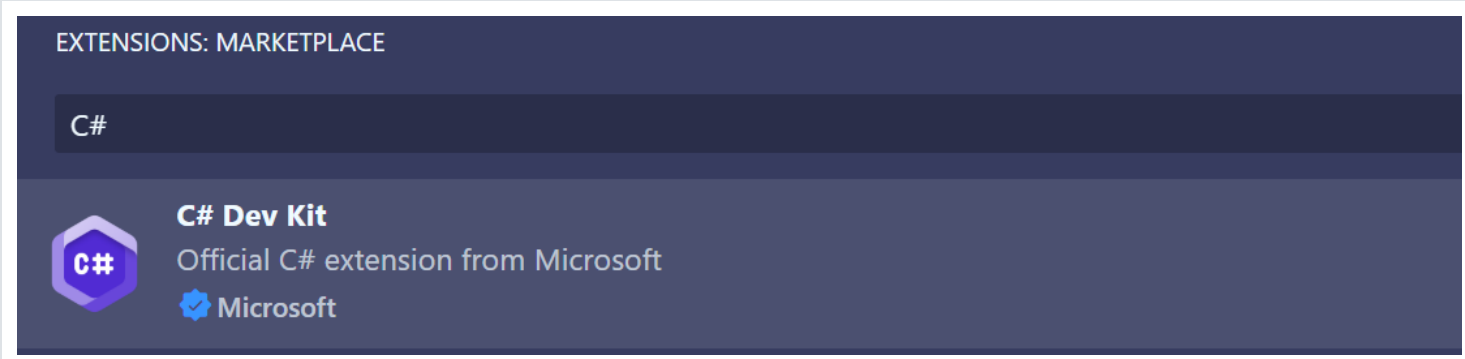


Figure 2-1: The C# Dev Kit Extension listed in Visual Studio Code

> **NOTE**
>
> When you search `C#` in the *Extension Panel* you may notice there is the C# Dev Kit extension and a base standard C# extension. When installing the C# Dev Kit extension, the base extension will also be installed as a requirement.

Installing the "MonoGame for VSCode" Extension

Throughout this tutorial, we will be using the MonoGame Content Builder (MGCB) Editor to add content to the game. MonoGame offers an official extension for Visual Studio 2022 that allows you to double-click the *Content.mgcb* file to automatically open it in the MGCB Editor. While there is no official tool for VSCode, there is a an extension developed by community member r88 to provide similar functionality and is regularly used by the MonoGame developers themselves. We will be using that extension throughout this tutorial.

To install it, with VSCode open:

1. Open the *Extensions Panel* by clicking the icon in the *Activity Bar* on the left or choosing *View* > *Extensions* from the top menu.
2. Enter `MonoGame for VSCode` in the *Search Box*
3. Click install on the *MonoGame for VSCode* extension by r88.

(Optional) Install the ".NET MAUI" extension to help with mobile development

This extension is provided by Microsoft and is built on top of the C# and C# Dev Kit extensions. They will be installed as dependencies if they were not already installed. While primarily designed for MAUI development, it provides essential mobile debugging capabilities that MonoGame mobile projects can utilize.

The .NET MAUI extension adds features for building mobile apps, including:

- F5 debugging support - Debug your MonoGame mobile apps on emulators, simulators, and physical devices
- Easy target switching - Change debug/deploy targets between different mobile platforms
- Enhanced C# Dev Kit integration - Leverages Solution Explorer, Test Explorer, and advanced language features

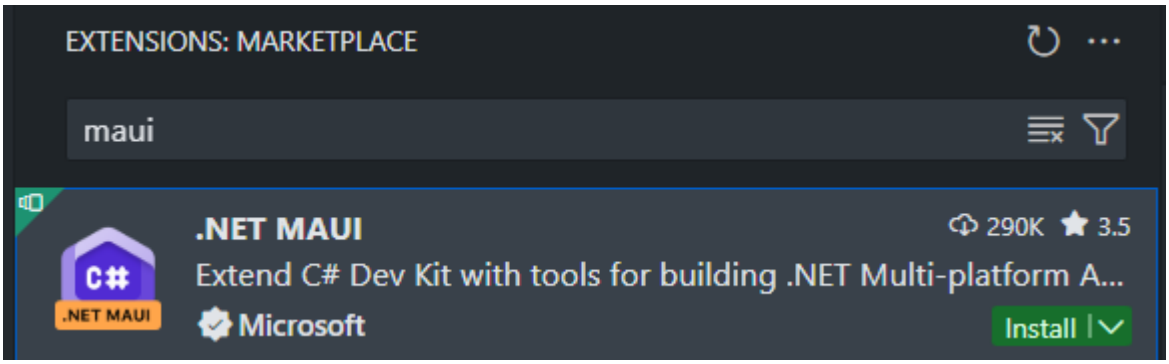


Figure 2-2: The .NET MAUI Extension listed in Visual Studio Code

> **NOTE**
>
> This extension is recommended if you wish to do iOS or Android development. It is required for debugging support when working on mobile games.

Setup WINE for Effect Compilation (macOS and Linux Only)

Effect (shader) compilation requires access to DirectX. This means it will not work natively on macOS and Linux systems, but it can be used through WINE. MonoGame provides a setup script that can be executed to setup the WINE environment. Below you can find the steps based on your operating system. To do this, follow the instructions for your operating system below:

Windows

> **NOTE**
> Setting up WINE for effect compilation is not required for Windows

macOS

Open a new *Terminal* window and execute the following commands:

```
brew install p7zip
brew install --cask wine-stable
xattr -dr com.apple.quarantine "/Applications/Wine Stable.app"
wget -qO- https://monogame.net/downloads/net8_mgfxc_wine_setup.sh | bash
```

> **NOTE**
> After performing these steps, a new folder called *.winemonogame* will be created in your home folder. If you ever wish to undo the setup performed by this script, you can simply delete this folder.

Linux

Open a new *Terminal* window and execute the following commands:

```
sudo apt-get update && sudo apt-get install -y curl p7zip-full wine64
wget -qO- https://monogame.net/downloads/net8_mgfxc_wine_setup.sh | bash
```

> **NOTE**
>
> After performing these steps, a new folder called *.winemonogame* will be created in your home folder. If you ever wish to undo the setup performed by this script, you can simply delete this folder.

Creating Your First MonoGame Application

With your development environment setup, it is time to create your first MonoGame application.

1. Launch the VSCode application
2. Open the *Command Palette* by clicking *View* > *Command Palette* or by using the keyboard shortcut `CTRL+SHIFT+P` (`CMD+SHIFT+P` on Mac).
3. Type `.NET New Project` in the *Command Palette* and choose the *.NET New Project* command
4. Next, you will be presented with a list of the available .NET project templates. Enter `MonoGame` into the prompt to filter the project templates to only show the MonoGame ones, then choose the *MonoGame Cross-Platform Desktop Application* project template.

> **NOTE**
>
> If the MonoGame templates are not showing up, then you skipped the step to install the templates, exit the project creation and click `Terminal -> New Terminal` in VSCode and run the following command:
>
> ```
> dotnet new install MonoGame.Templates.CSharp
> ```

5. After choosing the template, a dialog window will appear asking you to choose a location to save the project, this is a folder where your projects will be created by default.
6. Next, you will be prompted to enter a name for the project. Enter the name `DungeonSlime`, which will create your project in a new folder with the same name.
7. If this is your first time creating your project, you will be asked to choose a solution format, simply select the default `.sln` option and click `Next` to continue. (This does not occur with subsequent projects)
8. Finally, select the *Create Project* prompt.

After selecting *Create Project*, a new C# project will be generated based on the chosen MonoGame template and opened automatically in VSCode.

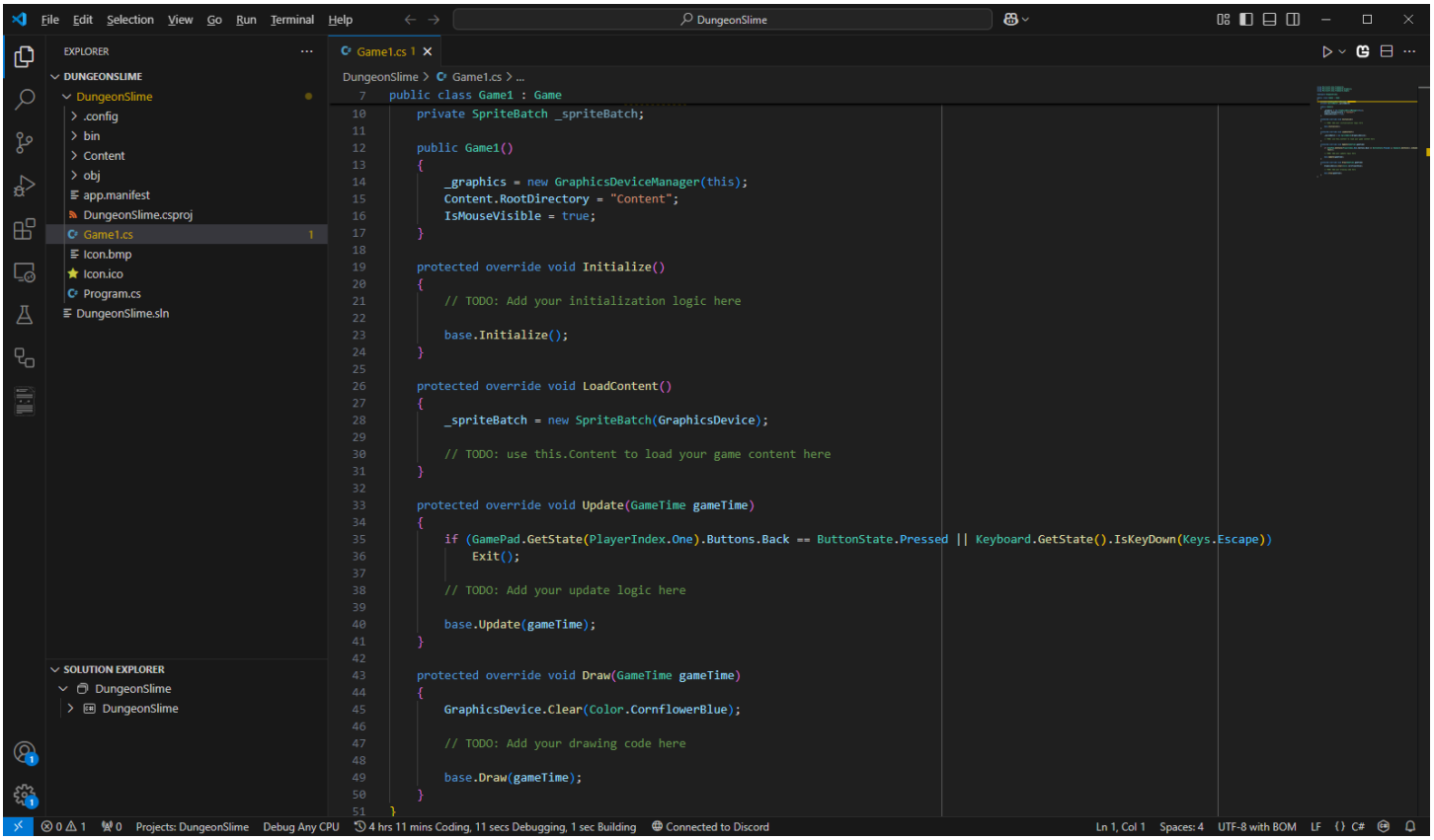


Figure 2-3: A new MonoGame project after being created in Visual Studio Code

Now that we have the project created, press the F5 key on your keyboard, or choose *Run* > *Start Debugging* from the top menu. If prompted for a configuration, choose *C#*. The project will compile and run, displaying a screen similar to the following:

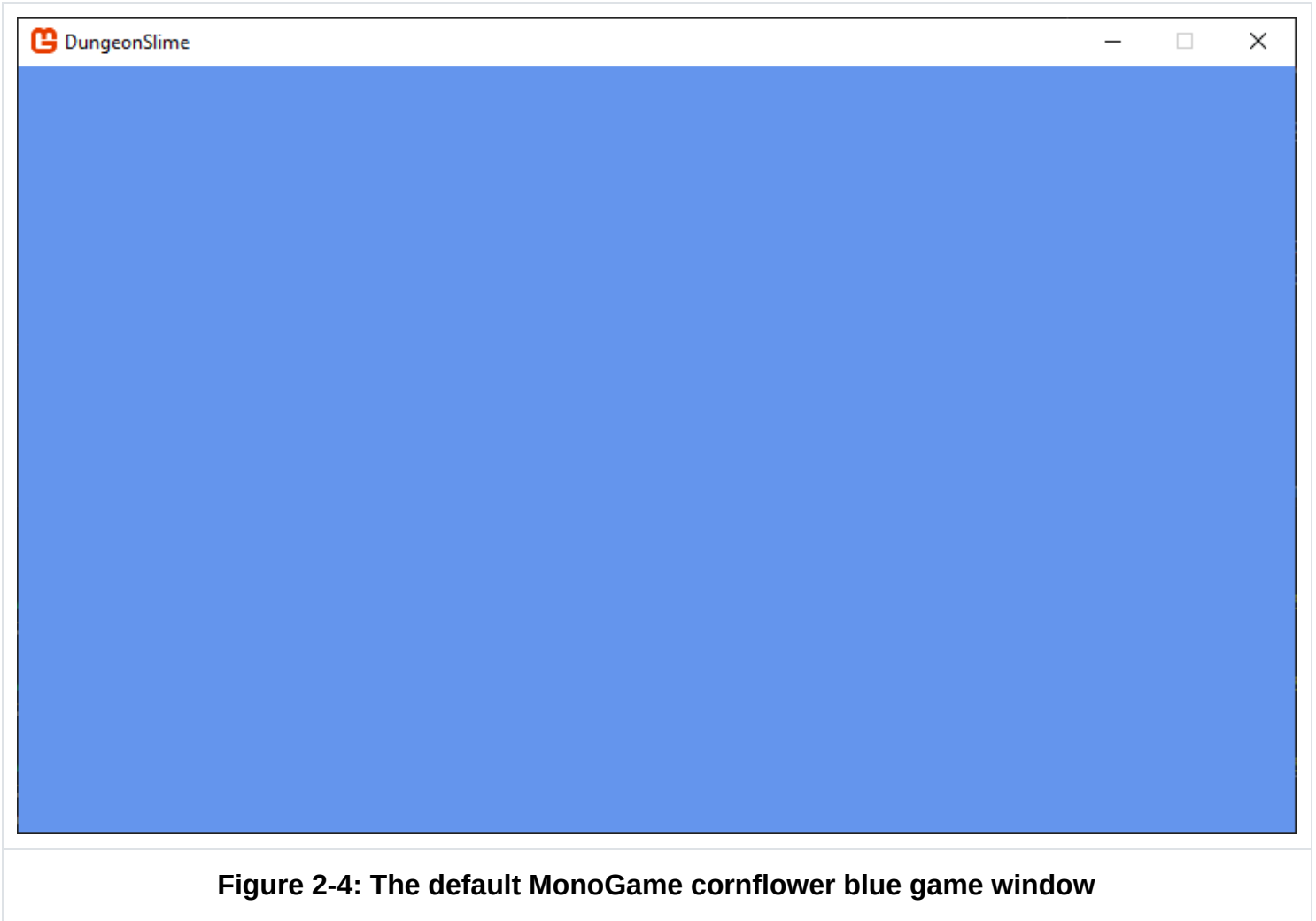


Figure 2-4: The default MonoGame cornflower blue game window

Be amazed, the default MonoGame Cornflower Blue game window. You have just created your very first MonoGame application. While there is not much happening here visually, there is a lot going on behind the scenes that the MonoGame framework is handling for you. When you ran the application, the following occurred:

1. The application started
2. The game window was created and graphics were initialized
3. A loop is entered which performs the following over and over, until the game is told to exit:
    1. The game is updated
    2. The game is rendered to the window

You can exit the game at any time by pressing the `Esc` key on your keyboard.

> **NOTE**
>
> Above, I mentioned that a loop is entered. This is commonly referred to as the *game loop*, which we will discuss in more detail in the next chapter. The reason the application enters this loop is because game applications work differently than traditional desktop applications, such as your web browser.
>
> Desktop applications are event based, meaning once loaded, they do not do much at all while waiting for input from the user. They respond to user interactions and redraw the window only when necessary.
>
> In games, things are always happening, such as objects moving around like the player or particle effects. To handle this, games implement a loop structure that runs continuously, first calling a method to `update` the game logic, and then a `draw` method to render the current frame, until it is told to exit.

Conclusion

In this chapter, you accomplished the following:

- You setup your operating system to develop .NET applications by installing the .NET SDK
- You installed the MonoGame project templates.
- You installed VSCode and the necessary extension to develop C# applications with VSCode
- You created and ran your first MonoGame project.

Now that your development environment is setup and ready to go, you can dive in and start building your first game. In the next chapter, we will cover the contents of the `Game1.cs` file that was included in the MonoGame project you just created.

Test Your Knowledge

1. What are two advantages of MonoGame being a framework rather than an engine?

   Any two of the following:

   - Flexibility: Developers can use their preferred development tools
   - Integration: MonoGame easily integrates with other .NET libraries
   - Cross-platform Development: Projects can be developed on Windows, macOS, or Linux
   - Version Control Friendly: The code-first approach works well with systems like Git

2. What is the primary reason that game applications implement a *game loop* structure instead of using an event-based approach like traditional desktop applications?

   Game applications implement a *game loop* structure because games need to continuously update and render, even when there is no user input. In games, objects might be moving, animations playing, and physics calculating regardless of user interaction, requiring constant updating and rendering until the game is told to exit.

3. What is the color of the game window when you run a MonoGame project for the first time?

   Cornflower Blue

Chapter 03: The Game1 File

Explore the contents of the Game1 file generated when creating a new MonoGame project.

After you created a new MonoGame project using the *MonoGame Cross-Platform Desktop Application* template in Chapter 02, you will notice the generated files and project structure that serve as a starting point for your game application. While MonoGame offers different templates based on target platform, all projects will contain the `Game1.cs` file.

Chapter Video

Live Recording

Exploring the Game1 Class

At the core of a MonoGame project is the **Game** class. This class handles the initialization of graphics services, initialization of the game, loading content, updating, and rendering the game. When you create a new MonoGame project, this **Game** class is implemented as the `Game1` class that you can customize as needed for your specific game.

> **TIP**
>
> While the default template names the class `Game1`, you are free to rename it to something more appropriate for your project. However, for consistency, the documentation will continue to refer to it as `Game1`.

Locate the `Game1.cs` file that was generated when you created the MonoGame project and open it. The default content will be:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace DungeonSlime;

public class Game1 : Game
{
    private GraphicsDeviceManager _graphics;
    private SpriteBatch _spriteBatch;

    public Game1()
    {
        _graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        IsMouseVisible = true;
    }

    protected override void Initialize()
    {
        base.Initialize();
    }

    protected override void LoadContent()
    {
        _spriteBatch = new SpriteBatch(GraphicsDevice);
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);

        base.Draw(gameTime);
    }
}
```

This class provides the following structure:

1. **Graphics and Rendering**: The class declares two core graphics components; the GraphicsDeviceManager for interacting with the Graphics Processing Unit (GPU) and the SpriteBatch for 2D rendering.
2. **Initialization**: The constructor and Initialize method handle the game's setup sequence.
3. **Content Loading**: The LoadContent method manages game asset loading during startup.
4. **Game Loop**: The *game loop* consists of the Update method for game logic and the Draw method for rendering, running continuously until the game is told to exit.

Figure 3-1 below shows the lifecycle of a MonoGame game including the Update and Draw methods that make up the *game loop*.

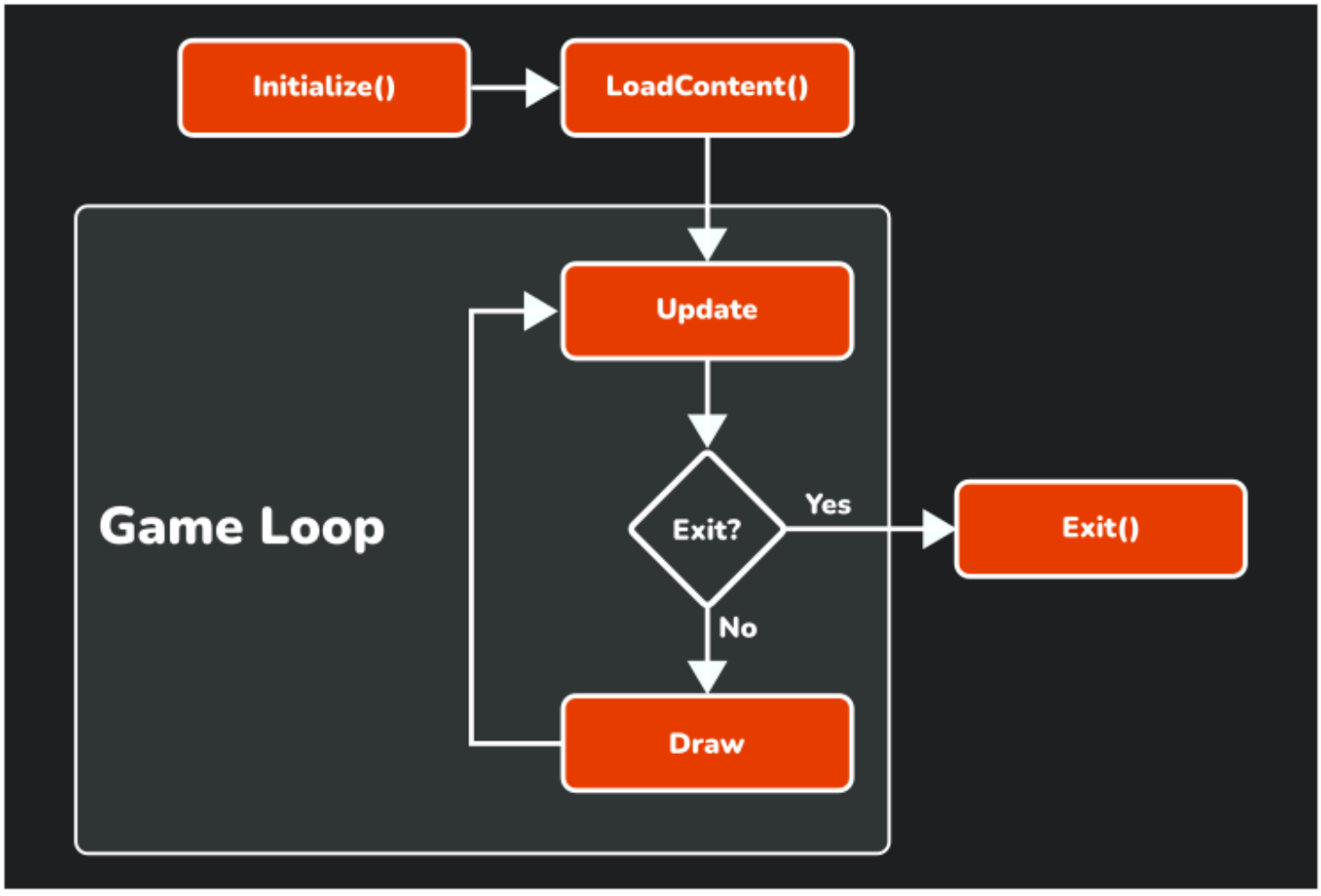


Figure 3-1: Lifecycle of a MonoGame game

Graphics and Rendering

The graphics pipeline in MonoGame starts with two components: the GraphicsDeviceManager and SpriteBatch.

```
private GraphicsDeviceManager _graphics;
private SpriteBatch _spriteBatch;
```

The **GraphicsDeviceManager** initializes and manages the connection to the graphics hardware. It handles tasks such as setting the screen resolution, toggling between fullscreen and windowed mode, and managing the **GraphicsDevice**, which is the interface between your game and the Graphics Processing Unit (GPU) the game is running on. The **SpriteBatch** optimizes 2D rendering by batching similar draw calls together, improving draw performance when rendering multiple sprites.

Initialization

MonoGame's initialization process for your game follows a specific sequence. The constructor runs first, which handles basic setup like creating the **GraphicsDeviceManager**, setting the content directory, and the visibility of the mouse.

```
public Game1()
{
    _graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    IsMouseVisible = true;
}
```

After that, the **Initialize** method executes, providing a dedicated place for additional configuration and initializations.

```
protected override void Initialize()
{
    base.Initialize();
}
```

This separation allows you to perform setup tasks in a logical order; core systems in the constructor and game-specific initializations in the **Initialize** method. The call to `base.Initialize()` should never be removed, as this is where the graphics device is initialized for the target platform.

> **TIP**
>
> You may be wondering why there is an **Initialize** method instead of performing all initializations in the constructor. The **Initialize** method is a `virtual` method that is overridden, and it is advised to not call overridable methods from within a constructor, as this can lead to unexpected states in the object constructor when called. Additionally, when the constructor is called, the base constructor will instantiate properties and services based on the target platform that may be needed first before performing initializations for the game itself.

Content Loading

The **LoadContent** method serves as the place for asset management. Here you can load textures, sound effects, music, and other game assets. We will cover loading assets in the coming chapters as we discuss each asset type that can be loaded. In a new project, the only task it performs is initializing a new instance of the **SpriteBatch**.

```
protected override void LoadContent()
{
    _spriteBatch = new SpriteBatch(GraphicsDevice);
}
```

This method is only called once during the startup of the game, but *when* it is called can be a little confusing at first. In the **Initialize** method shown above, when the `base.Initialize` call is executed, the final task it performs is calling the **LoadContent** method. This means any initializations you need to perform that have a dependency on assets being loaded should be done *after* the `base.Initialize` call and not *before* it.

The Game Loop

MonoGame implements a *game loop* by calling **Update** and **Draw** over and over until the game is told to exit. Recall at the end of Chapter 02 when you ran the project for the first time, I mentioned that there is a lot going on behind the scenes? This game loop is what I was referring to.

MonoGame is executing the **Update** method and then the **Draw** method 60 times per second.

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    base.Draw(gameTime);
}
```

The **Update** method at the moment is not doing much, only checking for input from a controller or keyboard to determine if the game should exit. However, the **Draw** method is doing more than what it appears to at first glance.

The first line is executing the **Clear** method of the **GraphicsDevice** property using the color **CornflowerBlue**. Recall that the **GraphicsDevice** object is your direct interface between the game and what is rendered to the screen. Every time the **Draw** method is called, this line of code is erasing the contents of the game window and refilling it with the color specified. Without clearing the contents of the screen first, every draw call would draw the new frame render on top of the previous render, and you'd end up with something like the old solitaire win screen.

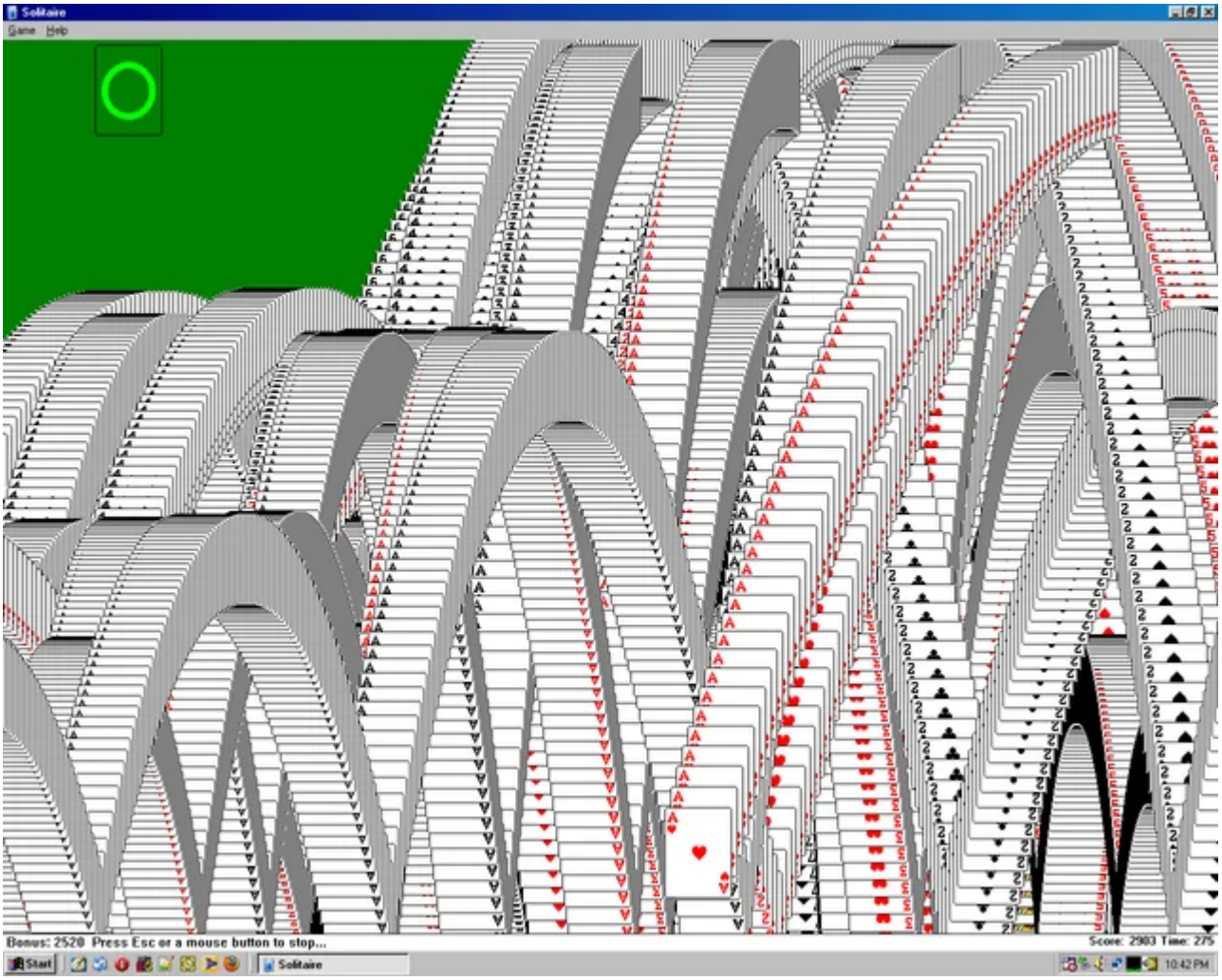


Figure 3-2: Windows XP Solitaire Win Screen

While this can make for a neat effect, it is not something you want all the time. So, the screen is cleared and refilled with a solid color.

NOTE

You can test this yourself by modifying the code to use a different color, such as **Color.MonoGameOrange**. (yes, there is a MonoGame Orange color).

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.MonoGameOrange);

    base.Draw(gameTime);
}
```

After making this change and running the game, the screen is cleared to the MonoGame Orange color.

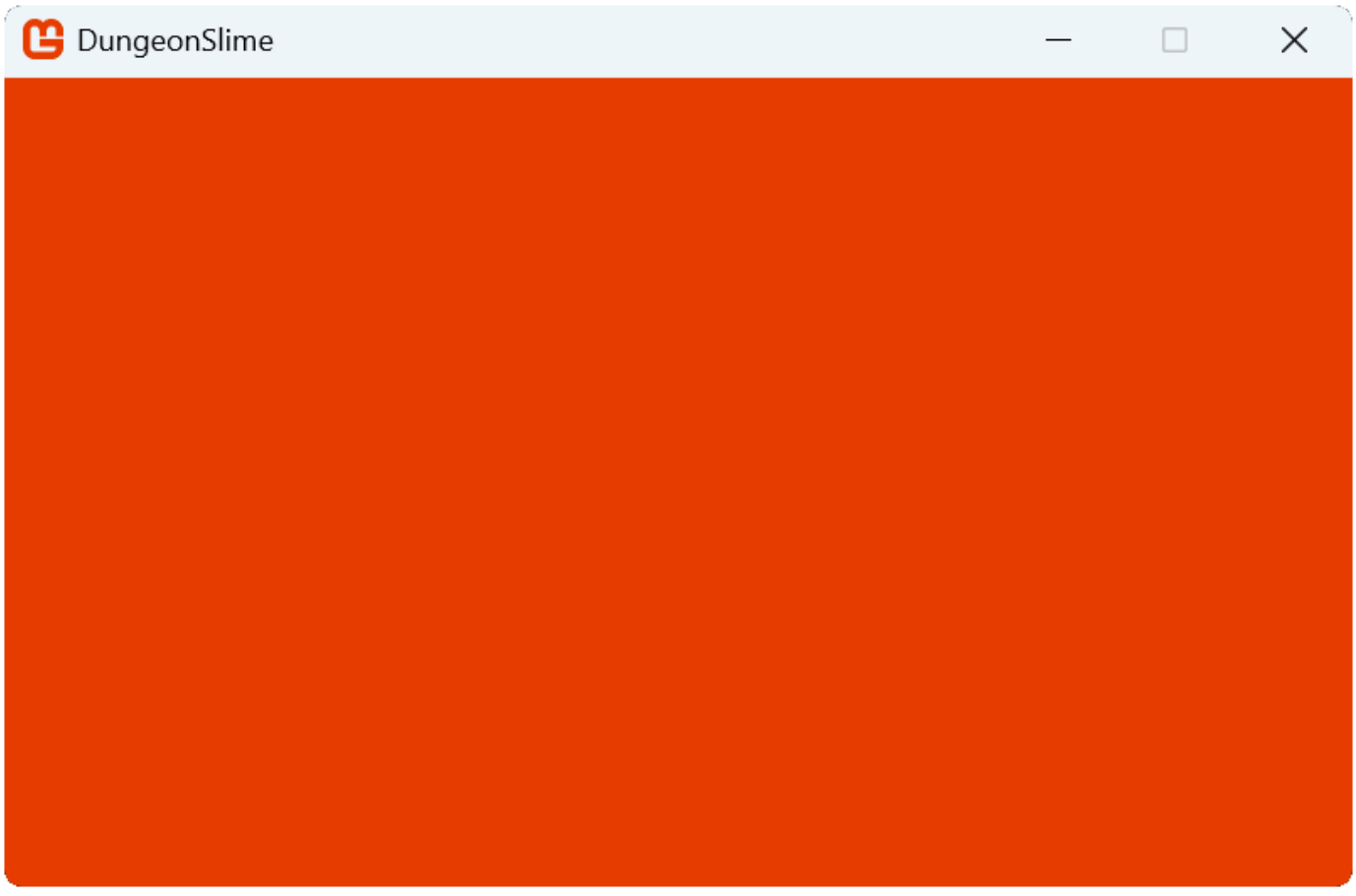


Figure 3-3: The game window clearing the screen using the MonoGame Orange color

Each time the game loop completes and the game is drawn to the screen, we call this a *frame*. So, if MonoGame is running the game loop at 60 frames per second, that means it is performing an update and a render of a frame in 16ms. Notice that both the **Update** and the **Draw** methods both receive a parameter of the type **GameTime**. The **GameTime** parameter provides a snapshot of the timing values for the game, including the amount of time that it took for the previous frame to execute. This is commonly referred to as the *delta time*.

Delta time allows you to track time accurately for things such as animations and events based on *game time* and not the speed of the processor (CPU) on the machine running the game. While in ideal circumstances, the delta time will always be 16ms, there are any number of things that could cause a temporary slow down or hiccup in a frame, and using the delta time ensures that time-based events are always correct.

Conclusion

In this chapter, you accomplished the following:

- You read through the default code provided in the `Game1.cs` file created by a MonoGame template.
- You learned about the lifecycle of a MonoGame game project.
- You learned what a game loop is and how it is implemented in MonoGame.

In the next chapter, you will start working with sprites and learn how to load and render them.

Test Your Knowledge

1. Can the `Game1` class be renamed or is it required to be called `Game1`

> It is not a requirement that it be called `Game1`. This is just the default name given to it by the templates when creating a new MonoGame game project. However, you cannot change the name of the *BASE* class `Game`, as this is a MonoGame construct.

2. What is the **SpriteBatch** used for?

> The **SpriteBatch** provides an optimized method of rendering 2D graphics, like sprites, onto the screen.

3. When is the **LoadContent** method executed and why is it important to know this?

> **LoadContent** is executed during the `base.Initialize()` method call within the **Initialize** method. It is important to know this because anything being initialized that is dependent on content loaded should be done **after** the `base.Initialize()` call and not **before**.

4. How does MonoGame provide a *delta time* value?

> Through the **GameTime** parameter that is given to both the **Update** and the **Draw** methods.

04: Creating a Class Library

Learn how to create and structure a reusable MonoGame class library to organize game components and share code between projects.

One of the goals of this tutorial is to create reusable modules that you can use to jump start your next game project after this. Rather than starting from scratch each time, we will build a collection of game components you can take with you from project to project.

In this chapter you will:

- Learn about class libraries and their benefits for game development.
- Create a MonoGame class library project using templates.
- Add library references to your game project.
- Structure your library for reusability.
- Set up the foundation for creating shared game components.

Chapter Video

Live Recording

What Is a Class Library

Think of a class library like a toolbox for your game development. Just as a mechanic keeps their most-used tools in a toolbox they bring to every job, a class library stores code components you will want to use in multiple game projects. Instead of recreating these tools for each new game (or copying and pasting code), you organize them in one place where they are easy to find, use, and improve over time.

The following diagrams show how this works:

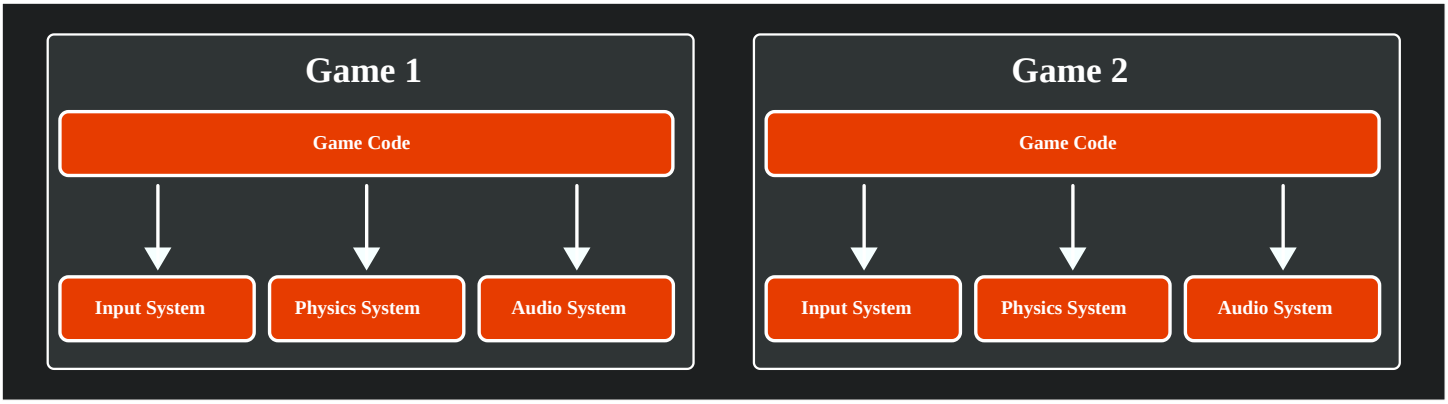


Figure 4-1: This diagram displays the block for Game 1 on the left and Game 2 on the right. In this example, when not using a class library, code for input, physics and audio systems is duplicated in both game projects. If a bug is fixed in one system, the effort has to be replicated in the same system in the other game projects.

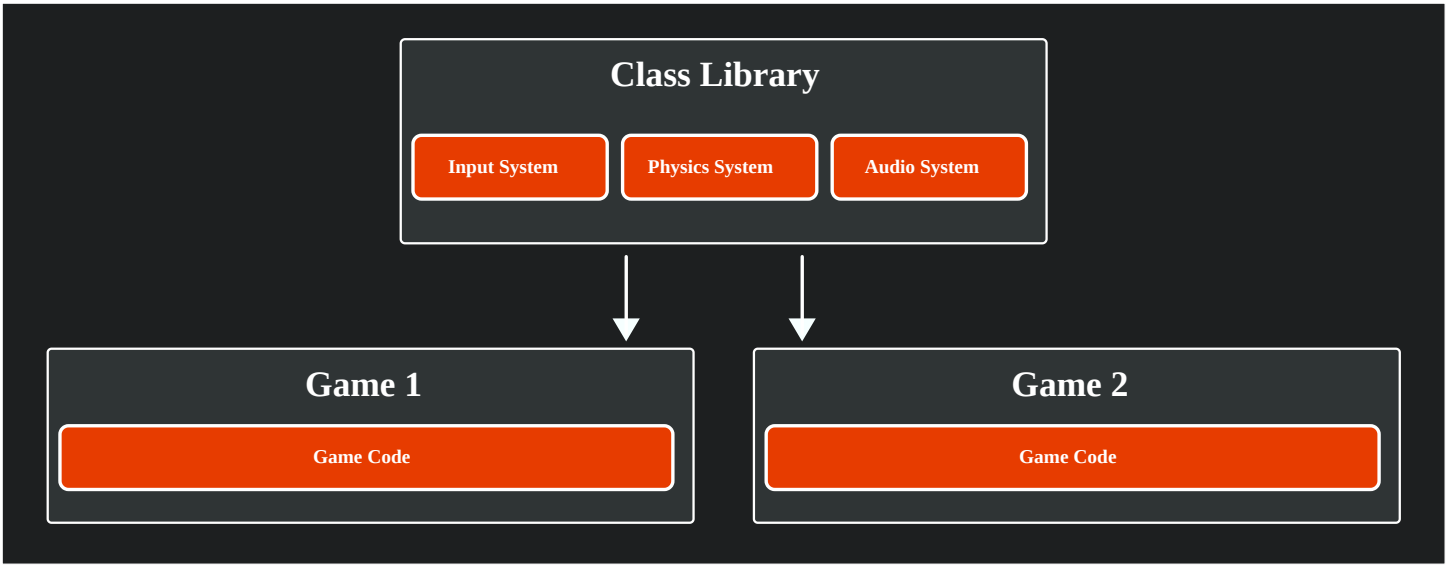


Figure 4-2: This diagram displays a block for a class library containing common modules at the top, which are then shared between the two game projects below. If a bug is found in a module, fixing the bug will fix it across all game projects that use the class library.

> **NOTE**
>
> A class library is a project type that compiles into a Dynamic Link Library (DLL) instead of an executable. It contains reusable code that can be referenced by other projects, making it perfect for sharing common functionality across multiple games.

Why Create a Class Library?

Creating a class library offers several important advantages, especially as your games grow in complexity:

1. **Reusability**: Instead of rewriting the same code for each new game project, you build it once in your library and reuse it everywhere. This is like creating a multi-tool that works across all your projects.
2. **Organization**: Your game code stays focused on the unique aspects of each game, while common functionality lives in the library. This keeps your project folder neat and makes code easier to find.
3. **Maintainability**: When you improve or fix a bug in your library code, all games using that library benefit automatically. This means you only fix a bug once, rather than in several places.
4. **Testing**: You can test your library code independently from any specific game. This helps ensure your core systems are solid before you build a game on top of them.

As your library grows, you will accumulate a personal collection of well-tested modules that make starting new projects much faster. The modules we will create in this library will handle common game tasks like input, audio, sprites, and animations.

Adding the Class Library

MonoGame offers the *MonoGame Game Library* project template to add a new class library project that is configured with the correct MonoGame framework references. Using this template saves time and ensures compatibility with MonoGame projects.

To use the template to add the class library, perform the following based on which development environment you are using:

Visual Studio Code

To add the class library using the MonoGame Game Library project template in Visual Studio Code, perform the following:

1. In the *Solution Explorer* panel, right-click the *DungeonSlime* solution.
2. Choose *New Project* from the context menu.
3. Enter "MonoGame Game Library" and select it as the template to use.
4. Name the project "MonoGameLibrary".
5. When prompted for a location, use the default option, which will put the new project in a folder next to your game project.
6. Select "Create Project".

To add the class library using the MonoGame Game Library project template in Visual Studio 2022, perform the following:

1. Right-click the *DungeonSlime* solution in the Solution Explorer panel.
2. Choose Add > New Project from the context menu.
3. Enter "MonoGame Game Library" in the search box, select that template, then click Next.
4. Name the project "MonoGameLibrary".
5. The default location will place the new project in a folder next to your game project; you do not need to adjust this setting.
6. Click "Create".

To add the class library using the MonoGame Game Library project template with the dotnet CLI, perform the following:

1. Open a new Command Prompt or Terminal window in the same folder as the *DungeonSlime.sln* solution file.
2. Enter the command `dotnet new mglib -n MonoGameLibrary` to create the project, placing it in a folder next to your game project.
3. Enter the command `dotnet sln DungeonSlime.sln add ./MonoGameLibrary/MonoGameLibrary.csproj` to add the newly created class library project to the *DungeonSlime.sln* solution file.

Adding a Reference To The Class Library

Now that the game library project has been created, a reference to it needs to be added in our game project. Without adding a reference, our game project will be unaware of anything we add to the class library. To do this, perform the following based on which development environment you are using:

Visual Studio Code

To add the game library project as a reference to the game project in Visual Studio Code:

1. In the Solution Explorer panel, right-click the *DungeonSlime* project.
2. Choose "Add Project Reference" from the context menu.
3. Choose *MonoGameLibrary* from the available options.

> **TIP**
>
> The Solution Explorer panel in VSCode is provided by the C# Dev Kit extension that was installed in Chapter 02. If you do not see this panel, you can open it by
>
> 1. Opening the *Command Palette* (View > Command Palette).
> 2. Enter "Explorer: Focus on Solution Explorer View" and select the command.

To add the game library project as a reference to the game project in Visual Studio 2022:

1. In the Solution Explorer panel, right-click the *DungeonSlime* project.
2. Select Add > Project Reference from the context menu.
3. Check the box for the *MonoGameLibrary* project.
4. Click Ok.

To add the game library project as a reference to the game project with the dotnet CLI:

1. Open a new Command Prompt or Terminal window in the same folder as the *DungeonSlime.csproj* C# project file.
2. Enter the command `dotnet add ./DungeonSlime.csproj reference ../MonoGameLibrary/MonoGameLibrary.csproj`. This will add the *MonoGameLibrary* reference to the *DungeonSlime* game project.

Clean Up

When using the *MonoGame Game Library* project template, the generated project contains file similar to a standard MonoGame game project, including a *dotnet-tools.json* manifest file, a *Content.mgcb* file, and a `Game1.cs` file. For the purposes of this tutorial, we will not need these. To clean these up, locate the following in the *MonoGameLibrary* project folder and delete them:

1. The *.config/* folder.
2. The *Content/* folder
3. The `Game1.cs` file.

> **TIP**
>
> These files are needed in more advanced scenarios, such as creating a central code base for game logic that is referenced by other projects, each targeting different platforms such as desktop, mobile, and console. Creating a project structure of this type is out of scope for this tutorial.
>
> If you would like more information on this, Simon Jackson has written the article Going cross-platform with MonoGame which covers this in more detail.
>
> Also the `2D StartKit` and `2D Blank StartKit` templates provide you with a richer startup project targeting all platforms, using a common Class Library to reuse code across them all. Although we recommend completing this tutorial first before tackling that beast.

Creating Our First Library Module

We will create a class for our library called `Core`. This class will extend the MonoGame **Game** class and provide a starting point for game development with a few common built-in functionalities. Creating this will also allow us to confirm that our class library reference setup is correct.

Create a new file called `Core.cs` in the *MonoGameLibrary* project and add the following code:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace MonoGameLibrary;

public class Core : Game
{
    internal static Core s_instance;

    /// <summary>
    /// Gets a reference to the Core instance.
    /// </summary>
    public static Core Instance => s_instance;

    /// <summary>
    /// Gets the graphics device manager to control the presentation of graphics.
    /// </summary>
    public static GraphicsDeviceManager Graphics { get; private set; }

    /// <summary>
    /// Gets the graphics device used to create graphical resources and perform
primitive rendering.
    /// </summary>
    public static new GraphicsDevice GraphicsDevice { get; private set; }

    /// <summary>
    /// Gets the sprite batch used for all 2D rendering.
    /// </summary>
    public static SpriteBatch SpriteBatch { get; private set; }

    /// <summary>
    /// Gets the content manager used to load global assets.
    /// </summary>
    public static new ContentManager Content { get; private set; }

    /// <summary>
    /// Creates a new Core instance.
    /// </summary>
    /// <param name="title">The title to display in the title bar of the game
window.</param>
    /// <param name="width">The initial width, in pixels, of the game
window.</param>
    /// <param name="height">The initial height, in pixels, of the game
window.</param>
```

```
    /// <param name="fullScreen">Indicates if the game should start in fullscreen
mode.</param>
    public Core(string title, int width, int height, bool fullScreen)
    {
        // Ensure that multiple cores are not created.
        if (s_instance != null)
        {
            throw new InvalidOperationException($"Only a single Core instance can
be created");
        }

        // Store reference to engine for global member access.
        s_instance = this;

        // Create a new graphics device manager.
        Graphics = new GraphicsDeviceManager(this);

        // Set the graphics defaults.
        Graphics.PreferredBackBufferWidth = width;
        Graphics.PreferredBackBufferHeight = height;
        Graphics.IsFullScreen = fullScreen;

        // Apply the graphic presentation changes.
        Graphics.ApplyChanges();

        // Set the window title.
        Window.Title = title;

        // Set the core's content manager to a reference of the base Game's
        // content manager.
        Content = base.Content;

        // Set the root directory for content.
        Content.RootDirectory = "Content";

        // Mouse is visible by default.
        IsMouseVisible = true;
    }

    protected override void Initialize()
    {
        base.Initialize();

        // Set the core's graphics device to a reference of the base Game's
        // graphics device.
        GraphicsDevice = base.GraphicsDevice;
```

```
            // Create the sprite batch instance.
            SpriteBatch = new SpriteBatch(GraphicsDevice);
        }
    }
```

The `Core` class provides the following features:

1. It extends the MonoGame **Game** class, so it inherits all of the base functionality.
2. It implements a singleton pattern through the `Instance` property, ensuring that only one core exists.
3. It provides static access to the graphics device manager, the graphics device, the sprite batch, and the content manager.
4. It simplifies the game window setup with a constructor that handles common initializations.

> **NOTE**
>
> The `new` keyword in the property declaration `public static new GraphicsDevice GraphicsDevice` and `public static new ContentManager Content` is used to intentionally hide (or "shadow") the inherited `GraphicsDevice` and `Content` properties from the base `Game` class. This creates new properties with the same name but different accessibility (static vs. instance) in the derived class.
>
> When you access `Core.GraphicsDevice` or `Core.Content` you will be using these static properties, while `base.GraphicsDevice` or `base.Content` within instance methods of the `Core` class will access the original property. This pattern allows us to provide convenient static access to the graphics device and content manager throughout our game without having to reference the Core instance every time.

This approach provides a consistent foundation for all our games, handling common setup tasks and providing convenient access to core functionality.

> **NOTE**
>
> As this tutorial progresses, we will be coming back to this `Core` class to add more to it.

Updating Our Game to Use the Core Class

Now that we have our `Core` class, we can modify our game project to use it. Doing this will also help ensure that the project references were setup correctly.

Open the `Game1.cs` file and make the following changes:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;

namespace DungeonSlime;

public class Game1 : Core
{
    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // TODO: use this.Content to load your game content here

        base.LoadContent();
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // TODO: Add your update logic here

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // TODO: Add your drawing code here

        base.Draw(gameTime);
```

```
    }
}
```

The key changes made here are:

1. Added `using MonoGameLibrary;` directive to reference our library.
2. Removed the **GraphicsDeviceManager** and **SpriteBatch** fields; these are now supplied through the `Core` class.
3. Changed `Game1` class to inherit from `Core` instead of `Game`.
4. Updated the constructor to call the `Core` base constructor with our game configuration.

Running the game now will show the same window as before, only now it is at a 1280x720 resolution as per the configuration and it is using the `Core` class from our library. This may not seem like a big change visually, but it demonstrates how our library can simplify and standardize game initializations.

> **NOTE**
>
> If you get any additional prompts when starting the project, simply accept the defaults. Adding the Game Library gives the compiler more projects to choose from when starting the project, however, we can only "run" the Game Project, libraries are not executables.

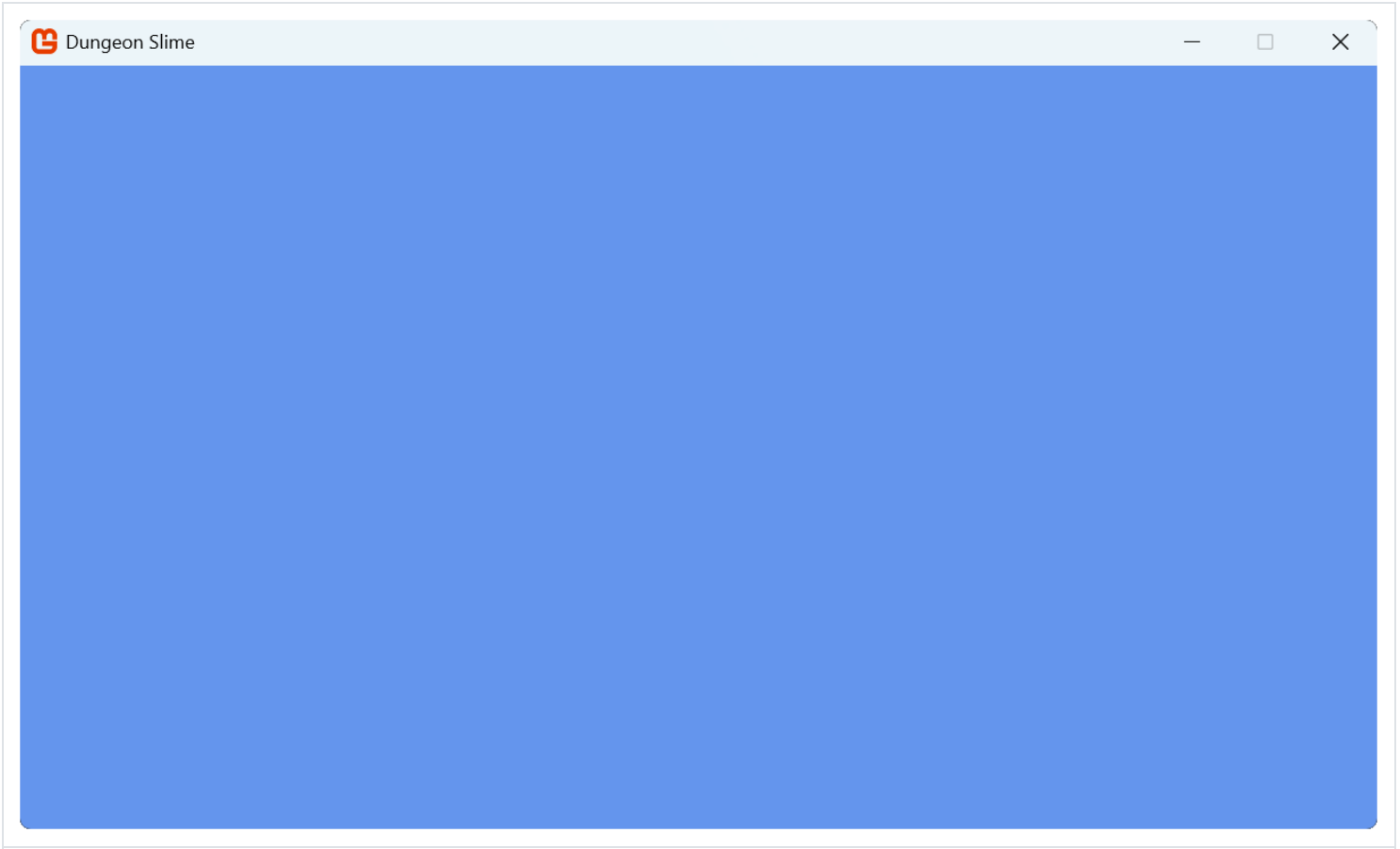


Figure 4-3: The game window at 1280x720 with the title Dungeon Slime

IMPORTANT

If you receive an error stating the following:

The type or namespace name 'Core' could not be found (are you missing a using directive or an assembly reference?)

This means either you forgot to add the `using MonoGameLibrary;` using directive to the top of the `Game1.cs` class file, or you did not add the project reference correctly. Ensure that the project reference was added correctly by revisiting the Add a Reference to the Class Library section above and that you added the using directive.

Conclusion

In this chapter, you accomplished the following:

- Learned about class libraries and their advantages for game development:
  - Code reusability across projects
  - Better organization and separation of concerns
  - Improved maintainability
  - Easier testing
- Created a MonoGame class library project
- Added the library as a reference to your game project
- Created your first reusable component, added a reference to it, and used it in the game project.

In the next chapter, we will learn about the Content Pipeline and how to load game assets.

Test Your Knowledge

1. What are the main benefits of using a class library for game development?

   The main benefits are:

   - **Reusability**: Code can be easily shared between different game projects
   - **Organization**: Separates reusable code from game-specific code
   - **Maintainability**: Changes to shared code benefit all games using the library
   - **Testing**: Library code can be tested independently of specific games

2. Why should you use the MonoGame Game Library template instead of a standard class library template?

   The MonoGame Game Library template automatically configures the correct MonoGame framework references and ensures compatibility with MonoGame projects, saving time and preventing potential setup issues.

3. What happens if you do not add a reference to your class library in your game project?

   > Without adding a reference, your game project will be unaware of any code in the class library. You will not be able to use any of the classes or components from the library in your game.

Chapter 05: Content Pipeline

Learn the advantages of using the Content Pipeline to load assets and go through the processes involved in loading your first asset

Every game has assets: images to represent the visual graphics to players, audio to provide sound effects and background music, fonts to render text with, and much more. These assets start out as raw files (e.g. *.png* image files or *.mp3* audio files), which you will need to load into the game to use.

Chapter Video

Live Recording

Loading Assets

Loading assets can be done during runtime directly from file, or it can be loaded through the **Content Pipeline**. Both of these methods are two sides of the same coin, and there are trade offs to each approach.

For instance, to load an image file directly at runtime, you would need to:

1. Add the image file to your project.
2. Configure the project to copy the image file on build to the build output folder.
3. Load the image file as a texture at runtime using the **Texture2D.FromFile** method.

> ⊗ **IMPORTANT**
>
> One major drawback of loading an image file as a texture directly at runtime is that the data is in a compressed format, such as .png or .jpg. These compression formats are not understood by a Graphics Processing Unit (GPU). In order to be used by your game, this data will need to be decompressed into raw bytes, a format that the GPU understands. This operation can potentially leave a larger memory footprint for your assets. You will also need to handle how different compression formats work on the platform you are targeting, such as desktops, mobile and consoles.

On the other side of the coin, MonoGame offers the **Content Pipeline**; a workflow for managing assets. The workflow is made up of a set of tools and utilities that are automatically added by default when you create a new MonoGame project using the MonoGame project templates. To use this workflow, you need to:

1. Add the asset file to your content project (*Content.mgcb* file) using the *MonoGame Content Builder Editor* (MGCB Editor).
2. Perform a project build. This will cause the *MonoGame.Content.Builder.Tasks* NuGet reference to compile the assets defined in the content project, optimized for the target platform, and automatically copy them to the game project build folder.
3. Load the compiled asset at runtime using the **ContentManager**.

For the same amount of steps, you also get the benefit of the assets being pre-processed and compiled to an optimized format for the target platform. For instance, image files can be compiled using DXT compression, which is a format that is understood by GPUs without needing to be decompressed first, reducing the memory footprint.

> **NOTE**
>
> For more information on the benefits of compiling assets and what optimizations it can offer, see the Content Pipeline documentation.

For this tutorial series, we are going to focus on using the content pipeline workflow to load assets. Doing this will allow you, as the developer, to get accustomed to using the content pipeline tools, and also give the benefits of having assets pre-compiled to optimized formats.

The MGCB Editor

As mentioned previously, the content pipeline workflow in MonoGame is made up of a set of tools that come with every new MonoGame project. At the center of this workflow is the MGCB Editor; a graphical tool for managing your game's content.

Opening the MGCB Editor can be done in different ways depending on which IDE and development environment you have. Choose the one you are using below to open the MGCB Editor so we can explore its interface:

Visual Studio Code

To open the *Content.mgcb* content project file in the MGCB Editor from Visual Studio Code, you can use the *MonoGame for VSCode* extension. You should already have installed this extension in Chapter 02. With this extension installed, anytime you have a code file open, you will see the MonoGame logo in the top-right of the code window like below:

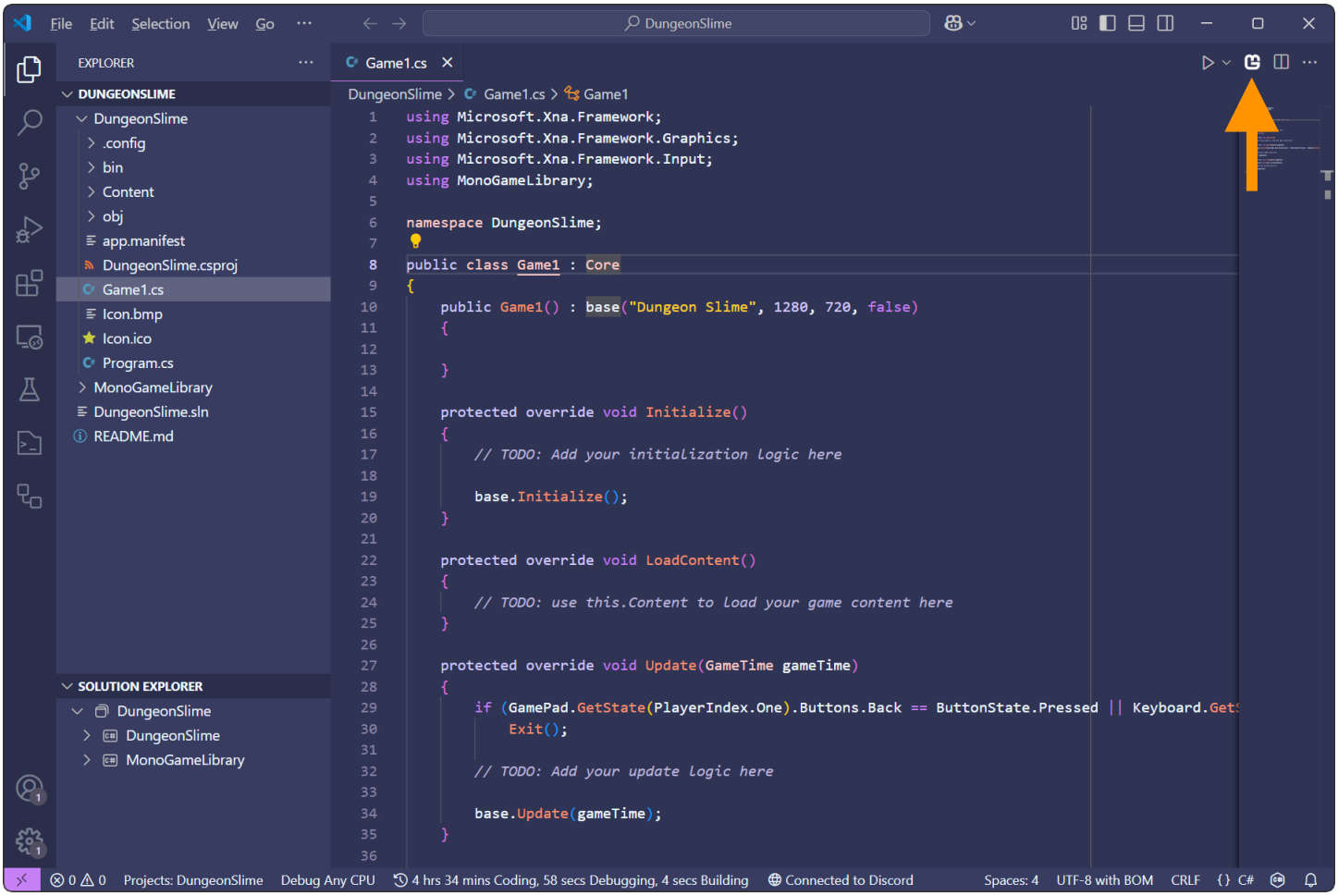


Figure 5-1: MonoGame for VSCode extension icon

Clicking the MonoGame logo here will open the *Content.mgcb* content project file from the current project in the MGCB Editor.

To open the *Content.mgcb* content project file in the MGCB Editor from Visual Studio 2022, you can use the *MonoGame Framework C# project templates* extension. Despite the name, this extension does more than just install the MonoGame project templates. With this extension installed, simply double-click the *Content.mgcb* content project file in the Solution Explorer panel to open it in the MGCB Editor.

To open the *Content.mgcb* content project file in the MGCB Editor using the dotnet CLI commands, perform the following:

1. Open a new Command Prompt or Terminal window in the same folder as the *DungeonSlime.csproj* project file (your main game project).
2. Enter the following dotnet CLI command

```
dotnet mgcb-editor ./Content/Content.mgcb
```

TIP

If for any reason the MGCB editor fails to load or you are hit with MGCB errors when you build your project, it is likely the MGCB references from the `dotnet-tools.json` configuration located in your projects `.config` folder have not been loaded/initialized.

To correct this, simply run the following from a terminal/command prompt in your projects directory (there the `.config` folder is located)

```
dotnet tool restore
```

This should restore the tool with the version that is configured. If at any time you update your `dotnet-tools.json` configuration, e.g. when upgrading to a newer version of MonoGame, **you will need to run this command again**

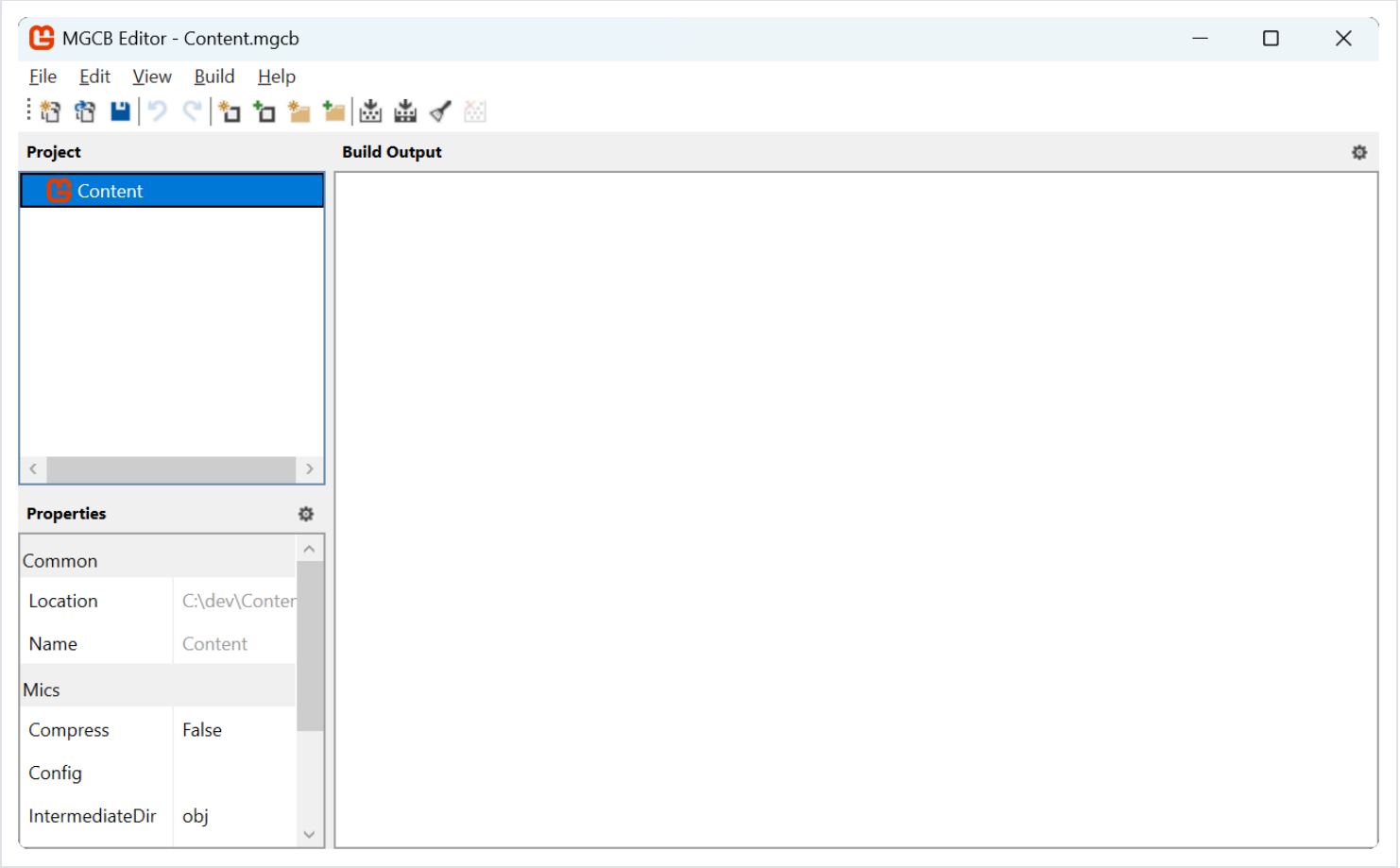


Figure 5-2: MonoGame Content Builder Editor (MGCB Editor) Window

In *Figure 5-2* above, you can see the user interface for the MGCB Editor:

- **Toolbar**: Contains icon buttons for common actions such as creating new items, opening files, saving changes, and building content.
- **Project Panel**: Located on the left side of the MGCB Editor, it displays a hierarchical tree view of all content items added to the content project. The root node *Content* represents the root of the content project.
- **Properties Panel**: Located in the lower left corner of the MGCB Editor, it lists the properties of the currently selected item in the project panel. The available properties are based on the selected element type.
- **Build Output Panel**: The large area to the right side outputs build messages, warnings, and errors when content is processed.

Creating Folders to Organize Content

Organizing your game assets into folders helps keep your content project manageable as it grows. For now, we will add a new folder that will hold the image assets we will add to the game throughout this tutorial series. In the MGCB Editor:

1. In the Project Panel, select the root `Content` node.
2. Right-click it and choose `Add > New Folder...` from the context menu.
3. Type `images` for the folder name and click the `Ok` button.

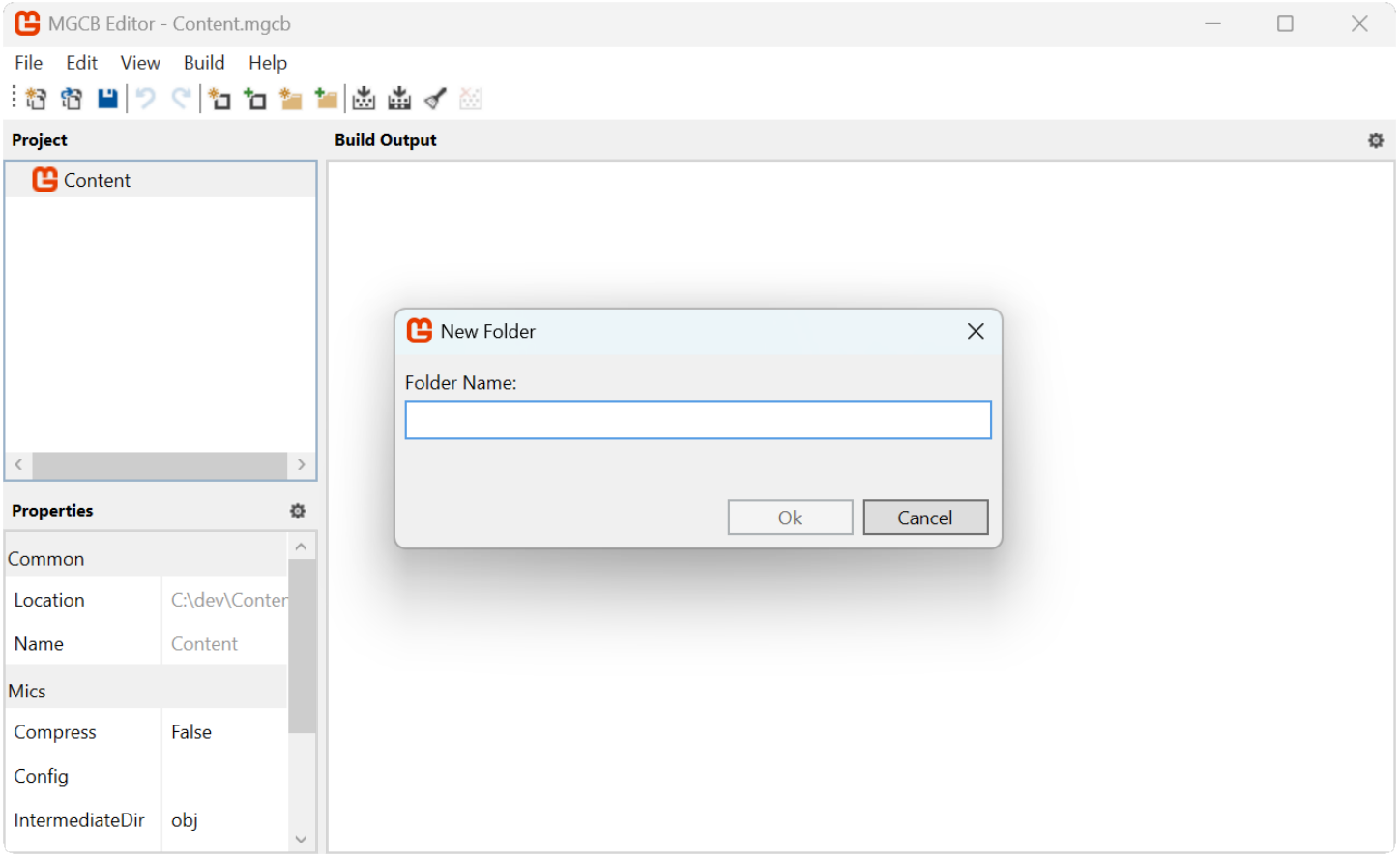


Figure 5-3: New folder pop-up

You have now created a folder that will help organize the game's image assets. As we continue through this tutorial series, we will be adding additional folders for organization of content such as audio, fonts, and effects.

> **NOTE**
>
> If you try to add a new folder that already exists in the file system but is not showing in the MGCB editor, you will get an error. Either remove the folder or use `Add Existing Folder` instead.

Adding Your First Asset

Now that we have a folder structure, we can add our first image asset to the project. For this example, we will use the MonoGame logo. Perform the following:

1. First, download the MonoGame logo by right-clicking the following image and saving it as `logo.png` somewhere on your computer:

Figure 5-4: MonoGame Horizontal Logo

2. In the MGCB Editor, select the `images` folder you created earlier.

3. Right-click it and choose `Add > Existing Item...` from the context menu.

4. Navigate to the location of the `logo.png` file you just downloaded and select it.

5. Click the `Open` button

6. When prompted in the add existing file popup, choose `Copy the file to the directory.`

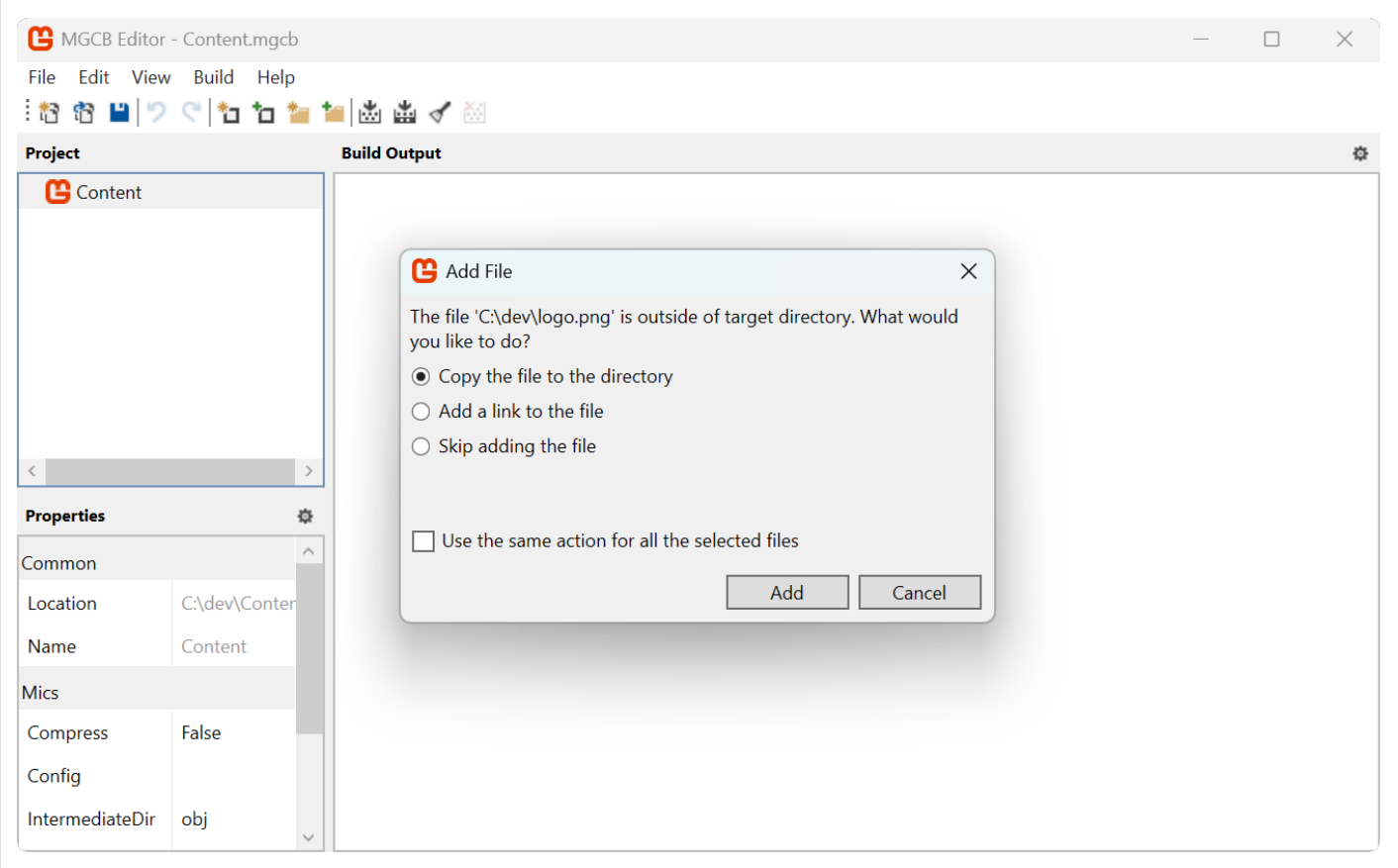


Figure 5-5: Add existing file pop-up

7. Save the changes to the content project by selecting `File > Save` from the top menu or pressing `CTRL+S`.

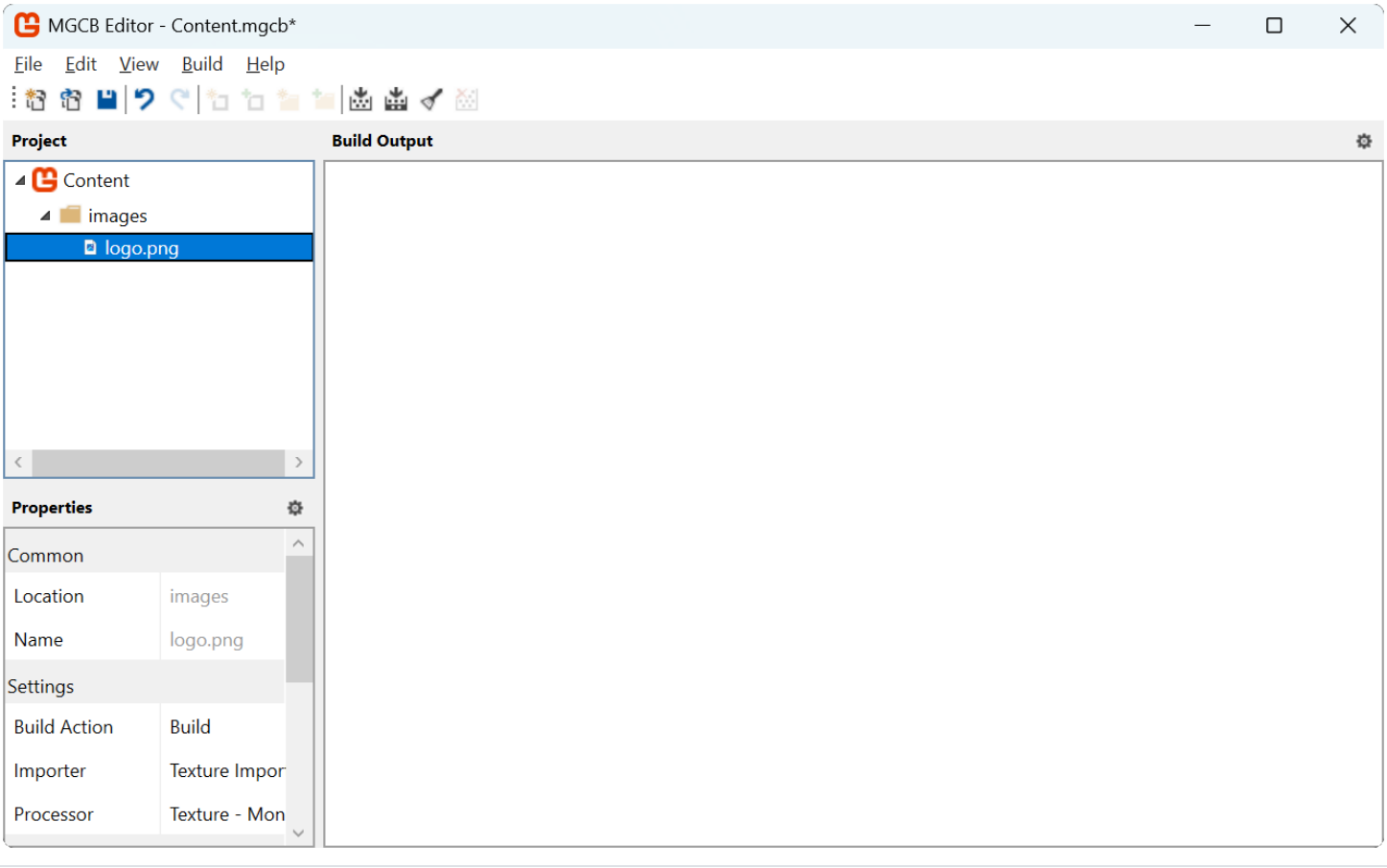


Figure 5-6: The logo image added to the content project in the MGCB Editor

> **IMPORTANT**
>
> After changes have been made in the MGCB Editor, ensure that you save the changes. They are not automatically saved, though you will be warned if you close the editor and have not saved changes. You can tell that changes have not been saved by looking at the title bar of the MGCB editor window. If it has an '*' at the end of the window title, this means changes have not been saved.

Understanding the Content Pipeline Workflow

Now that we have added our first asset, we can take a moment to understand what happens to this asset in the Content Pipeline workflow:

1. You create source files for your game assets such as images, audio, fonts, effects, and 3D models.
2. Using the MGCB Editor, add these assets your content project (the `Content.mgcb` file).
3. When you perform a build of your project, the `MonoGame.Content.Builder.Task` NuGet reference will:
    1. Compile the assets defined in the content project using the **MonoGame Content Builder (MGCB)** tool into `.xnb` files.
    2. Copy the compiled `.xnb` files from the content project's build folder to your game project's build folder.
4. At runtime, you load the compiled assets using the ContentManager.

The following diagram demonstrates this workflow:

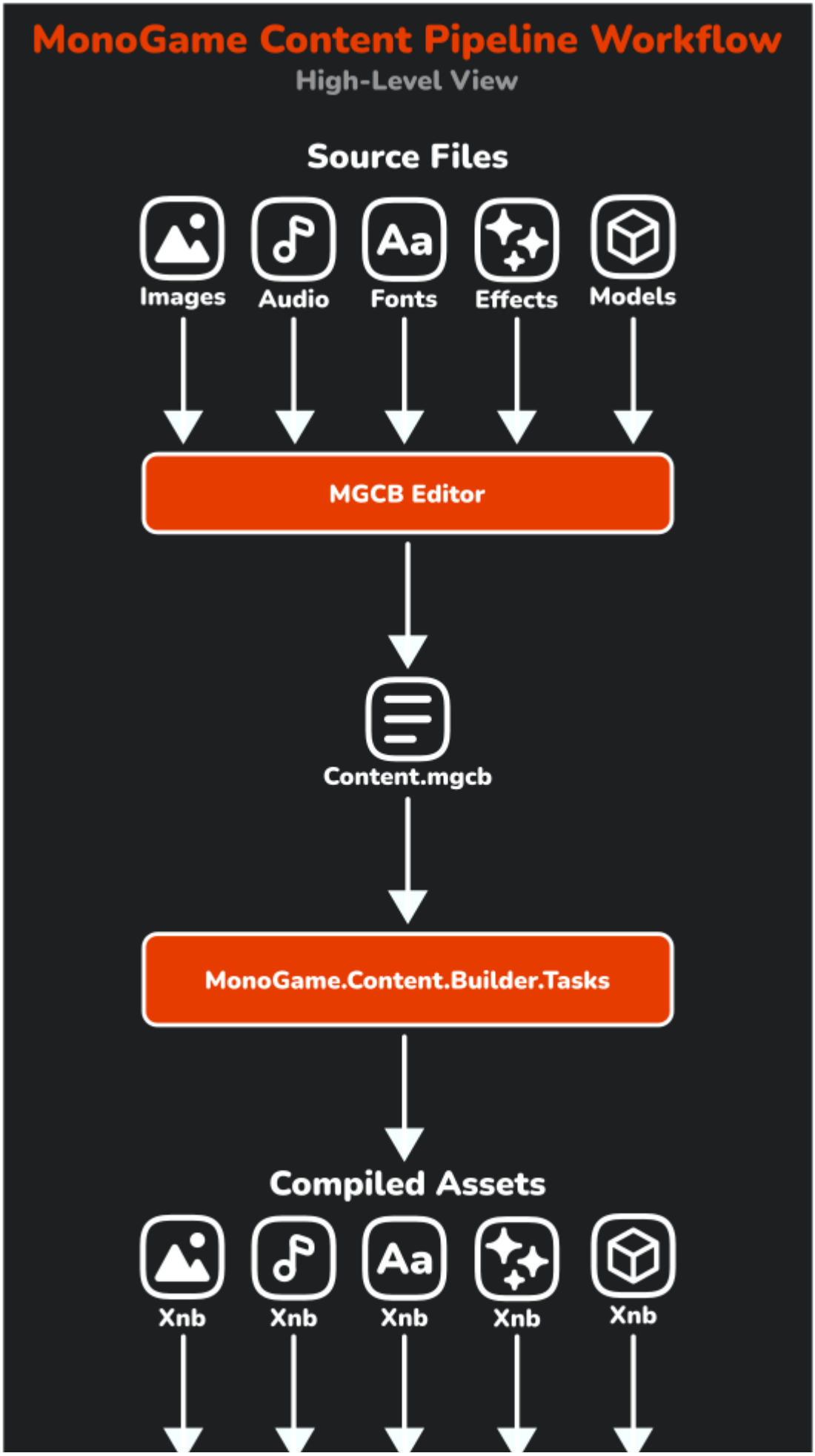

MonoGame Content Pipeline Workflow
High-Level View
Source Files
Images
Audio
Fonts
Effects
Models
MGCB Editor
Content.mgcb
MonoGame.Content.Builder.Tasks
Compiled Assets
Xnb
Xnb
Xnb
Xnb
Xnb

Figure 5-7: MonoGame Content Pipeline Workflow diagram showing the process flow from source files (Images, Audio, Fonts, Effects, Models) through the MGCB Editor to generate the Content.mgcb file, which is then processed by MonoGame.Content.Builder.Tasks to create compiled .xnb assets (Xnb formats for each type), which are finally loaded by the ContentManager at runtime

The Content Pipeline offers significant advantages:

- Assets are pre-processed and optimized for your target platform.
- Image files can be compiled using formats like DXT compression, which GPU's understand natively.
- Asset loading is simplified and consistent across platforms.

The ContentManager Class

To load assets in your game code, MonoGame provides the **ContentManager** class. The **Game** already has a **Content** property which is a ready-to-use instance of the **ContentManager**

ContentManager Methods

The key methods for asset loading are:

Method	Returns	Description
Load<T> (string)	`T`	Loads the asset of type `T` that has been processed by the content pipeline.
Unload	`void`	Unloads all assets that have been loaded by that content manager instance.

> **TIP**
>
> When an asset is loaded for the first time, the **ContentManager** internally caches it. Loading the same asset again will return the cached version, avoiding extra disk reads.

Understanding Content Paths

When loading content, you need to specify the path to the asset, minus the extension. This path is relative to the ContentManager's **RootDirectory** property, which is set to **"Content"** by default in the `Game1` constructor.

For example, with our newly added logo in the "images" folder,the path would be "images/logo" (without the file extension). The reason for this relates to the build process. When you build your project, the *MonoGame.Content.Builder.Tasks* NuGet reference compiles your assets and copies them to the game's output folder.

This creates a folder structure in your output directory similar to:

```
DungeonSlime/
    └── bin/
        └── Debug/
            └── net8.0/
                ├── DungeonSlime.exe
                └── Content/
                    └── images/
                        └── logo.xnb
```

> **NOTE**
>
> Notice that the compiled asset has an .xnb extension, but when loading the asset in code, you refer to it without any extension.

Loading and Displaying Your First Asset

Now that we have the MonoGame logo added as an asset in the content project, we can modify the game to display the logo. In the *DungeonSlime* project, open the `Game1.cs` file and perform the following:

1. Add a field to store the logo texture by inserting this line after the class declaration:

```
// The MonoGame logo texture
private Texture2D _logo;
```

2. In the **LoadContent** method, add this line to load the logo texture:

```
_logo = Content.Load<Texture2D>("images/logo");
```

3. Finally, in the **Draw** method, add these lines before the `base.Draw(gameTime);` call:

```
// Begin the sprite batch to prepare for rendering.
SpriteBatch.Begin();

// Draw the logo texture
SpriteBatch.Draw(_logo, Vector2.Zero, Color.White);

// Always end the sprite batch when finished.
SpriteBatch.End();
```

> **NOTE**
>
> We will go into more details about the **SpriteBatch** in the next chapter.

The complete updated `Game1.cs` file should now look like this:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;

namespace DungeonSlime;

public class Game1 : Core
{
    // The MonoGame logo texture
    private Texture2D _logo;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        _logo = Content.Load<Texture2D>("images/logo");
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // TODO: Add your update logic here

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin();
```

```
            // Draw the logo texture
            SpriteBatch.Draw(_logo, Vector2.Zero, Color.White);

            // Always end the sprite batch when finished.
            SpriteBatch.End();

            base.Draw(gameTime);
        }
    }
```

Running the game now will show the MonoGame logo displayed in the upper-left corner of the game window.

Figure 5-8: The MonoGame logo drawn to the game window

Adding Built-In Asset Types

The MGCB Editor can also create certain built-in asset types. In this section we will explore these types and this functionality. If not already open, open the MGCB Editor and perform the following:

1. Select the `Content` node.
2. Right-click it and choose `Add > New Item...` from the context menu.
3. In the dialog that appears, you will see the available built-in types.

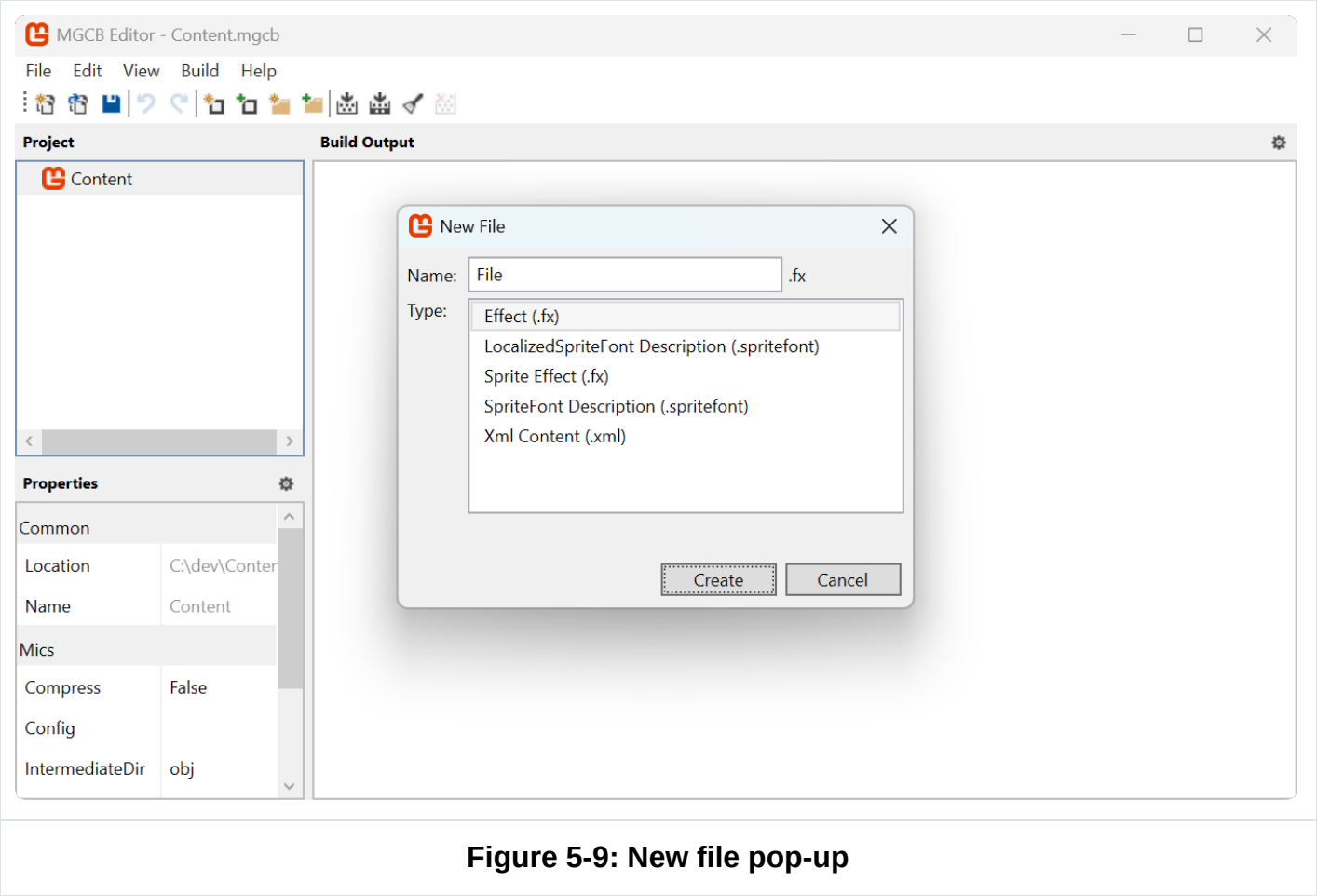


Figure 5-9: New file pop-up

The available built-in types include:

- **Effect (.fx)**: A shader file that creates custom visual effects by controlling how graphics are rendered on the GPU.
- **SpriteFont Description (.spritefont)**: A configuration file that defines how text will be displayed in your game, including character set and font properties.
- **Sprite Effect (.fx)**: A shader specifically designed for use with 2D sprites to create special visual effects.
- **Xml Content (.xml)**: A structured data file for storing game information like levels, dialogues, or configuration settings.
- **LocalizedSpriteFont Description (.spritefont)**: A configuration file for creating fonts with support for multiple languages.

> **NOTE**
>
> Each built-in asset type comes with a template that includes the minimum required structure and settings.

For now, click the `Cancel` button on the new file dialog. We will explore these built-in types further in later chapters when we need them.

Conclusion

In this chapter, you accomplished the following:

- You learned about the advantages of loading assets using the **Content Pipeline**.
- You opened the MGCB Editor and explored its interface.
- You created a folder structure to organize your game assets.
- You added an image file asset to the content project.
- You understood the Content Pipeline workflow and how MonoGame automates the process.
- You loaded and displayed your first asset using the **ContentManager**.

In the next chapter, we will explore working with textures in more detail and learn about different rendering options.

Test Your Knowledge

1. What are the two main ways of loading a texture, and what are the pros and cons of each approach?

   The two main ways to load a texture in MonoGame are:

   1. Directly from file using **Texture2D.FromFile**. This method requires manually setting up file copying, offers no pre-processing benefits, and can have a higher memory footprint.
   2. Using the content pipeline with **Content.Load<Texture2D>**. Using the content pipeline optimizes textures into formats for the target platform(s), automatically handles compiling and copying assets during build, and reduces memory footprint, but requires additional setup using the MGCB Editor.

2. During the MonoGame content pipeline workflow, assets are compiled and then copied to the project output folder. What is responsible for performing this task?

   The *MonoGame.Content.Builder.Tasks* NuGet reference.

Chapter 06: Working with Textures

Learn how to load and render textures using the MonoGame content pipeline and SpriteBatch.

Textures are images that are used in your game to represent the visual graphics to the player, commonly referred to as *Sprites*. In Chapter 05, you learned how to use the **Content Pipeline** to load the MonoGame *logo.png* texture and render it to the screen.

In this chapter, you will:

- Learn how to render a texture with the **SpriteBatch**.
- Explore how to manipulate the way the texture is rendered using the parameters of the **SpriteBatch.Draw** method.

Chapter Video

Live Recording

Drawing a Texture

When rendering in MonoGame, *render states*, properties of the **GraphicsDevice** that affect how rendering is performed, need to be set. When rendering 2D sprites, the **SpriteBatch** class simplifies rendering by managing these render states for you.

> **IMPORTANT**
>
> Although the **SpriteBatch** makes it easier to manage the render states for the **GraphicsDevice**, it can also change states that you may have set manually, such as when you are performing 3D rendering. Keep this in mind when mixing 2D and 3D rendering.

Three methods are used when rendering with the **SpriteBatch**:

1. **SpriteBatch.Begin** prepares the Graphics Device for rendering, including the render states.
2. **SpriteBatch.Draw** tells the **SpriteBatch** what to render. This is usually called multiple times before **SpriteBatch.End** and batches the draw calls for efficiency.
3. **SpriteBatch.End** submits the draw calls that were batched to the graphics device to be rendered.

> **NOTE**
>
> The order of method calls when rendering using the **SpriteBatch** is important. **SpriteBatch.Begin** must be called before any **SpriteBatch.Draw** calls are made. When finished, **SpriteBatch.End** must be called before another **SpriteBatch.Begin** can be called. If these methods are called out of order, an exception will be thrown.

As mentioned in Chapter 03, all rendering should be done inside the **Draw** method. The **Draw** method's responsibility is to render the game state that was calculated in **Update**; it should not contain any game logic or complex calculations.

At the end of Chapter 05, you added the following code to the **Draw** method inside the `Game1.cs` file:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(_logo, Vector2.Zero, Color.White);

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

These lines initialize the **SpriteBatch**, draw the logo at **Vector2.Zero** (0, 0), and submit the batch. When you ran the game, the logo appeared in the upper-left corner of the window:

Figure 6-1: The MonoGame logo drawn to the game window

The **SpriteBatch.Draw** method we just used can be given the following parameters:

Parameter	Type	Description
texture	Texture2D	The Texture2D to draw.
position	Vector2	The X and Y coordinates at which the texture will be rendered, with the texture's origin being the upper-left corner of the image.
color	Color	The color mask (tint) to apply to the image drawn. Specifying Color.White will render the texture with no tint.

> **TIP**
>
> Try adjusting the position and color parameters and see how they affect the image being drawn.

MonoGame uses a coordinate system where (0, 0) is at the screen's upper-left corner. X values increase moving right, and Y values increase moving down. Understanding this, we will try to center the logo on the game window.

To center content on the screen, we need to find the window's center point. We can access this using the **Window.ClientBounds** property from the **Game** class, which represents the rectangular bounds of the game window. **Window.ClientBounds** exposes both **Width** and **Height** properties for the window's dimensions in pixels. By dividing these dimensions in half, we can calculate the window's center coordinates. We can update our **Draw** method to use this:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the logo texture.
    SpriteBatch.Draw(
        _logo,          // texture
        new Vector2(    // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height)
            * 0.5f,
        Color.White     // color
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

> **TIP**
>
> In the example above, we multiply the **Vector2** created by `0.5f` to halve the values instead of dividing it by `2.0f`. If you are not used to seeing this, it might seem strange at first, but it is actually an optimization technique. CPUs are able to perform multiplication operations much faster than division operations and reading `* 0.5f` is easily understood to be the same thing as `/ 2.0f`.

We have now set the position to half the window's dimensions, which should center the logo. Run the game to see the result.

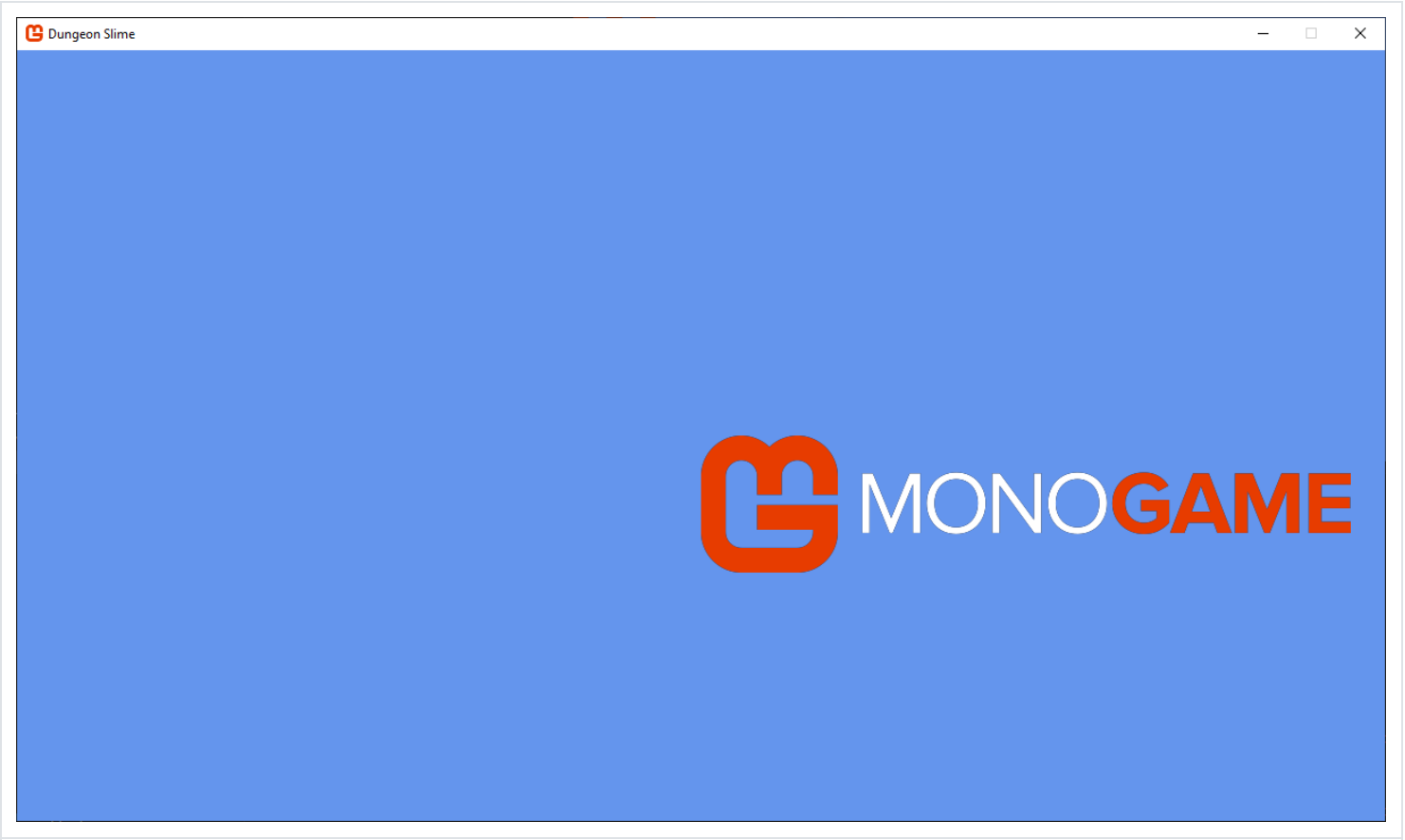


Figure 6-2: Attempting to draw the MonoGame logo centered on the game window

The logo is not centered as we expected it to be. Even though we set the *position* parameter to the center of the game window, the texture starts drawing from its *origin*, which is the upper-left corner in this example. So when we set the position to the screen's center, we are actually placing the logo's upper-left corner at that point, not the center of the texture.

One way to solve this is to subtract half the width and height of the texture from the center position of the game window, as shown below:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the logo texture.
    SpriteBatch.Draw(
        _logo,          // texture
        new Vector2(    // position
            (Window.ClientBounds.Width * 0.5f) - (_logo.Width * 0.5f),
            (Window.ClientBounds.Height * 0.5f) - (_logo.Height * 0.5f)),
        Color.White     // color
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

This offsets the position so that it correctly centers the image inside the game window.

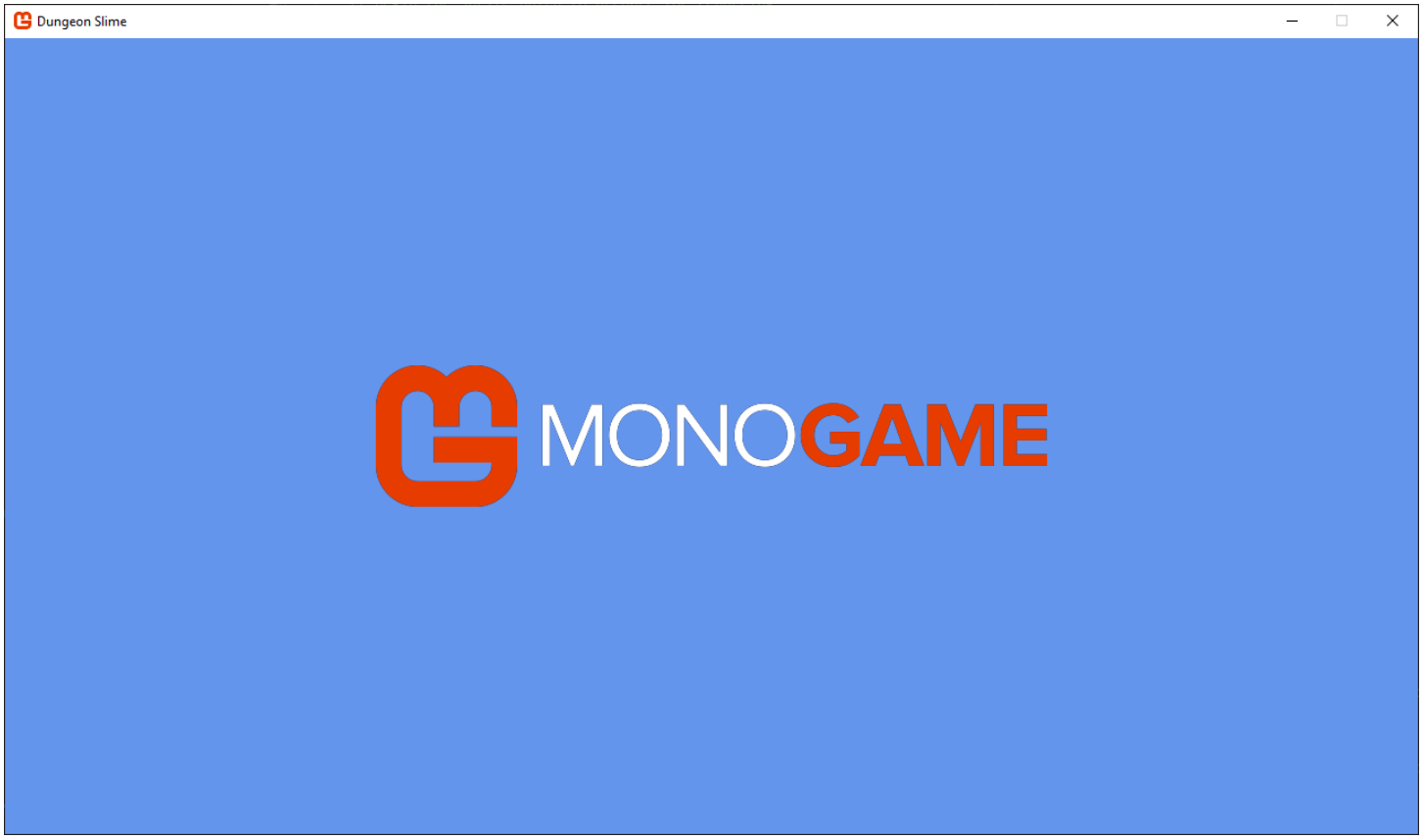


Figure 6-3: The MonoGame logo drawn centered on the game window

While this works, there is a better approach. There is a different overload of the **SpriteBatch.Draw** method that provides additional parameters for complete control over the draw operation, which we will use in the upcoming sections.

Update your code to:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            (Window.ClientBounds.Width * 0.5f) - (_logo.Width * 0.5f),
            (Window.ClientBounds.Height * 0.5f) - (_logo.Height * 0.5f)),
        null,               // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        Vector2.Zero,       // origin
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

This overload produces the same centered result but exposes all parameters that control rendering for a draw operation. Unlike engines that abstract much of these details away, MonoGame provides explicit control for a flexible custom rendering pipeline. Here is what each parameter does:

Parameter	Type	Description
texture	Texture2D	The Texture2D to draw.
position	Vector2	The X and Y position at which the texture will be rendered, relative to the *origin* parameter.
sourceRectangle	Rectangle	An optional region within the texture to be rendered in order to draw only a portion of the texture. Specifying `null` will render the entire texture.
color	Color	The color mask (tint) to apply to the image drawn. Specifying Color.White will render the texture with no tint.
rotation	`float`	The amount of rotation, in radians, to apply to the texture when rendering. Specifying `0.0f` will render the image with no rotation.
origin	Vector2	The X and Y coordinates defining the position of the texture origin. This affects the texture's offset during rendering and serves as a reference point around which the texture is rotated and scaled.
scale	`float`	The scale factor applied to the image across the x- and y-axes. Specifying `1.0f` will render the image at its original size with no scaling.
effects	SpriteEffects	A SpriteEffects enum value that specifies if the texture should be rendered flipped across the horizontal axis, the vertical axis, or both axes.
layerDepth	`float`	Specifies the depth at which the texture is rendered. Textures with a higher layer depth value are drawn on top of those with a lower layer depth value. **Note: This value will only apply when using `SpriteSortMode.FrontToBack` or `SpriteSortMode.BackToFront`. We will cover this in a moment.**

Rotation

First we will explore the `rotation` parameter. This value is the amount of rotation to apply to the sprite when rendering it. We will rotate the texture 90° to make it vertical. Since rotation is measured in radians, not degrees, we can use the built-in math library in MonoGame to make the conversion for us by calling MathHelper.ToRadians. Update the code to:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,                      // texture
        new Vector2(                // position
            (Window.ClientBounds.Width * 0.5f) - (_logo.Width * 0.5f),
            (Window.ClientBounds.Height * 0.5f) - (_logo.Height * 0.5f)),
        null,                       // sourceRectangle
        Color.White,                // color
        MathHelper.ToRadians(90),   // rotation
        Vector2.Zero,               // origin
        1.0f,                       // scale
        SpriteEffects.None,         // effects
        0.0f                        // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Running the code now shows the rotated image, but not in the expected position:

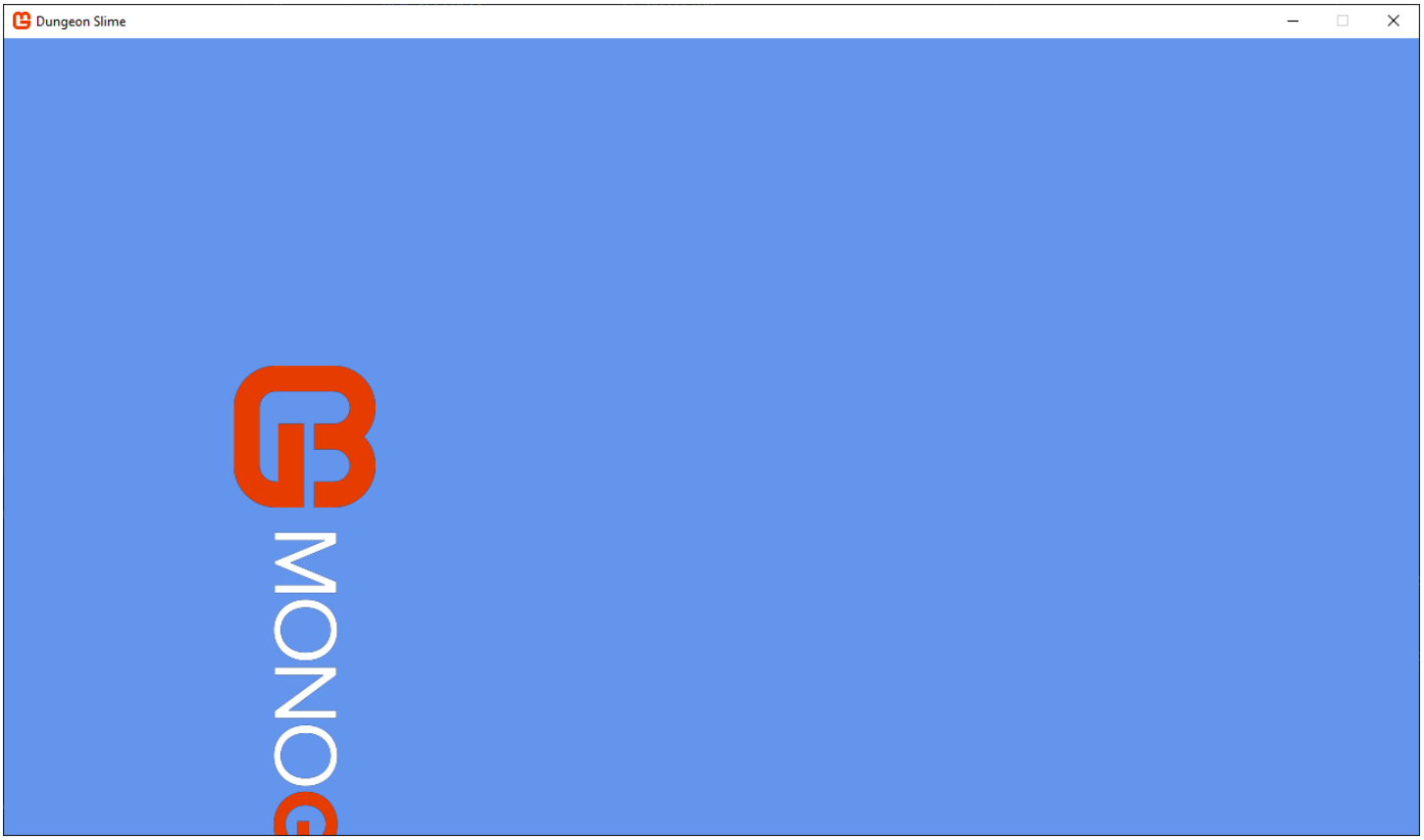


Figure 6-4: Attempting to draw the MonoGame logo rotated 90° and centered on the game window

The reason the sprite did not rotate as expected is because of the `origin` parameter.

Origin

The `origin` parameter determines the reference point from which the sprite is rendered, rotated, and scaled. By default, if no origin is set, it will be **Vector2.Zero**, the upper-left corner of the sprite. To visualize this, see *Figure 6-5* below. The red square represents where the origin is for the sprite, and we can see how it is rotated around this origin point.

Figure 6-5: Demonstration of how a sprite is rotated around its origin

To resolve this rotation issue, we need to need to change two things:

1. Set the `origin` parameter to the center of the sprite instead of defaulting to the upper-left corner.
2. Change the `position` parameter back to the center of the screen.

Update the code to:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,                      // texture
        new Vector2(                // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,                       // sourceRectangle
        Color.White,                // color
        MathHelper.ToRadians(90),   // rotation
        new Vector2(                // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        1.0f,                       // scale
        SpriteEffects.None,         // effects
        0.0f                        // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Moving the sprite's origin to its center not only solves the rotation problem, but also allows us to use the screen's center position directly, without the need for additional offset calculations. Running the game now shows the logo properly centered and rotated by 90°.

> **NOTE**
>
> The `origin` parameter is based on the sprite's width and height, so the center will be half the width and half the height of the sprite.

Figure 6-6: The MonoGame logo drawn rotated 90° and centered on the game window

Scale

The `scale` parameter specifies the amount of scaling to apply to the sprite when it is rendered. The default value is `1.0f`, which can be read as "rendering the sprite at 1x the size". Increasing this will scale up the size of the sprite and decreasing it will scale down the sprite.

For this example, we will first reset the rotation back to `0.0f` (removing the 90° rotation we applied above) so we can clearly see the scaling effect. Then we will set the scale of the logo sprite to `1.5f`

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
       _logo,              // texture
       new Vector2(        // position
           Window.ClientBounds.Width,
           Window.ClientBounds.Height) * 0.5f,
       null,               // sourceRectangle
       Color.White,        // color
       0.0f,               // rotation
       new Vector2(        // origin
           _logo.Width,
           _logo.Height) * 0.5f,
       1.5f,               // scale
       SpriteEffects.None, // effects
       0.0f                // layerDepth
   );

   // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

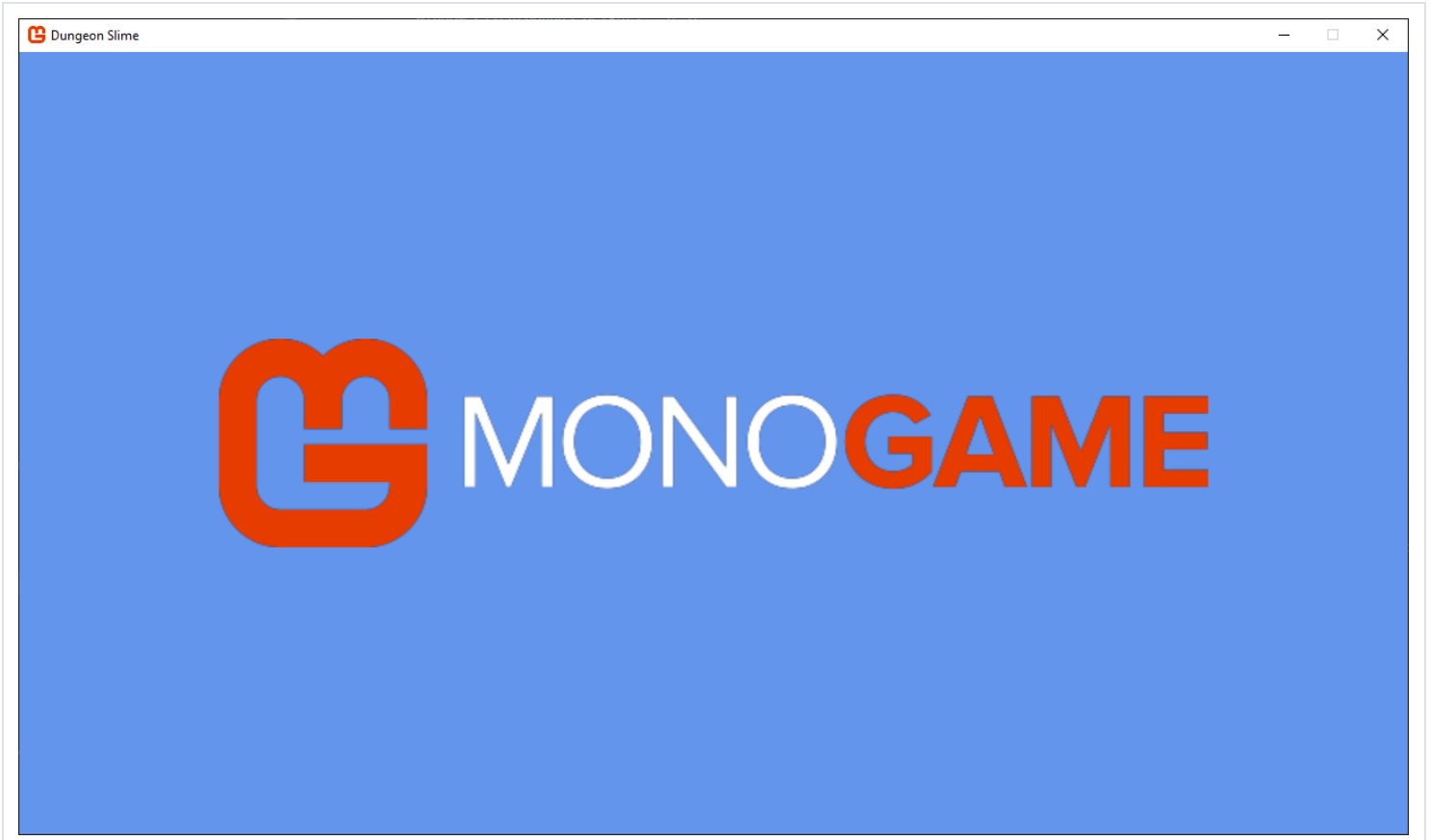


Figure 6-7: The MonoGame logo drawn scaled at 1.5x the size

Note that the sprite is scaled up from the center. This is because the `origin` parameter is still set to the center of the sprite. If we adjust the code so that the origin parameter is back in the upper-left corner, as in the listing below:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,               // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        Vector2.Zero,       // origin
        1.5f,               // scale
        SpriteEffects.None, // effects
        0.0f                //layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Then the scaling is applied from the origin in the upper-left corner, producing the following result:

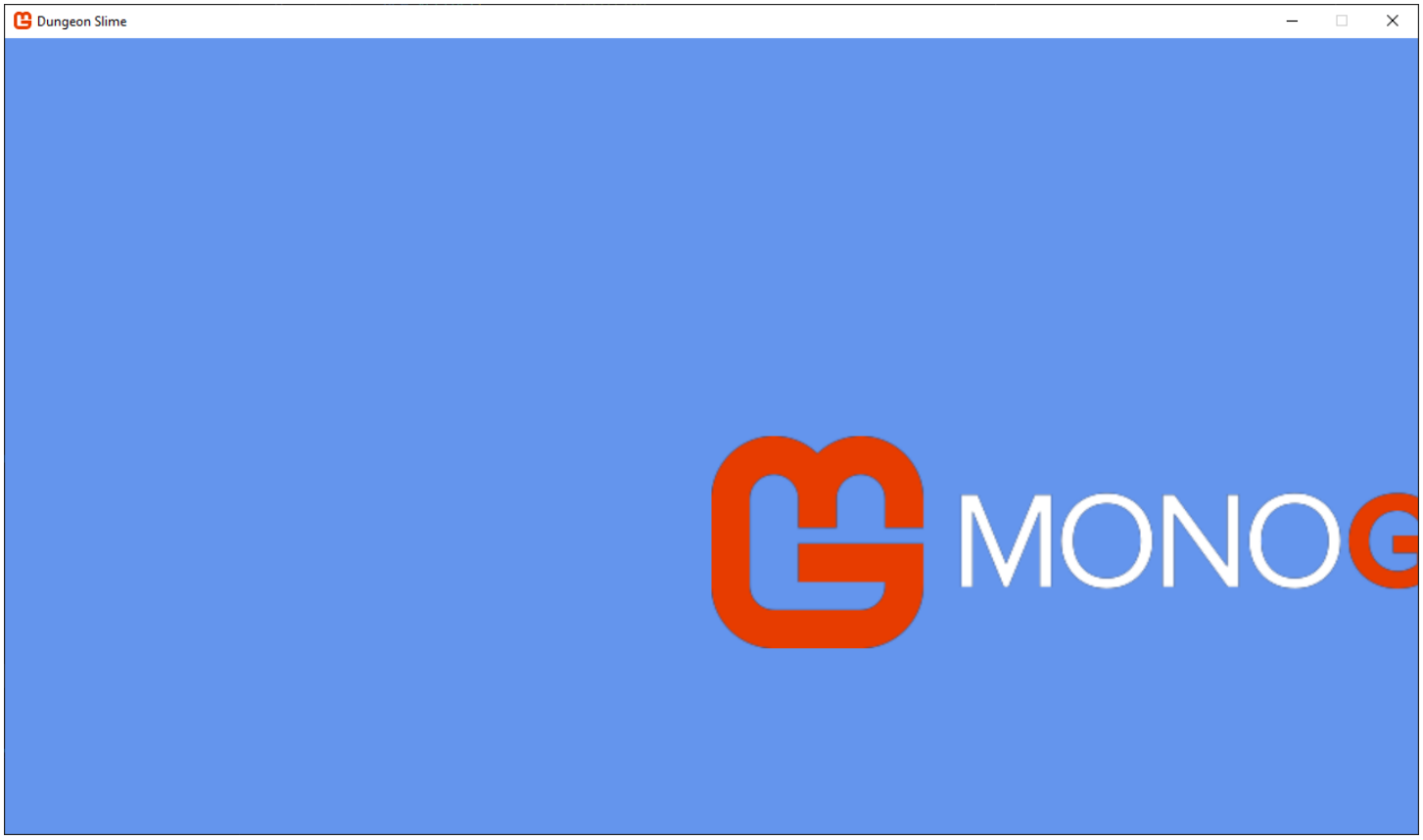


Figure 6-8: The MonoGame logo drawn with a scale factor of 1.5x and the origin set in the upper-left corner

Scaling can also be applied to the x- and y-axes independently by providing a **Vector2** value instead of a float value. For instance, we can scale the x-axis of the sprite by 1.5x and reduce the scale of the y-axis to 0.5x:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,                      // texture
        new Vector2(                // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,                       // sourceRectangle
        Color.White,                // color
        0.0f,                       // rotation
        new Vector2(                // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        new Vector2(1.5f, 0.5f),    // scale
        SpriteEffects.None,         // effects
        0.0f                        // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Which will produce the following result:

Figure 6-9: The MonoGame logo drawn scaled at 1.5x the size on the x-axis and 0.5x on the y-axis

SpriteEffects

The `effects` parameter is used to flip the sprite when rendered on either the horizontal axis, vertical axis, or both. The value for this parameter will be one of the **SpriteEffects** enum values.

SpriteEffect	Description
SpriteEffects.None	No effect is applied and the sprite is rendered normally.
SpriteEffects.FlipHorizontally	The sprite is rendered flipped along the horizontal axis.
SpriteEffects.FlipVertically	The sprite is rendered flipped along the vertical axis.

For this example, we will reset the scale back to `1.0f` and apply the **SpriteEffects.FlipHorizontally** value to the sprite:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,                          // texture
        new Vector2(                    // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,                           // sourceRectangle
        Color.White,                    // color
        0.0f,                           // rotation
        new Vector2(                    // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        1.0f,                           // scale
        SpriteEffects.FlipHorizontally, // effects
        0.0f                            // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Which will produce the following result:

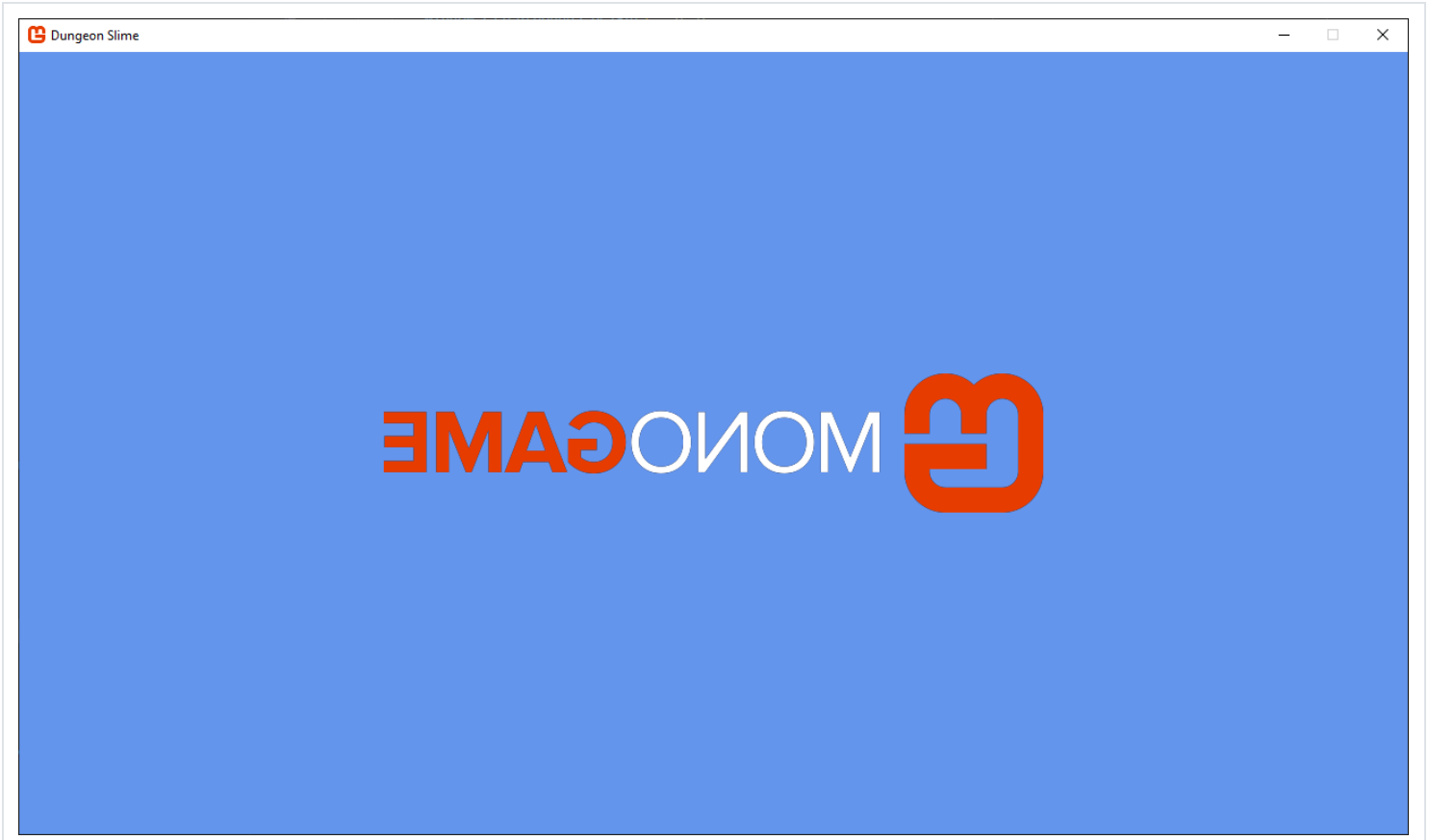


Figure 6-10: The MonoGame logo flipped horizontally

The **SpriteEffects** enum value also uses the `[Flag]` attribute, which means we can combine both horizontal and vertical flipping together. To do this, we use the bitwise OR operator `|`. Update the `effect` parameter value to the following:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,                              // texture
        new Vector2(                        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,                               // sourceRectangle
        Color.White,                        // color
        0.0f,                               // rotation
        new Vector2(                        // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        1.0f,                               // scale
        SpriteEffects.FlipHorizontally |    // effects
        SpriteEffects.FlipVertically,
        0.0f                                // layerDepth
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Now the sprite is flipped both horizontally and vertically

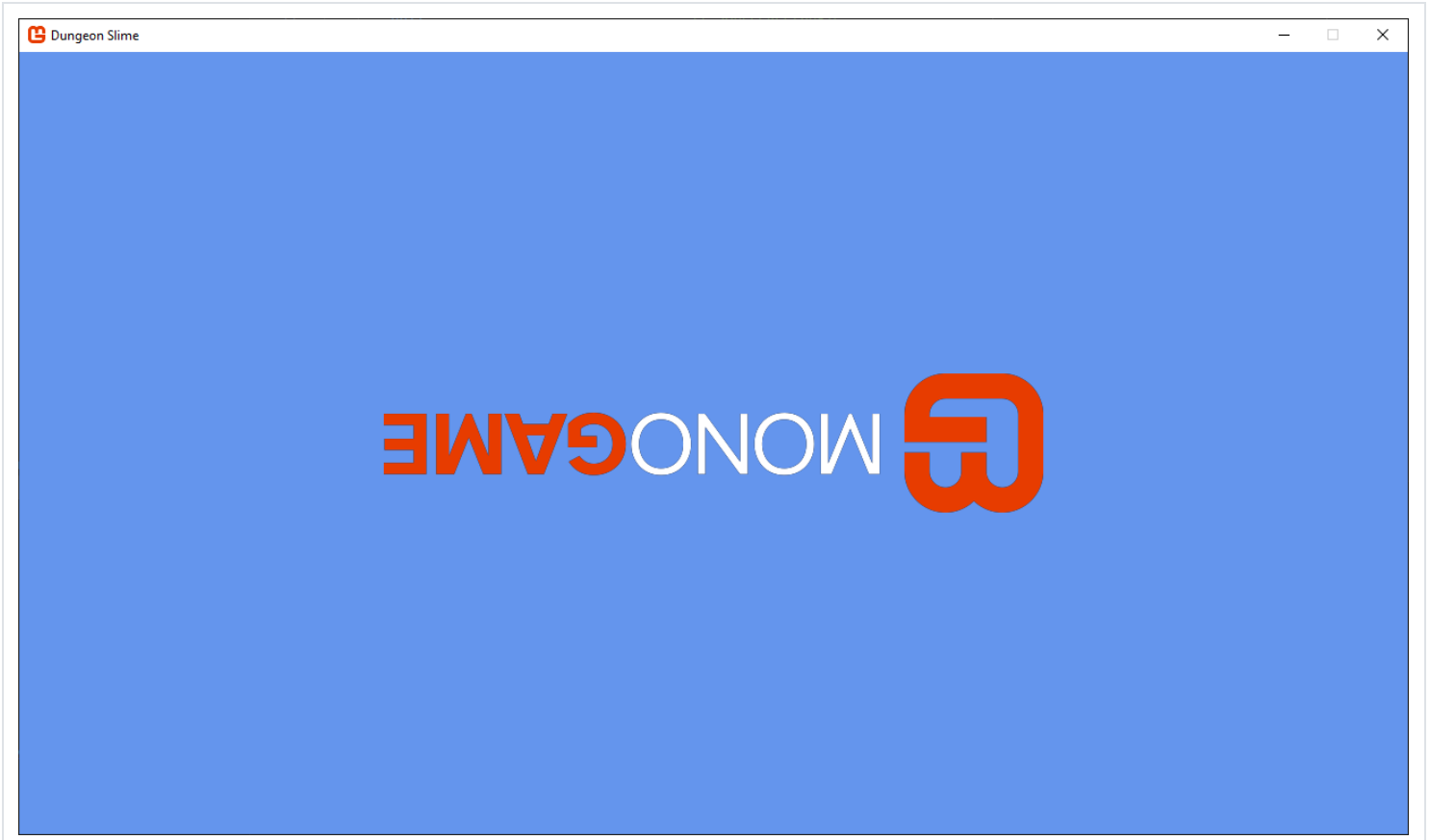


Figure 6-11: The MonoGame logo flipped horizontally and vertically

Color and Opacity

The `color` parameter applies a color mask to the sprite when it is rendered. Note that this is not setting the actual color of the image, just a mask that is applied on top of the image. The default value is **Color.White**. So if we are setting it to **Color.White**, why does this not affect the tinting of the sprite drawn?

When the `color` parameter is applied, each color channel (Red, Green, Blue) of the sprite is multiplied by the corresponding channel in the `color` parameter, where each channel is represented as a value between `0.0f` and `1.0f`. For **Color.White**, all color channels are set to `1.0f` (255 in byte form), so the multiplication looks like this:

```
Final Red = Sprite Red * 1.0f
Final Green = Sprite Green * 1.0f
Final Blue = Sprite Blue * 1.0f;
```

Since multiplying by `1.0f` does not change the value, **Color.White** essentially preserves the original colors of the sprite.

For this example, we will reset the `effects` parameter back to **SpriteEffects.None** and update the `color` parameter to use **Color.Green**:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,               // sourceRectangle
        Color.Green,        // color
        0.0f,               // rotation
        new Vector2(        // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

This produces the following result:

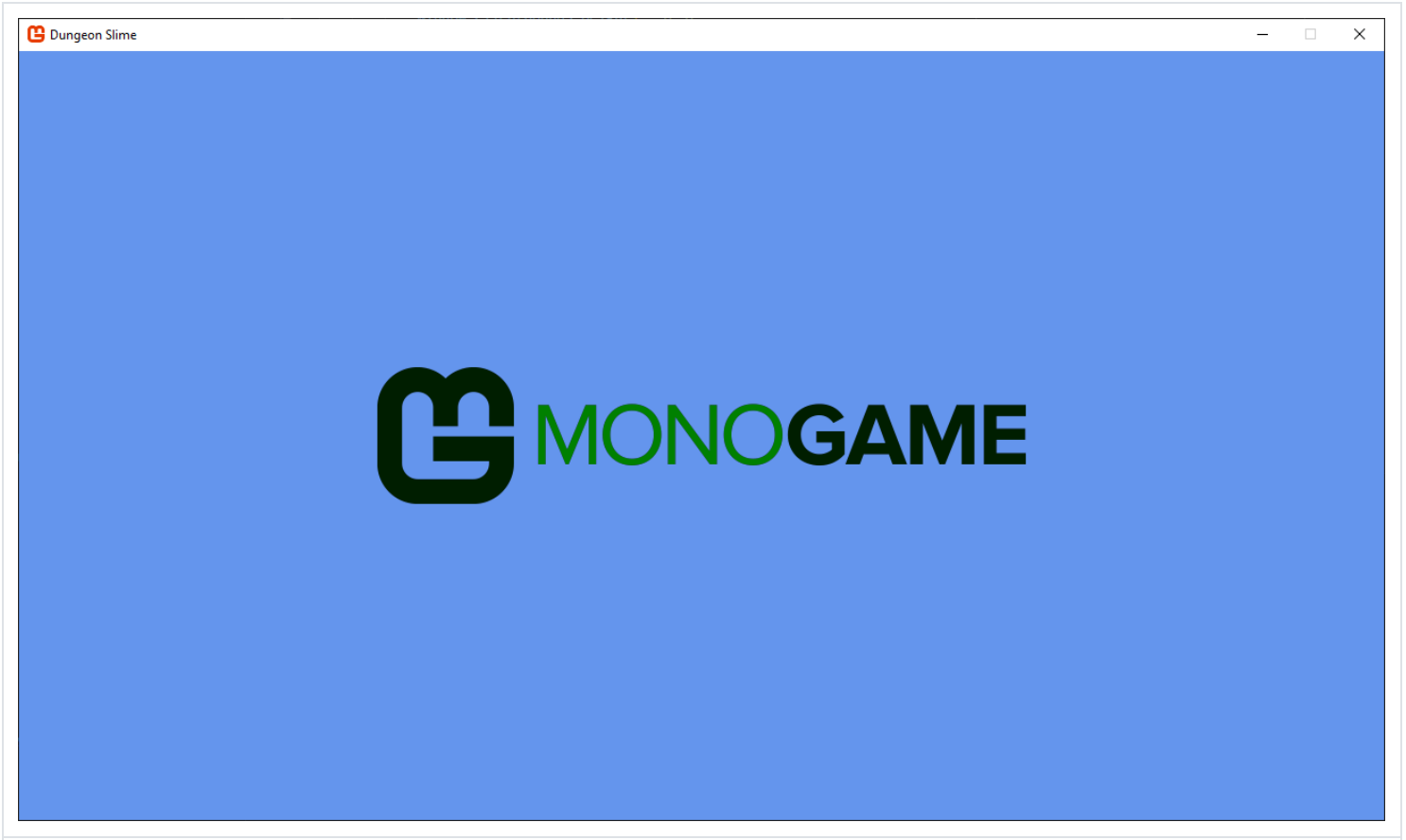


Figure 6-12: The MonoGame logo with a green color tint applied

> **NOTE**
>
> The icon and the word "GAME" in the logo look black after using **Color.Green** because the Red, Blue and Green components of that color are (`0.0f`, `0.5f`, `0.0f`). The Orange color used in the logo is **Color.MonoGameOrange**, which has the component values of (`0.9f`, `0.23f`, `0.0f`). When multiplying the component values, the result is (`0.0f`, `0.125f`, `0.0f`) which would be Red 0, Green 31, Blue 0 in byte values. So it is not quite fully black, but it is very close.
>
> This is why it is important to understand how the `color` parameter values are applied to the sprite when it is rendered.

To adjust the opacity of a sprite, we can multiply the `color` parameter value by a value between `0.0f` (fully transparent) and `1.0f` (fully opaque). For instance, if we wanted to render the logo with 50% transparency we can multiply the `color` parameter by `0.5f` like this:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        null,               // sourceRectangle
        Color.White * 0.5f, // color
        0.0f,               // rotation
        new Vector2(        // origin
            _logo.Width,
            _logo.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f
    );

    // Always end the sprite batch when finished.
    SpriteBatch.End();

    base.Draw(gameTime);
}
```

Which will produce the following result:

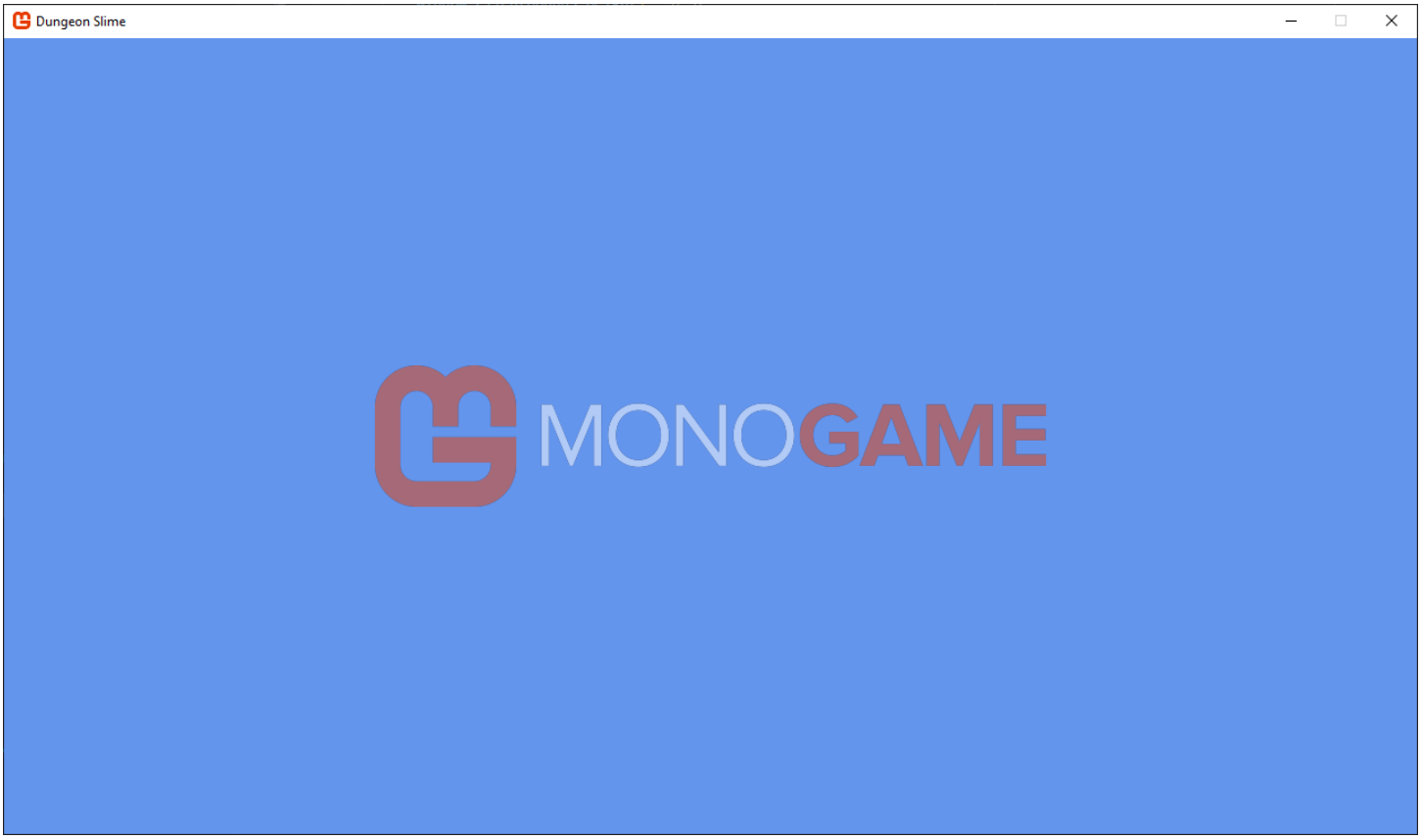


Figure 6-13: The MonoGame logo with half transparency

Source Rectangle

The `sourceRectangle` parameter specifies a region within the texture that should be rendered. So far, we have just set this parameter to `null`, which specifies that the full texture should be rendered. If we only wanted to render a portion of the texture as the sprite, we can set this parameter value.

For instance, take the logo image we have been using. We can break it down into two distinct regions; the MonoGame icon and the MonoGame wordmark.

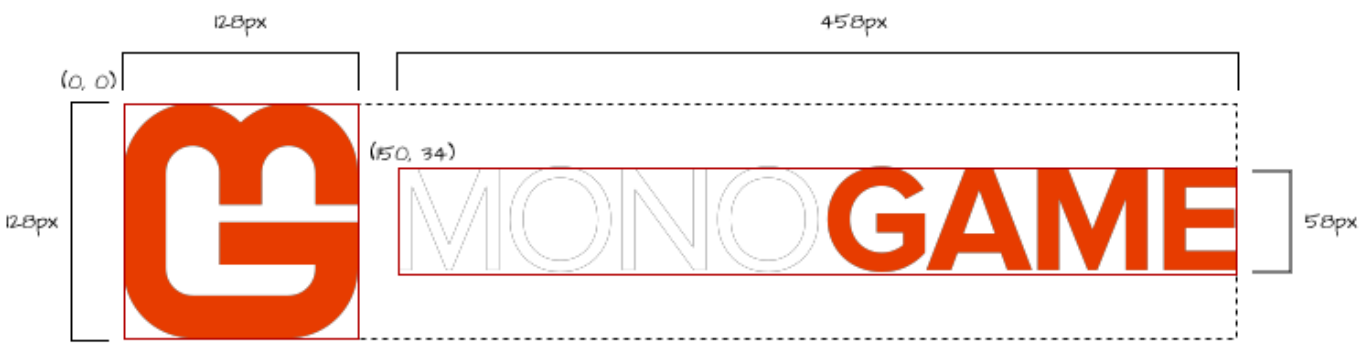


Figure 6-14: The MonoGame logo broken down into the icon and wordmark regions

We can see from *Figure 6-14* above that the actual icon starts at position (0, 0) and is 128px wide and 128px tall. Likewise, the wordmark starts at position (150, 34) and is 458px wide and 58px tall. Knowing the starting position and the width and height of the region gives us a defined rectangle that we can use as the `sourceRectangle`.

We can see this in action by drawing the icon and the wordmark separately from the same texture. Update the code to the following:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // The bounds of the icon within the texture.
    Rectangle iconSourceRect = new Rectangle(0, 0, 128, 128);

    // The bounds of the word mark within the texture.
    Rectangle wordmarkSourceRect = new Rectangle(150, 34, 458, 58);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw only the icon portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        iconSourceRect,     // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
            iconSourceRect.Width,
            iconSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
    );

    // Draw only the word mark portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
          Window.ClientBounds.Width,
          Window.ClientBounds.Height) * 0.5f,
        wordmarkSourceRect, // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
          wordmarkSourceRect.Width,
          wordmarkSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
```

```
        );

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
```

The following changes were made:

- Two new **Rectangle** values called `iconSourceRect` and `wordmarkSourceRect` that represent the boundaries of the MonoGame icon and wordmark regions within the logo texture were added.
- The *sourceRectangle* parameter of the `_spriteBatch.Draw` call was updated to use the new `iconSourceRect` value. **Notice that we are still telling it to draw the `_logo` for the *texture*, we have just supplied it with a source rectangle this time.**
- The *origin* parameter was updated to use the width and height of the `iconSourceRect`. Since the overall dimensions of what we will be rendering has changed due to supplying a source rectangle, the origin needs to be adjusted to those dimensions as well.
- Finally, a second `_spriteBatch.Draw` call is made, this time using the `wordmarkSourceRect` as the source rectangle so that the wordmark is drawn.

If you run the game now, you should see the following:

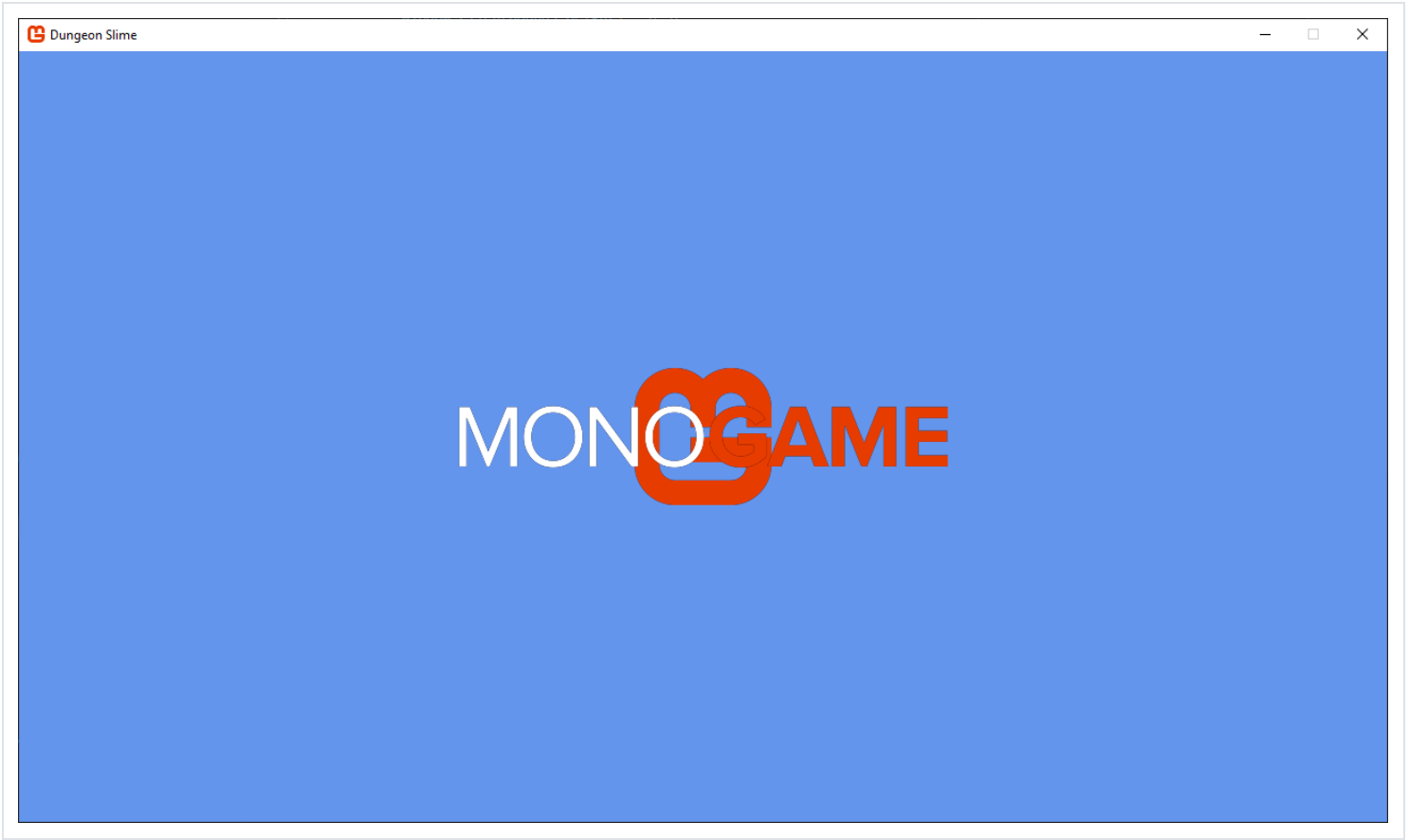


Figure 6-15: The MonoGame icon and wordmark, from the logo texture, centered in the game window

> **NOTE**
>
> Making use of the `sourceRectangle` parameter to draw different sprites from the same texture is an optimization technique that we will explore further in the next chapter.

Layer Depth

The final parameter to discuss is the `layerDepth` parameter. Notice that in *Figure 6-15* above, the word mark is rendered on top of the icon. This is because of the order the draw calls were made; first the icon was rendered, then the word mark was rendered.

The **SpriteBatch.Begin** method contains several optional parameters, one of which is the `sortMode` parameter. By default, this value is **SpriteSortMode.Deferred**, which means what is drawn is done so in the order of the **SpriteBatch.Draw** calls. Each subsequent call will be drawn visually on top of the previous call.

When **SpriteSortMode.Deferred** is used, then the `layerDepth` parameter in the **SpriteBatch.Draw** call is essentially ignored. For instance, in the first `_spriteBatch.Draw` method call, update the `layerDepth` parameter to `1.0f`.

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // The bounds of the icon within the texture.
    Rectangle iconSourceRect = new Rectangle(0, 0, 128, 128);

    // The bounds of the word mark within the texture.
    Rectangle wordmarkSourceRect = new Rectangle(150, 34, 458, 58);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin();

    // Draw only the icon portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        iconSourceRect,     // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
            iconSourceRect.Width,
            iconSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        1.0f                // layerDepth
    );

    // Draw only the word mark portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
          Window.ClientBounds.Width,
          Window.ClientBounds.Height) * 0.5f,
        wordmarkSourceRect, // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
          wordmarkSourceRect.Width,
          wordmarkSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
```

```
        );

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
```

Doing this should tell it to render on a layer above the wordmark since the icon is at `1.0f` and the wordmark is at `0.0f` for the `layerDepth`. However, if you run the game now, you will see that no change actually happens; the wordmark is still drawn on top of the icon.

To make use of the `layerDepth` parameter, you need to set the `sortMode` to either **SpriteSortMode.BackToFront** or **SpriteSortMode.FrontToBack**.

Sort Mode	Description
SpriteSortMode.BackToFront	Sprites are sorted by depth in back-to-front order prior to drawing.
SpriteSortMode.FrontToBack	Sprites are sorted by depth in front-to-back order prior to drawing.

Now we can see this in action. We have already set the `layerDepth` parameter of the icon to `1.0f`. Find the `_spriteBatch.Begin()` method call and update it to the following:

```
protected override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // The bounds of the icon within the texture.
    Rectangle iconSourceRect = new Rectangle(0, 0, 128, 128);

    // The bounds of the word mark within the texture.
    Rectangle wordmarkSourceRect = new Rectangle(150, 34, 458, 58);

    // Begin the sprite batch to prepare for rendering.
    SpriteBatch.Begin(sortMode: SpriteSortMode.FrontToBack);

    // Draw only the icon portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
            Window.ClientBounds.Width,
            Window.ClientBounds.Height) * 0.5f,
        iconSourceRect,     // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
            iconSourceRect.Width,
            iconSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        1.0f                // layerDepth
    );

    // Draw only the word mark portion of the texture.
    SpriteBatch.Draw(
        _logo,              // texture
        new Vector2(        // position
          Window.ClientBounds.Width,
          Window.ClientBounds.Height) * 0.5f,
        wordmarkSourceRect, // sourceRectangle
        Color.White,        // color
        0.0f,               // rotation
        new Vector2(        // origin
          wordmarkSourceRect.Width,
          wordmarkSourceRect.Height) * 0.5f,
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
```

```
        );

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
```

Now we are telling it to use the **SpriteSortMode.FrontToBack** sort mode, which will sort the draw calls so that those with a higher `layerDepth` will be drawn on top of those with a lower one. Even though we did not change the order of the `_spriteBatch.Draw` calls, if you run the game now, you will see the following:

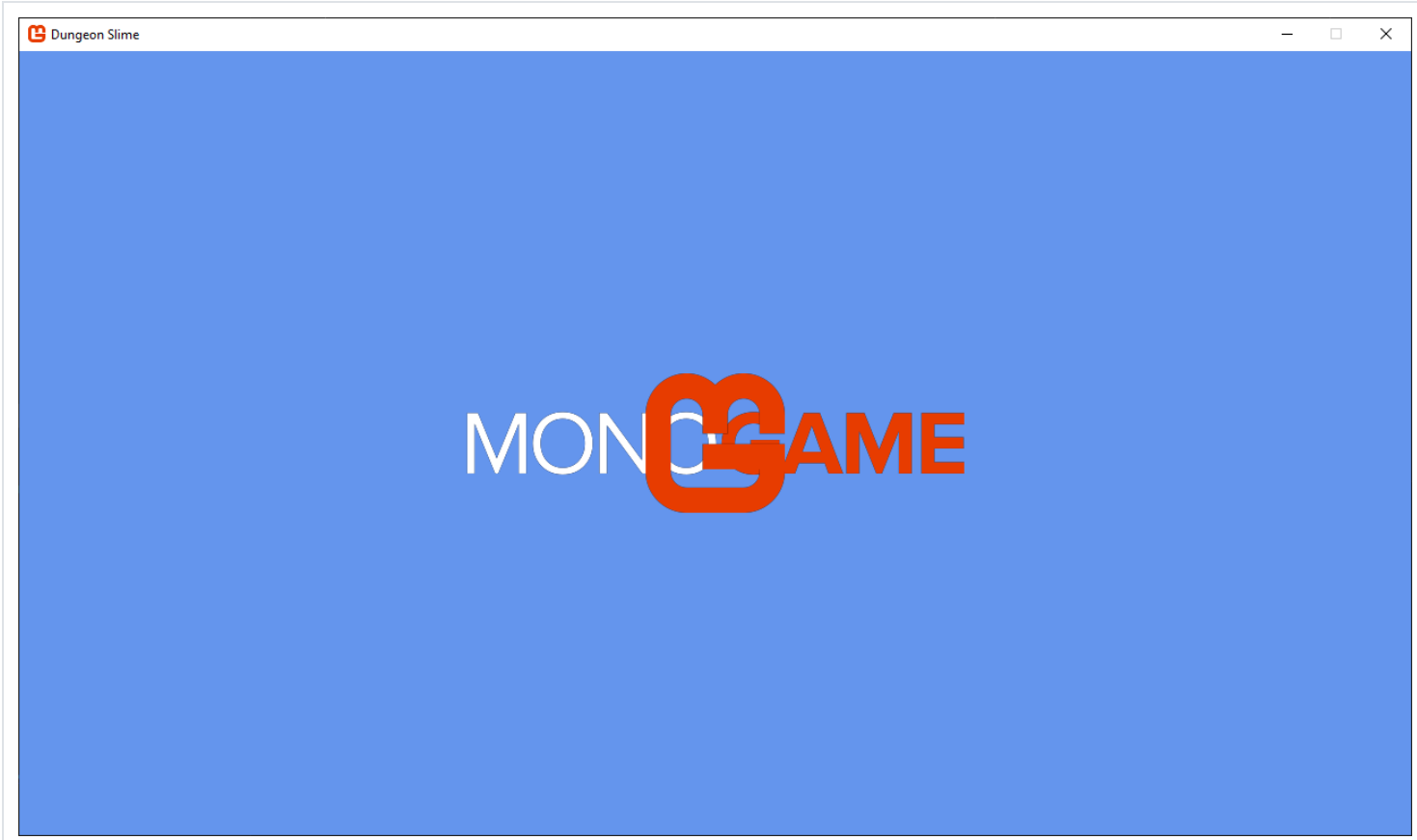


Figure 6-16: The MonoGame icon drawn on top of the wordmark

There are also two additional **SpriteSortMode** values that can be used. These, however, are situational and can have draw backs when using them, so understanding what they are for is important.

The first is **SpriteSortMode.Texture**. This works similar to **SpriteSortMode.Deferred** in that draw calls happen in the order they are made. However, before the draw calls are made, they are sorted by texture. This can be helpful when using multiple textures to reduce texture swapping, however it can have unintended results with layering if you are not careful.

The second is **SpriteSortMode.Immediate**. When using this sort mode, when a draw call is made, it is immediately flushed to the GPU and rendered to the screen, ignoring the layer depth, instead of batched and drawn when **SpriteBatch.End** is called. Using this can cause performance issues and should only be used when necessary. We will discuss an example of using this in a later chapter when we discuss shaders, since with **SpriteSortMode.Immediate** you can adjust shader parameters for each individual draw call.

Conclusion

In this chapter, you accomplished the following:

- You learned about the different parameters of the **SpriteBatch.Draw** method and how they affect sprite rendering.
- You learned how the `rotation` parameter works and how to convert between degrees and radians using **MathHelper.ToRadians**.
- You learned how the `origin` parameter affects sprite positioning, rotation, and scaling.
- You learned how to use the `scale` parameter to resize sprites uniformly or along individual axes.
- You explored the **SpriteEffects** enum to flip sprites horizontally and vertically.
- You learned how the `color` parameter can be used to tint sprites and adjust their opacity.
- You used the `sourceRectangle` parameter to draw specific regions from a texture.
- You explored sprite layering using the `layerDepth` parameter and different **SpriteSortMode** options.

In the next chapter, we will take what we have learned about working with textures and learn techniques to optimize rendering to reduce texture swapping.

Test Your Knowledge

1. What is the purpose of the `origin` parameter in SpriteBatch.Draw, and how does it affect position, rotation and scaling?

   The `origin` parameter determines the reference point for the sprite's position, rotation, and scaling. When set to **Vector2.Zero**, the sprite rotates and scales from its upper-left corner. When set to the center of the sprite, the sprite rotates and scales from its center. The origin point also affects where the sprite is positioned relative to the `position` parameter.

2. How can you adjust a sprite's opacity using **SpriteBatch.Draw**?

   A sprite's opacity can be adjusted by multiplying the `color` parameter by a value between `0.0f` (fully transparent) and `1.0f` (fully opaque). For example, `Color.White * 0.5f` will render the sprite at 50% opacity.

3. How can you flip a sprite horizontally and vertically at the same time using SpriteEffects?

   To flip a sprite both horizontally and vertically, you can combine the SpriteEffects values using the bitwise OR operator (`|`):

   ```
   SpriteEffects.FlipHorizontally | SpriteEffects.FlipVertically
   ```

4. When using the `sourceRectangle` parameter, what information do you need to specify, and what is its purpose?

   The `sourceRectangle` parameter requires a **Rectangle** value where the x- and y-coordinates specify the upper-left corner of the region within the texture and the width and height, in pixels, of the region.

   Its purpose is to specify a specific region within a texture to draw, allowing multiple sprites to be drawn from different parts of the same texture.

Chapter 07: Optimizing Texture Rendering

Explore optimization techniques when rendering textures using a texture atlas.

In Chapter 06, you learned how to load and render textures using **SpriteBatch**. While rendering individual textures works well for simple games, it can lead to performance issues as your game grows more complex. In this chapter, we will explore how to optimize texture rendering by reducing texture swaps and creating reusable components for better organization.

In this chapter, you will:

- Learn about texture swapping and its impact on performance.
- Explore texture atlases as a solution for optimizing texture rendering.
- Create reusable classes to optimize and simplify texture management and rendering.

By the end of this chapter, you will understand how to organize your game's textures for optimal performance and have a flexible texture atlas management system for your future game projects.

Chapter Video

Live Recording

Texture Swapping

Every time the **SpriteBatch.Draw** method is executed with a different *texture* parameter than the previous **SpriteBatch.Draw** method call, a *texture swap* occurs, unbinding the current texture on the GPU and binding the new texture.

> **NOTE**
> A texture swap occurs when the GPU needs to switch between different textures during rendering. While each individual swap may seem trivial, the cumulative effect in a complex game can significantly impact performance.

For example, here are the simplified draw calls for an example Pong game:

```
// Using the paddle texture to render the left player paddle.
// The paddle texture is bound to the GPU.
_spriteBatch.Draw(paddleTexture, leftPaddlePosition, Color.White);

// Using the ball texture to render the ball
// A texture swap occurs, unbinding the paddle texture to bind the ball texture.
_spriteBatch.Draw(ballTexture, ballPosition, Color.White);

// Reusing the paddle texture to draw the right player paddle.
// A texture swap occurs again, unbinding the ball texture to bind the
paddle texture.
_spriteBatch.Draw(paddleTexture, rightPaddlePosition, Color.White);
```

In the above example:

1. The paddle texture is bound to the GPU so the left player paddle can be drawn.
2. The paddle texture is unbound from the GPU and the ball texture is bound so that the ball can be drawn (Texture Swap #1).
3. The ball texture is unbound from the GPU and the paddle texture is bound again so the right player paddle can be drawn (Texture Swap #2).

These texture swaps, while negligible in this example, can become a performance issue in a full game where you might be drawing hundreds or thousands of sprites per frame.

Attempting to Optimize Draw Order

One approach to get around this could be to optimize the order of the draw calls to minimize texture swaps. For example, if we reorder the draw calls from the previous example so that both paddles are drawn first and then the ball, the number of texture swaps is reduced from two to one:

```
// Render the left and right paddles first.
// This reduces the number of texture swaps needed from two to one.
_spriteBatch.Draw(paddleTexture, _leftPaddlePosition, Color.White);
_spriteBatch.Draw(paddleTexture, _rightPaddlePosition, Color.White);
_spriteBatch.Draw(ballTexture, _ballPosition, Color.White);
```

However this is not a scalable solution. In a real game with dozens of different textures and complex draw orders for layered sprites, UI elements, particles, etc., managing draw order by texture becomes impractical and will conflict with desired visual layering.

What is a Texture Atlas

A texture atlas (also known as a sprite sheet) is a large image file that contains multiple smaller images packed together. Instead of loading separate textures for each sprite, you load the single texture file with all the images combined like a scrapbook where all your photos are arranged on the same page.

> **NOTE**
>
> Using a texture atlas not only eliminates texture swaps but also reduces memory usage and simplifies asset management since you are loading and tracking a single texture instead of many individual ones.

In the Pong example, imagine taking the paddle and ball image and combining them into a single image file like in *Figure 7-1* below:

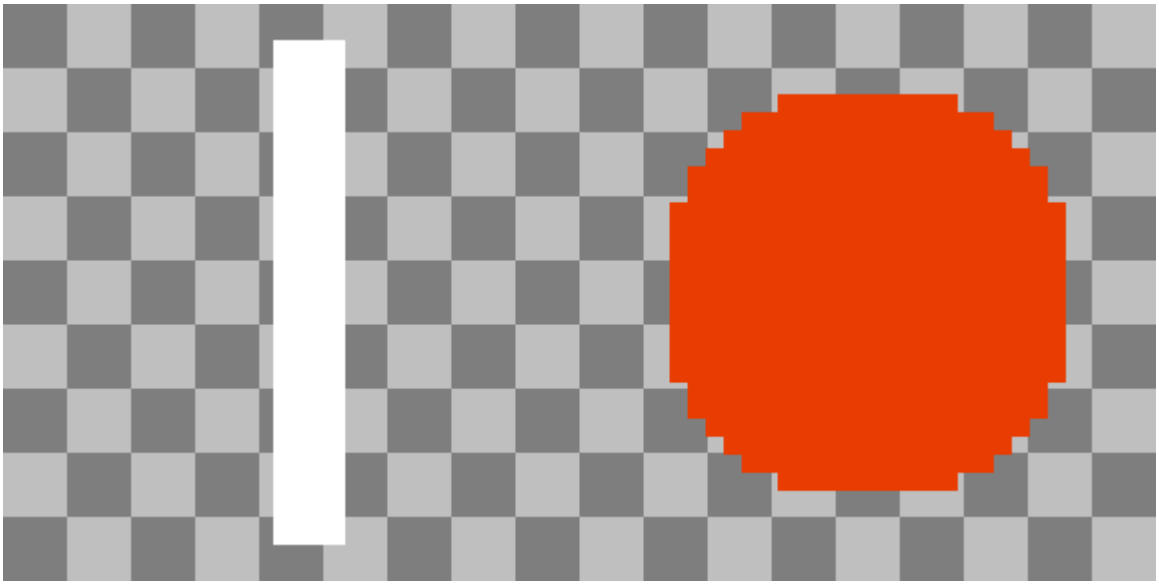


Figure 7-1: Pong Texture Atlas Example

Now when we draw these images, we would be using the same texture and just specify the source rectangles for the paddle or ball when needed, completely eliminating texture swaps.

```
private Texture2D _textureAtlas;
private Rectangle _paddleSourceRect;
private Rectangle _ballSourceRect;

protected override void LoadContent()
{
    _textureAtlas = Content.Load<Texture2D>("pong-atlas");
    _paddleSourceRect = new Rectangle(0, 0, 32, 32);
    _ballSourceRect = new Rectangle(32, 0, 32, 32);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    _spriteBatch.Begin();

    // All draw calls use the same texture, so there is no texture swapping!
    _spriteBatch.Draw(_textureAtlas, _leftPaddlePosition,
_paddleSourceRect, Color.White);
    _spriteBatch.Draw(_textureAtlas, _rightPaddlePosition,
_paddleSourceRect, Color.White);
    _spriteBatch.Draw(_textureAtlas, _ballPosition, _ballSourceRect, Color.White);

    _spriteBatch.End();
}
```

While using the single texture with source rectangles solves the potential performance issues, managing multiple source rectangles in variables can become complex as your game grows. In the Pong example above, we are already tracking the source rectangles for both the paddle and ball sprites. Imagine scaling this up to a game with dozens of different images, each potentially needing their own position, rotation, scale, and other rendering properties.

To better organize this complexity, we can apply object-oriented design principles to create classes that encapsulates the information needed.

The TextureRegion Class

In Chapter 06, we learned about using the `sourceRectangle` parameter to reuse the same texture when rendering sprites but specifying different regions within the texture to render. For our next step, we will build on this and create a class called `TextureRegion`.

We are going to add this class to the class library we created in Chapter 04. Perform the following:

1. Add new folder in the *MonoGameLibrary* project named `Graphics`
2. Create a new file named `TextureRegion.cs` inside the *Graphics* folder you just created.
3. Add the following code for the foundation of the `TextureRegion` class to the file:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MonoGameLibrary.Graphics;

/// <summary>
/// Represents a rectangular region within a texture.
/// </summary>
public class TextureRegion
{

}
```

> **NOTE**
>
> The `TextureRegion.cs` class file is placed in the *MonoGame/Graphics* folder and the class uses the `MonoGameLibrary.Graphics` namespace to keep graphics-related classes organized together. As we add more functionality to the library, we will continue to use directories and namespaces to maintain a clean structure.

We will add several components to this class in sequence. Each section below should be added to the `TextureRegion` class in the order presented between the brackets `{ }` of the class definition. As we go through each part, the class will gradually take shape to handle all the texture handling behavior we need.

TextureRegion Members

The `TextureRegion` class will utilize four properties to define and manage a region within a texture. Add the following properties:

```
/// <summary>
/// Gets or Sets the source texture this texture region is part of.
/// </summary>
public Texture2D Texture { get; set; }

/// <summary>
/// Gets or Sets the source rectangle boundary of this texture region within the
source texture.
/// </summary>
public Rectangle SourceRectangle { get; set; }

/// <summary>
/// Gets the width, in pixels, of this texture region.
/// </summary>
public int Width => SourceRectangle.Width;

/// <summary>
/// Gets the height, in pixels, of this texture region.
/// </summary>
public int Height => SourceRectangle.Height;
```

The `Texture` and `SourceRectangle` properties work together to define where the region is located: `Texture` specifies which texture contains the region, while `SourceRectangle` defines its exact location and size within that texture. The `Width` and `Height` properties provide convenient access to the region's dimensions without having to access the SourceRectangle property directly.

TextureRegion Constructor

The `TextureRegion` class provides two ways to create a new texture region.

Add the following constructors:

```
/// <summary>
/// Creates a new texture region.
/// </summary>
public TextureRegion() { }

/// <summary>
/// Creates a new texture region using the specified source texture.
/// </summary>
/// <param name="texture">The texture to use as the source texture for this texture
region.</param>
/// <param name="x">The x-coordinate position of the upper-left corner of this
texture region relative to the upper-left corner of the source texture.</param>
/// <param name="y">The y-coordinate position of the upper-left corner of this
texture region relative to the upper-left corner of the source texture.</param>
/// <param name="width">The width, in pixels, of this texture region.</param>
/// <param name="height">The height, in pixels, of this texture region.</param>
public TextureRegion(Texture2D texture, int x, int y, int width, int height)
{
    Texture = texture;
    SourceRectangle = new Rectangle(x, y, width, height);
}
```

- The default constructor creates an empty texture region that can be configured later.
- The parameterized constructor allows you to define the region's source texture and boundary in a single step.

The second constructor provides a convenient way to create texture regions when you know the exact location and dimensions within the source texture upfront.

TextureRegion Methods

Finally, the `TextureRegion` class will provide three overloaded Draw methods to render the texture region. Add the following methods:

```
/// <summary>
/// Submit this texture region for drawing in the current batch.
/// </summary>
/// <param name="spriteBatch">The spritebatch instance used for batching draw
calls.</param>
/// <param name="position">The xy-coordinate location to draw this texture region on
the screen.</param>
/// <param name="color">The color mask to apply when drawing this texture region on
screen.</param>
public void Draw(SpriteBatch spriteBatch, Vector2 position, Color color)
{
    Draw(spriteBatch, position, color, 0.0f, Vector2.Zero, Vector2.One,
SpriteEffects.None, 0.0f);
}

/// <summary>
/// Submit this texture region for drawing in the current batch.
/// </summary>
/// <param name="spriteBatch">The spritebatch instance used for batching draw
calls.</param>
/// <param name="position">The xy-coordinate location to draw this texture region on
the screen.</param>
/// <param name="color">The color mask to apply when drawing this texture region on
screen.</param>
/// <param name="rotation">The amount of rotation, in radians, to apply when drawing
this texture region on screen.</param>
/// <param name="origin">The center of rotation, scaling, and position when drawing
this texture region on screen.</param>
/// <param name="scale">The scale factor to apply when drawing this texture region
on screen.</param>
/// <param name="effects">Specifies if this texture region should be flipped
horizontally, vertically, or both when drawing on screen.</param>
/// <param name="layerDepth">The depth of the layer to use when drawing this texture
region on screen.</param>
public void Draw(SpriteBatch spriteBatch, Vector2 position, Color color, float
rotation, Vector2 origin, float scale, SpriteEffects effects, float layerDepth)
{
    Draw(
        spriteBatch,
        position,
        color,
        rotation,
        origin,
        new Vector2(scale, scale),
        effects,
        layerDepth
```

```
    );
}

/// <summary>
/// Submit this texture region for drawing in the current batch.
/// </summary>
/// <param name="spriteBatch">The spritebatch instance used for batching draw
calls.</param>
/// <param name="position">The xy-coordinate location to draw this texture region on
the screen.</param>
/// <param name="color">The color mask to apply when drawing this texture region on
screen.</param>
/// <param name="rotation">The amount of rotation, in radians, to apply when drawing
this texture region on screen.</param>
/// <param name="origin">The center of rotation, scaling, and position when drawing
this texture region on screen.</param>
/// <param name="scale">The amount of scaling to apply to the x- and y-axes when
drawing this texture region on screen.</param>
/// <param name="effects">Specifies if this texture region should be flipped
horizontally, vertically, or both when drawing on screen.</param>
/// <param name="layerDepth">The depth of the layer to use when drawing this texture
region on screen.</param>
public void Draw(SpriteBatch spriteBatch, Vector2 position, Color color, float
rotation, Vector2 origin, Vector2 scale, SpriteEffects effects, float layerDepth)
{
    spriteBatch.Draw(
        Texture,
        position,
        SourceRectangle,
        color,
        rotation,
        origin,
        scale,
        effects,
        layerDepth
    );
}
```

These methods provide flexible options for rendering the texture region, similar to what the **SpriteBatch.Draw** method does:

- The simplest overload requires only position and color.
- A second overload exposes all rendering parameters while allowing for a single float value to be applied to both axes for scaling.
- The third overload is the most flexible, offering all rendering parameters and independent x- and y-axis scaling.

Only the last `Draw` method actually uses the `Texture` specified in the class, as the other two methods both depend on this final implementation, simplifying the code needed to actually do the drawing.

The TextureAtlas Class

In the What is a Texture Atlas section above, a texture atlas was described as a scrap book that holds all of the individual sprites for the game. These individual sprites can now be represented by the `TextureRegion` class we just created. Now, we will create the `TextureAtlas` class to represent the collection of the regions that make up all of our sprites.

Just like the `TextureRegion` class, we are going to add this to the class library. In the *Graphics* folder within the *MonoGameLibrary* project, add a new file named `TextureAtlas.cs`, then add the following code for the foundation of the `TextureAtlas` class:

```
using System.Collections.Generic;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace MonoGameLibrary.Graphics;

public class TextureAtlas
{

}
```

As before, each section below adds more functionality to this class, one after the other within the `TextureAtlas` class.

TextureAtlas Members

The `TextureAtlas` class needs two key members to manage texture regions. Add the following:

```
private Dictionary<string, TextureRegion> _regions;

/// <summary>
/// Gets or Sets the source texture represented by this texture atlas.
/// </summary>
public Texture2D Texture { get; set; }
```

The private `_regions` dictionary stores named texture regions, allowing us to retrieve specific regions by name, while the `Texture` property holds the source texture that contains all the regions. Together, these members enable the atlas to manage multiple texture regions from a single source texture.

TextureAtlas Constructors

The `TextureAtlas` class provides two ways to create a new atlas.

Add the following constructors:

```
/// <summary>
/// Creates a new texture atlas.
/// </summary>
public TextureAtlas()
{
    _regions = new Dictionary<string, TextureRegion>();
}

/// <summary>
/// Creates a new texture atlas instance using the given texture.
/// </summary>
/// <param name="texture">The source texture represented by the texture
atlas.</param>
public TextureAtlas(Texture2D texture)
{
    Texture = texture;
    _regions = new Dictionary<string, TextureRegion>();
}
```

- The default constructor creates an empty atlas that can be configured later.
- The parameterized constructor allows you to specify the source texture immediately.

Both constructors initialize the `_regions` dictionary so that it is ready to be used either way.

TextureAtlas Methods

Finally, The `TextureAtlas` class will provide methods for managing texture regions and creating atlases from configuration files. Add the following methods:

```
/// <summary>
/// Creates a new region and adds it to this texture atlas.
/// </summary>
/// <param name="name">The name to give the texture region.</param>
/// <param name="x">The top-left x-coordinate position of the region boundary
relative to the top-left corner of the source texture boundary.</param>
/// <param name="y">The top-left y-coordinate position of the region boundary
relative to the top-left corner of the source texture boundary.</param>
/// <param name="width">The width, in pixels, of the region.</param>
/// <param name="height">The height, in pixels, of the region.</param>
public void AddRegion(string name, int x, int y, int width, int height)
{
    TextureRegion region = new TextureRegion(Texture, x, y, width, height);
    _regions.Add(name, region);
}

/// <summary>
/// Gets the region from this texture atlas with the specified name.
/// </summary>
/// <param name="name">The name of the region to retrieve.</param>
/// <returns>The TextureRegion with the specified name.</returns>
public TextureRegion GetRegion(string name)
{
    return _regions[name];
}

/// <summary>
/// Removes the region from this texture atlas with the specified name.
/// </summary>
/// <param name="name">The name of the region to remove.</param>
/// <returns></returns>
public bool RemoveRegion(string name)
{
    return _regions.Remove(name);
}

/// <summary>
/// Removes all regions from this texture atlas.
/// </summary>
public void Clear()
{
    _regions.Clear();
}

/// <summary>
/// Creates a new texture atlas based on a texture atlas xml configuration file.
```

```
/// </summary>
/// <param name="content">The content manager used to load the texture for the
atlas.</param>
/// <param name="fileName">The path to the xml file, relative to the content root
directory.</param>
/// <returns>The texture atlas created by this method.</returns>
public static TextureAtlas FromFile(ContentManager content, string fileName)
{
    TextureAtlas atlas = new TextureAtlas();

    string filePath = Path.Combine(content.RootDirectory, fileName);

    using (Stream stream = TitleContainer.OpenStream(filePath))
    {
        using (XmlReader reader = XmlReader.Create(stream))
        {
            XDocument doc = XDocument.Load(reader);
            XElement root = doc.Root;

            // The <Texture> element contains the content path for the Texture2D
to load.
            // So we will retrieve that value then use the content manager to load
the texture.
            string texturePath = root.Element("Texture").Value;
            atlas.Texture = content.Load<Texture2D>(texturePath);

            // The <Regions> element contains individual <Region> elements, each
one describing
            // a different texture region within the atlas.
            //
            // Example:
            // <Regions>
            //      <Region name="spriteOne" x="0" y="0" width="32" height="32" />
            //      <Region name="spriteTwo" x="32" y="0" width="32" height="32" />
            // </Regions>
            //
            // So we retrieve all of the <Region> elements then loop through
each one
            // and generate a new TextureRegion instance from it and add it to
this atlas.
            var regions = root.Element("Regions")?.Elements("Region");

            if (regions != null)
            {
                foreach (var region in regions)
                {
```

```
                    string name = region.Attribute("name")?.Value;
                    int x = int.Parse(region.Attribute("x")?.Value ?? "0");
                    int y = int.Parse(region.Attribute("y")?.Value ?? "0");
                    int width = int.Parse(region.Attribute("width")?.Value ?? "0");
                    int height = int.Parse(region.Attribute("height")?.Value
?? "0");

                    if (!string.IsNullOrEmpty(name))
                    {
                        atlas.AddRegion(name, x, y, width, height);
                    }
                }
            }

            return atlas;
        }
    }
}
```

These methods serve different purposes in managing the texture atlas:

1. Region Management
    - `AddRegion`: Creates a new `TextureRegion` at the specified location in the atlas.
    - `GetRegion`: Retrieves a previously added region by its name.
    - `RemoveRegion`: Removes a specific region by its name.
    - `Clear`: Removes all regions from the atlas.
2. Atlas Creation
    - `FromFile`: creates a new `TextureAtlas` from an XML configuration file. This method will load the source texture then create and add the regions based on the XML configuration. We will look more into using the XML configuration in a moment.

Using the TextureAtlas Class

Now we can put our new `TextureAtlas` class to use by exploring two approaches; creating an atlas manually and using XML configuration. So far, we have been practicing using textures with the MonoGame logo. Now we will use a new texture atlas that contains various sprites we will need for our game.

Download the texture atlas by right-clicking the following image and saving it as atlas.png:

Figure 7-2: The texture atlas for our game

> **TIP**
>
> You may notice that our texture atlas image has some empty areas, which seems like a waste. Its dimensions are 256x256 pixels when it could have just been 240x160 pixels.
>
> **This is intentional.**
>
> Game graphics often use texture dimensions that are powers of 2 (128, 256, 512, 1024, etc.) for technical reasons. While modern graphics hardware can handle any texture size, power-of-2 dimensions provide better memory efficiency and more precise rendering. When pixel coordinates are converted to texture coordinates during rendering, powers of 2 can be represented more precisely in floating-point calculations, helping prevent visual artifacts like texture seams.
>
> This is not critical for simple 2D games, but adopting this practice early will serve you well as your games become more complex.

Add this texture atlas to your content project using the MGCB Editor:

1. Open the *Content.mgcb* file in the MGCB Editor
2. In the editor, right-click the *images* folder and choose *Add* > *Existing item...*.
3. Navigate to and choose the *atlas.png* file you downloaded to add it.
4. Save the changes and close the MGCB Editor.

> **TIP**
>
> If you need a refresher on adding content using the MGCB Editor, you can revisit the Chapter 05: The Content Pipeline.

First, we will explore creating the texture atlas and defining the texture regions directly in code. Replace the contents of `Game1.cs` with the following:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime;

public class Game1 : Core
{
    // texture region that defines the slime sprite in the atlas.
    private TextureRegion _slime;

    // texture region that defines the bat sprite in the atlas.
    private TextureRegion _bat;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Load the atlas texture using the content manager
        Texture2D atlasTexture = Content.Load<Texture2D>("images/atlas");

        //  Create a TextureAtlas instance from the atlas
        TextureAtlas atlas = new TextureAtlas(atlasTexture);

        // add the slime region to the atlas.
        atlas.AddRegion("slime", 0, 0, 20, 20);

        // add the bat region to the atlas.
        atlas.AddRegion("bat", 20, 0, 20, 20);

        // retrieve the slime region from the atlas.
        _slime = atlas.GetRegion("slime");

        // retrieve the bat region from the atlas.
```

```
        _bat = atlas.GetRegion("bat");
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // TODO: Add your update logic here

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime texture region at a scale of 4.0
        _slime.Draw(SpriteBatch, Vector2.Zero, Color.White, 0.0f, Vector2.One, 4.0f,
SpriteEffects.None, 0.0f);

        // Draw the bat texture region 10px to the right of the slime at a scale
of 4.0
        _bat.Draw(SpriteBatch, new Vector2(_slime.Width * 4.0f + 10, 0),
Color.White, 0.0f, Vector2.One, 4.0f, SpriteEffects.None, 1.0f);

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes in this implementation are:

1. The `_logo` field was removed.
2. Added `TextureRegion` members for the slime and bat sprites.
3. In **LoadContent**:
    - Removed loading the logo texture.
    - Created a `TextureAtlas` with the atlas texture.
    - Added regions for both the slime and the bat.
    - Retrieved the regions using their names.
4. Updated **Draw** to:
    - Draw the slime at a scale factor of 4.
    - Draw the bat 10 pixels to the right of the slime based on the slime's `Width` property, at a scale of 4.

Running the game now shows both sprites in the upper-left corner:

Figure 7-3: The slime and bat texture regions being rendered in the upper-left corner of the game window

While manual creation works for a few sprites, managing many regions becomes cumbersome. Now we will explore the `TextureAtlas.FromFile` method to load our atlas configuration from XML instead. Perform the following:

1. Open the *Content.mgcb* file in the MGCB Editor

2. Create a new file named `atlas-definition` in the *Content/images* folder using "Add -> New Item -> XML Content".

3. In the properties panel at the bottom for the `atlas-definition.xml` file, change the ***`Build Action`*** property from `Build` to ***`Copy`***.

4. Save the changes and close the MGCB Editor

5. Open the new `atlas-definition.xml` file in your code editor (The MGCB editor cannot edit files, only manage the what content it builds)

6. Replace the contents of the XML file with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TextureAtlas>
    <Texture>images/atlas</Texture>
    <Regions>
        <Region name="slime" x="0" y="0" width="20" height="20" />
        <Region name="bat" x="20" y="0" width="20" height="20" />
    </Regions>
</TextureAtlas>
```

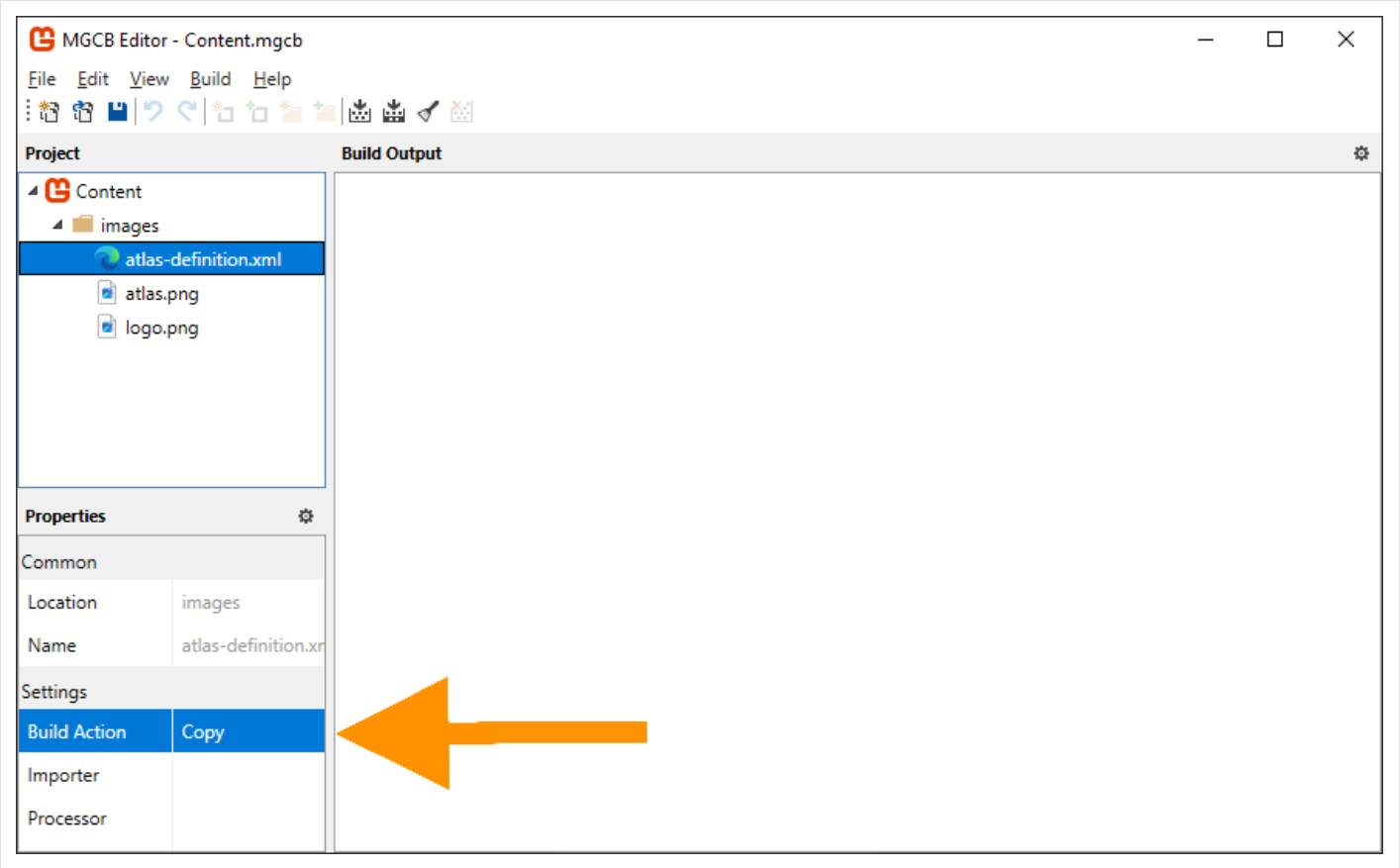


Figure 7-4: The atlas-definition.xml file added to the content project with the Build Action property set to Copy

> **TIP**
>
> Using the content pipeline to copy files ensures they are placed in the correct location alongside other game content. While there are other methods (like editing the .csproj), this approach keeps asset management centralized

7. Update the contents of `Game1.cs` with the following code:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime;

public class Game1 : Core
{
    // texture region that defines the slime sprite in the atlas.
    private TextureRegion _slime;

    // texture region that defines the bat sprite in the atlas.
    private TextureRegion _bat;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // retrieve the slime region from the atlas.
        _slime = atlas.GetRegion("slime");

        // retrieve the bat region from the atlas.
        _bat = atlas.GetRegion("bat");
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();
```

```
            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            // Clear the back buffer.
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // Begin the sprite batch to prepare for rendering.
            SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

            // Draw the slime texture region at a scale of 4.0
            _slime.Draw(SpriteBatch, Vector2.Zero, Color.White, 0.0f, Vector2.One,
4.0f, SpriteEffects.None, 0.0f);

            // Draw the bat texture region 10px to the right of the slime at a scale
of 4.0
            _bat.Draw(SpriteBatch, new Vector2(_slime.Width * 4.0f + 10, 0),
Color.White, 0.0f, Vector2.One, 4.0f, SpriteEffects.None, 1.0f);

            // Always end the sprite batch when finished.
            SpriteBatch.End();

            base.Draw(gameTime);
        }
    }
```

The key improvements here is in **LoadContent**, where we now:

- Create an atlas from the XML configuration file.
- Let the `TextureAtlas.FromFile` method handle texture loading and region creation.
- Removed the manual creation of regions in code.

This configuration based approached is advantageous because we can now add new and modify existing regions within the atlas without having to change code and/or recompile. This also keeps the sprite definitions separate from the game logic.

Running the game now will show the same results as *Figure 7-4* above, with the slime and bat texture regions rendered in the upper-left corner of the game window.

Conclusion

In this chapter, you accomplished the following:

- Learned about texture swapping and its impact on performance
- Explored texture atlases as a solution for optimizing texture rendering
- Learned what a class library is and the benefits of using one.
- Created reusable `TextureRegion` and `TextureAtlas` classes to optimize and simplify texture management.
- Learned how to include assets in the content pipeline that should only be copied and not processed.

In the next chapter, we will build on the concepts of the `TextureAtlas` and explore creating the `Sprite` and `AnimatedSprite` classes to further simplify managing and rendering sprites.

Test Your Knowledge

1. What is a texture swap and why can it impact performance?

   A texture swap occurs when the GPU needs to unbind one texture and bind another between draw calls. While individual swaps may seem trivial, they can significantly impact performance in games with many sprites as each swap is an expensive GPU operation.

2. Name a benefit of using a texture atlas.

   Any of the following are benefits of using a texture atlas:

   - Eliminates texture swaps by using a single texture
   - Reduces memory usage
   - Simplifies asset management
   - Improves rendering performance

Chapter 08: The Sprite Class

Explore creating a reusable Sprite class to efficiently manage sprites and their rendering properties, including position, rotation, scale, and more.

In Chapter 07, you learned how to use texture atlases to optimize rendering performance. While this solved the issue of texture swapping, managing individual sprites and their properties becomes increasingly complex as your game grows. Even in our simple example with just a slime and a bat, we would eventually need to track various properties for each sprite:

- Color mask for tinting.
- Origin for rotation and scale.
- Scale for size adjustments.
- Rotation for orientation.
- Sprite effects to flip horizontally and/or vertically.
- Layer depth for draw order layering.

Imagine scaling this up to dozens of sprites, each with multiple instances on screen. Tracking all these properties through individual variables quickly becomes unmanageable. In this chapter, we will solve this by creating a class that encapsulates sprite information and handles rendering.

Chapter Video

Live Recording

The Sprite Class

A sprite in our game represents a visual object created from a texture region, along with its rendering properties. While multiple sprites might use the same texture region (like multiple enemies of the same type), each sprite can have unique properties that control how it appears on screen; its position, rotation, scale, and other visual characteristics.

By creating a `Sprite` class, we can encapsulate both the texture region and its rendering parameters into a single, reusable component. This not only makes our code more organized but also makes it easier to manage multiple instances of the same type of sprite.

In the *Graphics* folder within the *MonoGameLibrary* project, add a new file named `Sprite.cs` and add the following code for the foundation of the `Sprite` class to the file:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MonoGameLibrary.Graphics;

public class Sprite
{

}
```

Properties

The `Sprite` class will utilize properties that mirror the parameters used in **SpriteBatch.Draw** so the rendering parameter for each sprite is self contained. Add the following properties:

```
/// <summary>
/// Gets or Sets the source texture region represented by this sprite.
/// </summary>
public TextureRegion Region { get; set; }

/// <summary>
/// Gets or Sets the color mask to apply when rendering this sprite.
/// </summary>
/// <remarks>
/// Default value is Color.White
/// </remarks>
public Color Color { get; set; } = Color.White;

/// <summary>
/// Gets or Sets the amount of rotation, in radians, to apply when rendering
this sprite.
/// </summary>
/// <remarks>
/// Default value is 0.0f
/// </remarks>
public float Rotation { get; set; } = 0.0f;

/// <summary>
/// Gets or Sets the scale factor to apply to the x- and y-axes when rendering
this sprite.
/// </summary>
/// <remarks>
/// Default value is Vector2.One
/// </remarks>
public Vector2 Scale { get; set; } = Vector2.One;

/// <summary>
/// Gets or Sets the xy-coordinate origin point, relative to the top-left corner, of
this sprite.
/// </summary>
/// <remarks>
/// Default value is Vector2.Zero
/// </remarks>
public Vector2 Origin { get; set; } = Vector2.Zero;

/// <summary>
/// Gets or Sets the sprite effects to apply when rendering this sprite.
/// </summary>
/// <remarks>
/// Default value is SpriteEffects.None
/// </remarks>
```

```
public SpriteEffects Effects { get; set; } = SpriteEffects.None;

/// <summary>
/// Gets or Sets the layer depth to apply when rendering this sprite.
/// </summary>
/// <remarks>
/// Default value is 0.0f
/// </remarks>
public float LayerDepth { get; set; } = 0.0f;

/// <summary>
/// Gets the width, in pixels, of this sprite.
/// </summary>
/// <remarks>
/// Width is calculated by multiplying the width of the source texture region by the
x-axis scale factor.
/// </remarks>
public float Width => Region.Width * Scale.X;

/// <summary>
/// Gets the height, in pixels, of this sprite.
/// </summary>
/// <remarks>
/// Height is calculated by multiplying the height of the source texture region by
the y-axis scale factor.
/// </remarks>
public float Height => Region.Height * Scale.Y;
```

The `TextureRegion` property works to provide the texture and source rectangle when rendering the sprite. Other properties directly correspond to **SpriteBatch.Draw** parameters with the same default values, making it easy to understand how each property affects the sprite's appearance.

> **TIP**
>
> The calculated `Width` and `Height` properties make it easier to position sprites relative to each other without manually applying scale factors.

Constructors

The `Sprite` class provides two ways to create a new sprite.

Add the following constructors:

```
/// <summary>
/// Creates a new sprite.
/// </summary>
public Sprite() { }

/// <summary>
/// Creates a new sprite using the specified source texture region.
/// </summary>
/// <param name="region">The texture region to use as the source texture region for
this sprite.</param>
public Sprite(TextureRegion region)
{
    Region = region;
}
```

- The default constructor creates an empty sprite that can be configured later.
- The parameterized constructor allows you to specify the source texture region for the sprite.

Methods

Finally, add the following two method to the `Sprite` class:

```
/// <summary>
/// Sets the origin of this sprite to the center.
/// </summary>
public void CenterOrigin()
{
    Origin = new Vector2(Region.Width, Region.Height) * 0.5f;
}

/// <summary>
/// Submit this sprite for drawing to the current batch.
/// </summary>
/// <param name="spriteBatch">The SpriteBatch instance used for batching draw
calls.</param>
/// <param name="position">The xy-coordinate position to render this sprite
at.</param>
public void Draw(SpriteBatch spriteBatch, Vector2 position)
{
    Region.Draw(spriteBatch, position, Color, Rotation, Origin, Scale,
Effects, LayerDepth);
}
```

- `CenterOrigin`: Sets the origin point of the sprite to its center.

> **NOTE**
> The origin needs to be set based on the width and height of the source texture region itself, regardless of the scale the sprite is rendered at.

- `Draw`: Uses the `TextureRegion` property to submit the sprite for rendering using the properties of the sprite itself.

Create Sprites With The TextureAtlas Class

While the `GetRegion` method of the `TextureAtlas` class we created in Chapter 07 works well for retrieving regions, creating sprites requires multiple steps:

1. Get the region by name.
2. Store it in a variable.
3. Create a new sprite with that region.

We can simplify this process by adding a sprite creation method to the `TextureAtlas` class. Open the `TextureAtlas.cs` and add the following method:

```
/// <summary>
/// Creates a new sprite using the region from this texture atlas with the
specified name.
/// </summary>
/// <param name="regionName">The name of the region to create the sprite
with.</param>
/// <returns>A new Sprite using the texture region with the specified
name.</returns>
public Sprite CreateSprite(string regionName)
{
    TextureRegion region = GetRegion(regionName);
    return new Sprite(region);
}
```

Using the Sprite Class

Now we can adjust our game to use the `Sprite` class instead of just the texture regions. Update the contents of `Game1.cs` with the following:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime sprite.
    private Sprite _slime;

    // Defines the bat sprite.
    private Sprite _bat;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime sprite from the atlas.
        _slime = atlas.CreateSprite("slime");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat sprite from the atlas.
        _bat = atlas.CreateSprite("bat");
        _bat.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
```

```
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // TODO: Add your update logic here

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, Vector2.Zero);

        // Draw the bat sprite 10px to the right of the slime.
        _bat.Draw(SpriteBatch, new Vector2(_slime.Width + 10, 0));

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes in this implementation are:

- The `_slime` and `_bat` members were changed from `TextureRegion` to `Sprite`.
- In **LoadContent**
  - The `_slime` and `_bat` sprites are now created using the new `TextureAtlas.CreateSprite` method.
  - Both the `_slime` and `_bat` sprites are given a scale of 4.0f.
- In **Draw**, the draw calls were updated to use the `Sprite.Draw` method.

Running the game now will produce the same result as in the previous chapter.

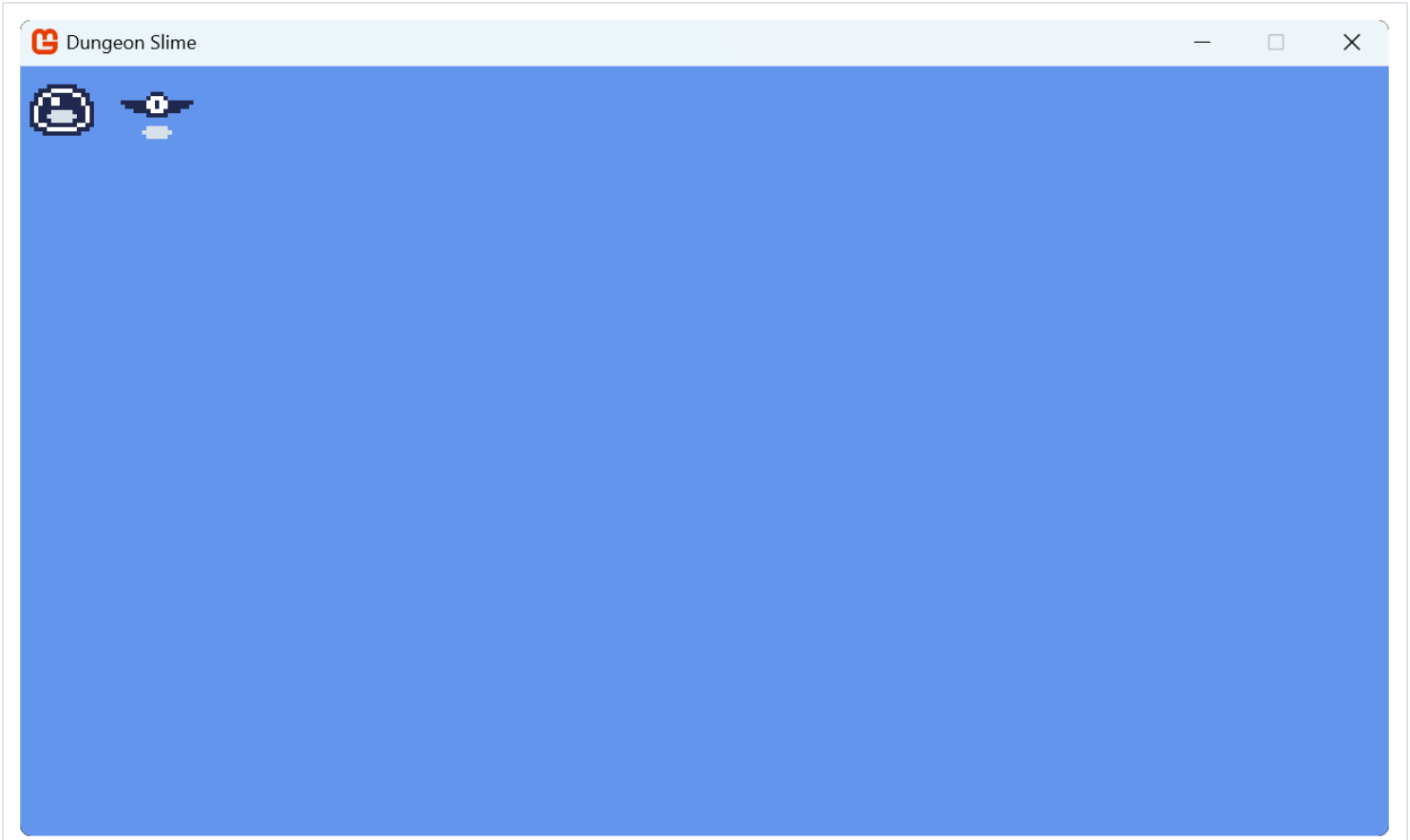


Figure 8-1: The slime and bat sprites being rendered in the upper-left corner of the game window

> **NOTE**
>
> Notice how even though we increased the scale of both sprites, the bat sprite is still only 10px to the right of the slime. This is because the `Width` property we created for the `Sprite` class takes into account the scale factor of the sprite as well.

Try adjusting the various properties available for the slime and the bat sprites to see how they affect the rendering.

Conclusion

In this chapter, we created a reusable `Sprite` class that encapsulates the properties for each sprite that we would render. The `TextureAtlas` class was updated to simplify sprite creation based on the `Sprite` class we created.

In the next chapter, we will build upon the `Sprite` class to create an `AnimatedSprite` class that will allow us to bring our sprites to life through animation.

Test Your Knowledge

1. What is the benefit of using a Sprite class instead of managing texture regions directly?

   The `Sprite` class encapsulates all rendering properties (position, rotation, scale, etc.) into a single, reusable component. This makes it easier to manage multiple instances of the same type of sprite without having to track properties through individual variables.

2. Why do the `Width` and `Height` properties of a Sprite take the Scale property into account?

   The `Width` and `Height` properties account for scaling to make it easier to position sprites relative to each other without having to manually calculate the scaled dimensions. This is particularly useful when sprites are rendered at different scales.

3. When using the `CenterOrigin` method, why is the origin calculated using the region's dimensions rather than the sprite's scaled dimensions?

   The origin needs to be set based on the texture region's actual dimensions because it represents the point around which scaling and rotation are applied. Using the scaled dimensions would result in incorrect positioning since the origin would change based on the current scale factor.

4. What advantage does the `TextureAtlas.CreateSprite` method provide over using `GetRegion`?

   The `CreateSprite` method simplifies sprite creation by combining multiple steps (getting the region, storing it, creating a sprite) into a single method call. This reduces code repetition and makes sprite creation more straightforward.

Chapter 09: The AnimatedSprite Class

Create an AnimatedSprite class that builds upon our Sprite class to support frame-based animations.

While packing images into a texture atlas and managing them through our `Sprite` class improves performance and organization, games need animation to bring their visuals to life. 2D animation in games works much like a flip book; a sequence of individual images (*frames*) displayed in rapid succession creates the illusion of movement. Each frame represents a specific point in the sprite's animation, and when these frames are cycled through quickly, our eyes perceive fluid motion.

Chapter Video

Live Recording

> **NOTE**
>
> The term "frame" in animation refers to a single image in an animation sequence. This is different from a game frame, which represents one complete render cycle of your game.

In MonoGame, we can create these animations by cycling through different regions of our texture atlas, with each region representing a single frame of the animation. For example, *Figure 9-1* below shows three frames that make up a bat's wing-flapping animation:

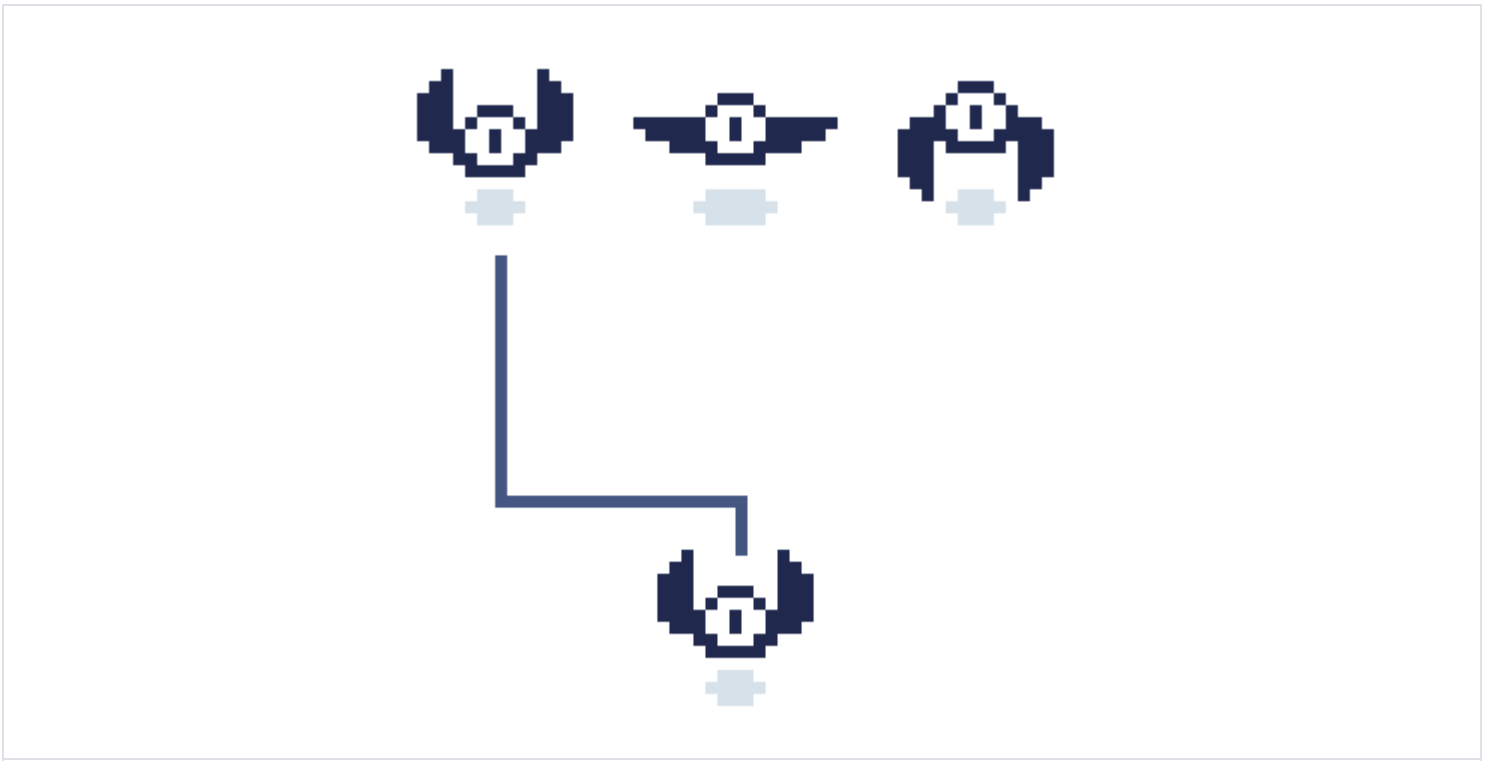


Figure 9-1: Animation example of a bat flapping its wings

By drawing each frame sequentially over time, we create the illusion that the bat is flapping its wings. The speed at which we switch between frames determines how smooth or rapid the animation appears.

In this chapter, we will build off of the `Sprite` class we created in Chapter 08 to create an `AnimatedSprite` class we can use to bring animations to life.

The Animation Class

Before we can create animated sprites, we need a way to manage animation data. We will create an `Animation` class to encapsulate this information. In the *Graphics* folder within the *MonoGameLibrary* project, add a new file named `Animation.cs` with this initial structure:

```
using System;
using System.Collections.Generic;

namespace MonoGameLibrary.Graphics;

public class Animation
{

}
```

Animation Properties

An animation requires two key pieces of information: the sequence of frames to display and the timing between them. Add these properties to the `Animation` class:

```
/// <summary>
/// The texture regions that make up the frames of this animation.  The order of the
regions within the collection
/// are the order that the frames should be displayed in.
/// </summary>
public List<TextureRegion> Frames { get; set; }

/// <summary>
/// The amount of time to delay between each frame before moving to the next frame
for this animation.
/// </summary>
public TimeSpan Delay { get; set; }
```

- The `Frames` property stores the collection of texture regions that make up the animation sequence. The order of regions in this collection is important; they will be displayed in the same sequence they are added, creating the animation's movement. For example, in our bat animation, the frames would be ordered to show the wings moving up, then fully extended, then down.
- The `Delay` property defines how long each frame should be displayed before moving to the next one. This timing control allows us to adjust the speed of our animations; a shorter delay creates faster animations, while a longer delay creates slower ones.

> **NOTE**
>
> Using `TimeSpan` for the delay allows us to specify precise timing intervals, making it easier to synchronize animations with game time. In other scenarios, you could opt to just use `float` values instead.

Animation Constructors

The `Animation` class provides two ways to create an animation.

Add the following constructors:

```
/// <summary>
/// Creates a new animation.
/// </summary>
public Animation()
{
    Frames = new List<TextureRegion>();
    Delay = TimeSpan.FromMilliseconds(100);
}

/// <summary>
/// Creates a new animation with the specified frames and delay.
/// </summary>
/// <param name="frames">An ordered collection of the frames for this
animation.</param>
/// <param name="delay">The amount of time to delay between each frame of this
animation.</param>
public Animation(List<TextureRegion> frames, TimeSpan delay)
{
    Frames = frames;
    Delay = delay;
}
```

- The default constructor creates an animation with an empty collection of frames and a default delay of 100 milliseconds between each frame.
- The parameterized constructor allows you to specify the frames of animation and the delay for the animation.

> **TIP**
>
> The default 100 milliseconds delay provides a good starting point for most animations, roughly equivalent to 10 animation frame changes per second.

Creating Animations With The TextureAtlas Class

The `TextureAtlas` class we created in Chapter 07 can do more than just manage texture regions and create sprites; it can also store and manage animation data to create animated sprites. The `atlas.png` image we are currently using contains the frames of animation for both a slime and a bat, as well as sprites for other things. We will first update our `atlas-definition.xml` file to include all regions in the atlas, as well as add new `<Animation>` elements to define the animations.

Open the `atlas-definition.xml` file in your code editor and replace the contents with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TextureAtlas>
    <Texture>images/atlas</Texture>
    <Regions>
        <Region name="slime-1" x="0" y="0" width="20" height="20" />
        <Region name="slime-2" x="0" y="20" width="20" height="20" />
        <Region name="bat-1" x="20" y="0" width="20" height="20" />
        <Region name="bat-2" x="20" y="20" width="20" height="20" />
        <Region name="bat-3" x="40" y="0" width="20" height="20" />
    </Regions>
    <Animations>
        <Animation name="slime-animation" delay="200">
            <Frame region="slime-1" />
            <Frame region="slime-2" />
        </Animation>
        <Animation name="bat-animation" delay="200">
            <Frame region="bat-1" />
            <Frame region="bat-2" />
            <Frame region="bat-1" />
            <Frame region="bat-3" />
        </Animation>
    </Animations>
</TextureAtlas>
```

The key changes here are:

- Regions have been added for all regions within the atlas.
- The slime and bat regions have been renamed to reflect the frame number of the animation.
- A new `<Animations>` element has been added that defines `<Animation>` elements.

> **NOTE**
>
> In the bat animation, we reuse frame "bat-1" in the sequence (bat-1, bat-2, bat-1, bat-3). This creates a smoother wing-flapping animation by returning to the neutral position between up and down wing positions.

Now that we have a fully configured XML configuration for the atlas, we need to update the `TextureAtlas` class to manage animation data. Open the `TextureAtlas.cs` file and make the following changes:

1. Add the following using statement so we can reference the `TimeSpan` struct:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
```

2. Add storage for animations after the `Texture` property:

```
// Stores animations added to this atlas.
private Dictionary<string, Animation> _animations;
```

3. Update the constructors so that the animations dictionary is initialized:

```
/// <summary>
/// Creates a new texture atlas.
/// </summary>
public TextureAtlas()
{
    _regions = new Dictionary<string, TextureRegion>();
    _animations = new Dictionary<string, Animation>();
}

/// <summary>
/// Creates a new texture atlas instance using the given texture.
/// </summary>
/// <param name="texture">The source texture represented by the texture
atlas.</param>
public TextureAtlas(Texture2D texture)
{
    Texture = texture;
    _regions = new Dictionary<string, TextureRegion>();
    _animations = new Dictionary<string, Animation>();
}
```

4. Add methods to manage animations, similar to those that we use to manage regions:

```
/// <summary>
/// Adds the given animation to this texture atlas with the specified name.
/// </summary>
/// <param name="animationName">The name of the animation to add.</param>
/// <param name="animation">The animation to add.</param>
public void AddAnimation(string animationName, Animation animation)
{
    _animations.Add(animationName, animation);
}

/// <summary>
/// Gets the animation from this texture atlas with the specified name.
/// </summary>
/// <param name="animationName">The name of the animation to retrieve.</param>
/// <returns>The animation with the specified name.</returns>
public Animation GetAnimation(string animationName)
{
    return _animations[animationName];
}

/// <summary>
/// Removes the animation with the specified name from this texture atlas.
/// </summary>
/// <param name="animationName">The name of the animation to remove.</param>
/// <returns>true if the animation is removed successfully; otherwise,
false.</returns>
public bool RemoveAnimation(string animationName)
{
    return _animations.Remove(animationName);
}
```

5. Update the `FromFile` method to parse the new `<Animation>` animation definitions from the XML configuration file

```
/// <summary>
/// Creates a new texture atlas based a texture atlas xml configuration file.
/// </summary>
/// <param name="content">The content manager used to load the texture for the
atlas.</param>
/// <param name="fileName">The path to the xml file, relative to the content
root directory..</param>
/// <returns>The texture atlas created by this method.</returns>
public static TextureAtlas FromFile(ContentManager content, string fileName)
{
    TextureAtlas atlas = new TextureAtlas();

    string filePath = Path.Combine(content.RootDirectory, fileName);

    using (Stream stream = TitleContainer.OpenStream(filePath))
    {
        using (XmlReader reader = XmlReader.Create(stream))
        {
            XDocument doc = XDocument.Load(reader);
            XElement root = doc.Root;

            // The <Texture> element contains the content path for the Texture2D
to load.
            // So we will retrieve that value then use the content manager to
load the texture.
            string texturePath = root.Element("Texture").Value;
            atlas.Texture = content.Load<Texture2D>(texturePath);

            // The <Regions> element contains individual <Region> elements, each
one describing
            // a different texture region within the atlas.
            //
            // Example:
            // <Regions>
            //      <Region name="spriteOne" x="0" y="0" width="32"
height="32" />
            //      <Region name="spriteTwo" x="32" y="0" width="32"
height="32" />
            // </Regions>
            //
            // So we retrieve all of the <Region> elements then loop through
each one
            // and generate a new TextureRegion instance from it and add it to
this atlas.
            var regions = root.Element("Regions")?.Elements("Region");
```

```
            if (regions != null)
            {
                foreach (var region in regions)
                {
                    string name = region.Attribute("name")?.Value;
                    int x = int.Parse(region.Attribute("x")?.Value ?? "0");
                    int y = int.Parse(region.Attribute("y")?.Value ?? "0");
                    int width = int.Parse(region.Attribute("width")?.Value
?? "0");
                    int height = int.Parse(region.Attribute("height")?.Value
?? "0");

                    if (!string.IsNullOrEmpty(name))
                    {
                        atlas.AddRegion(name, x, y, width, height);
                    }
                }
            }

            // The <Animations> element contains individual <Animation>
elements, each one describing
            // a different animation within the atlas.
            //
            // Example:
            // <Animations>
            //      <Animation name="animation" delay="100">
            //          <Frame region="spriteOne" />
            //          <Frame region="spriteTwo" />
            //      </Animation>
            // </Animations>
            //
            // So we retrieve all of the <Animation> elements then loop through
each one
            // and generate a new Animation instance from it and add it to
this atlas.
            var animationElements =
root.Element("Animations").Elements("Animation");

            if (animationElements != null)
            {
                foreach (var animationElement in animationElements)
                {
                    string name = animationElement.Attribute("name")?.Value;
                    float delayInMilliseconds =
float.Parse(animationElement.Attribute("delay")?.Value ?? "0");
                    TimeSpan delay =
```

```
    TimeSpan.FromMilliseconds(delayInMilliseconds);

                    List<TextureRegion> frames = new List<TextureRegion>();

                    var frameElements = animationElement.Elements("Frame");

                    if (frameElements != null)
                    {
                        foreach (var frameElement in frameElements)
                        {
                            string regionName =
frameElement.Attribute("region").Value;
                            TextureRegion region = atlas.GetRegion(regionName);
                            frames.Add(region);
                        }
                    }

                    Animation animation = new Animation(frames, delay);
                    atlas.AddAnimation(name, animation);
                }
            }

            return atlas;
        }
    }
}
```

The updated `FromFile` method now handles both region and animation definitions from the XML configuration. For animations, it:

- Reads the `<Animations>` section from the XML.
- For each animation:
  - Gets the name and frame delay.
  - Collects the referenced texture regions.
  - Creates and stores a new `Animation` instance.

The AnimatedSprite Class

With our `Animation` class handling animation data, and the `TextureAtlas` updated to store the animation data, we can now create a class that represents an animated sprites. Since an animated sprite is essentially a sprite that changes its texture region over time, we can build upon our existing `Sprite` class through inheritance.

> **NOTE**
>
> By inheriting from `Sprite`, our `AnimatedSprite` class automatically gets all the rendering properties (position, rotation, scale, etc.) while adding new animation-specific functionality.

The key to this design is the `Sprite.Region` property. Our `Sprite` class already knows how to render whatever region is currently set, so our `AnimatedSprite` class just needs to update this region property to the correct animation frame at the right time.

We will now create the initial structure for our `AnimatedSprite` class. In the *Graphics* folder within the *MonoGameLibrary* project, add a new file named `AnimatedSprite.cs`:

```
using System;
using Microsoft.Xna.Framework;

namespace MonoGameLibrary.Graphics;

public class AnimatedSprite : Sprite
{

}
```

AnimatedSprite Members

An animated sprite needs to track both its current animation state and timing information. Add the following members to the `AnimatedSprite` class:

```
private int _currentFrame;
private TimeSpan _elapsed;
private Animation _animation;

/// <summary>
/// Gets or Sets the animation for this animated sprite.
/// </summary>
public Animation Animation
{
    get => _animation;
    set
    {
        _animation = value;
        Region = _animation.Frames[0];
    }
}
```

The class uses three private fields to manage its animation state:

- `_currentFrame`: Tracks which frame of the animation is currently being displayed.
- `_elapsed`: Keeps track of how much time has passed since the last frame change.
- `_animation`: Stores the current animation being played.

The `Animation` property provides access to the current animation and ensures the sprite always starts with the first frame when a new animation is set. When you assign a new animation, the property's setter automatically updates the sprite's region to display the first frame of that animation.

> ⓘ **NOTE**
>
> Starting with the first frame when setting a new animation ensures consistent behavior when switching between different animations.

AnimatedSprite Constructors

The `AnimatedSprite` class provides two ways to create an animated sprite.

Add the following constructors:

```
/// <summary>
/// Creates a new animated sprite.
/// </summary>
public AnimatedSprite() { }

/// <summary>
/// Creates a new animated sprite with the specified frames and delay.
/// </summary>
/// <param name="animation">The animation for this animated sprite.</param>
public AnimatedSprite(Animation animation)
{
    Animation = animation;
}
```

- The default constructor creates an empty animated sprite that can be configured later.
- The parameterized constructor creates an animated sprite with a specified animation, which automatically sets the sprite's initial region to the first frame of that animation through the `Animation` property.

> **NOTE**
>
> Both constructors inherit from the base `Sprite` class, so an `AnimatedSprite` will have all the same rendering properties (position, rotation, scale, etc.) as a regular sprite.

AnimatedSprite Methods

The `AnimatedSprite` class needs a way to update its animation state over time. This is handled by adding an `Update` method:

```
/// <summary>
/// Updates this animated sprite.
/// </summary>
/// <param name="gameTime">A snapshot of the game timing values provided by the
framework.</param>
public void Update(GameTime gameTime)
{
    _elapsed += gameTime.ElapsedGameTime;

    if (_elapsed >= _animation.Delay)
    {
        _elapsed -= _animation.Delay;
        _currentFrame++;

        if (_currentFrame >= _animation.Frames.Count)
        {
            _currentFrame = 0;
        }

        Region = _animation.Frames[_currentFrame];
    }
}
```

The `Update` method manages the animation timing and frame progression:

1. Accumulates the time passed since the last update in `_elapsed`.
2. When enough time has passed (defined by the animation's delay):
    - Resets the elapsed time counter
    - Advances to the next frame
    - Loops back to the first frame if we have reached the end
    - Updates the sprite's region to display the current frame

> **NOTE**
> Unlike the `Sprite` class which only needs a `Draw` method, the `AnimatedSprite` requires this additional `Update` method to handle frame changes over time. This follows MonoGame's update/draw pattern we first saw in Chapter 03

The `Draw` method inherited from the base `Sprite` class remains unchanged, as it will automatically use whatever frame is currently set as the sprite's region.

Creating AnimatedSprites With The TextureAtlas Class

Similar to the update we did to the `TextureAtlas` class in Chapter 08, creating an `AnimatedSprite` from the atlas would require

1. Get the animation by name.
2. Store it in a variable.
3. Create a new animated sprite with that animation.

We can simplify this process by adding an animated sprite creation method to the `TextureAtlas` class. Open `TextureAtlas.cs` and add the following method:

```
/// <summary>
/// Creates a new animated sprite using the animation from this texture atlas with
the specified name.
/// </summary>
/// <param name="animationName">The name of the animation to use.</param>
/// <returns>A new AnimatedSprite using the animation with the specified
name.</returns>
public AnimatedSprite CreateAnimatedSprite(string animationName)
{
    Animation animation = GetAnimation(animationName);
    return new AnimatedSprite(animation);
}
```

Using the AnimatedSprite Class

We can now adjust our game to use the `AnimatedSprite` class to see our sprites come to life. Update the contents of `Game1.cs` with the following:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
```

```
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, Vector2.One);

        // Draw the bat sprite 10px to the right of the slime.
        _bat.Draw(SpriteBatch, new Vector2(_slime.Width + 10, 0));

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

Here are the key changes in this implementation:

- The `_slime` and `_bat` members were changed from `Sprite` to `AnimatedSprite`.
- In **LoadContent** the `_slime` and `_bat` sprites are now created using the new `TextureAtlas.CreateAnimatedSprite` method.
- In **Update**, the animations are updated based on the game time using the `AnimatedSprite.Update` method.

Running the game now shows both sprites animating automatically:

- The slime bounces between two frames
- The bat's wings flap in a continuous cycle

Figure 9-2: The slime and bat sprite animating

Conclusion

In this chapter, you accomplished the following:

- Created an `Animation` class to manage frame sequences and timing.
- Extended the `TextureAtlas` class to support animation definitions.
- Built an `AnimatedSprite` class that inherits from `Sprite`.
- Applied inheritance to add animation capabilities while maintaining existing sprite functionality.
- Used XML configuration to define animations separately from code.

Now that we can efficiently manage and render sprites and animations, in the next chapter we will start taking a look at user input.

Test Your Knowledge

1. Why did we create a separate `Animation` class instead of putting animation properties directly in `AnimatedSprite`?

   Separating animation data into its own class allows multiple `AnimatedSprite` instances to share the same animation definition. This is more efficient than each sprite having its own copy of the frame sequence and timing information.

2. What is the benefit of using `TimeSpan` for animation delays instead of float values?

   `TimeSpan` provides precise timing control and makes it easier to synchronize animations with game time. It also makes the delay values more explicit (milliseconds vs arbitrary numbers) and helps prevent timing errors.

3. Why does the `AnimatedSprite` class need an `Update` method while the base `Sprite` class does not?

   The `AnimatedSprite` needs to track elapsed time and change frames based on the animation's timing. This requires updating its state over time, while a regular sprite's appearance remains static until explicitly changed.

4. In the `TextureAtlas` XML configuration, why might you want to reuse a frame in an animation sequence, like we did with the bat animation?

   Reusing frames in an animation sequence can create smoother animations by providing transition states. In the bat animation, reusing the neutral position (bat-1) between wing movements creates a more natural flapping motion without requiring additional sprite frames.

Chapter 10: Handling Input

Learn how to handle keyboard, mouse, and gamepad input in MonoGame.

When you play a game, you need ways to control what is happening; using a keyboard or gamepad to control a character or clicking the mouse to navigate a menu, MonoGame helps us handle all these different types of controls through dedicated input classes:

- **Keyboard**: Detects which keys are being pressed.
- **Mouse**: Tracks mouse movement, button clicks, and scroll wheel use.
- **GamePad**: Manages controller input like button presses and thumbstick movement.
- **TouchPanel**: Manages touch input on devices with a touch panel such as mobile phones and tablets.

Each of these input types has a `GetState` method that, when called, checks what is happening with that device at that moment. Think of it like taking a snapshot; when you call `GetState`, MonoGame looks at that exact moment to see which buttons are pressed, where the mouse is, or how the controller is being used.

In this chapter, you will learn how to use each of these dedicated input classes to handle player input.

Chapter Video

Live Recording

Keyboard Input

The keyboard is often the primary input device for PC games, used for everything from character movement to menu navigation. MonoGame provides the **Keyboard** class to handle keyboard input, making it easy to detect which keys are being pressed at any time. Calling **Keyboard.GetState** will retrieve the current state of the keyboard as a **KeyboardState** struct.

KeyboardState Struct

The **KeyboardState** struct contains methods that can be used to determine if a keyboard key is currently down or up:

Method	Description
IsKeyDown(Keys)	Returns `true` if the specified key is down; otherwise, returns `false`.
IsKeyUp(Keys)	Returns `true` if the specified key is up; otherwise, returns `false`.

For example, if we wanted to see if the Space key is down, you could use the following:

```
// Get the current state of keyboard input.
KeyboardState keyboardState = Keyboard.GetState();

// Check if the space key is down.
if(keyboardState.IsKeyDown(Keys.Space))
{
    // The space key is down, so do something.
}
```

> **TIP**
>
> Notice we store the keyboard state in a variable instead of calling Keyboard.GetState multiple times. This is more efficient and ensures consistent input checking within a single frame.

Mouse Input

The mouse is often the secondary input device for PC games, used for various actions from camera movement to interacting with menus and objects. MonoGame provides the Mouse class to handle mouse input, making it easy to detect which buttons are pressed, the position of the mouse cursor, and the value of the scroll wheel. Calling Mouse.GetState will retrieve the current state of the mouse as a MouseState struct.

MouseState Struct

The MouseState struct contains properties that can be used to determine the state of the mouse buttons, the mouse position, and the scroll wheel value:

Property	Type	Description
LeftButton	ButtonState	Returns the state of the left mouse button.
MiddleButton	ButtonState	Returns the state of the middle mouse button. This is often the button activated by pressing the scroll wheel down.
Position	Point	Returns the position of the mouse cursor relative to the bounds of the game window.
RightButton	ButtonState	Returns the state of the right mouse button.
ScrollWheelValue	`int`	Returns the **cumulative** scroll wheel value since the start of the game
XButton1	ButtonState	Returns the state of the first extended button on the mouse.
XButton2	ButtonState	Returns the state of the second extended button on the mouse.

> **NOTE**
>
> ScrollWheelValue returns the cumulative value of the scroll wheel since the start of the game, not how much it moved since the last update. To determine how much it moved between one update and the next, you would need to compare it with the previous frame's value. We will discuss comparing previous and current frame values for inputs in the next chapter.

Unlike keyboard input which uses IsKeyDown(Keys)/IsKeyUp(Keys) methods mouse buttons return a ButtonState:

- ButtonState.Pressed: The button is being held down.
- ButtonState.Released: The button is not being pressed.

For example, if we wanted to see if the left mouse button is down, you could use the following

```
// Get the current state of mouse input.
MouseState mouseState = Mouse.GetState();

// Check if the left mouse button is pressed down.
if(mouseState.LeftButton == ButtonState.Pressed)
{
    // The left button is down, so do something.
}
```

Gamepad Input

Gamepads are often used as a primary input for a game or an alternative for keyboard and mouse controls. MonoGame provides the **GamePad** class to handle gamepad input, making it easy to detect which buttons are pressed and the value of the thumbsticks. Calling **GamePad.GetState** will retrieve the state of the gamepad as a **GamePadState** struct. Since multiple gamepads can be connected, you will need to supply a **PlayerIndex** value to specify which gamepad state to retrieve.

GamePadState Struct

The **GamePadState** struct and properties that can be used to get the state of the buttons, dpad, triggers, and thumbsticks:

Property	Type	Description
Buttons	GamePadButtons	Returns a struct that identifies which buttons on the controller are pressed.
DPad	GamePadDPad	Returns a struct that identifies which directions on the DPad are pressed.
IsConnected	`bool`	Returns a value that indicates whether the controller is connected.
ThumbSticks	GamePadThumbSticks	Returns a struct that contains the direction of each thumbstick. Each thumbstick (left and right) are represented as a Vector2 value between `-1.0f` and `1.0` for the x- and y-axes.
Triggers	GamePadTriggers	Returns a struct that contains the value of each trigger. Each trigger (left and right) are represented as a `float` value between `0.0f`, meaning not pressed, and `1.0f`, meaning fully pressed.

Buttons

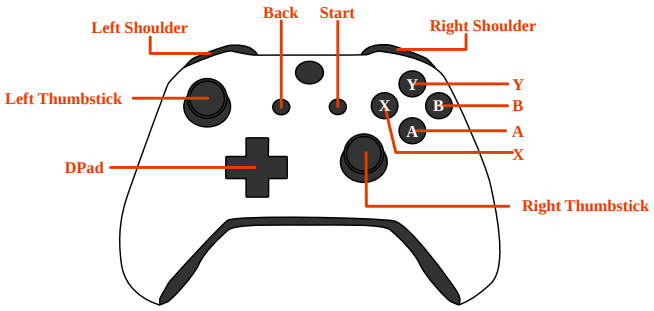
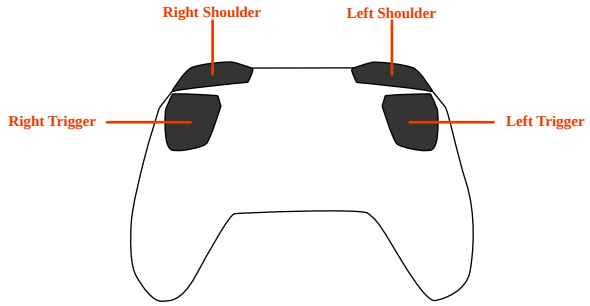
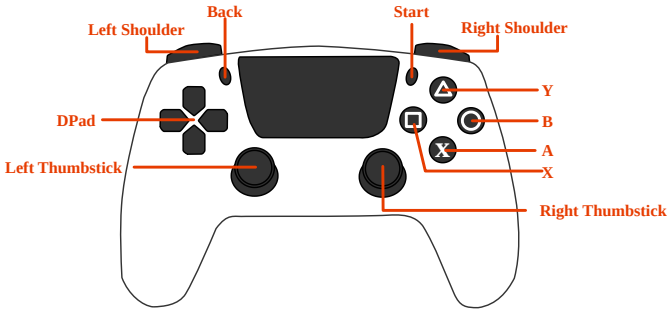
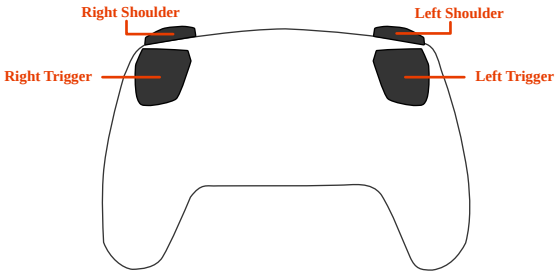
The GamePadState.Buttons property returns a GamePadButtons struct that can be used to identify which buttons on the controller are pressed. This struct contains the following properties:

Property	Type	Description
A	ButtonState	Returns the state of the A button
B	ButtonState	Returns the state of the B button
Back	ButtonState	Returns the state of the Back button
BigButton	ButtonState	Returns the state of the BigButton button
LeftShoulder	ButtonState	Returns the state of the LeftShoulder button
LeftStick	ButtonState	Returns the state of the LeftStick button
RightShoulder	ButtonState	Returns the state of the RightShoulder button
RightStick	ButtonState	Returns the state of the RightStick button
Start	ButtonState	Returns the state of the Start button
X	ButtonState	Returns the state of the X button
Y	ButtonState	Returns the state of the Y button

NOTE

Recall from Chapter 01 that MonoGame is a implementation the XNA API. Since XNA was originally created for making games on Windows PC and Xbox 360, the names of the gamepad buttons match those of an Xbox 360 controller.

The **BigButton** refers to the large, centrally located button on special Xbox 360 controllers created for games like "Scene It?" - this button is not present on standard controllers and is not mapped to any button on modern controllers. It remains in the API for backward compatibility with XNA.

Front	Back
Xbox	
	
Playstation	
	

Like with the mouse input, each of these buttons are represented by a **ButtonState** enum value. For instance, if you wanted to check if the A button is being pressed you could do the following:

```
// Get the current state of player one's gamepad.
GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

// Check if the A button is pressed down.
if(gamePadState.Buttons.A == ButtonState.Pressed)
{
    // Button A is pressed, do something.
}
```

> **NOTE**
>
> You may notice however, that the GamePadState also requires a controller index, as more than one can be connected at the same time. The latest Xbox console for instance can support up to 8 controllers at a time, for this reason you need to specify which controller you are listening for. Additionally, if you want ANY controller to start your game, you will need to loop through all possible controllers each frame until the first one "picks up".

DPad

The **DPad** property returns a **GamePadDPad** struct that can be used to identify which DPad buttons on the controller are pressed. This struct contains the following properties:

Property	Type	Description
Down	**ButtonState**	Returns the state of the DPad Down button.
Left	**ButtonState**	Returns the state of the DPad Left button.
Right	**ButtonState**	Returns the state of the DPad Right button.
Up	**ButtonState**	Returns the state of the DPad Up Button.

Like with the Buttons, these also return a **ButtonState** enum value to represent the state of the DPad button. For instance, if you wanted to check if the DPad up button is being pressed, you could do the following:

```
// Get the current state of player one's gamepad.
GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

// Check if the down on the DPad is pressed.
if(gamePadState.DPad.Down == ButtonState.Pressed)
{
    // DPad down is pressed, do something.
}
```

Thumbsticks

The **ThumbSticks** property returns a **GamePadThumbSticks** struct that can be used to retrieve the values of the left and right thumbsticks. This struct contains the following properties:

Property	Type	Description
Left	**Vector2**	The direction the left thumbstick is pressed.
Right	**Vector2**	The direction the right thumbstick is pressed.

The thumbstick values are represented as a **Vector2** value:

- X-axis: A value between `-1.0f` (pushed fully to the left) and `1.0f` (pushed fully to the right).
- Y-axis: A value between `-1.0f` (pushed fully downward) and `1.0f` (pushed fully upward).

For example, if you wanted to move a sprite using the left thumbstick, you could do the following:

```
// Get the current state of player one's gamepad.
GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

// Get the value of the left thumbstick.
Vector2 leftStick = gamePadState.Thumbsticks.Left;

// Invert the y-axis value.
leftStick.Y *= -1.0f;

// Apply the value to the position of the sprite.
sprite.Position += leftStick;
```

IMPORTANT

Notice that we inverted the y-axis value of the thumbstick by multiplying it by `-1.0f`. This is necessary because the thumbstick y-axis values range from `-1.0f` (down) to `1.0f` (up). The y-axis of the screen coordinates in MonoGame **increases** downward, as we saw in Chapter 06.

This inversion aligns the thumbstick's y-axis value with the screen movement.

Triggers

The **Triggers** property returns a **GamePadTriggers** struct that can be used to retrieve the values of the left and right triggers. This struct contains the following properties:

Property	Type	Description
Left	`float`	The value of the left trigger.
Right	`float`	The value of the left trigger.

The trigger values are represented as a float value between `0.0f` (not pressed) to `1.0f` (fully pressed). The triggers on a gamepad, however, can be either *analog* or *digital* depending on the gamepad manufacturer. For gamepads with *digital* triggers, the value will always be either `0.0f` or `1.0f`, as a digital trigger does not register values in between based on the amount of pressure applied to the trigger.

For example, if we were creating a racing game, the right trigger could be used for acceleration like the following:

```
// Get the current state of player one's gamepad.
GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

// Get the acceleration based on how far the right trigger is pushed down.
float acceleration = gamePadState.Triggers.Right;
```

GamePadState Methods

The **GamePadState** struct also contains two methods that can be used to get information about the device's inputs as either being up or down:

Method	Description
IsButtonDown(Buttons)	Returns a value that indicates whether the specified button is down. Multiple Buttons values can be given using the bitwise OR | operator. When multiple buttons are given, the return value indicates if all buttons specified are down, not just one of them.
IsButtonUp(Buttons)	Returns a value that indicates whether the specified button is up. Multiple Buttons values can be given using the bitwise OR | operator. When multiple buttons are given, the return value indicates if all buttons specified are up, not just one of them.

You can use the IsButtonDown(Buttons) and IsButtonUp(Buttons) methods to get the state of all buttons, including the DPad. The following is a complete list of all of the Buttons enum values:

- Buttons.A
- Buttons.B
- Buttons.Back
- Buttons.BigButton
- Buttons.DPadDown
- Buttons.DPadLeft
- Buttons.DPadRight
- Buttons.DPadUp
- Buttons.LeftShoulder
- Buttons.LeftStick
- Buttons.LeftThumbstickDown
- Buttons.LeftThumbstickLeft
- Buttons.LeftThumbstickRight
- Buttons.LeftThumbstickUp
- Buttons.LeftTrigger
- Buttons.None
- Buttons.RightShoulder
- Buttons.RightStick
- Buttons.RightStickDown
- Buttons.RightStickLeft
- Buttons.RightStickRight
- Buttons.RightStickUp
- Buttons.RightTrigger
- Buttons.Start
- Buttons.X
- Buttons.Y

> **CAUTION**
>
> While you can use these methods to get the state of any of these button inputs, the state will only tell you if it is being pressed or released. For the actual thumbstick values and trigger values, you would need to use the properties instead.

For example, if we wanted to check if the A button on the first gamepad is pressed, you could use the following:

```
// Get the current state of player one's gamepad.
GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

// Check if the A button is down.
if(gamePadState.IsButtonDown(Buttons.A))
{
    // The A button is pressed, do something.
}
```

GamePad Vibration

Another capability of gamepads is haptic feedback through vibration motors. MonoGame allows you to control this feature using the **GamePad.SetVibration** method. This method takes three parameters:

1. The **PlayerIndex** of the gamepad to vibrate.
2. The intensity of the left motor (from `0.0f` for no vibration to `1.0f` for maximum vibration).
3. The intensity of the right motor (using the same scale).

Most modern gamepads have two vibration motors, a larger one (usually the left motor) for low-frequency rumble and a smaller one (usually the right motor) for high-frequency feedback. By controlling these independently, you can create various haptic effects:

```
// Make the gamepad vibrate at full intensity.
GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);

// Stop all vibration.
GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);

// Create a subtle, low-intensity vibration.
GamePad.SetVibration(PlayerIndex.One, 0.3f, 0.1f);
```

TouchPanel Input

For mobile devices such as Android/iOS phones and tablets, the primary input device is the touch panel screen. Touching a location on the screen is similar to clicking a location on your computer with a mouse. MonoGame provides the **TouchPanel** class to handle touch input.

The **TouchPanel** class offers two ways of retrieving information about touch input:

- **TouchPanel.GetState** retrieves a **TouchCollection** struct that contains **TouchLocation** values for each point of touch on the touch panel.
- **TouchPanel.ReadGesture** retrieves a **GestureSample** struct that contains information about recent gestures that have been performed like a vertical or horizontal drag across the screen.

TouchCollection

When calling **TouchPanel.GetState** a **TouchCollection** struct is returned. This collection contains a **TouchLocation** value for each point of touch.

TouchLocation

Each **TouchLocation** value in a touch collection contains the following properties:

Property	Type	Description
Id	`int`	The id of the touch location.
Position	**Vector2**	The position of the touch location.
Pressure	`float`	The amount of pressure applied at the touch location. **(Only available for Android devices.)**
State	**TouchLocationState**	The current state of the touch location.

The important properties of the location are the **Position** and the **State** The position property will tell us the location of the touch event, and the state can be one of the following values:

State	Description
Invalid	This touch location position is invalid.
Moved	This touch location position was updated or pressed at the same position.
Pressed	This touch location was pressed.

State	Description
Released	This touch location was released.

When the state is moved or pressed, then we know that location on the touch panel is being touched. So we can capture it and use it like the following:

```
// Get the current state of touch input.
TouchCollection touchCollection = TouchPanel.GetState();

foreach(TouchLocation touchLocation in touchCollection)
{
    if(touchLocation.State == TouchLocationState.Pressed || touchLocation.State
== TouchLocationState.Moved)
    {
        // The the location at touchLocation.Position is currently being pressed,
        // so we can act on that information.
    }
}
```

> **NOTE**
>
> Unlike mouse input which only tracks a single point, **TouchPanel** supports multiple simultaneous touch points. The **TouchCollection** contains all active touch points, which is why we loop through them in the sample above.

The state of a touch location progresses through the states typically in order of:

- **Pressed**: Initial contact with the screen.
- **Moved** : Touch point moved while maintaining contact.
- **Released**: Contact with screen ended.
- **Invalid** : Touch data is invalid (used when tracking data is lost).

GestureSample

When calling **TouchPanel.ReadGesture** a **GestureSample** struct containing the information about recent gestures that have been performed is returned. The **GestureSample** struct contains the following properties:

Property	Type	Description
Delta	Vector2	Gets the delta information about the first touch-point in the gesture sample.
Delta2	Vector2	Gets the delta information about the second touch-point in the gesture sample.
GestureType	GestureType	Gets the type of the gesture.
Position	Vector2	Gets the position of the first touch-point in the gesture sample.
Position2	Vector2	Gets the position of the second touch-point in the gesture sample.

> **NOTE**
> Gestures have two delta properties and two position properties. This is because some gestures require multiple touch inputs to perform, such as performing a pinch to zoom in or out. You would need the location of both touch points to determine the correct zoom to apply during the gesture.

To determine what type of gesture is performed, we can get that from the GestureType property which will be one of the following values:

Gesture Type	Description
DoubleTap	The user double tapped the device twice which is always preceded by a Tap gesture.
DragComplete	States completion of a drag gesture (VerticalDrag, HorizontalDrag, or FreeDrag).
Flick	States that a touch was combined with a quick swipe.
FreeDrag	The user touched a point and then performed a free-form drag.
Hold	The user touched a single point for approximately one second.
HorizontalDrag	The user touched the screen and performed either a left-to-right or right-to-left drag gesture.
None	No gesture.
Pinch	The user converged or diverged two touch-points on the screen which is like a two-finger drag.
PinchComplete	An in-progress pinch gesture was completed.
Tap	The user touched a single point.
VerticalDrag	The user touched the screen and performed either a top-to-bottom or bottom-to-top drag gesture.

IMPORTANT

Before gestures can be detected, they have to be enabled using **TouchPanel.EnabledGestures**. This can be done in **Game.Initialize** like the following:

```
protected override void Initialize()
{
    base.Initialize();

    // Enable gestures we want to handle.
    TouchPanel.EnabledGestures =
        GestureType.Tap |
        GestureType.HorizontalDrag |
        GestureType.VerticalDrag;
}
```

The following is an example of using a gesture to detect horizontal and vertical drags:

```
while(TouchPanel.IsGestureAvailable)
{
    GestureSample gesture = TouchPanel.ReadGesture();

    if(gesture.GestureType == GestureType.HorizontalDrag)
    {
        // A horizontal drag from left-to-right or right-to-left occurred.
        // You can use the Delta property to determine how much movement
        // occurred during the swipe.
        float xDragAmount = gesture.Delta.X;

        // Now do something with that information.
    }

    if(gesture.GestureType == GestureType.VerticalDrag)
    {
        // A vertical drag from top-to-bottom or bottom-to-top occurred.
        // You can use the Delta property to determine how much movement
        // occurred during the swipe.
        float yDragAmount = gesture.Delta.Y;

        // Now do something with that information.
    }
}
```

IMPORTANT

Notice above that we use a `while` loop with **TouchPanel.IsGestureAvailable** as the condition for the loop. The reason we do this is because when a user performs a gesture, such as a horizontal drag across the screen, very quickly, what can often occurs is a series of multiple small drag gestures are registered and queued.

Each time **TouchPanel.ReadGesture** is called, it will dequeue the next gesture. So to ensure that we handle the complete gesture, we loop the gesture queue until there are none left.

Implementing Input in Our Game

For our game, we are going to implement keyboard and gamepad controls based on the following criteria:

Keyboard Input	Gamepad Input	Description
[Keys.W] and [Keys.Up]	[Thumbstick.Left.Y] and [Buttons.DPadUp]	Moves the slime up the screen.
[Keys.S] and [Keys.Down]	[Thumbstick.Left.Y] and [Buttons.DPadDown]	Moves the slime down the screen
[Keys.A] and [Keys.Left]	[Thumbstick.Left.X] and [Buttons.DPadLeft]	Moves the slime left on the screen.
[Keys.D] and [Keys.Right]	[Thumbstick.Left.X] and [Buttons.DPadRight]	Moves the slime right on the screen.
[Keys.Space]	[Buttons.A]	Increased the speed of the slime.

Open `Game1.cs` in the *DungeonSlime* project and update it with the following:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
```

```
        _bat.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
            Exit();

        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        // Check for keyboard input and handle it.
        CheckKeyboardInput();

        // Check for gamepad input and handle it.
        CheckGamePadInput();

        base.Update(gameTime);
    }

    private void CheckKeyboardInput()
    {
        // Get the state of keyboard input
        KeyboardState keyboardState = Keyboard.GetState();

        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (keyboardState.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (keyboardState.IsKeyDown(Keys.W) || keyboardState.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (keyboardState.IsKeyDown(Keys.S) || keyboardState.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
```

```
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (keyboardState.IsKeyDown(Keys.A) || keyboardState.IsKeyDown(Keys.Left))
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (keyboardState.IsKeyDown(Keys.D) || keyboardState.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }
    }

    private void CheckGamePadInput()
    {
        GamePadState gamePadState = GamePad.GetState(PlayerIndex.One);

        // If the A button is held down, the movement speed increases by 1.5
        // and the gamepad vibrates as feedback to the player.
        float speed = MOVEMENT_SPEED;
        if (gamePadState.IsButtonDown(Buttons.A))
        {
            speed *= 1.5f;
            GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
        }
        else
        {
            GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
        }

        // Check thumbstick first since it has priority over which gamepad input
        // is movement.  It has priority since the thumbstick values provide a
        // more granular analog value that can be used for movement.
        if (gamePadState.ThumbSticks.Left != Vector2.Zero)
        {
            _slimePosition.X += gamePadState.ThumbSticks.Left.X * speed;
            _slimePosition.Y -= gamePadState.ThumbSticks.Left.Y * speed;
        }
        else
        {
            // If DPadUp is down, move the slime up on the screen.
            if (gamePadState.IsButtonDown(Buttons.DPadUp))
            {
                _slimePosition.Y -= speed;
```

```
            }

            // If DPadDown is down, move the slime down on the screen.
            if (gamePadState.IsButtonDown(Buttons.DPadDown))
            {
                _slimePosition.Y += speed;
            }

            // If DPapLeft is down, move the slime left on the screen.
            if (gamePadState.IsButtonDown(Buttons.DPadLeft))
            {
                _slimePosition.X -= speed;
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadState.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite 10px to the right of the slime.
        _bat.Draw(SpriteBatch, new Vector2(_slime.Width + 10, 0));

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes made here are:

1. The `_slimePosition` field was added to track the position of the slime as it moves.
2. The `MOVEMENT_SPEED` constant was added to use as the base multiplier for the movement speed.
3. The `CheckKeyboardInput` method was added which checks for input from the keyboard based on the input table above and moves the slime based on the keyboard input detected.
4. The `CheckGamePadInput` method was added which checks for input from the gamepad based on the input table above and moves the slime based the gamepad input detected.

> **NOTE**
>
> The gamepad implementation includes a priority system for directional input. The code prioritizes the analog thumbstick values over the digital DPad buttons. This design choice provides players with more nuanced control, as analog inputs allow for a variable movement speed based on how far the thumbstick is pushed, while DPad buttons only provide on/off input states. The code first checks if either thumbstick axis has a non-zero value, and only falls back to DPad input when the thumbstick is centered.
>
> To enhance player experience, the gamepad implementation also includes gamepad vibration when the speed boost is activated. Haptic feedback like this creates a more immersive experience by engaging additional senses for the player beyond just visual and auditory feedback.

5. In **Update** `CheckKeyboardInput` and `CheckGamePadInput` methods are called.
6. In **Draw**, the slime is now drawn using `_slimePosition` as the position.

Running the game now, you can move the slime around using the keyboard with the arrow keys or WASD keys. If you have a gamepad plugged in you can also use the DPad and left thumbstick.

Figure 10-1: The slime moving around based on device input

> **NOTE**
>
> You may notice that the slime is capable of moving completely off the screen, this is completely normal as we have not yet implemented any logic to prevent it from doing so, it is only doing what we currently tell it to do.

Input Buffering

While checking for input every frame works well for continuous actions like movement, many games benefit from more sophisticated input handling techniques. One such technique is **input buffering**, which can significantly improve how responsive controls feel to players.

Understanding Input Buffering

Input buffering is a technique where the game temporarily stores player inputs that cannot be immediately processed. Instead of discarding these inputs, they are placed in a queue and processed in order when the game is ready to handle them.

Input buffering is particularly valuable in games where:

- Actions occur at fixed intervals rather than continuously (like turn-based games or grid movement).
- Precise timing is required for complex input sequences (like fighting games).
- Multiple rapid inputs need to be remembered in order (like quick directional changes).

Without input buffering, players must time their inputs perfectly to align with the game's update cycle. With buffering, the game becomes more forgiving and responsive by:

1. Storing inputs that arrive between action updates.
2. Preserving the order of inputs for more predictable behavior.
3. Creating a sense that the game is actually listening to the player.

Implementing a Simple Input Buffer

A basic input buffer can be implemented using a queue data structure, which follows a First-In-First-Out (FIFO) pattern:

```
// Use a queue directly for input buffering
private Queue<Vector2> _inputBuffer;
private const int MAX_BUFFER_SIZE = 2;

// In initialization code:
_inputBuffer = new Queue<Vector2>(MAX_BUFFER_SIZE);

// In the input handling code:
KeyboardState keyboard = Keyboard.GetState();
Vector2 newDirection = Vector2.Zero;

if(keyboard.IsKeyDown(Keys.Up))
{
    newDirection = -Vector2.UnitY;
}
else if(keyboard.IsKeyDown(Keys.Down))
{
    newDirection = Vector2.UnitY;
}
else if(keyboard.IsKeyDown(Keys.Left))
{
    newDirection = -Vector2.UnitX;
}
else if(keyboard.IsKeyDown(Keys.Right))
{
    newDirection = Vector2.UnitX;
}

// Only add if a valid direction and does not exceed the buffer size.
if(newDirection != Vector2.Zero && _inputBuffer.Count < MAX_BUFFER_SIZE)
{
    _inputBuffer.Enqueue(newDirection);
}

// In movement update code.
if(_inputBuffer.Count > 0)
{
    Vector2 nextDirection = _inputBuffer.Dequeue();
    _position += nextDirection * _speed;
}
```

> **NOTE**
>
> The `Queue<T>` is a First In, First Out (FIFO) collection in C#. When you add items with `Enqueue()`, they join the end of the line, and when you retrieve items with `Dequeue()`, you always get the oldest item (the one at the front of the line). Think of it like people waiting in line - the first person to arrive is the first one served.
>
> This contrasts with a `Stack<T>`, which follows Last In, First Out (LIFO) behavior, where the most recently added item is the first one retrieved.

The size of an input buffer is an important design decision. If it is too small, players might still feel the game is not responsive enough. If it is too large, the game might feel like it is playing itself by working through a backlog of commands.

When to Use Input Buffering

Consider implementing input buffering in your game when:

- Players complain about the game feeling "unresponsive".
- Your game uses fixed-interval updates for certain mechanics.
- Actions require precise timing that is difficult for players to hit consistently.
- You want to allow players to "queue up" their next few moves.

We will see a practical implementation of input buffering in Chapter 23 when we finalize our snake-like game mechanics, where timing and direction changes are critical to gameplay.

Conclusion

In this chapter, you accomplished the following:

- Handle keyboard input to detect key presses.
- Handle mouse input including button clicks and cursor position.
- Work with gamepad controls including buttons, thumbsticks, and vibration.
- Understand touch input for mobile devices including touch points and gestures.
- Implement movement controls using different input methods.
- Consider controller-specific details like coordinate systems and analog vs digital input.

In the next chapter, we will learn how to track previous input states to handle single-press events and implement an input management system to simplify some of the complexity of handling input.

Test Your Knowledge

1. Why do we store the result of `GetState` in a variable instead of calling it multiple times?

> Storing the state in a variable is more efficient and ensures consistent input checking within a frame. Each `GetState` call polls the device, which can impact performance if called repeatedly.

2. What is the main difference between how keyboard and mouse/gamepad button states are checked?

> Keyboard input uses **IsKeyUp**/**IsKeyDown** methods, while mouse and gamepad buttons return a **ButtonState** enum value (Pressed or Released).

3. When using thumbstick values for movement, why do we multiply the Y value by -1?

> The thumbstick Y-axis values (-1.0f down to 1.0f up) are inverted compared to MonoGame's screen coordinate system (Y increases downward). Multiplying by -1 aligns the thumbstick direction with screen movement.

4. What is the difference between analog and digital trigger input on a gamepad?

> Analog triggers provide values between 0.0f and 1.0f based on how far they are pressed, while digital triggers only report 0.0f (not pressed) or 1.0f (pressed). This affects how you handle trigger input in your game.

5. What is the key difference between **TouchPanel.GetState** and **TouchPanel.ReadGesture**?

> **TouchPanel.GetState** returns information about current touch points on the screen, while **TouchPanel.ReadGesture** provides information about specific gesture patterns like taps, drags, and pinches that have been performed.

6. Why do we use a while loop with **TouchPanel.IsGestureAvailable** when reading gestures?

> Quick gestures can generate multiple gesture events that are queued. Using a while loop with **TouchPanel.IsGestureAvailable** ensures we process all queued gestures, as **TouchPanel.ReadGesture** only returns one gesture at a time.

7. How does touch input differ from mouse input in terms of handling multiple input points?

> Touch input can handle multiple simultaneous touch points through the **TouchCollection**, while mouse input only tracks a single cursor position. This allows touch input to support features like multi-touch gestures that are not possible with a mouse.

8. What are the different states a **TouchLocation** can have and what do they indicate?

> A **TouchLocation** can have four states:
>
> - **Pressed**: Initial contact with the screen
> - **Moved**: Touch point moved while maintaining contact
> - **Released**: Contact with the screen ended
> - **Invalid**: Touch data is not valid or tracking was lost

Chapter 11: Input Management

Learn how to create an input management system to handle keyboard, mouse, and gamepad input, including state tracking between frames and creating a reusable framework for handling player input.

In Chapter 10, you learned how to handle input from various devices like keyboard, mouse, and gamepad. While checking if an input is currently down works well for continuous actions like movement, many game actions should only happen once when an input is first pressed; think firing a weapon or jumping. To handle these scenarios, we need to compare the current input state with the previous frame's state to detect when an input changes from up to down.

In this chapter you will:

- Learn the difference between an input being down versus being pressed
- Track input states between frames
- Create a reusable input management system
- Simplify handling input across multiple devices

We will start by understanding the concept of input state changes and how we can detect them.

Understanding Input States

When handling input in games, there are two key scenarios we need to consider:

- An input is being held down (like holding a movement key).
- An input was just pressed for one frame (like pressing a jump button).

Now, we will look at the difference using keyboard input as an example. With our current implementation, we can check if a key is down using **KeyboardState.IsKeyDown**:

```
// Get the current state of keyboard input.
KeyboardState keyboardState = Keyboard.GetState();

// Check if the space key is down.
if (keyboardState.IsKeyDown(Keys.Space))
{
    // This runs EVERY frame the space key is held down
}
```

However, many game actions should not repeat while a key is held. For instance, if the Space key makes your character jump, you probably do not want them to jump repeatedly just because the player is holding the key down. Instead, you want the jump to happen only on the first frame when Space is pressed.

To detect this "just pressed" state, we need to compare two states:

1. Is the key down in the current frame?
2. Was the key up in the previous frame?

If both conditions are true, we know the key was just pressed. If we were to modify the above code to track the previous keyboard state it would look something like this:

```
// Track the state of keyboard input during the previous frame.
private KeyboardState _previousKeyboardState;

protected override void Update(GameTime gameTime)
{
    // Get the current state of keyboard input.
    KeyboardState keyboardState = Keyboard.GetState();

    // Compare if the space key is down on the current frame but was up on the
previous frame.
    if (keyboardState.IsKeyDown(Keys.Space) &&
_previousKeyboardState.IsKeyUp(Keys.Space))
    {
        // This will only run on the first frame Space is pressed and will not
        // happen again until it has been released and then pressed again.
    }

    // At the end of update, store the current state of keyboard input into the
    // previous state tracker.
    _previousKeyboardState = keyboardState;

    base.Update(gameTime);
}
```

If you need to know the inverse state, when the key was just released, then it is simply a matter of switching the checking of the states, for example, is the key up this frame and was it down in the previous frame.

This same concept applies to mouse buttons and gamepad input as well. Any time you need to detect a "just pressed" or "just released" state, you will need to compare the current input state with the previous frame's state.

So far, we have only been working with our game within the `Game1.cs` file. This has been fine for the examples given. Over time, as the game grows, we are going to have a more complex system set up with different scenes, and each scene will need a way to track the state of input over time. We could do this by creating a lot of variables in each scene to track this information, or we can use object-oriented design concepts to create a reusable `InputManager` class to simplify this for us.

Before we create the `InputManager` class, we should first create classes for the keyboard, mouse, and gamepad that encapsulates the information about those inputs, which will then be exposed through the `InputManager`.

To get started, create a new folder called `Input` in the *MonoGameLibrary* project. We will put all of our input related classes here.

The KeyboardInfo Class

We will start our input management system by creating a class to handle keyboard input. The `KeyboardInfo` class will encapsulate all keyboard-related functionality, making it easier to:

- Track current and previous keyboard states
- Detect when keys are pressed or released
- Check if keys are being held down

In the `Input` folder of the *MonoGameLibrary* project, add a new file named `KeyboardInfo.cs` with this initial structure:

```
using Microsoft.Xna.Framework.Input;

namespace MonoGameLibrary.Input;

public class KeyboardInfo { }
```

KeyboardInfo Properties

To detect changes in keyboard input between frames, we need to track both the previous and current keyboard states. Add these properties to the `KeyboardInfo` class:

```
/// <summary>
/// Gets the state of keyboard input during the previous update cycle.
/// </summary>
public KeyboardState PreviousState { get; private set; }

/// <summary>
/// Gets the state of keyboard input during the current input cycle.
/// </summary>
public KeyboardState CurrentState { get; private set; }
```

> **NOTE**
>
> These properties use a public getter but private setter pattern. This allows other parts of the game to read the keyboard states if needed, while ensuring only the `KeyboardInfo` class can update them.

KeyboardInfo Constructor

The `KeyboardInfo` class constructor needs to initialize the keyboard states.

Add this constructor:

```
/// <summary>
/// Creates a new KeyboardInfo.
/// </summary>
public KeyboardInfo()
{
    PreviousState = new KeyboardState();
    CurrentState = Keyboard.GetState();
}
```

The constructor:

- Creates an empty state for `PreviousState` since there is no previous input yet
- Gets the current keyboard state as our starting point for `CurrentState`

This initialization ensures we have valid states to compare against in the first frame of our game, preventing any potential null reference issues when checking for input changes.

KeyboardInfo Methods

The `KeyboardInfo` class needs methods both for updating states and checking key states. First, we will start with our update method:

```
/// <summary>
/// Updates the state information about keyboard input.
/// </summary>
public void Update()
{
    PreviousState = CurrentState;
    CurrentState = Keyboard.GetState();
}
```

> **NOTE**
>
> Each time `Update` is called, the current state becomes the previous state, and we get a fresh current state. This creates our frame-to-frame comparison chain.

Next, we will add methods to check various key states:

```
/// <summary>
/// Returns a value that indicates if the specified key is currently down.
/// </summary>
/// <param name="key">The key to check.</param>
/// <returns>true if the specified key is currently down; otherwise,
false.</returns>
public bool IsKeyDown(Keys key)
{
    return CurrentState.IsKeyDown(key);
}

/// <summary>
/// Returns a value that indicates whether the specified key is currently up.
/// </summary>
/// <param name="key">The key to check.</param>
/// <returns>true if the specified key is currently up; otherwise, false.</returns>
public bool IsKeyUp(Keys key)
{
    return CurrentState.IsKeyUp(key);
}

/// <summary>
/// Returns a value that indicates if the specified key was just pressed on the
current frame.
/// </summary>
/// <param name="key">The key to check.</param>
/// <returns>true if the specified key was just pressed on the current frame;
otherwise, false.</returns>
public bool WasKeyJustPressed(Keys key)
{
    return CurrentState.IsKeyDown(key) && PreviousState.IsKeyUp(key);
}

/// <summary>
/// Returns a value that indicates if the specified key was just released on the
current frame.
/// </summary>
/// <param name="key">The key to check.</param>
/// <returns>true if the specified key was just released on the current frame;
otherwise, false.</returns>
public bool WasKeyJustReleased(Keys key)
{
    return CurrentState.IsKeyUp(key) && PreviousState.IsKeyDown(key);
}
```

These methods serve two distinct purposes. For checking continuous states:

- `IsKeyDown`: Returns true as long as the specified key is being held down.
- `IsKeyUp`: Returns true as long as the specified key is not being pressed.

And for detecting state changes:

- `WasKeyJustPressed`: Returns true only on the frame when the specified key changes from up-to-down.
- `WasKeyJustReleased`: Returns true only on the frame when the specified key changes from down-to-up.

> **TIP**
> Use continuous state checks (`IsKeyDown`/`IsKeyUp`) for actions that should repeat while a key is held, like movement. Use single-frame checks (`WasKeyJustPressed`/`WasKeyJustReleased`) for actions that should happen once per key press, like jumping or shooting.

That's it for the `KeyboardInfo` class, now we can move on to mouse input next.

MouseButton Enum

Recall from the Mouse Input section of the previous chapter that the **MouseState** struct provides button states through properties rather than methods like `IsButtonDown`/`IsButtonUp`. To keep our input management API consistent across devices, we will create a `MouseButton` enum that lets us reference mouse buttons in a similar way to how we use **Keys** for keyboard input and **Buttons** for gamepad input.

In the `Input` folder of the *MonoGameLibrary* project, add a new file named `MouseButton.cs` with the following code:

```
namespace MonoGameLibrary.Input;

public enum MouseButton
{
    Left,
    Middle,
    Right,
    XButton1,
    XButton2
}
```

> **NOTE**
>
> Each enum value corresponds directly to a button property in MouseState:
>
> - `Left`: Maps to **MouseState.LeftButton**.
> - `Middle`: Maps to **MouseState.MiddleButton**.
> - `Right`: Maps to **MouseState.RightButton**.
> - `XButton1`: Maps to **MouseState.XButton1**.
> - `XButton2`: Maps to **MouseState.XButton2**.

The MouseInfo Class

To manage mouse input effectively, we need to track both current and previous states, as well as provide easy access to mouse position, scroll wheel values, and button states. The `MouseInfo` class will encapsulate all of this functionality, making it easier to:

- Track current and previous mouse states.
- Track the mouse position.
- Check the change in mouse position between frames and if it was moved.
- Track scroll wheel changes.
- Detect when mouse buttons are pressed or released
- Check if mouse buttons are being held down

To get started, in the `Input` folder of the *MonoGameLibrary* project, create a new file named `MouseInfo.cs` with the following initial structure:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;

namespace MonoGameLibrary.Input;

public class MouseInfo
{

}
```

MouseInfo Properties

The `MouseInfo` class needs properties to track both mouse states and provide easy access to common mouse information. Add the following properties to the `MouseInfo` class:

First, we need properties for tracking mouse states:

```
/// <summary>
/// The state of mouse input during the previous update cycle.
/// </summary>
public MouseState PreviousState { get; private set; }

/// <summary>
/// The state of mouse input during the current update cycle.
/// </summary>
public MouseState CurrentState { get; private set; }
```

Next, we will add properties for handling cursor position:

```
/// <summary>
/// Gets or Sets the current position of the mouse cursor in screen space.
/// </summary>
public Point Position
{
    get => CurrentState.Position;
    set => SetPosition(value.X, value.Y);
}

/// <summary>
/// Gets or Sets the current x-coordinate position of the mouse cursor in
screen space.
/// </summary>
public int X
{
    get => CurrentState.X;
    set => SetPosition(value, CurrentState.Y);
}

/// <summary>
/// Gets or Sets the current y-coordinate position of the mouse cursor in
screen space.
/// </summary>
public int Y
{
    get => CurrentState.Y;
    set => SetPosition(CurrentState.X, value);
}
```

> **NOTE**
>
> The position properties use a `SetPosition` method that we will implement later. This method will handle the actual cursor positioning on screen.

These properties provide different ways to work with the cursor position:

- `Position`: Gets/sets the cursor position as a **Point**.
- `X`: Gets/sets just the horizontal position.
- `Y`: Gets/sets just the vertical position.

Next, we will add properties for determining if the mouse cursor moved between game frames and if so how much:

```
/// <summary>
/// Gets the difference in the mouse cursor position between the previous and
current frame.
/// </summary>
public Point PositionDelta => CurrentState.Position - PreviousState.Position;

/// <summary>
/// Gets the difference in the mouse cursor x-position between the previous and
current frame.
/// </summary>
public int XDelta => CurrentState.X - PreviousState.X;

/// <summary>
/// Gets the difference in the mouse cursor y-position between the previous and
current frame.
/// </summary>
public int YDelta => CurrentState.Y - PreviousState.Y;

/// <summary>
/// Gets a value that indicates if the mouse cursor moved between the previous and
current frames.
/// </summary>
public bool WasMoved => PositionDelta != Point.Zero;
```

The properties provide different ways of detecting mouse movement between frames:

- `PositionDelta`: Gets how much the cursor moved between frames as a **Point**.
- `XDelta`: Gets how much the cursor moved horizontally between frames.
- `YDelta`: Gets how much the cursor moved vertically between frames.
- `WasMoved`: Indicates if the cursor moved between frames.

Finally, we will add properties for handling the scroll wheel:

```
/// <summary>
/// Gets the cumulative value of the mouse scroll wheel since the start of the game.
/// </summary>
public int ScrollWheel => CurrentState.ScrollWheelValue;

/// <summary>
/// Gets the value of the scroll wheel between the previous and current frame.
/// </summary>
public int ScrollWheelDelta => CurrentState.ScrollWheelValue -
PreviousState.ScrollWheelValue;
```

The scroll wheel properties serve different purposes:

- `ScrollWheel`: Gets the total accumulated scroll value since game start.
- `ScrollWheelDelta`: Gets the change in scroll value just in this frame.

> **TIP**
>
> Use `ScrollWheelDelta` when you need to respond to how much the user just scrolled, rather than tracking the total scroll amount.

MouseInfo Constructor

The `MouseInfo` class constructor needs to initialize the mouse states.

Add this constructor:

```
/// <summary>
/// Creates a new MouseInfo.
/// </summary>
public MouseInfo()
{
    PreviousState = new MouseState();
    CurrentState = Mouse.GetState();
}
```

The constructor:

- Creates an empty state for `PreviousState` since there is no previous input yet.
- Gets the current mouse state as our starting point for `CurrentState`.

This initialization ensures we have valid states to compare against in the first frame of our game, preventing any potential null reference issues when checking for input changes.

MouseInfo Methods

The `MouseInfo` class needs methods for updating states, checking button states, and setting the cursor position. Add the following method to the `MouseInfo` class:

```
/// <summary>
/// Updates the state information about mouse input.
/// </summary>
public void Update()
{
    PreviousState = CurrentState;
    CurrentState = Mouse.GetState();
}
```

Next, we will add methods to check various button states:

```
/// <summary>
/// Returns a value that indicates whether the specified mouse button is
currently down.
/// </summary>
/// <param name="button">The mouse button to check.</param>
/// <returns>true if the specified mouse button is currently down; otherwise,
false.</returns>
public bool IsButtonDown(MouseButton button)
{
    switch (button)
    {
        case MouseButton.Left:
            return CurrentState.LeftButton == ButtonState.Pressed;
        case MouseButton.Middle:
            return CurrentState.MiddleButton == ButtonState.Pressed;
        case MouseButton.Right:
            return CurrentState.RightButton == ButtonState.Pressed;
        case MouseButton.XButton1:
            return CurrentState.XButton1 == ButtonState.Pressed;
        case MouseButton.XButton2:
            return CurrentState.XButton2 == ButtonState.Pressed;
        default:
            return false;
    }
}

/// <summary>
/// Returns a value that indicates whether the specified mouse button is current up.
/// </summary>
/// <param name="button">The mouse button to check.</param>
/// <returns>true if the specified mouse button is currently up; otherwise,
false.</returns>
public bool IsButtonUp(MouseButton button)
{
    switch (button)
    {
        case MouseButton.Left:
            return CurrentState.LeftButton == ButtonState.Released;
        case MouseButton.Middle:
            return CurrentState.MiddleButton == ButtonState.Released;
        case MouseButton.Right:
            return CurrentState.RightButton == ButtonState.Released;
        case MouseButton.XButton1:
            return CurrentState.XButton1 == ButtonState.Released;
        case MouseButton.XButton2:
            return CurrentState.XButton2 == ButtonState.Released;
```

```
        default:
            return false;
    }
}

/// <summary>
/// Returns a value that indicates whether the specified mouse button was just
pressed on the current frame.
/// </summary>
/// <param name="button">The mouse button to check.</param>
/// <returns>true if the specified mouse button was just pressed on the current
frame; otherwise, false.</returns>
public bool WasButtonJustPressed(MouseButton button)
{
    switch (button)
    {
        case MouseButton.Left:
            return CurrentState.LeftButton == ButtonState.Pressed &&
PreviousState.LeftButton == ButtonState.Released;
        case MouseButton.Middle:
            return CurrentState.MiddleButton == ButtonState.Pressed &&
PreviousState.MiddleButton == ButtonState.Released;
        case MouseButton.Right:
            return CurrentState.RightButton == ButtonState.Pressed &&
PreviousState.RightButton == ButtonState.Released;
        case MouseButton.XButton1:
            return CurrentState.XButton1 == ButtonState.Pressed &&
PreviousState.XButton1 == ButtonState.Released;
        case MouseButton.XButton2:
            return CurrentState.XButton2 == ButtonState.Pressed &&
PreviousState.XButton2 == ButtonState.Released;
        default:
            return false;
    }
}

/// <summary>
/// Returns a value that indicates whether the specified mouse button was just
released on the current frame.
/// </summary>
/// <param name="button">The mouse button to check.</param>
/// <returns>true if the specified mouse button was just released on the current
frame; otherwise, false.</returns>
public bool WasButtonJustReleased(MouseButton button)
{
    switch (button)
```

```
    {
        case MouseButton.Left:
            return CurrentState.LeftButton == ButtonState.Released &&
PreviousState.LeftButton == ButtonState.Pressed;
        case MouseButton.Middle:
            return CurrentState.MiddleButton == ButtonState.Released &&
PreviousState.MiddleButton == ButtonState.Pressed;
        case MouseButton.Right:
            return CurrentState.RightButton == ButtonState.Released &&
PreviousState.RightButton == ButtonState.Pressed;
        case MouseButton.XButton1:
            return CurrentState.XButton1 == ButtonState.Released &&
PreviousState.XButton1 == ButtonState.Pressed;
        case MouseButton.XButton2:
            return CurrentState.XButton2 == ButtonState.Released &&
PreviousState.XButton2 == ButtonState.Pressed;
        default:
            return false;
    }
}
```

These methods serve two distinct purposes. For checking continuous states:

- `IsButtonDown`: Returns true as long as the specified button is being held down.
- `IsButtonUp`: Returns true as long as the specified button is not being pressed.

And for detecting state changes:

- `WasButtonJustPressed`: Returns true only on the frame when the specified button changes from up-to-down.
- `WasButtonJustReleased`: Returns true only on the frame when the specified button changes from down-to-up.

> **NOTE**
>
> Each method uses a switch statement to check the appropriate button property from the **MouseState** based on which `MouseButton` enum value is provided. This provides a consistent API while handling the different button properties internally.

Finally, we need a method to handle setting the cursor position:

```
/// <summary>
/// Sets the current position of the mouse cursor in screen space and updates the
CurrentState with the new position.
/// </summary>
/// <param name="x">The x-coordinate location of the mouse cursor in screen
space.</param>
/// <param name="y">The y-coordinate location of the mouse cursor in screen
space.</param>
public void SetPosition(int x, int y)
{
    Mouse.SetPosition(x, y);
    CurrentState = new MouseState(
        x,
        y,
        CurrentState.ScrollWheelValue,
        CurrentState.LeftButton,
        CurrentState.MiddleButton,
        CurrentState.RightButton,
        CurrentState.XButton1,
        CurrentState.XButton2
    );
}
```

> **TIP**
>
> Notice that after setting the position, we immediately update the `CurrentState`. This ensures our state tracking remains accurate even when manually moving the cursor.

That's it for the `MouseInfo` class, next we will move onto gamepad input.

The GamePadInfo Class

To manage gamepad input effectively, we need to track both current and previous states, is the gamepad still connected, as well as provide easy access to the thumbstick values, trigger values, and button states. The `GamePadInfo` class will encapsulate all of this functionality, making it easier to:

- Track current and previous gamepad states.
- Check if the gamepad is still connected.
- Track the position of the left and right thumbsticks.
- Check the values of the left and right triggers.
- Detect when gamepad buttons are pressed or released.
- Check if gamepad buttons are being held down.
- Start and Stop vibration of a gamepad.

To get started, in the `Input` folder of the *MonoGameLibrary* project, create a new file named `GamePadInfo.cs` with the following initial structure:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;

namespace MonoGameLibrary.Input;

public class GamePadInfo
{

}
```

GamePadInfo Properties

We use vibration in gamepads to provide haptic feedback to the player. The **GamePad** class provides the **SetVibration** method to tell the gamepad to vibrate, but it does not provide a timing mechanism for it if we wanted to only vibrate for a certain period of time. Add the following private field to the `GamePadInfo` class:

```
private TimeSpan _vibrationTimeRemaining = TimeSpan.Zero;
```

If you recall from the previous chapter, a **PlayerIndex** value needs to be supplied when calling **Gamepad.GetState**. Doing this returns the state of the gamepad connected at that player index. So we will need a property to track the player index this gamepad info is for.

```
/// <summary>
/// Gets the index of the player this gamepad is for.
/// </summary>
public PlayerIndex PlayerIndex { get; }
```

To detect changes in the gamepad input between frames, we need to track both the previous and current gamepad states. Add these properties to the `GamePadInfo` class:

```
/// <summary>
/// Gets the state of input for this gamepad during the previous update cycle.
/// </summary>
public GamePadState PreviousState { get; private set; }

/// <summary>
/// Gets the state of input for this gamepad during the current update cycle.
/// </summary>
public GamePadState CurrentState { get; private set; }
```

There are times that a gamepad can disconnect for various reasons; being unplugged, bluetooth disconnection, or battery dying are just some examples. To track if the gamepad is connected, add the following property:

```
/// <summary>
/// Gets a value that indicates if this gamepad is currently connected.
/// </summary>
public bool IsConnected => CurrentState.IsConnected;
```

The values of the thumbsticks and triggers can be accessed through the `CurrentState`. However, instead of having to navigate through multiple property chains to get this information, add the following properties to get direct access to the values:

```
/// <summary>
/// Gets the value of the left thumbstick of this gamepad.
/// </summary>
public Vector2 LeftThumbStick => CurrentState.ThumbSticks.Left;

/// <summary>
/// Gets the value of the right thumbstick of this gamepad.
/// </summary>
public Vector2 RightThumbStick => CurrentState.ThumbSticks.Right;

/// <summary>
/// Gets the value of the left trigger of this gamepad.
/// </summary>
public float LeftTrigger => CurrentState.Triggers.Left;

/// <summary>
/// Gets the value of the right trigger of this gamepad.
/// </summary>
public float RightTrigger => CurrentState.Triggers.Right;
```

GamePadInfo Constructor

The `GamePadInfo` class constructor needs to initialize the gamepad states.

Add this constructor:

```
/// <summary>
/// Creates a new GamePadInfo for the gamepad connected at the specified
player index.
/// </summary>
/// <param name="playerIndex">The index of the player for this gamepad.</param>
public GamePadInfo(PlayerIndex playerIndex)
{
    PlayerIndex = playerIndex;
    PreviousState = new GamePadState();
    CurrentState = GamePad.GetState(playerIndex);
}
```

This constructor

- Requires a **PlayerIndex** value which is stored and will be used to get the states for the correct gamepad
- Creates an empty state for `PreviousState` since there is no previous state yet.
- Gets the current gamepad state as our starting `CurrentState`.

This initialization ensures we have valid states to compare against in the first frame of our game, preventing any potential null reference issues when checking for input changes.

GamePadInfo Methods

The `GamePadInfo` class needs methods for updating states, checking button states, and controlling vibration. Add the following method to the `GamePadInfo` class:

```
/// <summary>
/// Updates the state information for this gamepad input.
/// </summary>
/// <param name="gameTime"></param>
public void Update(GameTime gameTime)
{
    PreviousState = CurrentState;
    CurrentState = GamePad.GetState(PlayerIndex);

    if (_vibrationTimeRemaining > TimeSpan.Zero)
    {
        _vibrationTimeRemaining -= gameTime.ElapsedGameTime;

        if (_vibrationTimeRemaining <= TimeSpan.Zero)
        {
            StopVibration();
        }
    }
}
```

> **NOTE**
>
> Unlike keyboard and mouse input, the gamepad update method takes a **GameTime** parameter. This allows us to track and manage timed vibration effects.

Next, we will add methods to check various button states:

```
/// <summary>
/// Returns a value that indicates whether the specified gamepad button is
current down.
/// </summary>
/// <param name="button">The gamepad button to check.</param>
/// <returns>true if the specified gamepad button is currently down; otherwise,
false.</returns>
public bool IsButtonDown(Buttons button)
{
    return CurrentState.IsButtonDown(button);
}

/// <summary>
/// Returns a value that indicates whether the specified gamepad button is
currently up.
/// </summary>
/// <param name="button">The gamepad button to check.</param>
/// <returns>true if the specified gamepad button is currently up; otherwise,
false.</returns>
public bool IsButtonUp(Buttons button)
{
    return CurrentState.IsButtonUp(button);
}

/// <summary>
/// Returns a value that indicates whether the specified gamepad button was just
pressed on the current frame.
/// </summary>
/// <param name="button">The gamepad button to check.</param>
/// <returns>true if the specified gamepad button was just pressed on the current
frame; otherwise, false.</returns>
public bool WasButtonJustPressed(Buttons button)
{
    return CurrentState.IsButtonDown(button) && PreviousState.IsButtonUp(button);
}

/// <summary>
/// Returns a value that indicates whether the specified gamepad button was just
released on the current frame.
/// </summary>
/// <param name="button">The gamepad button to check.</param>
/// <returns>true if the specified gamepad button was just released on the current
frame; otherwise, false.</returns>
public bool WasButtonJustReleased(Buttons button)
{
```

```
    return CurrentState.IsButtonUp(button) && PreviousState.IsButtonDown(button);
}
```

These methods serve two distinct purposes. For checking continuous states:

- `IsButtonDown`: Returns true as long as a button is being held down.
- `IsButtonUp`: Returns true as long as a button is not being pressed.

And for detecting state changes:

- `WasButtonJustPressed`: Returns true only on the frame when a button changes from up-to-down.
- `WasButtonJustReleased`: Returns true only on the frame when a button changes from down-to-up.

Finally, we will add methods for controlling gamepad vibration:

```
/// <summary>
/// Sets the vibration for all motors of this gamepad.
/// </summary>
/// <param name="strength">The strength of the vibration from 0.0f (none) to 1.0f
(full).</param>
/// <param name="time">The amount of time the vibration should occur.</param>
public void SetVibration(float strength, TimeSpan time)
{
    _vibrationTimeRemaining = time;
    GamePad.SetVibration(PlayerIndex, strength, strength);
}

/// <summary>
/// Stops the vibration of all motors for this gamepad.
/// </summary>
public void StopVibration()
{
    GamePad.SetVibration(PlayerIndex, 0.0f, 0.0f);
}
```

The vibration methods provide control over the gamepad's haptic feedback:

- `SetVibration`: Starts vibration at the specified strength for a set duration.
- `StopVibration`: Immediately stops all vibration.

> **TIP**
>
> When setting vibration, you can specify both the strength (`0.0f` to `1.0f`) and duration. The vibration will automatically stop after the specified time has elapsed, so you do not need to manage stopping it manually.

That's it for the `GamePadInfo` class. Next, we can create the actual input manager.

The InputManager Class

Now that we have classes to handle keyboard, mouse, and gamepad input individually, we can create a centralized manager class to coordinate all input handling.

In the `Input` folder of the *MonoGameLibrary* project, add a new file named `InputManager.cs` with this initial structure:

```
using Microsoft.Xna.Framework;

namespace MonoGameLibrary.Input;

public class InputManager { }
```

InputManager Properties

The `InputManager` class needs properties to access each type of input device. Add these properties:

```
/// <summary>
/// Gets the state information of keyboard input.
/// </summary>
public KeyboardInfo Keyboard { get; private set; }

/// <summary>
/// Gets the state information of mouse input.
/// </summary>
public MouseInfo Mouse { get; private set; }

/// <summary>
/// Gets the state information of a gamepad.
/// </summary>
public GamePadInfo[] GamePads { get; private set; }
```

> **NOTE**
>
> The `GamePads` property is an array because MonoGame supports up to four gamepads simultaneously. Each gamepad is associated with a PlayerIndex (0-3).

InputManager Constructor

The constructor for the `InputManager` initializes the keyboard, mouse, and gamepad states.

Add the following constructor:

```
/// <summary>
/// Creates a new InputManager.
/// </summary>
public InputManager()
{
    Keyboard = new KeyboardInfo();
    Mouse = new MouseInfo();

    GamePads = new GamePadInfo[4];
    for (int i = 0; i < 4; i++)
    {
        GamePads[i] = new GamePadInfo((PlayerIndex)i);
    }
}
```

InputManager Methods

The `Update` method for the `InputManager` calls update for each device, so that they can update their internal states.

```
/// <summary>
/// Updates the state information for the keyboard, mouse, and gamepad inputs.
/// </summary>
/// <param name="gameTime">A snapshot of the timing values for the current
frame.</param>
public void Update(GameTime gameTime)
{
    Keyboard.Update();
    Mouse.Update();

    for (int i = 0; i < 4; i++)
    {
        GamePads[i].Update(gameTime);
    }
}
```

Implementing the InputManager Class

Now that we have our input management system complete, we will update our game to use it. We will do this in two steps:

1. First, update the `Core` class to add the `InputManager` globally.
2. Update the `Game1` class to use the global input manager from `Core`.

Updating the Core Class

The `Core` class serves as our base game class, so we will update it to add and expose the `InputManager` globally. Open the *Core.cs* file in the *MonoGameLibrary* project and update it to the following:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary.Input;

namespace MonoGameLibrary;

public class Core : Game
{
    internal static Core s_instance;

    /// <summary>
    /// Gets a reference to the Core instance.
    /// </summary>
    public static Core Instance => s_instance;

    /// <summary>
    /// Gets the graphics device manager to control the presentation of graphics.
    /// </summary>
    public static GraphicsDeviceManager Graphics { get; private set; }

    /// <summary>
    /// Gets the graphics device used to create graphical resources and perform
primitive rendering.
    /// </summary>
    public static new GraphicsDevice GraphicsDevice { get; private set; }

    /// <summary>
    /// Gets the sprite batch used for all 2D rendering.
    /// </summary>
    public static SpriteBatch SpriteBatch { get; private set; }

    /// <summary>
    /// Gets the content manager used to load global assets.
    /// </summary>
    public static new ContentManager Content { get; private set; }

    /// <summary>
    /// Gets a reference to the input management system.
    /// </summary>
    public static InputManager Input { get; private set; }

    /// <summary>
    /// Gets or Sets a value that indicates if the game should exit when the esc key
```

```
on the keyboard is pressed.
    /// </summary>
    public static bool ExitOnEscape { get; set; }

    /// <summary>
    /// Creates a new Core instance.
    /// </summary>
    /// <param name="title">The title to display in the title bar of the game
window.</param>
    /// <param name="width">The initial width, in pixels, of the game
window.</param>
    /// <param name="height">The initial height, in pixels, of the game
window.</param>
    /// <param name="fullScreen">Indicates if the game should start in fullscreen
mode.</param>
    public Core(string title, int width, int height, bool fullScreen)
    {
        // Ensure that multiple cores are not created.
        if (s_instance != null)
        {
            throw new InvalidOperationException($"Only a single Core instance can
be created");
        }

        // Store reference to engine for global member access.
        s_instance = this;

        // Create a new graphics device manager.
        Graphics = new GraphicsDeviceManager(this);

        // Set the graphics defaults.
        Graphics.PreferredBackBufferWidth = width;
        Graphics.PreferredBackBufferHeight = height;
        Graphics.IsFullScreen = fullScreen;

        // Apply the graphic presentation changes.
        Graphics.ApplyChanges();

        // Set the window title.
        Window.Title = title;

        // Set the core's content manager to a reference of the base Game's
        // content manager.
        Content = base.Content;

        // Set the root directory for content.
```

```
            Content.RootDirectory = "Content";

            // Mouse is visible by default.
            IsMouseVisible = true;

            // Exit on escape is true by default
            ExitOnEscape = true;
        }

        protected override void Initialize()
        {
            base.Initialize();

            // Set the core's graphics device to a reference of the base Game's
            // graphics device.
            GraphicsDevice = base.GraphicsDevice;

            // Create the sprite batch instance.
            SpriteBatch = new SpriteBatch(GraphicsDevice);

            // Create a new input manager.
            Input = new InputManager();
        }

        protected override void Update(GameTime gameTime)
        {
            // Update the input manager.
            Input.Update(gameTime);

            if (ExitOnEscape && Input.Keyboard.IsKeyDown(Keys.Escape))
            {
                Exit();
            }

            base.Update(gameTime);
        }
    }
```

The key changes to the `Core` class are:

1. Added the `using MonoGameLibrary.Input;` directive to access the `InputManager` class.
2. Added a static `Input` property to provide global access to the input manager.
3. Added a static `ExitOnEscape` property to set whether the game should exit when the Escape key on the keyboard is pressed.
4. In the `Core` constructor, `ExitOnEscape` is set to true by default to mirror how the default MonoGame `Game1` class template has this functionality by default.
5. In `Initialize` the input manager is created.
6. Added an override for the `Update` method where:
    1. The input manager is updated
    2. A check is made to see if `ExitOnEscape` is true and if the Escape keyboard key is pressed.

Updating the Game1 Class

Now we can update our `Game1` class to use the new input management system through the `Core` class. Open `Game1.cs` in the game project and update it to the following:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);
```

```
        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        // Check for keyboard input and handle it.
        CheckKeyboardInput();

        // Check for gamepad input and handle it.
        CheckGamePadInput();

        base.Update(gameTime);
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
        {
```

```
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }
    }

    private void CheckGamePadInput()
    {
        GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];

        // If the A button is held down, the movement speed increases by 1.5
        // and the gamepad vibrates as feedback to the player.
        float speed = MOVEMENT_SPEED;
        if (gamePadOne.IsButtonDown(Buttons.A))
        {
            speed *= 1.5f;
            gamePadOne.SetVibration(1.0f, TimeSpan.FromSeconds(1));
        }
        else
        {
            gamePadOne.StopVibration();
        }

        // Check thumbstick first since it has priority over which gamepad input
        // is movement.  It has priority since the thumbstick values provide a
        // more granular analog value that can be used for movement.
        if (gamePadOne.LeftThumbStick != Vector2.Zero)
        {
            _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
            _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
        }
        else
        {
            // If DPadUp is down, move the slime up on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadUp))
            {
                _slimePosition.Y -= speed;
            }

            // If DPadDown is down, move the slime down on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadDown))
```

```
            {
                _slimePosition.Y += speed;
            }

            // If DPapLeft is down, move the slime left on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
            {
                _slimePosition.X -= speed;
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite 10px to the right of the slime.
        _bat.Draw(SpriteBatch, new Vector2(_slime.Width + 10, 0));

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes to the `Game1` class are:

1. In **Update**, the check for the gamepad back button or keyboard escape key being pressed was removed. This is now handled by the `ExitOnEscape` property and the `Update` method of the `Core` class.
2. In `CheckKeyboardInput` and `CheckGamepadInput`, instead of getting the keyboard and gamepad states and then using the states, calls to check those devices are now done through the input.

Running the game now, you will be able to control it the same as before, only now we are using our new `InputManager` class instead.

Figure 11-1: The slime moving around based on device input

Conclusion

In this chapter, you accomplished the following:

- Detect the difference between continuous and single-frame input states.
- Create classes to manage different input devices.
- Build a centralized `InputManager` to coordinate all input handling that is:
  - Reusable across different game projects
  - Easy to maintain and extend
  - Consistent across different input devices
- Integrate the input system into the `Core` class for global access.
- Update the game to use the new input management system.

Test Your Knowledge

1. What is the difference between checking if an input is "down" versus checking if it was "just pressed"?

   "Down" checks if an input is currently being held, returning true every frame while held. "Just pressed" only returns true on the first frame when the input changes from up to down, requiring comparison between current and previous states.

2. Why do we track both current and previous input states?

   Tracking both states allows us to detect when input changes occur by comparing the current frame's state with the previous frame's state. This is essential for implementing "just pressed" and "just released" checks.

3. What advantage does the `InputManager` provide over handling input directly?

   The `InputManager` centralizes all input handling, automatically tracks states between frames, and provides a consistent API across different input devices. This makes the code more organized, reusable, and easier to maintain.

Chapter 12: Collision Detection

Learn how to implement collision detection between game objects and handle collision responses like blocking, triggering events, and bouncing.

In Chapter 11, you learned how to manage player input to control game objects. However, for objects in your game to interact with each other, collecting items, hitting obstacles, or triggering events, you need to detect when these objects come into contact. This is accomplished through collision detection.

In this chapter you will:

- Understand different collision shapes and their use cases.
- Implement rectangle-based collision detection.
- Create circle-based collision detection.
- Learn how to handle object overlap and response.
- Build a reusable collision system for your game.

We will first start by understanding the basics of collision detection and the different approaches that can be used.

> **NOTE**
>
> There is a lot to understand when it comes to collision detection and the many complex ways that two objects can be considered IN collision or NEAR collision. It is critical to get an understanding of the basics before jumping into code. So buckle up, we have a story to tell before you can get back to the keyboard.
>
> Feel free to keep coming back to this chapter and refer to the content when you need to, with a fresh cup of coffee.

Understanding Collision Detection

Before we start implementing collision detection, we should discuss what collision detection actually is. In 2D games, collision detection involves checking if two objects interact with each other in some way. There are several approaches to detecting collisions, ranging from simple to complex:

Proximity Collision Detection

The simplest form is checking if objects are within a certain range of each other. This is useful when you only need to know if objects are "near" each other like detecting if an enemy is close enough to chase a player or if two objects are close enough to perform a more complex collision check.

Simple Shape Based Collision Detection

Shaped based collision detection checks if two shapes overlap. The most common and simple shapes used are circles and rectangles:

Circle Collision Detection

Circle collision detection is computationally a simpler check than rectangles. There are also no special considerations if the circles are rotated, which makes them easier to use. To determine if two circle shapes are overlapping, we only need to check if the square of the sum of the radii between the two circles is less than the squared distance between the two circles with the following formula:

To find the distance between two circles, imagine drawing a line from the center of one circle to the center of the other. The length of this line is the distance, but we could also calculate it by first walking up or down and then walking left or right from the center of one circle to another, forming a right triangle.

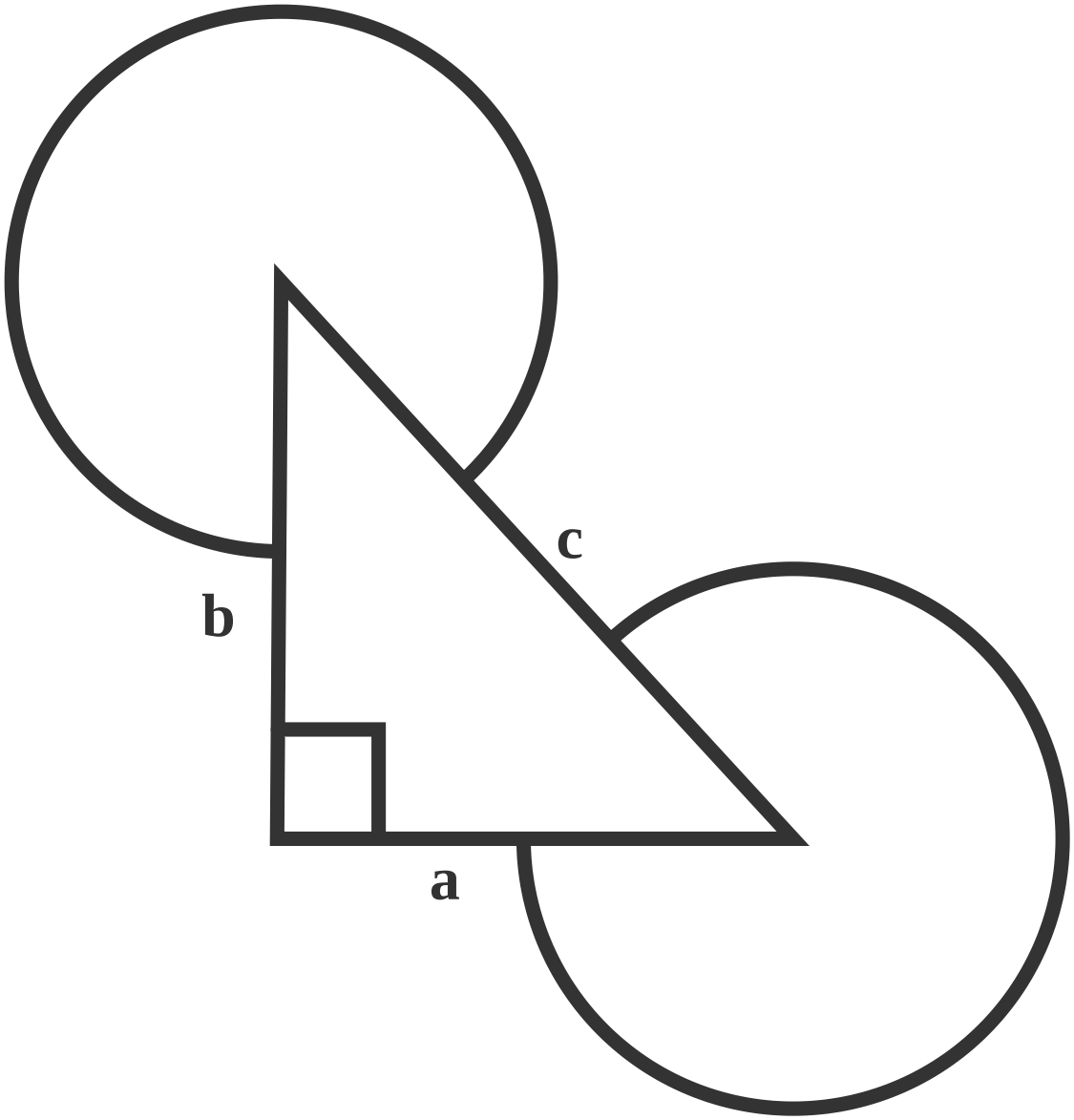


Figure 12-1: Showing the distance between the center of two circles forms a right triangle

In *Figure 12-1* above

- a is the horizontal distance between the centers of the two circles (x-axis).
- b is the vertical distance between the centers of the two circles (y-axis).
- c is the total distance between the centers of the two circles.

Since this forms a right triangle, we can use Pythagorean's Theorem to calculate c^2 given a^2 and b^2:

$$c^2 = a^2 + b^2$$

To check for overlap of two circles, we compare whether the *squared sum of the radii* of the two circles is greater than the *squared distance*:

$$(radius_{circle1} + radius_{circle2})^2 > a^2 + b^2$$

It is easy to confuse the direction of the inequality sign. As a quick mental test, think of how the math works when the origin of two circles are at the same position, i.e., when the *squared distance* is zero.

To calculate the squared distance between two points, MonoGame provides the **Vector2.DistanceSquared** method:

```
Vector2 circle1Position = new Vector2(8, 10);
Vector2 circle2Position = new Vector2(5, 6);

float circle1Radius = 5;
float circle2Radius = 5;

// c^2 = a^2 + b^2
// c^2 = (8 - 5)^2 + (10 - 6)^2
// c^2 = 3^2 + 4^2
// c^2 = 9 + 16
// c^2 = 25
float distanceSquared = Vector2.DistanceSquared(circle1Position, circle2Position);

// r^2 = (5 + 5)^2
// r^2 = (10)^2
// r^2 = 100
int radiiSquared = (circle1Radius + circle2Radius) * (circle1Radius
+ circle2Radius);

// The circles overlap because 100 is greater than 25.
if(radiiSquared > distanceSquared)
{

}
```

TIP

MonoGame also provides a distance calculation method with **Vector2.Distance** which returns the distance by providing the square root of the distance squared. So why not use this instead?

Square root operations are more computationally complex for a CPU. So instead of getting the normal distance, which would require the square root operation, it is more efficient for the cpu to multiply the sum of the radii by itself to get the squared sum and use that for comparison instead.

Rectangle Collision Detection

Rectangles, often called *bounding boxes*, typically uses what is called *Axis-Aligned Bounding Box* (AABB) collision detection to determine if two rectangle shapes overlap. Unlike circles, to perform AABB collision detection, the x- and y-axes of both rectangles must be aligned with the x- and y-axes of the screen. This is just another way of saying that the rectangles cannot be rotated.

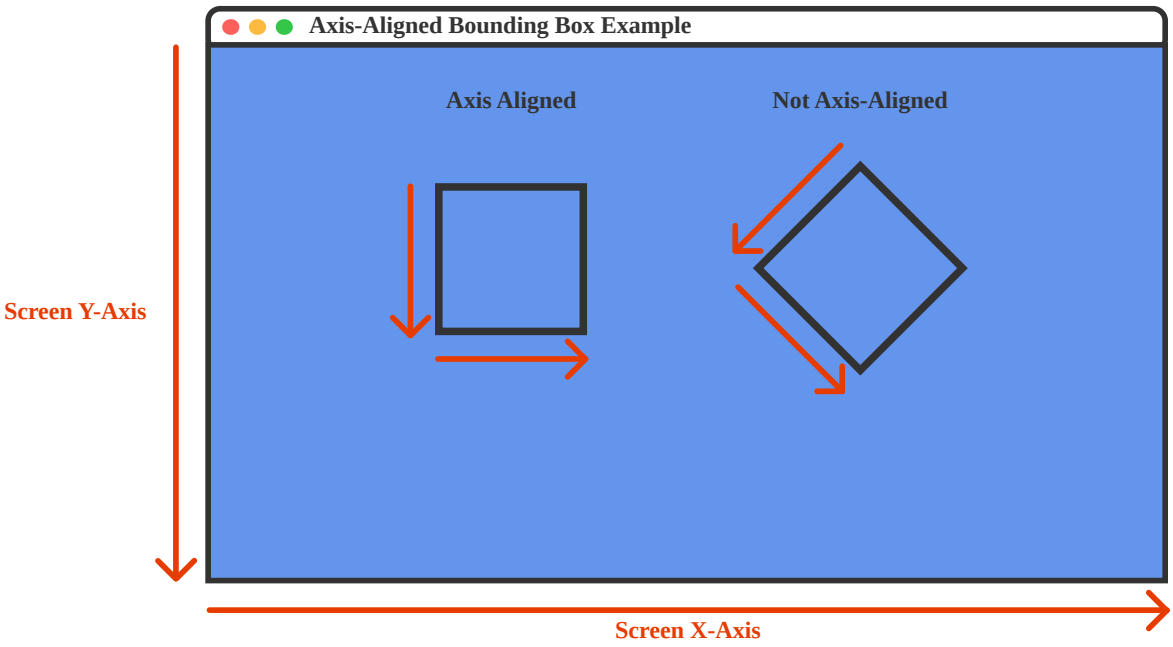


Figure 12-2: The rectangle on the left is axis-aligned since both the axes are aligned with the screen axes. The rectangle on the right is non axis-aligned since it is rotated and the axes do not align with the screen axes

MonoGame provides the **Rectangle** struct which represents a rectangle by its position (X,Y) and size (Width,Height). The following table shows some of the properties of the **Rectangle** struct:

Property	Type	Description
Bottom	`int`	Returns the y-coordinate location of the bottom edge of the rectangle. This is equal to Rectangle.Y plus the height of the rectangle.
Left	`int`	Returns the x-coordinate location of the left edge of the rectangle. This is equal to Rectangle.X.
Right	`int`	Returns the x-coordinate location of the right edge of the rectangle. This is equal to Rectangle.X plus the width of the rectangle.
Top	`int`	Returns the y-coordinate location of the top edge of the rectangle. This is equal to Rectangle.Y.

To determine if two rectangles overlap using AABB collision detection, there are four conditions that need to be checked, and all four conditions must be true. Given two rectangles A and B, these conditions are:

1. A_{Left} must be less than B_{Right}.
2. A_{Right} must be greater than B_{Left}.
3. A_{Top} must be less than B_{Bottom}.
4. A_{Bottom} must be greater than B_{Top}.

If even a single one of these conditions is false, then the rectangles are not overlapping and thus not colliding.

MonoGame provides the Rectangle.Intersects method which will perform an AABB collision check for us:

```
// Rectangle 1
//                Top: 0
//          ----------------
//         |                |
//         |                |
// Left: 0 |                |  Right: 32
//         |                |
//         |                |
//          ----------------
//              Bottom: 32
Rectangle rect1 = new Rectangle(0, 0, 32, 32);

// Rectangle 2
//                Top: 16
//           ----------------
//          |                |
//          |                |
// Left: 16 |                |  Right: 48
//          |                |
//          |                |
//           ----------------
//              Bottom: 48
Rectangle rect2 = new Rectangle (16, 16, 32, 32);

// rect1.Left (0)  < rect2.Right (48) = true
// rect1.Right (32) > rect3.Left (16) = true
// rect1.Top (0) < rect2.Bottom (48) = true
// rect1.Bottom (32) > rect2.Top (16) = true
bool isColliding = rect1.Intersects(rect2); // returns true
```

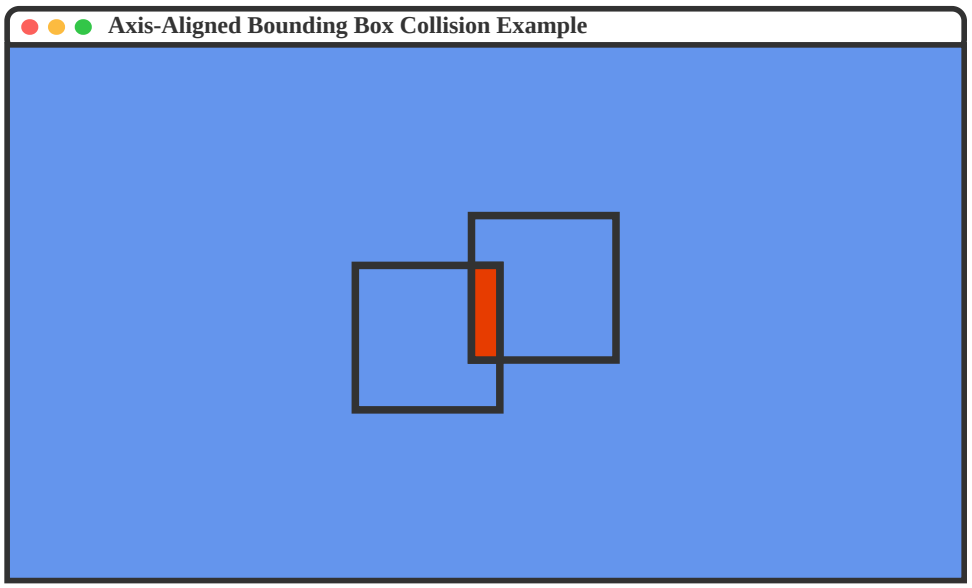


Figure 12-3: The rectangle on the left is overlapping the rectangle on the right based on the conditions required for the Axis-Aligned Bounding Box collision check

Complex Polygon Collision Detection

Complex polygon collision detection uses a method called *Separating Axis Theorem* (SAT) to determine if two polygon shapes overlap. SAT uses more complex calculations that can determine if any polygon shape overlaps with another polygon shape, including if they are rotated. There are performance considerations to consider when using SAT.

Implementing SAT is out-of-scope for this tutorial. If you are interested in further reading about this, please see the following articles as a good starting point:

- Separating Axis Theorem (SAT) Explanation.
- Collision Detection Using the Separating Axis Theorem by Kah Shiu Chong.
- N Tutorial A - Collision Detection and Response.

Choosing a Collision Detection Method

When determining which collision detection method to use, you should start with the simplest one that meets the needs of your game. If distance checks work for your game mechanic, there's no need to implement more complex shape based detections. Similarly, if a circle can represent the bounding area of a game object, start with that before moving on to rectangles.

Some other points to consider are

- Circles:
  - Better for round objects like balls and coins.
  - More accurate for rotating objects.
  - A simpler check for overlap than rectangles.
- Rectangles:
  - Great for walls, platforms, and most game objects.
  - Easy to visualize and debug.
  - Works well with tile-based games.

Collision Detection vs Collision Response

Often times when talking about collision detection, the term is used to mean both the detection of overlapping shapes and what to do once a positive check has occurred. What you do after a positive collision check has occurred is called the *collision response*. Some of the common responses are:

Blocking Collision Response

A blocking collision response is the most basic response which just prevents the two objects from overlapping. This is commonly used for walls, platforms and other solid objects. To perform a blocking collision response:

1. Store the object's current location and calculate the new position it should move to.
2. Check if it is overlapping an object at the new location:

- If it is overlapping, then set the position to the position before it was moved.
- If it is not overlapping, set the position to the new calculated position.

For example:

```
// Store the current location.
Vector2 previousLocation = _spriteLocation;

// Calculate a new location.
Vector2 newLocation = _spriteLocation + new Vector2(10, 0);

// Create a bounding box for the sprite object.
Rectangle spriteBounds = new Rectangle(
    (int)newLocation.X,
    (int)newLocation.Y,
    (int)_sprite.Width,
    (int)_sprite.Height
);

// Create a bounding box for the blocking object.
Rectangle blockingBounds = new Rectangle(
    (int)_blockingLocation_.X,
    (int)_blockingLocation_.Y,
    (int)_blockingSprite_.Width,
    (int)_blockingSprite_.Height
);

// Detect if they are colliding
if(spriteBounds.Intersects(blockingBounds))
{
    // Respond by not allowing the sprite to move by setting
    // the location back to the previous location.
    newLocation = previousLocation;
}

_spriteLocation = newLocation;
```

Sometimes, instead of preventing an object from moving onto another object, we want to ensure an object remains contained within a certain bounding area. MonoGame also provides the **Rectangle.Contains** method that we can use to determine this. **Rectangle.Contains** can check if any of the following are completely contained within the bounds of the rectangle;

- **Point**
- **Rectangle**
- **Vector2**

For example, if we wanted to perform a blocking collision response that ensure a sprite remained contained within the bounds of the game screen:

```
// Store the current location
Vector2 previousLocation = _spriteLocation;

// Calculate a new location
Vector2 newLocation = _spriteLocation + new Vector2(10, 0);

// Create a bounding box for the sprite object
Rectangle spriteBounds = new Rectangle(
    (int)newLocation.X,
    (int)newLocation.Y,
    (int)_sprite.Width,
    (int)_sprite.Height
);

// Get the bounds of the screen as a rectangle
Rectangle screenBounds = new Rectangle(
    0,
    0,
    GraphicsDevice.PresentationParameters.BackBufferWidth,
    GraphicsDevice.PresentationParameters.BackBufferHeight
);

// Detect if the sprite is contained within the bounds of the screen
if(!screenBounds.Contains(spriteBounds))
{
    // Respond by not allowing the sprite to move outside the screen
    // bounds by setting the location back to the previous location.
    newLocation = previousLocation;
}

_spriteLocation = newLocation;
```

TIP

Use **GraphicsDevice.PresentationParameters** to get the actual screen dimensions instead of **GraphicsDeviceManager.PreferredBackBufferWidth** and **GraphicsDeviceManager.PreferredBackBufferHeight**. The preferred values are only hints and may not reflect the actual back buffer size.

Trigger Collision Response

Sometimes you want to trigger an event, rather than block movement, when a collision occurs. Common examples include:

- Collecting items.
- Activating switches.
- Entering zones or areas.
- Triggering cutscenes.

Performing a trigger collision response is just simply checking if the game object is overlapping with the bounding area of the trigger zone, and if so trigger the event.

For example:

```
// Create a bounding box for the sprite object.
Rectangle spriteBounds = new Rectangle(
    (int)_spriteLocation.X,
    (int)_spriteLocation.Y,
    (int)_sprite.Width,
    (int)_sprite.Height
);

// Detect if the sprite object is within the trigger zone.
if(_spriteBounds.Intersects(_triggerBounds))
{
    // Perform some event.
    CollectItem();
}
```

Bounce Collision Response

For games that need objects to bounce off each other (like the ball in a Pong game), we need to calculate how their velocity should change after the collision. MonoGame provides the **Vector2.Reflect** method to handle this calculation for us. The method needs two pieces of information:

1. The incoming vector (the direction the object is moving in before the collision).
2. The normal vector (the direction perpendicular to the surface).

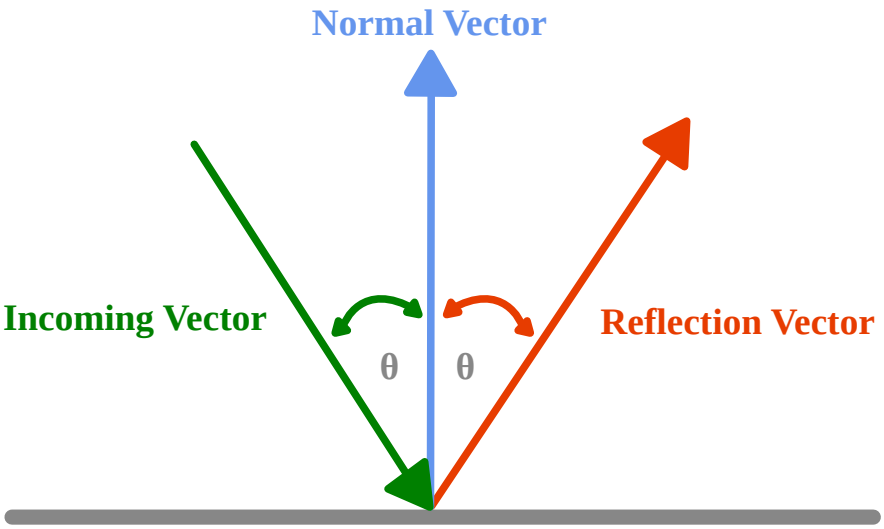


Figure 12-4: A diagram showing how an incoming vector reflects off of a surface base around the normal vector of the surface

As shown in the diagram above, when an incoming vector hits a surface, it reflects at the same angle (θ) relative to the normal vector.

> **TIP**
>
> Think of the normal vector like the line you'd draw perpendicular to a mirror's surface. The angle between your incoming path and this line will be the same as the angle between your reflection and this line.

For example, if we had a ball moving around the screen and wanted it to bounce off the edges of the screen:

```
// Calculate the new position of the ball based on the velocity.
Vector2 newPosition = _ballPosition + _ballVelocity;

// Get the bounds of the ball as a rectangle.
Rectangle ballBounds = new Rectangle(
    (int)_ballPosition.X,
    (int)_ballPosition.Y,
    (int)_ball.Width,
    (int)_ball.Height
);

// Get the bounds of the screen as a rectangle.
Rectangle screenBounds = new Rectangle(
    0,
    0,
    GraphicsDevice.PresentationParameters.BackBufferWidth,
    GraphicsDevice.PresentationParameters.BackBufferHeight
);

// Detect if the ball object is within the screen bounds.
if(!screenBounds.Contains(ballBounds))
{
    // Ball would move outside the screen
    // First find the distance from the edge of the ball to each edge of the screen.
    float distanceLeft = Math.Abs(screenBounds.Left - ballBounds.Left);
    float distanceRight = Math.Abs(screenBounds.Right - ballBounds.Right);
    float distanceTop = Math.Abs(screenBounds.Top - ballBounds.Top);
    float distanceBottom = Math.Abs(screenBounds.Bottom - ballBounds.Bottom);

    // Determine which screen edge is the closest.
    float minDistance = Math.Min(
        Math.Min(distanceLeft, distanceRight),
        Math.Min(distanceTop, distanceBottom)
    );

    // Determine the normal vector based on which screen edge is the closest.
    Vector2 normal;
    if (minDistance == distanceLeft)
    {
        // Closest to the left edge.
        normal = Vector2.UnitX;
        newPosition.X = 0;
    }
    else if (minDistance == distanceRight)
    {
        // Closest to the right edge.
```

```
            normal = -Vector2.UnitX;
            newPosition.X = screenBounds.Right - _ball.Width;
        }
        else if (minDistance == distanceTop)
        {
            // Closest to the top edge.
            normal = Vector2.UnitY;
            newPosition.Y = 0;
        }
        else
        {
            // Closest to the bottom edge.
            normal = -Vector2.UnitY;
            newPosition.Y = screenBounds.Bottom - _ball.Height;
        }

        // Reflect the velocity about the normal.
        _ballVelocity = Vector2.Reflect(_ballVelocity, normal);
    }

    // Set the new position of the ball.
    _ballVelocity = newPosition;
```

> **TIP**
>
> Vector2.UnitX is $(1, 0)$ and Vector2.UnitY is $(0, 1)$. We use these to get the screen edge normal since the edges of the screen are not at an angle. For more complex surfaces, you would need to calculate the appropriate normal vector based on the surface angle.

Optimizing Collision Performance

When checking for collisions between multiple objects, testing every object against every other object (often called brute force checking) becomes inefficient as your game grows. Brute force checking can be calculated as $(n * (n - 1))/2$ where n is the total number of objects. For example, if you have 100 objects in your game, that's $(100 * 99)/2 = 4950$ collision checks every frame. To improve performance, we can use a two-phase approach:

1. Broad Phase: A quick, simple check to rule out objects that definitely are not colliding.
2. Narrow Phase: A more precise check that is only performed on objects that have passed the broad phase.

For our simple game with just two objects, this optimization is not necessary. However, as you develop more complex games, implementing a broad-phase check can significantly improve performance.

> **NOTE**
>
> Time to get back to the code! The fun starts again here.

The Circle Struct

For our game, we are going to implement circle based collision detection. MonoGame does not have a `Circle` struct to represent a circle like it does with **Rectangle**. Before we can perform circle collision, we will need to create our own.

In the root of the *MonoGameLibrary* project, add a new file named `Circle.cs`. Add the following code as the foundation of the `Circle` struct:

```
using System;
using Microsoft.Xna.Framework;

namespace MonoGameLibrary;

public readonly struct Circle : IEquatable<Circle>
{

}
```

> **NOTE**
>
> Notice that the struct has declared it will implement the `IEquatable<T>` interface. When creating value types like this, it is recommended to implement `IEquatable<T>` because it has better performance for comparing objects and can help avoid boxing.
>
> For more information on recommended design guidelines for structs, see Struct Design - Framework Design Guidelines | Microsoft Learn
>
> Ignore the red squiggles for now, they will be resolved as we add more code.

Circle Fields

The `Circle` struct uses both private and public fields to store its state.

First, add the following private static field that stores a reusable empty circle:

```
private static readonly Circle s_empty = new Circle();
```

Next, add the following public fields that define the circle's position and size:

```
/// <summary>
/// The x-coordinate of the center of this circle.
/// </summary>
public readonly int X;

/// <summary>
/// The y-coordinate of the center of this circle.
/// </summary>
public readonly int Y;

/// <summary>
/// The length, in pixels, from the center of this circle to the edge.
/// </summary>
public readonly int Radius;
```

These public fields store the fundamental properties of the circle:

- `X` and `Y` define the center point location.
- `Radius` defines how far the circle extends from its center.

Circle Properties

The `Circle` struct provides properties to access its location, state, and its boundaries.

Add the following property to get the location of the circle as a **Point** value:

```
/// <summary>
/// Gets the location of the center of this circle.
/// </summary>
public readonly Point Location => new Point(X, Y);
```

Add the following properties to track empty circles:

```
/// <summary>
/// Gets a circle with X=0, Y=0, and Radius=0.
/// </summary>
public static Circle Empty => s_empty;

/// <summary>
/// Gets a value that indicates whether this circle has a radius of 0 and a location
of (0, 0).
/// </summary>
public readonly bool IsEmpty => X == 0 && Y == 0 && Radius == 0;
```

> **NOTE**
>
> The `Empty` property returns a reusable instance of an empty circle stored in the private static field `s_empty`. This is more efficient than creating new empty circles each time one is needed, as it reuses the same instance in memory.

Add the following properties for getting the circle's boundaries:

```
/// <summary>
/// Gets the y-coordinate of the highest point on this circle.
/// </summary>
public readonly int Top => Y - Radius;

/// <summary>
/// Gets the y-coordinate of the lowest point on this circle.
/// </summary>
public readonly int Bottom => Y + Radius;

/// <summary>
/// Gets the x-coordinate of the leftmost point on this circle.
/// </summary>
public readonly int Left => X - Radius;

/// <summary>
/// Gets the x-coordinate of the rightmost point on this circle.
/// </summary>
public readonly int Right => X + Radius;
```

> **TIP**
>
> These boundary properties are particularly useful when you need to know the extent of a circle in screen space, such as determining if a circle is visible on screen or creating a bounding box around the circle.

Circle Constructors

The `Circle` struct provides two ways to create a new circle:

```
/// <summary>
/// Creates a new circle with the specified position and radius.
/// </summary>
/// <param name="x">The x-coordinate of the center of the circle.</param>
/// <param name="y">The y-coordinate of the center of the circle..</param>
/// <param name="radius">The length from the center of the circle to an
edge.</param>
public Circle(int x, int y, int radius)
{
    X = x;
    Y = y;
    Radius = radius;
}

/// <summary>
/// Creates a new circle with the specified position and radius.
/// </summary>
/// <param name="location">The center of the circle.</param>
/// <param name="radius">The length from the center of the circle to an
edge.</param>
public Circle(Point location, int radius)
{
    X = location.X;
    Y = location.Y;
    Radius = radius;
}
```

- The first constructor accepts individual x and y coordinates for the circle's center.
- The second accepts a **Point** struct that combines both coordinates.

Both constructors require a radius value that defines the circle's size.

Circle Methods

The `Circle` struct implements several methods to support equality comparison between circles. These methods allow us to check if two circles are identical (have the same center position and radius).

First, add the following method that will check if two circles are overlapping with each other:

```
/// <summary>
/// Returns a value that indicates whether the specified circle intersects with
this circle.
/// </summary>
/// <param name="other">The other circle to check.</param>
/// <returns>true if the other circle intersects with this circle; otherwise,
false.</returns>
public bool Intersects(Circle other)
{
    int radiiSquared = (this.Radius + other.Radius) * (this.Radius + other.Radius);
    float distanceSquared = Vector2.DistanceSquared(this.Location.ToVector2(),
other.Location.ToVector2());
    return distanceSquared < radiiSquared;
}
```

Next we start implementing the `IEquatable<T>` Interface, add the following methods for comparing a circle with another object:

```
/// <summary>
/// Returns a value that indicates whether this circle and the specified object
are equal
/// </summary>
/// <param name="obj">The object to compare with this circle.</param>
/// <returns>true if this circle and the specified object are equal; otherwise,
false.</returns>
public override readonly bool Equals(object obj) => obj is Circle other
&& Equals(other);

/// <summary>
/// Returns a value that indicates whether this circle and the specified circle
are equal.
/// </summary>
/// <param name="other">The circle to compare with this circle.</param>
/// <returns>true if this circle and the specified circle are equal; otherwise,
false.</returns>
public readonly bool Equals(Circle other) => this.X == other.X &&
                                             this.Y == other.Y &&
                                             this.Radius == other.Radius;
```

Next, add the following override for `GetHashCode` to support using circles in hash-based collections:

```
/// <summary>
/// Returns the hash code for this circle.
/// </summary>
/// <returns>The hash code for this circle as a 32-bit signed integer.</returns>
public override readonly int GetHashCode() => HashCode.Combine(X, Y, Radius);
```

Finally, add the following operator overloads to support using == and != with circles:

```
/// <summary>
/// Returns a value that indicates if the circle on the left hand side of the
equality operator is equal to the
/// circle on the right hand side of the equality operator.
/// </summary>
/// <param name="lhs">The circle on the left hand side of the equality
operator.</param>
/// <param name="rhs">The circle on the right hand side of the equality
operator.</param>
/// <returns>true if the two circles are equal; otherwise, false.</returns>
public static bool operator ==(Circle lhs, Circle rhs) => lhs.Equals(rhs);

/// <summary>
/// Returns a value that indicates if the circle on the left hand side of the
inequality operator is not equal to the
/// circle on the right hand side of the inequality operator.
/// </summary>
/// <param name="lhs">The circle on the left hand side of the inequality
operator.</param>
/// <param name="rhs">The circle on the right hand side fo the inequality
operator.</param>
/// <returns>true if the two circle are not equal; otherwise, false.</returns>
public static bool operator !=(Circle lhs, Circle rhs) => !lhs.Equals(rhs);
```

IEquatable interface implemented, red squiggles be gone.

> **TIP**
>
> The operator overloads allow you to compare circles using familiar syntax:
>
> ```
> Circle circle1 = new Circle(0, 0, 5);
> Circle circle2 = new Circle(0, 0, 5);
> bool areEqual = circle1 == circle2;    // Returns true
> ```

Now that we have a struct to represent a circle and check for overlapping, we will update our game to implement collision detection and responses.

Adding Collision To Our Game

If you run the game right now and move the slime around, you will notice a few issues that can be fixed by adding collision detection and response:

1. You can move the slime outside the bounds of the screen.
2. Nothing occurs when the slime collides with the bat.
3. The bat does not move, providing no challenge in the game.

We can now implement these features using collision detection and response in our game. In the *DungeonSlime* project (your main game project), open the `Game1.cs` file and make the following changes to the `Game1` class:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    // Tracks the position of the bat.
    private Vector2 _batPosition;

    // Tracks the velocity of the bat.
    private Vector2 _batVelocity;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        base.Initialize();

        // Set the initial position of the bat to be 10px
        // to the right of the slime.
        _batPosition = new Vector2(_slime.Width + 10, 0);

        // Assign the initial random velocity to the bat.
        AssignRandomBatVelocity();
    }
```

```
    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        // Check for keyboard input and handle it.
        CheckKeyboardInput();

        // Check for gamepad input and handle it.
        CheckGamePadInput();

        // Create a bounding rectangle for the screen.
        Rectangle screenBounds = new Rectangle(
            0,
            0,
            GraphicsDevice.PresentationParameters.BackBufferWidth,
            GraphicsDevice.PresentationParameters.BackBufferHeight
        );

        // Creating a bounding circle for the slime
        Circle slimeBounds = new Circle(
            (int)(_slimePosition.X + (_slime.Width * 0.5f)),
            (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
            (int)(_slime.Width * 0.5f)
        );

        // Use distance based checks to determine if the slime is within the
```

```
// bounds of the game screen, and if it is outside that screen edge,
// move it back inside.
if (slimeBounds.Left < screenBounds.Left)
{
    _slimePosition.X = screenBounds.Left;
}
else if (slimeBounds.Right > screenBounds.Right)
{
    _slimePosition.X = screenBounds.Right - _slime.Width;
}

if (slimeBounds.Top < screenBounds.Top)
{
    _slimePosition.Y = screenBounds.Top;
}
else if (slimeBounds.Bottom > screenBounds.Bottom)
{
    _slimePosition.Y = screenBounds.Bottom - _slime.Height;
}

// Calculate the new position of the bat based on the velocity.
Vector2 newBatPosition = _batPosition + _batVelocity;

// Create a bounding circle for the bat.
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
);

Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal.
if (batBounds.Left < screenBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = screenBounds.Left;
}
else if (batBounds.Right > screenBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = screenBounds.Right - _bat.Width;
}
```

```
        if (batBounds.Top < screenBounds.Top)
        {
            normal.Y = Vector2.UnitY.Y;
            newBatPosition.Y = screenBounds.Top;
        }
        else if (batBounds.Bottom > screenBounds.Bottom)
        {
            normal.Y = -Vector2.UnitY.Y;
            newBatPosition.Y = screenBounds.Bottom - _bat.Height;
        }

        // If the normal is anything but Vector2.Zero, this means the bat had
        // moved outside the screen edge so we should reflect it about the
        // normal.
        if (normal != Vector2.Zero)
        {
            normal.Normalize();
            _batVelocity = Vector2.Reflect(_batVelocity, normal);
        }

        _batPosition = newBatPosition;

        if (slimeBounds.Intersects(batBounds))
        {
            // Divide the width  and height of the screen into equal columns and
            // rows based on the width and height of the bat.
            int totalColumns = GraphicsDevice.PresentationParameters.BackBufferWidth
/ (int)_bat.Width;
            int totalRows = GraphicsDevice.PresentationParameters.BackBufferHeight
/ (int)_bat.Height;

            // Choose a random row and column based on the total number of each
            int column = Random.Shared.Next(0, totalColumns);
            int row = Random.Shared.Next(0, totalRows);

            // Change the bat position by setting the x and y values equal to
            // the column and row multiplied by the width and height.
            _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

            // Assign a new random velocity to the bat
            AssignRandomBatVelocity();
        }

        base.Update(gameTime);
    }
```

```
    private void AssignRandomBatVelocity()
    {
        // Generate a random angle.
        float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

        // Convert angle to a direction vector.
        float x = (float)Math.Cos(angle);
        float y = (float)Math.Sin(angle);
        Vector2 direction = new Vector2(x, y);

        // Multiply the direction vector by the movement speed.
        _batVelocity = direction * MOVEMENT_SPEED;
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
```

```
    }
}

private void CheckGamePadInput()
{
    GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];

    // If the A button is held down, the movement speed increases by 1.5
    // and the gamepad vibrates as feedback to the player.
    float speed = MOVEMENT_SPEED;
    if (gamePadOne.IsButtonDown(Buttons.A))
    {
        speed *= 1.5f;
        GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
    }
    else
    {
        GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
    }

    // Check thumbstick first since it has priority over which gamepad input
    // is movement.  It has priority since the thumbstick values provide a
    // more granular analog value that can be used for movement.
    if (gamePadOne.LeftThumbStick != Vector2.Zero)
    {
        _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
        _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
    }
    else
    {
        // If DPadUp is down, move the slime up on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadUp))
        {
            _slimePosition.Y -= speed;
        }

        // If DPadDown is down, move the slime down on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadDown))
        {
            _slimePosition.Y += speed;
        }

        // If DPapLeft is down, move the slime left on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
        {
            _slimePosition.X -= speed;
```

```
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite.
        _bat.Draw(SpriteBatch, _batPosition);

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes made here are:

1. The field `_batPosition` was added to track the position of the bat.
2. The field `_batVelocity` was added to track the velocity of the bat.
3. The `AssignRandomBatVelocity()` method was added which calculates a random x and y velocity for the bat to move at when called.
4. In **Initialize**, the initial position of the bat is set and `AssignRandomVelocity` is called to assign the initial velocity for the bat.
5. In **Update**, collision detection and response logic was added to perform the following in order:
    1. A **Rectangle** bound is created to represent the bounds of the screen.
    2. A `Circle` bound is created to represent the bounds of the slime.
    3. Distance based checks are performed to ensure that the slime cannot move outside of the screen, the resolution of which is to perform a blocking response.
    4. A new position for the bat is calculated based on the current velocity of the bat.
    5. A `Circle` bound is created to represent the bounds of the bat.
    6. Distance based checks are performed to ensure the bat cannot move outside of the screen, the resolution of which is to perform a bounce response.
    7. A collision check is made to determine if the slime and bat are colliding (bat "eating" the slime). If so, the bat is assigned a new random position within the screen and assigned a new random velocity.
6. In **Draw**, the bat is now drawn using the `_batPosition` value.

Running the game now

- The bat will start moving with a random velocity and bounce off the edges of the screen
- You can move the slime around, but cannot leave the bounds of the screen with the slime.
- If you move the slime to collide ("eat") the bat, the bat will respawn at a new location with a new velocity.

Figure 12-5: When the slime collides ("eats") the bat, the bat respawns in a new location on the screen with a random velocity assigned

Conclusion

In this chapter, you accomplished the following:

- Learned about different approaches to collision detection:
  - Distance-based checks for simple proximity detection.
  - Shape-based checks using circles and rectangles.
  - Complex polygon checks using SAT.
- Understood when to use different collision shapes:
  - Circles for round objects and rotation.
  - Rectangles for walls and platforms.
- Explored different types of collision responses:
  - Blocking to prevent objects from overlapping.
  - Triggering events when objects collide.
  - Bouncing to reflect objects off surfaces.
- Created reusable components:
  - Implemented a Circle struct for circle-based collision.
  - Added methods to detect circle intersections.
- Applied collision concepts to our game:
  - Added screen boundary collision for the slime.
  - Implemented bouncing behavior for the bat.
  - Created a trigger response when the slime "eats" the bat.

In the next chapter, we will explore using tilesets and tilemaps to create tile based environments for our game.

Test Your Knowledge

1. What is the difference between collision detection and collision response?

   Collision detection is determining when two objects overlap or intersect, while collision response is what happens after a collision is detected (like blocking movement, triggering events, or bouncing objects off each other).

2. When using Rectangle.Intersects for AABB collision, what four conditions must all be true for a collision to occur?

   For two rectangles A and B to collide:

   1. A's left edge must be less than B's right edge.
   2. A's right edge must be greater than B's left edge.
   3. A's top edge must be less than B's bottom edge.
   4. A's bottom edge must be greater than B's top edge.

3. When implementing circle collision, why do we compare the distance between centers to the sum of the radii?

   Two circles are colliding if the distance between their centers is less than the sum of their radii. If the distance is greater, they are separate. If the distance equals the sum of radii, they are just touching at one point.

4. When implementing bounce collision response, what two pieces of information does **Vector2.Reflect** need?

   Vector2.Reflect needs:

   1. The incoming vector (direction the object is moving).
   2. The normal vector (direction perpendicular to the surface being hit).

5. Why might you choose to use circle collision over rectangle collision for certain objects?

Circle collision might be chosen because:

- It is more accurate for round objects.
- It handles rotating objects better.
- It is simpler for continuous collision detection.
- It is natural for radius-based interactions.

6. In the blocking collision response example, why do we store the previous position before handling input?

We store the previous position so that if a collision occurs after movement, we can reset the object back to its last valid position. This prevents objects from moving through each other by undoing any movement that would cause overlap.

Chapter 13: Working with Tilemaps

Learn how to implement tile-based game environments using tilemaps and tilesets, including creating reusable classes for managing tiles and loading level designs from XML configuration files.

In the previous chapters, you have learned how to draw individual sprites and animated sprites from a texture atlas and handle collision detection. However, the game so far is lacking an actual world or environment to exist in; it is just sprites on a cornflower blue background. Most 2D games feature game worlds built from many tiles arranged in a grid-like patten. These *tilemaps* allow you to efficiently create large game environments without managing thousands of individual sprites.

In this chapter you will:

- Learn what tilemaps are and how they are used in game development.
- Create a `Tileset` class to manage collections of related tiles.
- Build a `Tilemap` class to render tile-based game worlds.
- Implement an XML-based tilemap loading system.
- Update our game to use tilemaps for the game environment.

Understanding Tilemaps

Tilemaps are a common technique used in 2D game development to create game worlds. Instead of positioning individual sprites for each element in the game world, a tilemap divides the world into a grid and places tiles from a *tileset* at each grid position.

What is a Tileset?

A tileset is a collection of small images (tiles) that can be combined and arranged to create game environments. Typically these are stored in a single texture atlas, similar to how we have been handing sprites and animations. Common examples of tiles might include:

- Floor and ground tiles.
- Walls and obstacle tiles.
- Decorative elements like plants and furniture.
- Special tiles like doors, ladders, or water.

Each tile in a tileset is assigned an ID number, which the tilemap uses to reference which tile goes where. For example, in *Figure 13-1* below, the tileset we will add to our game in a moment is shown on the left and on the right is the same tileset with an overlay showing how each tile is assigned an ID number.

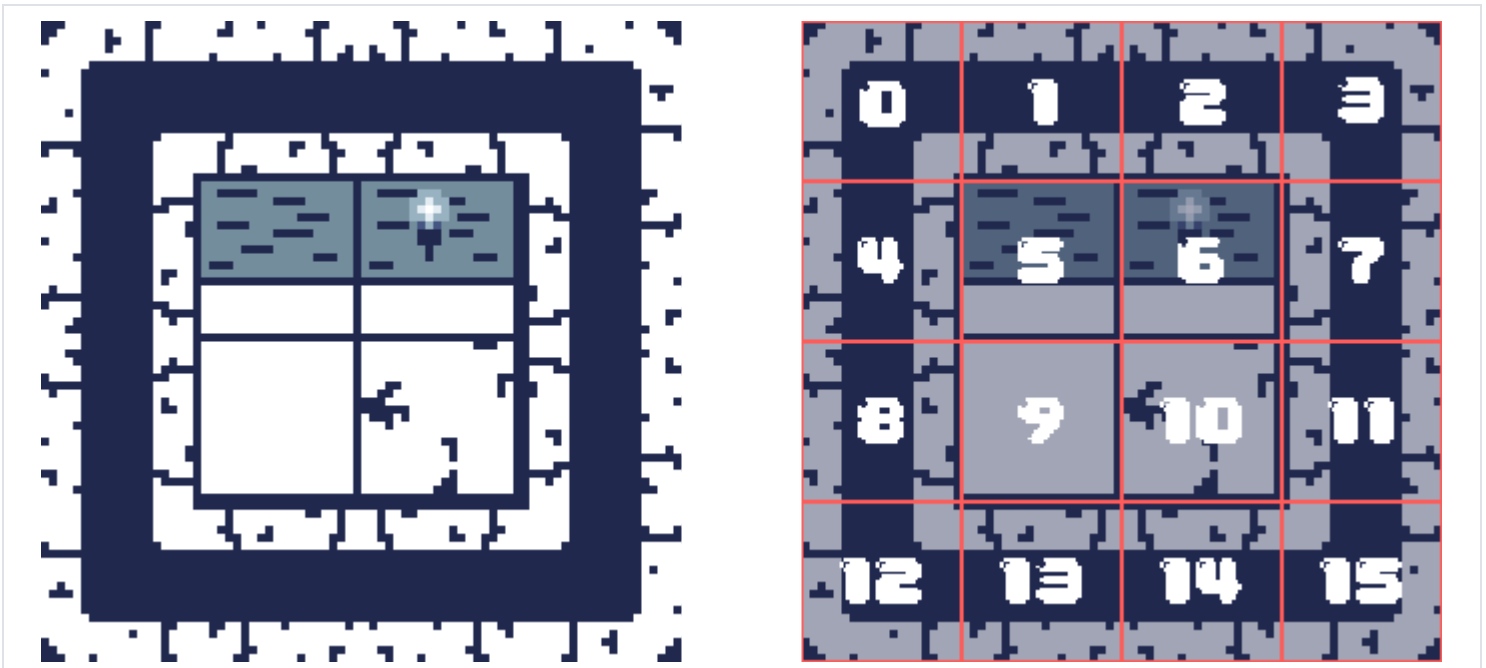


Figure 13-1: Left: Original dungeon tileset. Right: The same tileset with an overlay showing how each tile is assigned a numeric ID

What is a Tilemap?

A tilemap is a grid-based data structure that defines which tiles from a tileset appear at each position in the game world. The tilemap stores an ID for each cell in the grid, where the ID corresponds to a specific tile in the tileset.

For example, a simple tilemap may look like this conceptually:

```
00 01 02 01 03
04 05 06 05 07
08 09 10 09 11
04 09 09 09 07
12 13 14 13 15
```

If we took the above tilemap data and mapped each cell to the tile in the related tileset, it would look something similar to *Figure 13-2* below:

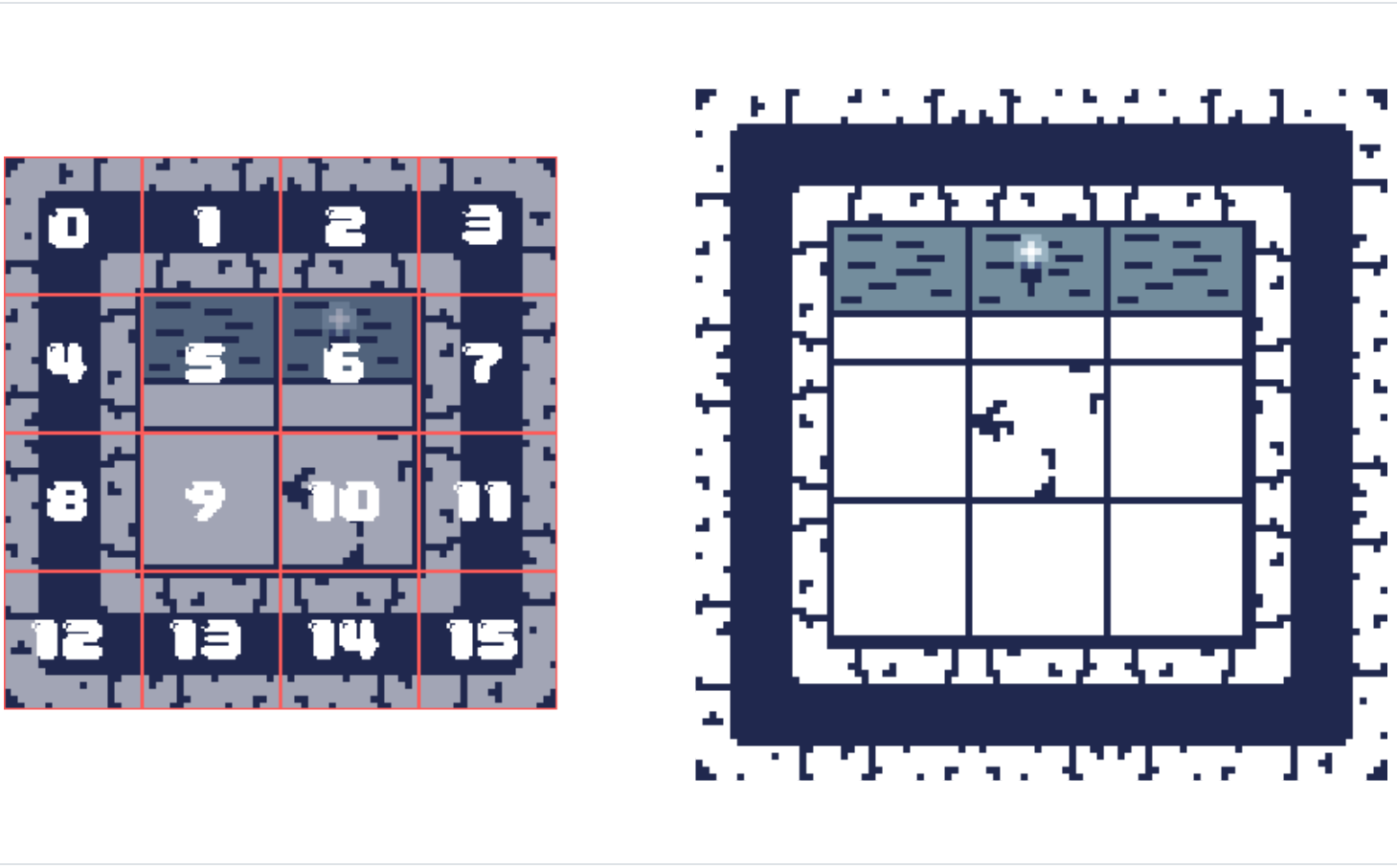


Figure 13-2: From tileset to tilemap. Left: Tileset with an overlay showing the tile IDs. Right: The tilemap created using the tiles arranged with the pattern from the code example above

This approach offers several advantage:

1. **Memory efficiency**: Instead of storing complete information about each tile's appearance, you only need to store a reference ID.
2. **Performance**: Drawing a tilemap can be optimized to reduce texture swapping compared to rendering many individual sprites.
3. **Design flexibility**: Tilemaps make it easy to create, modify, and load level designs from external files.

We will now take this concept and implement it in our game by creating a `Tileset` class and a `Tilemap` class.

The Tileset Class

The `Tileset` class will manage a collection of tiles from a texture atlas. Each tile will be represented as a `TextureRegion`, building on the tools in the library we created earlier.

In the `Graphics` folder of the *MonoGameLibrary* project, create a new file named `Tileset.cs` with the following code as the initial structure:

```
namespace MonoGameLibrary.Graphics;

public class Tileset
{

}
```

Tileset Properties and Fields

The `Tileset` class needs to store a `TextureRegion` for each of the individual tiles in the tile set and provide the dimensions (width and height) of the tiles. It should also offers additional properties that provide the total number of rows and columns in the tileset and the total number of tiles. Add the following fields and properties:

```
private readonly TextureRegion[] _tiles;

/// <summary>
/// Gets the width, in pixels, of each tile in this tileset.
/// </summary>
public int TileWidth { get; }

/// <summary>
/// Gets the height, in pixels, of each tile in this tileset.
/// </summary>
public int TileHeight { get; }

/// <summary>
/// Gets the total number of columns in this tileset.
/// </summary>
public int Columns { get; }

/// <summary>
/// Gets the total number of rows in this tileset.
/// </summary>
public int Rows { get; }

/// <summary>
/// Gets the total number of tiles in this tileset.
/// </summary>
public int Count { get; }
```

Tileset Constructor

The `Tileset` class constructor requires a source `TextureRegion` that represents the tileset and the width and height of the tiles. Based on these parameters provided, it can automatically divide the source `TextureRegion` into a grid of smaller texture regions and calculate the total number of rows, columns, and tiles.

Add the following constructor:

```
/// <summary>
/// Creates a new tileset based on the given texture region with the specified
/// tile width and height.
/// </summary>
/// <param name="textureRegion">The texture region that contains the tiles for the
tileset.</param>
/// <param name="tileWidth">The width of each tile in the tileset.</param>
/// <param name="tileHeight">The height of each tile in the tileset.</param>
public Tileset(TextureRegion textureRegion, int tileWidth, int tileHeight)
{
    TileWidth = tileWidth;
    TileHeight = tileHeight;
    Columns = textureRegion.Width / tileWidth;
    Rows = textureRegion.Height / tileHeight;
    Count = Columns * Rows;

    // Create the texture regions that make up each individual tile
    _tiles = new TextureRegion[Count];

    for (int i = 0; i < Count; i++)
    {
        int x = i % Columns * tileWidth;
        int y = i / Columns * tileHeight;
        _tiles[i] = new TextureRegion(textureRegion.Texture,
textureRegion.SourceRectangle.X + x, textureRegion.SourceRectangle.Y + y,
tileWidth, tileHeight);
    }
}
```

Tileset Methods

The `Tileset` class needs to provide methods to retrieve the `TextureRegion` of a tile based on the index (tile ID) or by the location (row and column) of the tile in the tileset. Add the following methods:

```
/// <summary>
/// Gets the texture region for the tile from this tileset at the given index.
/// </summary>
/// <param name="index">The index of the texture region in this tile set.</param>
/// <returns>The texture region for the tile form this tileset at the given
index.</returns>
public TextureRegion GetTile(int index) => _tiles[index];

/// <summary>
/// Gets the texture region for the tile from this tileset at the given location.
/// </summary>
/// <param name="column">The column in this tileset of the texture region.</param>
/// <param name="row">The row in this tileset of the texture region.</param>
/// <returns>The texture region for the tile from this tileset at given
location.</returns>
public TextureRegion GetTile(int column, int row)
{
    int index = row * Columns + column;
    return GetTile(index);
}
```

The Tilemap Class

Now that we have a `Tileset` class to define our tile collection, we need a `Tilemap` class to arrange these tiles into a game level. The `Tilemap` class will store which tile goes where in our game world and provide methods to draw the entire map.

In the `Graphics` folder of the *MonoGameLibrary* project, create a new file named `Tilemap.cs` with the following code as the initial structure:

```
using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace MonoGameLibrary.Graphics;

public class Tilemap
{

}
```

Tilemap Properties and Fields

The `Tilemap` class needs to store a reference to the tileset being used, along with an array of the tile IDs representing each tile in the map. It should also offer additional properties that provide the total number of rows and columns in the tilemap and the total number of tiles. Add the following fields and properties:

```
private readonly Tileset _tileset;
private readonly int[] _tiles;

/// <summary>
/// Gets the total number of rows in this tilemap.
/// </summary>
public int Rows { get; }

/// <summary>
/// Gets the total number of columns in this tilemap.
/// </summary>
public int Columns { get; }

/// <summary>
/// Gets the total number of tiles in this tilemap.
/// </summary>
public int Count { get; }

/// <summary>
/// Gets or Sets the scale factor to draw each tile at.
/// </summary>
public Vector2 Scale { get; set; }

/// <summary>
/// Gets the width, in pixels, each tile is drawn at.
/// </summary>
public float TileWidth => _tileset.TileWidth * Scale.X;

/// <summary>
/// Gets the height, in pixels, each tile is drawn at.
/// </summary>
public float TileHeight => _tileset.TileHeight * Scale.Y;
```

Tilemap Constructor

The `Tilemap` constructor requires the `Tilemap` to reference for each tile, the total number of columns and rows in the map, and the size (width and height) of each tile.

Add the following constructor:

```
/// <summary>
/// Creates a new tilemap.
/// </summary>
/// <param name="tileset">The tileset used by this tilemap.</param>
/// <param name="columns">The total number of columns in this tilemap.</param>
/// <param name="rows">The total number of rows in this tilemap.</param>
public Tilemap(Tileset tileset, int columns, int rows)
{
    _tileset = tileset;
    Rows = rows;
    Columns = columns;
    Count = Columns * Rows;
    Scale = Vector2.One;
    _tiles = new int[Count];
}
```

Tilemap Tile Management Methods

The `Tilemap` class should provide methods to set and retrieve tiles, either by index or location (rows and column). Add the following methods:

```
/// <summary>
/// Sets the tile at the given index in this tilemap to use the tile from
/// the tileset at the specified tileset id.
/// </summary>
/// <param name="index">The index of the tile in this tilemap.</param>
/// <param name="tilesetID">The tileset id of the tile from the tileset to
use.</param>
public void SetTile(int index, int tilesetID)
{
    _tiles[index] = tilesetID;
}

/// <summary>
/// Sets the tile at the given column and row in this tilemap to use the tile
/// from the tileset at the specified tileset id.
/// </summary>
/// <param name="column">The column of the tile in this tilemap.</param>
/// <param name="row">The row of the tile in this tilemap.</param>
/// <param name="tilesetID">The tileset id of the tile from the tileset to
use.</param>
public void SetTile(int column, int row, int tilesetID)
{
    int index = row * Columns + column;
    SetTile(index, tilesetID);
}

/// <summary>
/// Gets the texture region of the tile from this tilemap at the specified index.
/// </summary>
/// <param name="index">The index of the tile in this tilemap.</param>
/// <returns>The texture region of the tile from this tilemap at the specified
index.</returns>
public TextureRegion GetTile(int index)
{
    return _tileset.GetTile(_tiles[index]);
}

/// <summary>
/// Gets the texture region of the tile from this tilemap at the specified
/// column and row.
/// </summary>
/// <param name="column">The column of the tile in this tilemap.</param>
/// <param name="row">The row of the tile in this tilemap.</param>
/// <returns>The texture region of the tile from this tilemap at the specified
column and row.</returns>
public TextureRegion GetTile(int column, int row)
```

```
{
    int index = row * Columns + column;
    return GetTile(index);
}
```

Tilemap Draw Method

The `Tilemap` class needs a method to draw the tilemap by iterating through each of the tiles and drawing the `TextureRegion` for that tile at its correct position. Add the following method:

```
/// <summary>
/// Draws this tilemap using the given sprite batch.
/// </summary>
/// <param name="spriteBatch">The sprite batch used to draw this tilemap.</param>
public void Draw(SpriteBatch spriteBatch)
{
    for (int i = 0; i < Count; i++)
    {
        int tilesetIndex = _tiles[i];
        TextureRegion tile = _tileset.GetTile(tilesetIndex);

        int x = i % Columns;
        int y = i / Columns;

        Vector2 position = new Vector2(x * TileWidth, y * TileHeight);
        tile.Draw(spriteBatch, position, Color.White, 0.0f, Vector2.Zero, Scale,
SpriteEffects.None, 1.0f);
    }
}
```

Tilemap FromFile Method

The `Tilemap` class also requires a method to load and create an instance of the tilemap from an external configuration file. This allows us to separate level design from code. Add the following method:

```
/// <summary>
/// Creates a new tilemap based on a tilemap xml configuration file.
/// </summary>
/// <param name="content">The content manager used to load the texture for the
tileset.</param>
/// <param name="filename">The path to the xml file, relative to the content root
directory.</param>
/// <returns>The tilemap created by this method.</returns>
public static Tilemap FromFile(ContentManager content, string filename)
{
    string filePath = Path.Combine(content.RootDirectory, filename);

    using (Stream stream = TitleContainer.OpenStream(filePath))
    {
        using (XmlReader reader = XmlReader.Create(stream))
        {
            XDocument doc = XDocument.Load(reader);
            XElement root = doc.Root;

            // The <Tileset> element contains the information about the tileset
            // used by the tilemap.
            //
            // Example
            // <Tileset region="0 0 100 100" tileWidth="10"
tileHeight="10">contentPath</Tileset>
            //
            // The region attribute represents the x, y, width, and height
            // components of the boundary for the texture region within the
            // texture at the contentPath specified.
            //
            // the tileWidth and tileHeight attributes specify the width and
            // height of each tile in the tileset.
            //
            // the contentPath value is the contentPath to the texture to
            // load that contains the tileset
            XElement tilesetElement = root.Element("Tileset");

            string regionAttribute = tilesetElement.Attribute("region").Value;
            string[] split = regionAttribute.Split(" ",
StringSplitOptions.RemoveEmptyEntries);
            int x = int.Parse(split[0]);
            int y = int.Parse(split[1]);
            int width = int.Parse(split[2]);
            int height = int.Parse(split[3]);

            int tileWidth = int.Parse(tilesetElement.Attribute("tileWidth").Value);
```

```
            int tileHeight =
int.Parse(tilesetElement.Attribute("tileHeight").Value);
            string contentPath = tilesetElement.Value;

            // Load the texture 2d at the content path
            Texture2D texture = content.Load<Texture2D>(contentPath);

            // Create the texture region from the texture
            TextureRegion textureRegion = new TextureRegion(texture, x, y,
width, height);

            // Create the tileset using the texture region
            Tileset tileset = new Tileset(textureRegion, tileWidth, tileHeight);

            // The <Tiles> element contains lines of strings where each line
            // represents a row in the tilemap.  Each line is a space
            // separated string where each element represents a column in that
            // row.  The value of the column is the id of the tile in the
            // tileset to draw for that location.
            //
            // Example:
            // <Tiles>
            //      00 01 01 02
            //      03 04 04 05
            //      03 04 04 05
            //      06 07 07 08
            // </Tiles>
            XElement tilesElement = root.Element("Tiles");

            // Split the value of the tiles data into rows by splitting on
            // the new line character
            string[] rows = tilesElement.Value.Trim().Split('\n',
StringSplitOptions.RemoveEmptyEntries);

            // Split the value of the first row to determine the total number
of columns
            int columnCount = rows[0].Split(" ",
StringSplitOptions.RemoveEmptyEntries).Length;

            // Create the tilemap
            Tilemap tilemap = new Tilemap(tileset, columnCount, rows.Length);

            // Process each row
            for (int row = 0; row < rows.Length; row++)
            {
                // Split the row into individual columns
```

```
                string[] columns = rows[row].Trim().Split(" ",
StringSplitOptions.RemoveEmptyEntries);

                // Process each column of the current row
                for (int column = 0; column < columnCount; column++)
                {
                    // Get the tileset index for this location
                    int tilesetIndex = int.Parse(columns[column]);

                    // Add that region to the tilemap at the row and column location
                    tilemap.SetTile(column, row, tilesetIndex);
                }
            }

            return tilemap;
        }
    }
}
```

Updating the Game

Now that we have the `Tilemap` and `Tileset` classes defined, we can update our game to use them. We will need to

1. Update the texture atlas to include the tileset.
2. Create a tilemap xml configuration file.
3. Update the game to load the tilemap from the configuration file and draw it.

Update the Texture Atlas

Currently, the texture atlas we have been using only contains the sprites for the slime and bat animations. We need update it to a new version that contains the tileset as well.

Right-click the following image and save it as `atlas.png` in the `Content/images` directory of the *DungeonSlime* project (your main game project), overwriting the existing one.

> **NOTE**
>
> You do not need to do this in the MGCB editor as you are simply replacing the file and not altering any of its import properties.

Figure 13-3: The texture atlas for our game updated to include the tileset for the tilemap

> **NOTE**
>
> Since the slime and bat sprites are in the same position in the new texture atlas, we do not need to update the atlas XML configuration file.

Creating a Tilemap XML Configuration

Now that we have the texture atlas updated to include the tileset, we need to create a tilemap configuration that our game can load. The configuration will be an XML file that specifies the tileset to use and the arrangement of tiles in the tilemap.

We need to add this configuration file to our content project in the *Content/images* folder with the MGCB Editor, in the same way we did with the "atlas-definition.xml":

1. Open the *Content.mgcb* content project file in the MGCB Editor.
2. Right-click the *images* folder and choose *Add* > *New Item...*.
3. Select the *Xml Content (.xml)* type and name it `tilemap-definition`
4. Select the `tilemap-definition.xml` file you just created.
5. In the Properties panel, change the *Build Action* property from *Build* to *Copy*.
6. Save the changes in the MGCB Editor.

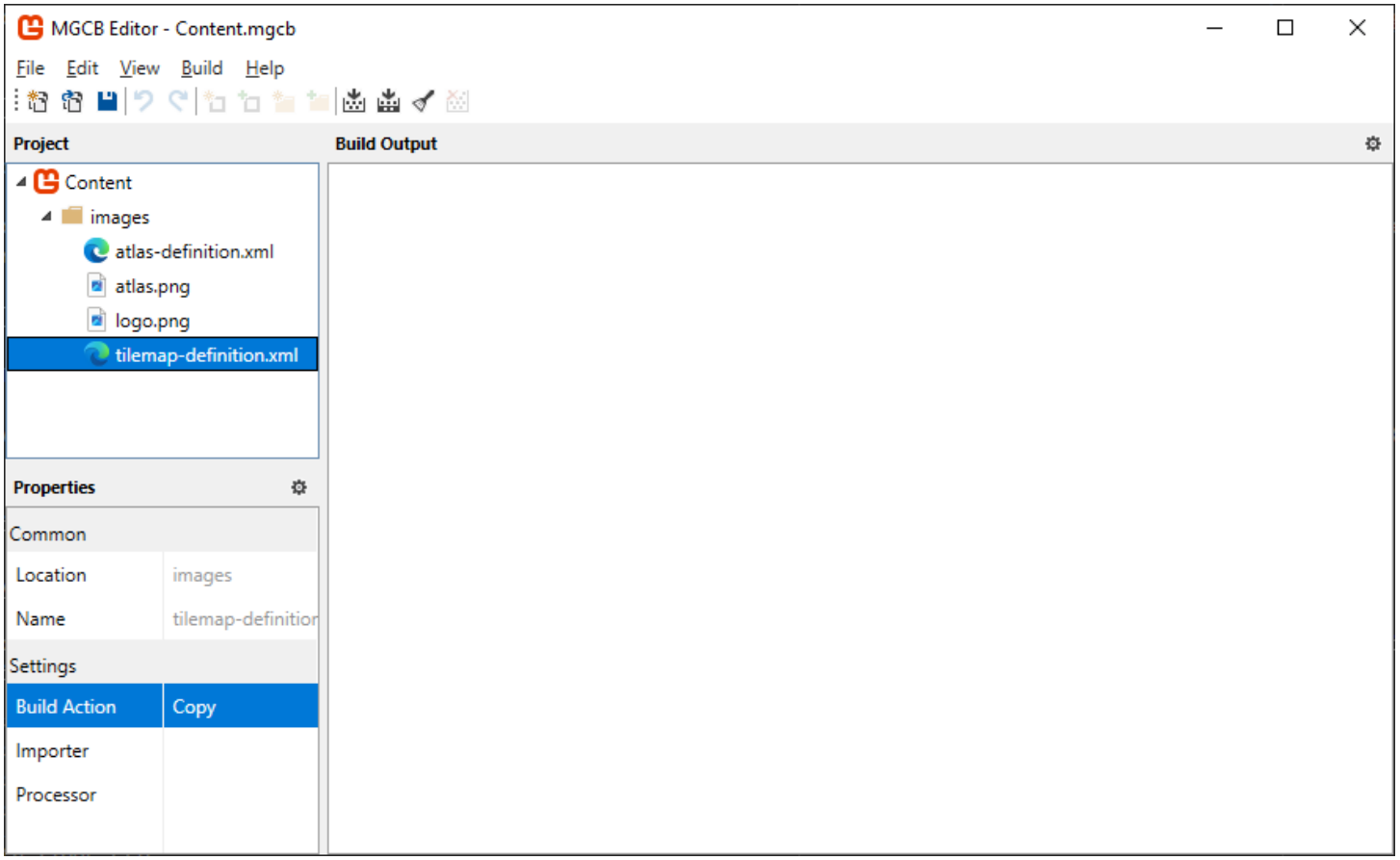


Figure 13-4: The Content project in the MGCB Editor with the tilemap-definition.xml file added and the Build Action property set to copy

7. Open the `tilemap-definition.xml` file in your code editor and replace its contents with the following and save it:

```
<?xml version="1.0" encoding="utf-8"?>
<Tilemap>
    <Tileset region="0 40 80 80" tileWidth="20"
tileHeight="20">images/atlas</Tileset>
    <Tiles>
        00 01 02 01 02 01 02 01 02 01 02 01 02 01 02 03
        04 05 05 06 05 05 06 05 05 06 05 05 06 05 05 07
        08 09 09 09 09 09 09 09 09 09 09 09 09 09 09 11
        04 09 09 09 09 09 09 09 10 09 09 09 09 10 09 07
        08 09 10 09 09 09 09 09 09 09 09 09 09 09 09 11
        04 09 09 09 09 09 09 09 09 09 09 09 09 09 09 07
        08 10 09 09 09 09 09 09 09 09 10 09 09 09 09 11
        04 09 09 09 09 09 10 09 09 09 09 09 09 09 09 07
        12 13 14 13 14 13 14 13 14 13 14 13 14 13 14 15
    </Tiles>
</Tilemap>
```

This tilemap configuration creates a simple dungeon layout with walls around the perimeter and an open floor in the middle. The tile IDs correspond to specific tiles in the tileset:

- `00`, `03`, `12`, `15`: Corner wall tiles (top-left, top-right, bottom-left, bottom-right).
- `01`, `02`, `13`, `14`: Horizontal wall tiles (top and bottom walls).
- `04`, `07`, `08`, `11`: Vertical wall tiles (left and right walls).
- `05` and `06`: Top floor edge tiles.
- `09`: Standard floor tile.
- `10`: Decorated floor tile with a crack in it.

Update the Game1 Class

With all of the assets now in place and configured, we can update the `Game1` class to load the tilemap and draw it. We will also need to update the collision logic so that the boundary is no longer the edge of the screen, but instead the edges of the wall tiles of the tilemap. Open `Game1.cs` and make the following updates:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    // Tracks the position of the bat.
    private Vector2 _batPosition;

    // Tracks the velocity of the bat.
    private Vector2 _batVelocity;

    // Defines the tilemap to draw.
    private Tilemap _tilemap;

    // Defines the bounds of the room that the slime and bat are contained within.
    private Rectangle _roomBounds;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        base.Initialize();

        Rectangle screenBounds = GraphicsDevice.PresentationParameters.Bounds;
```

```
        _roomBounds = new Rectangle(
            (int)_tilemap.TileWidth,
            (int)_tilemap.TileHeight,
            screenBounds.Width - (int)_tilemap.TileWidth * 2,
            screenBounds.Height - (int)_tilemap.TileHeight * 2
        );

        // Initial slime position will be the center tile of the tile map.
        int centerRow = _tilemap.Rows / 2;
        int centerColumn = _tilemap.Columns / 2;
        _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

        // Initial bat position will be in the top left corner of the room
        _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

        // Assign the initial random velocity to the bat.
        AssignRandomBatVelocity();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);

        // Create the tilemap from the XML configuration file.
        _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
        _tilemap.Scale = new Vector2(4.0f, 4.0f);
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
```

```
_bat.Update(gameTime);

// Check for keyboard input and handle it.
CheckKeyboardInput();

// Check for gamepad input and handle it.
CheckGamePadInput();

// Creating a bounding circle for the slime
Circle slimeBounds = new Circle(
    (int)(_slimePosition.X + (_slime.Width * 0.5f)),
    (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
    (int)(_slime.Width * 0.5f)
);

// Use distance based checks to determine if the slime is within the
// bounds of the game screen, and if it is outside that screen edge,
// move it back inside.
if (slimeBounds.Left < _roomBounds.Left)
{
    _slimePosition.X = _roomBounds.Left;
}
else if (slimeBounds.Right > _roomBounds.Right)
{
    _slimePosition.X = _roomBounds.Right - _slime.Width;
}

if (slimeBounds.Top < _roomBounds.Top)
{
    _slimePosition.Y = _roomBounds.Top;
}
else if (slimeBounds.Bottom > _roomBounds.Bottom)
{
    _slimePosition.Y = _roomBounds.Bottom - _slime.Height;
}

// Calculate the new position of the bat based on the velocity
Vector2 newBatPosition = _batPosition + _batVelocity;

// Create a bounding circle for the bat
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
);
```

```
Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal
if (batBounds.Left < _roomBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Left;
}
else if (batBounds.Right > _roomBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Right - _bat.Width;
}

if (batBounds.Top < _roomBounds.Top)
{
    normal.Y = Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Top;
}
else if (batBounds.Bottom > _roomBounds.Bottom)
{
    normal.Y = -Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Bottom - _bat.Height;
}

// If the normal is anything but Vector2.Zero, this means the bat had
// moved outside the screen edge so we should reflect it about the
// normal.
if (normal != Vector2.Zero)
{
    normal.Normalize();
    _batVelocity = Vector2.Reflect(_batVelocity, normal);
}

_batPosition = newBatPosition;

if (slimeBounds.Intersects(batBounds))
{
    // Choose a random row and column based on the total number of each
    int column = Random.Shared.Next(1, _tilemap.Columns - 1);
    int row = Random.Shared.Next(1, _tilemap.Rows - 1);

    // Change the bat position by setting the x and y values equal to
    // the column and row multiplied by the width and height.
```

```
            _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

            // Assign a new random velocity to the bat
            AssignRandomBatVelocity();
        }

        base.Update(gameTime);
    }

    private void AssignRandomBatVelocity()
    {
        // Generate a random angle
        float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

        // Convert angle to a direction vector
        float x = (float)Math.Cos(angle);
        float y = (float)Math.Sin(angle);
        Vector2 direction = new Vector2(x, y);

        // Multiply the direction vector by the movement speed
        _batVelocity = direction * MOVEMENT_SPEED;
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
```

```
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }
    }

    private void CheckGamePadInput()
    {
        GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];

        // If the A button is held down, the movement speed increases by 1.5
        // and the gamepad vibrates as feedback to the player.
        float speed = MOVEMENT_SPEED;
        if (gamePadOne.IsButtonDown(Buttons.A))
        {
            speed *= 1.5f;
            GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
        }
        else
        {
            GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
        }

        // Check thumbstick first since it has priority over which gamepad input
        // is movement.  It has priority since the thumbstick values provide a
        // more granular analog value that can be used for movement.
        if (gamePadOne.LeftThumbStick != Vector2.Zero)
        {
            _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
            _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
        }
        else
        {
            // If DPadUp is down, move the slime up on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadUp))
            {
                _slimePosition.Y -= speed;
            }

            // If DPadDown is down, move the slime down on the screen.
```

```
            if (gamePadOne.IsButtonDown(Buttons.DPadDown))
            {
                _slimePosition.Y += speed;
            }

            // If DPapLeft is down, move the slime left on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
            {
                _slimePosition.X -= speed;
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the tilemap.
        _tilemap.Draw(SpriteBatch);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite.
        _bat.Draw(SpriteBatch, _batPosition);

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes to the `Game1` class include:

1. The `_tilemap` field was added to hold the loaded tilemap.
2. The `_roombounds` **Rectangle** was added to define the playable area within the tilemap to keep the slime and bat inside the walls.
3. In **Initialize**:
    1. The `_roomBounds` is set based on the tilemap's tile size.
    2. The starting position of the slime is now set to be in the center of the room.
4. In **LoadContent**:
    1. The tilemap is loaded from the XML configuration file.
    2. The scale of the tilemap is set to a factor of 4.0.
5. In **Update**, the `screenBounds` variable was removed and the collision logic has been updated to instead use the `_roomBounds` instead.
6. In **Draw** the tilemap is drawn.

Running the game now with these changes, our game now visually transforms from a simple screen with sprites to a proper game environment with walls and floors. The slime and bat are now confined within the walls of the dungeon defined by our tilemap.

Figure 13-5: Gameplay with the tilemap rendered and the bat and slime contained within the dungeon walls

Additional Notes

While the method provided in this chapter offers a straightforward approach to loading tilemaps from external configuration files, several dedicated tools exist specifically for creating tilemaps for games. Popular options include Tiled, LDtk, and Ogmo. These specialized tools export map configurations in various formats such as XML (similar to what we implemented) or JSON, and often include additional features like multiple layers, object placement, and custom properties for tiles.

> **NOTE**
>
> In order to utilize Tilemaps from other tools, you will need a way to import those maps as MonoGame does not natively support them, one option is to use MonoGame.Extended which provides Tilemap importers for Tiled (and more in the future) as well as a host of other features.

Although these tools are more robust than our implementation, the underlying concept remains the same: a tilemap is fundamentally a grid layout where each cell references a tile ID from a tileset. The principles you have learned in this chapter form the foundation for working with any tilemap system, regardless of which tool you might use.

Conclusion

In this chapter, you accomplished the following:

- Learned about tilemaps and how they are used in 2D game development.
- Created a `Tileset` class to manage collections of tiles from a texture atlas.
- Implemented a `Tilemap` class to render grid-based game environments.
- Created an XML-based tilemap definition system for storing level layouts.
- Updated our game to use tilemaps for the game environment.

In the next chapter, we will start exploring audio to add sound effects when a collision occurs and background music to our game.

Test Your Knowledge

1. What is the main advantage of using tilemaps for game environments rather than individual sprites?

> Tilemaps offer several advantages: memory efficiency (reusing tiles instead of storing complete environments), performance optimization (batched rendering), and design flexibility (easier to create and modify levels). They allow creating large game worlds by reusing a small set of tiles in different arrangements.

2. What is the relationship between a tileset and a tilemap?

> A tileset is a collection of individual tiles stored in a texture atlas, where each tile has a unique ID. A tilemap is a grid-based structure that references tiles from the tileset by their IDs to create a complete game environment. The tileset provides the visual elements, while the tilemap defines their arrangement.

3. Why might you use an XML definition for a tilemap instead of hardcoding the tile layout?

> Using XML definitions for tilemaps separates level design from game code, offering several benefits: easier level editing (without changing code), support for multiple levels, ability to create external level editors, and better organization of game content. It also allows non-programmers like game designers to create and modify levels.

4. In our implementation, how does the Tilemap's Draw method work?

> The Tilemap's Draw method iterates through each position in the grid. For each position, it:
>
> 1. Retrieves the tile ID stored at that position.
> 2. Gets the corresponding texture region from the tileset.
> 3. Calculates the screen position based on the grid coordinates and tile size.
> 4. Draws the texture region at that position using the sprite batch.

Chapter 14: SoundEffects and Music

Learn how to load and play sound effects and background music in MonoGame including managing audio volume, looping, and handling multiple sound effects at once.

In Chapter 12, we implemented collision detection to enable interactions between game objects; the slime can now "eat" the bat, which respawns in a random location, while the bat bounces off walls of the dungeon. While these mechanics work visually, our game lacks an important element of player feedback: audio.

Audio plays a crucial role in game development by providing immediate feedback for player actions and creating atmosphere. Sound effects alert players when events occur (like collisions or collecting items), while background music helps establish mood and atmosphere.

In this chapter, you will:

- Learn how MonoGame handles different types of audio content.
- Learn how to load and play sound effects and music using the content pipeline.
- Implement sound effects for collision events.
- Add background music to enhance atmosphere.

We will first start by understanding how MonoGame approaches audio content.

Understanding Audio in MonoGame

Recall from Chapter 01 that MonoGame is an implementation of the XNA API. With XNA, there were two methods for implementing audio in your game: the *Microsoft Cross-Platform Audio Creation Tool* (XACT) and the simplified sound API.

IMPORTANT

XACT is a mini audio engineering studio where you can easily edit the audio for your game like editing volume, pitch, looping, applying effects, and other properties without having to do it in code. At that time, XACT for XNA games was akin to what FMOD Studio is today for game audio.

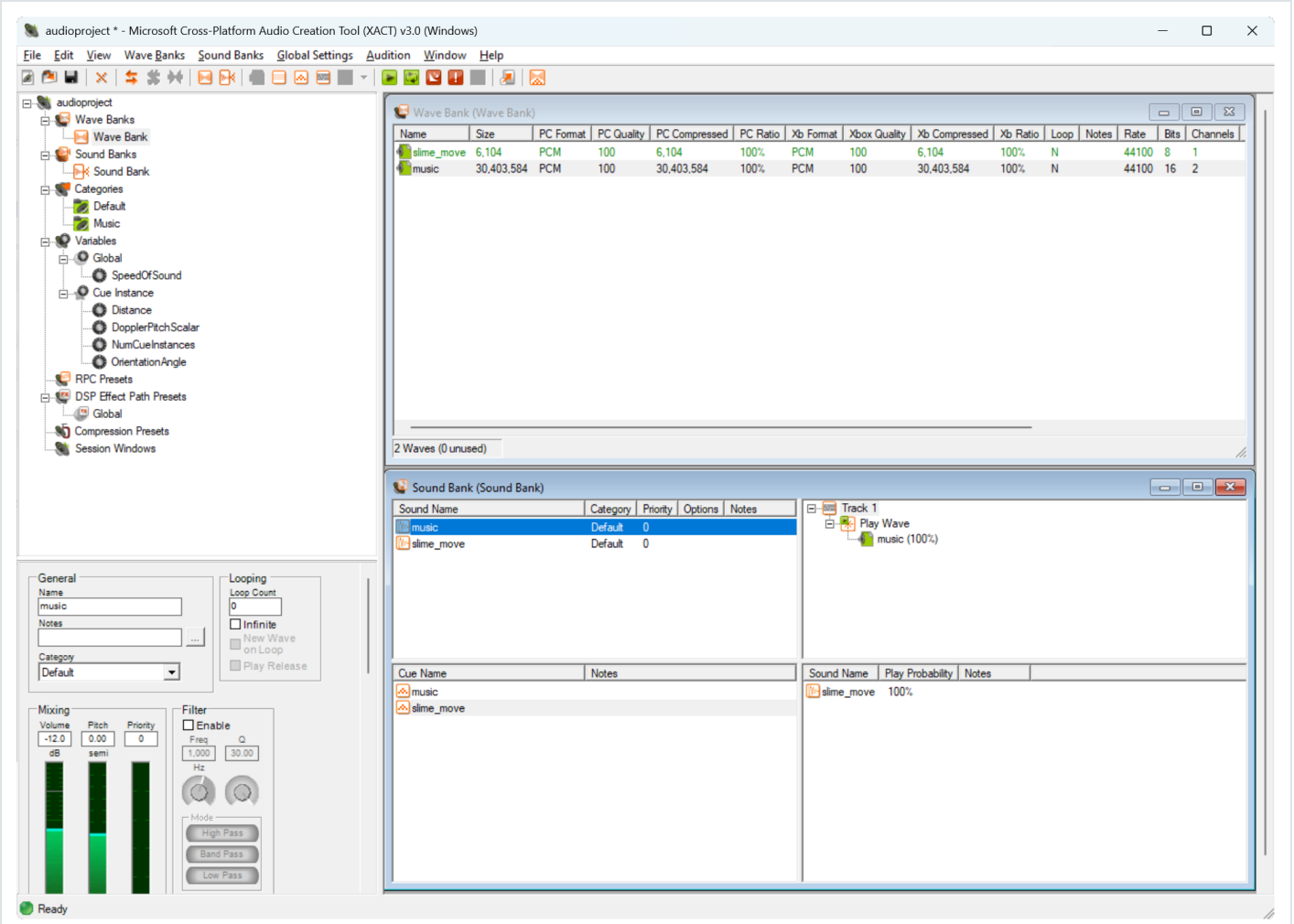


Figure 14-1: Microsoft Cross-Platform Audio Creation Tool

While XACT projects are still fully supported in MonoGame, it remains a Windows-only tool that has not been updated since Microsoft discontinued the original XNA, nor has its source code been made open source. Though it is possible to install XACT on modern Windows, the process can be complex.

For these reasons, this tutorial will focus on the simplified sound API, which provides all the core functionality needed for most games while remaining cross-platform compatible.

The simplified sound API approaches audio management through two distinct paths, each optimized for different use cases in games. When adding audio to your game, you need to consider how different types of sounds should be handled:

- **Sound Effects**: Short audio clips that need to play immediately and often simultaneously, like the bounce of a ball or feedback for picking up a collectable.
- **Music**: Longer audio pieces that play continuously in the background, like level themes.

MonoGame addresses these different needs through two main classes:

Sound Effects

The **SoundEffect** class handles short audio clips like:

- Collision sounds.
- Player action feedback (jumping, shooting, etc.).
- UI interactions (button clicks, menu navigation).
- Environmental effects (footsteps, ambient sounds).

The key characteristics of sound effects are:

- Loaded entirely into memory for quick access
- Can play multiple instances simultaneously:
  - Mobile platforms can have a maximum of 32 sounds playing simultaneously.
  - Desktop platforms have a maximum of 256 sounds playing simultaneously.
  - Consoles and other platforms have their own constraints, and you would need to refer to the SDK documentation for that platform.
- Lower latency playback (ideal for immediate feedback)
- Individual volume control per instance.

Music

The **Song** class handles longer audio pieces like background music. The key characteristics of songs are:

- Streamed from storage rather than loaded into memory.
- Only one song can be played at a time.
- Higher latency, but lower memory usage.

Throughout this chapter, we will use both classes to add audio feedback to our game; sound effects for the bat bouncing and being eaten by the slime, and background music to create atmosphere.

Loading Audio Content

Just like textures, audio content in MonoGame can be loaded through the content pipeline, optimizing the format for your target platform.

Supported Audio Formats

MonoGame supports several audio file formats for both sound effects and music:

- `.wav`: Uncompressed audio, ideal for short sound effects
- `.mp3`: Compressed audio, better for music and longer sounds
- `.ogg`: Open source compressed format, supported on all platforms
- `.wma`: Windows Media Audio format (not recommended for cross-platform games)

> **TIP**
>
> For sound effects, `.wav` files provide the best loading and playback performance since they do not need to be decompressed. For music, `.mp3` or `.ogg` files are better choices as they reduce file size while maintaining good quality.

Adding Audio Files

Adding audio files can be done through the content pipeline, just like we did for image files, using the MGCB Editor. When you add an audio file to the content project, the MGCB Editor will automatically select the appropriate importer and processor for the audio file based on the file extension.

The processor that are available for audio files file:

- **Sound Effects**: Processes the audio file as a **SoundEffect**. This is automatically selected for *.wav* files.
- **Song**: Processes the audio file as a **Song**. This is automatically selected for *.mp3*, *.ogg*, and *.wma* files.

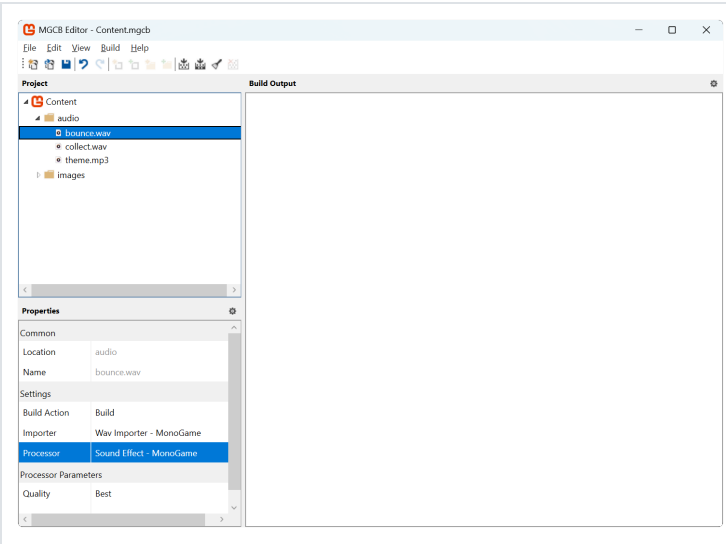


Figure 14-2: MGCB Editor properties panel showing Sound Effect content processor settings for .wav files

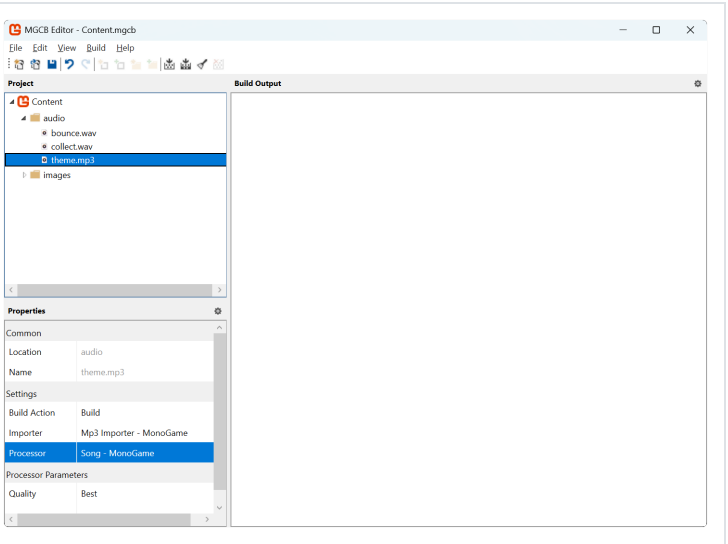


Figure 14-3: MGCB Editor properties panel showing Song content processor settings for .mp3 files

> **NOTE**
>
> While you typically will not need to change the processor it automatically selects, there may be times where you add files, such as *.mp3* files that are meant to be sound effects and not songs. Always double check that the processor selected is for the intended type.

Loading Sound Effects

To load a sound effect, we use **ContentManager.Load** with the **SoundEffect** type:

```
// Loading a SoundEffect using the content pipeline
SoundEffect soundEffect = Content.Load<SoundEffect>("soundEffect");
```

Loading Music

Loading music is similar, only we specify the **Song** type instead.

```
// Loading a Song using the content pipeline
Song song = Content.Load<Song>("song");
```

Playing Sound Effects

Sound effects are played using the **SoundEffect** class. This class provides two ways to play sounds:

1. Direct playback using **SoundEffect.Play**:

```
// Loading a SoundEffect using the content pipeline
SoundEffect soundEffect = Content.Load<SoundEffect>("soundEffect");

// Play the sound effect with default settings
soundEffect.Play();
```

2. Creating an instance using **SoundEffect.CreateInstance**:

```
// Loading a SoundEffect using the content pipeline
SoundEffect soundEffect = Content.Load<SoundEffect>("soundEffect");

// Create an instance we can control
SoundEffectInstance soundEffectInstance = soundEffect.CreateInstance();

// Adjust the properties of the instance as needed
soundEffectInstance.IsLooped = true;    // Make it loop
soundEffectInstance.Volume = 0.5f;      // Set half volume.

// Play the sound effect using the instance.
soundEffectInstance.Play();
```

- Use **SoundEffect.Play** for simple sound effects that you just want to play once.
- Use **SoundEffect.CreateInstance** when you need more control over the sound effect, like adjusting volume, looping, or managing multiple instances of the same sound.

SoundEffectInstance contains several properties that can be used to control how the sound effect is played:

Property	Type	Description
IsLooped	`bool`	Whether the sound should loop when it reaches the end.
Pan	`float`	Stereo panning between -1.0f (full left) and 1.0f (full right).
Pitch	`float`	Pitch adjustment between -1.0f (down one octave) and 1.0f (up one octave).

Property	Type	Description
State	SoundState	Current playback state (Playing, Paused, or Stopped).
Volume	`float`	Volume level between 0.0f (silent) and 1.0f (full volume).

> **NOTE**
> There is a lot more behind SoundEffectInstances such as the ability to play 3D sounds, as well as the advanced capabilities of the DynamicSoundEffectInstance which can build audio streams. However, these are beyond the scope of this beginners guide. If you wish to know more, then check the MonoGame documentation

Playing Music

Unlike sound effects, music is played through the **MediaPlayer** class. This static class manages playback of **Song** instances and provides global control over music playback:

```
// Loading a Song using the content pipeline
Song song = Content.Load<Song>("song");

// Set whether the song should repeat when finished
MediaPlayer.IsRepeating = true;

// Adjust the volume (0.0f to 1.0f)
MediaPlayer.Volume = 0.5f;

// Check if the media player is already playing, if so, stop it
if(MediaPlayer.State == MediaState.Playing)
{
    MediaPlayer.Stop();
}

// Start playing the background music
MediaPlayer.Play(song);
```

> **IMPORTANT**
>
> While **SoundEffect** instances can be played simultaneously, trying to play a new **Song** while another is playing will stop the current song in the best case, and in the worst case cause a crash on some platforms. In the example above, the state of the media player is checked first before we tell it to play a song. Checking the state first and stopping it manually if it is playing is best practice to prevent potential crashes.

Adding Audio To Our Game

Before we can add audio to our game, we need some sound files to work with. Download the following audio files:

- bounce.wav - For when the bat bounces off screen edges
- collect.wav - For when the slime eats the bat
- theme.ogg - Background music

> **NOTE**
>
> - *bounce.wav* is "Retro Impact Punch 07" by Davit Masia (https://kronbits.itch.io/retrosfx).
> - *collect.wav* is "Retro Jump Classic 08" by Davit Masia (https://kronbits.itch.io/retrosfx).
> - *theme.mp3* is "Exploration" by Luis Zuno (@ansimuz).

Add these files to your content project using the MGCB Editor:

1. Open the *Content.mgcb* file in the MGCB Editor.
2. Create a new folder called `audio` (right-click *Content* > *Add* > *New Folder*).
3. Right-click the new *audio* folder and choose *Add* > *Existing Item...*.
4. Navigate to and select the audio files you downloaded.
5. For each file that is added, check its properties in the Properties panel:
    - For `.wav` files, ensure the *Processor* is set to `Sound Effect`.
    - For `.mp3` files, ensure the *Processor* is set to `Song`.
6. Save the changes and close the MGCB Editor.

Next, open the `Game1.cs` file and update it to the following:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    // Tracks the position of the bat.
    private Vector2 _batPosition;

    // Tracks the velocity of the bat.
    private Vector2 _batVelocity;

    // Defines the tilemap to draw.
    private Tilemap _tilemap;

    // Defines the bounds of the room that the slime and bat are contained within.
    private Rectangle _roomBounds;

    // The sound effect to play when the bat bounces off the edge of the screen.
    private SoundEffect _bounceSoundEffect;

    // The sound effect to play when the slime eats a bat.
    private SoundEffect _collectSoundEffect;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {
```

```
    }

    protected override void Initialize()
    {
        base.Initialize();

        Rectangle screenBounds = GraphicsDevice.PresentationParameters.Bounds;

        _roomBounds = new Rectangle(
            (int)_tilemap.TileWidth,
            (int)_tilemap.TileHeight,
            screenBounds.Width - (int)_tilemap.TileWidth * 2,
            screenBounds.Height - (int)_tilemap.TileHeight * 2
        );

        // Initial slime position will be the center tile of the tile map.
        int centerRow = _tilemap.Rows / 2;
        int centerColumn = _tilemap.Columns / 2;
        _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

        // Initial bat position will the in the top left corner of the room
        _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

        // Assign the initial random velocity to the bat.
        AssignRandomBatVelocity();
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);

        // Create the tilemap from the XML configuration file.
        _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
        _tilemap.Scale = new Vector2(4.0f, 4.0f);
```

```
        // Load the bounce sound effect
        _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

        // Load the collect sound effect
        _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

        // Load the background theme music
        Song theme = Content.Load<Song>("audio/theme");

        // Ensure media player is not already playing on device, if so, stop it
        if (MediaPlayer.State == MediaState.Playing)
        {
            MediaPlayer.Stop();
        }

        // Play the background theme music.
        MediaPlayer.Play(theme);

        // Set the theme music to repeat.
        MediaPlayer.IsRepeating = true;
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        // Check for keyboard input and handle it.
        CheckKeyboardInput();

        // Check for gamepad input and handle it.
        CheckGamePadInput();

        // Creating a bounding circle for the slime
        Circle slimeBounds = new Circle(
            (int)(_slimePosition.X + (_slime.Width * 0.5f)),
            (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
            (int)(_slime.Width * 0.5f)
        );

        // Use distance based checks to determine if the slime is within the
        // bounds of the game screen, and if it is outside that screen edge,
```

```
// move it back inside.
if (slimeBounds.Left < _roomBounds.Left)
{
    _slimePosition.X = _roomBounds.Left;
}
else if (slimeBounds.Right > _roomBounds.Right)
{
    _slimePosition.X = _roomBounds.Right - _slime.Width;
}

if (slimeBounds.Top < _roomBounds.Top)
{
    _slimePosition.Y = _roomBounds.Top;
}
else if (slimeBounds.Bottom > _roomBounds.Bottom)
{
    _slimePosition.Y = _roomBounds.Bottom - _slime.Height;
}

// Calculate the new position of the bat based on the velocity
Vector2 newBatPosition = _batPosition + _batVelocity;

// Create a bounding circle for the bat
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
);

Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal
if (batBounds.Left < _roomBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Left;
}
else if (batBounds.Right > _roomBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Right - _bat.Width;
}

if (batBounds.Top < _roomBounds.Top)
```

```
        {
            normal.Y = Vector2.UnitY.Y;
            newBatPosition.Y = _roomBounds.Top;
        }
        else if (batBounds.Bottom > _roomBounds.Bottom)
        {
            normal.Y = -Vector2.UnitY.Y;
            newBatPosition.Y = _roomBounds.Bottom - _bat.Height;
        }

        // If the normal is anything but Vector2.Zero, this means the bat had
        // moved outside the screen edge so we should reflect it about the
        // normal.
        if (normal != Vector2.Zero)
        {
            normal.Normalize();
            _batVelocity = Vector2.Reflect(_batVelocity, normal);

            // Play the bounce sound effect
            _bounceSoundEffect.Play();
        }

        _batPosition = newBatPosition;

        if (slimeBounds.Intersects(batBounds))
        {
            // Choose a random row and column based on the total number of each
            int column = Random.Shared.Next(1, _tilemap.Columns - 1);
            int row = Random.Shared.Next(1, _tilemap.Rows - 1);

            // Change the bat position by setting the x and y values equal to
            // the column and row multiplied by the width and height.
            _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

            // Assign a new random velocity to the bat
            AssignRandomBatVelocity();

            // Play the collect sound effect
            _collectSoundEffect.Play();
        }

        base.Update(gameTime);
    }

    private void AssignRandomBatVelocity()
    {
```

```
        // Generate a random angle
        float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

        // Convert angle to a direction vector
        float x = (float)Math.Cos(angle);
        float y = (float)Math.Sin(angle);
        Vector2 direction = new Vector2(x, y);

        // Multiply the direction vector by the movement speed
        _batVelocity = direction * MOVEMENT_SPEED;
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }
    }
```

```
private void CheckGamePadInput()
{
    GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];

    // If the A button is held down, the movement speed increases by 1.5
    // and the gamepad vibrates as feedback to the player.
    float speed = MOVEMENT_SPEED;
    if (gamePadOne.IsButtonDown(Buttons.A))
    {
        speed *= 1.5f;
        GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
    }
    else
    {
        GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
    }

    // Check thumbstick first since it has priority over which gamepad input
    // is movement.  It has priority since the thumbstick values provide a
    // more granular analog value that can be used for movement.
    if (gamePadOne.LeftThumbStick != Vector2.Zero)
    {
        _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
        _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
    }
    else
    {
        // If DPadUp is down, move the slime up on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadUp))
        {
            _slimePosition.Y -= speed;
        }

        // If DPadDown is down, move the slime down on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadDown))
        {
            _slimePosition.Y += speed;
        }

        // If DPapLeft is down, move the slime left on the screen.
        if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
        {
            _slimePosition.X -= speed;
        }
```

```
            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the tilemap
        _tilemap.Draw(SpriteBatch);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite.
        _bat.Draw(SpriteBatch, _batPosition);

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes here are:

1. Added the `using Microsoft.Xna.Framework.Audio;` and `using Microsoft.Xna.Framework.Media;` directories to access the **Song** and **SoundEffect** classes.
2. Added the `_boundSoundEffect` and `_collectSoundEffect` fields to store those sound effects when loaded and use them for playback.
3. In **LoadContent**
    1. The bounce and collect sound effects are loaded using the content manager.
    2. The background theme music is loaded using the content manager.
    3. The background music is played using the media player, checking its state first.
    4. The **MediaPlayer.IsRepeating** is set to `true` so the background music loops.
4. In **Update**:
    1. The bounce sound effect is played when the bat bounces off the edge of the screen.
    2. The collect sound effect is played when the slime eats the bat.

Running the game now, the theme music plays in the background, you can hear the bat bounce off the edge of the screen, and if you move the slime to eat the bat, you hear that as well.

Figure 14-4: Gameplay with audio.

Conclusion

In this chapter, you accomplished the following:

- Learned about MonoGame's audio system including sound effects and music.
- Explored the key differences between:
    - Sound effects (short, multiple simultaneous playback).
    - Music (longer, streamed, single playback).
- Added audio content to your game project through the content pipeline.
- Loaded audio files using the ContentManager.
- Implemented audio feedback in your game:
    - Background music to set atmosphere.
    - Sound effects for bat bouncing and collection events.
- Learned best practices for handling audio playback across different platforms.

In the next chapter, we will explore additional ways to manage audio by creating an audio controller module that will help with common tasks such as volume control, muting, and state management.

Test Your Knowledge

1. What are the two main classes MonoGame provides for audio playback and how do they differ?

   MonoGame provides:

   - **SoundEffect** for short audio clips (loaded entirely into memory, multiple can play at once) and
   - **Song** for longer audio like music (streamed from storage, only one can play at a time).

2. Why is it important to check if **MediaPlayer** is already playing before starting a new song?

   Checking if MediaPlayer is already playing and stopping it if necessary helps prevent crashes on some platforms. Since only one song can play at a time, properly stopping the current song before starting a new one ensures reliable behavior across different platforms.

3. What file formats are best suited for sound effects and music, respectively, and why?

   For sound effects, .wav files are generally best because they are uncompressed and load quickly into memory for immediate playback. For music, compressed formats like .mp3 or .ogg are better suited because they greatly reduce file size while maintaining good audio quality, which is important for longer audio that's streamed rather than fully loaded.

4. What is the difference between using **SoundEffect.Play** directly and creating a **SoundEffectInstance**?

   - **SoundEffect.Play** is simpler but provides limited control - it plays the sound once with basic volume/pitch/pan settings.
   - Creating a **SoundEffectInstance** gives more control including the ability to pause, resume, loop, and change properties during playback, as well as track the sound's state.

5. How many sound effects can play simultaneously on different platforms?

The number of simultaneous sound effects varies by platform:

- Mobile platforms: maximum of 32 sounds.
- Desktop platforms: maximum of 256 sounds.
- Consoles and other platforms have their own constraints specified in their respective SDK documentation.

Chapter 15: Audio Controller

Learn how to create a reusable audio controller class to manage sound effects and music, including volume control, muting/unmuting, and proper resource cleanup.

While playing sounds and music using the simplified sound API is straightforward, a game needs to handle various audio states and resource cleanup including:

- Track and manage sound effect instances that are created.
- Dispose of sound effect instances when they are finished.
- Handle volume control for songs and sound effects.
- Manage audio states (pause/resume, mute/unmute).

In this chapter you will:

- Learn how to create a central audio management system.
- Implement proper resource tracking and cleanup for sound effects.
- Build methods to control audio state (play/pause, mute/unmute).
- Add global volume control for different audio types.
- Integrate the audio controller with your game's core systems.
- Implement keyboard shortcuts for audio control.

By the end of this chapter, you will have an audio control system that can be easily reused in future game projects.

The AudioController Class

To get started, in the *MonoGameLibrary* project:

1. Create a new folder named `Audio`.

2. Add a new class file named `AudioController.cs` to the `Audio` folder you just created.

3. Add the following code as the initial structure for the class

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Media;

namespace MonoGameLibrary.Audio;

public class AudioController : IDisposable
{

}
```

> **NOTE**
>
> The `AudioController` class will implement the `IDisposable` interface, This interface is part of .NET and provides a standardized implementation for an object to release resources. Implementing `IDisposable` allows other code to properly clean up the resources held by our audio controller when it is no longer needed. For more information on `IDisposable`, you can read the Implement a Dispose Method article on Microsoft Learn.

AudioController Properties and Fields

The `AudioController` will need to track sound effect instances created for cleanup and track the state and volume levels of songs and sound effects when toggling between mute states.

Add the following fields and properties:

```
    // Tracks sound effect instances created so they can be paused, unpaused,
and/or disposed.
    private readonly List<SoundEffectInstance> _activeSoundEffectInstances;

    // Tracks the volume for song playback when muting and unmuting.
    private float _previousSongVolume;

    // Tracks the volume for sound effect playback when muting and unmuting.
    private float _previousSoundEffectVolume;

    /// <summary>
    /// Gets a value that indicates if audio is muted.
    /// </summary>
    public bool IsMuted { get; private set; }

/// <summary>
    /// Gets or Sets the global volume of songs.
    /// </summary>
    /// <remarks>
    /// If IsMuted is true, the getter will always return back 0.0f and the
    /// setter will ignore setting the volume.
    /// </remarks>
    public float SongVolume
    {
        get
        {
            if(IsMuted)
            {
                return 0.0f;
            }

            return MediaPlayer.Volume;
        }
        set
        {
            if(IsMuted)
            {
                return;
            }

            MediaPlayer.Volume = Math.Clamp(value, 0.0f, 1.0f);
        }
    }

    /// <summary>
    /// Gets or Sets the global volume of sound effects.
```

```
/// </summary>
/// <remarks>
/// If IsMuted is true, the getter will always return back 0.0f and the
/// setter will ignore setting the volume.
/// </remarks>
public float SoundEffectVolume
{
    get
    {
        if(IsMuted)
        {
            return 0.0f;
        }

        return SoundEffect.MasterVolume;
    }
    set
    {
        if(IsMuted)
        {
            return;
        }

        SoundEffect.MasterVolume = Math.Clamp(value, 0.0f, 1.0f);
    }
}

/// <summary>
/// Gets a value that indicates if this audio controller has been disposed.
/// </summary>
public bool IsDisposed { get; private set; }
```

AudioController Constructor

The constructor just initializes the collection used to track the sound effect instances.

Add the following constructor and finalizer:

```
/// <summary>
/// Creates a new audio controller instance.
/// </summary>
public AudioController()
{
    _activeSoundEffectInstances = new List<SoundEffectInstance>();
}

// Finalizer called when object is collected by the garbage collector.
~AudioController() => Dispose(false);
```

> **NOTE**
>
> The `AudioController` class implements a finalizer method `~AudioManager()`. This method is called when an instance of the class is collected by the garbage collector and is here as part of the `IDisposable` implementation.

AudioController Methods

The `AudioController` needs methods to:

- Update it to check for resources to clean up.
- Playing sound effects and songs
- State management (play/pause, mute/unmute)
- Volume control
- Implement the `IDisposable` interface.

So lets add them below.

AudioController Update

The `Update` method will check for existing sound effect instances that have expired and properly dispose of them. Add the following method:

```
/// <summary>
/// Updates this audio controller.
/// </summary>
public void Update()
{
    for (int i = _activeSoundEffectInstances.Count - 1; i >= 0; i--)
    {
        SoundEffectInstance instance = _activeSoundEffectInstances[i];

        if (instance.State == SoundState.Stopped)
        {
            if (!instance.IsDisposed)
            {
                instance.Dispose();
            }
            _activeSoundEffectInstances.RemoveAt(i);
        }
    }
}
```

AudioController Playback

While the MonoGame simplified audio API allows sound effects to be played in a fire and forget manner, doing it this way does not work if you need to pause them because the game paused. Instead, we can add playback methods through the `AudioController` that can track the sound effect instances and pause them if needed, as well as checking the media player state before playing a song.

Add the following methods:

```
/// <summary>
/// Plays the given sound effect.
/// </summary>
/// <param name="soundEffect">The sound effect to play.</param>
/// <returns>The sound effect instance created by this method.</returns>
public SoundEffectInstance PlaySoundEffect(SoundEffect soundEffect)
{
    return PlaySoundEffect(soundEffect, 1.0f, 0.0f, 0.0f, false);
}

/// <summary>
/// Plays the given sound effect with the specified properties.
/// </summary>
/// <param name="soundEffect">The sound effect to play.</param>
/// <param name="volume">The volume, ranging from 0.0 (silence) to 1.0 (full
volume).</param>
/// <param name="pitch">The pitch adjustment, ranging from -1.0 (down an octave) to
0.0 (no change) to 1.0 (up an octave).</param>
/// <param name="pan">The panning, ranging from -1.0 (left speaker) to 0.0
(centered), 1.0 (right speaker).</param>
/// <param name="isLooped">Whether the the sound effect should loop after
playback.</param>
/// <returns>The sound effect instance created by playing the sound
effect.</returns>
/// <returns>The sound effect instance created by this method.</returns>
public SoundEffectInstance PlaySoundEffect(SoundEffect soundEffect, float volume,
float pitch, float pan, bool isLooped)
{
    // Create an instance from the sound effect given.
    SoundEffectInstance soundEffectInstance = soundEffect.CreateInstance();

    // Apply the volume, pitch, pan, and loop values specified.
    soundEffectInstance.Volume = volume;
    soundEffectInstance.Pitch = pitch;
    soundEffectInstance.Pan = pan;
    soundEffectInstance.IsLooped = isLooped;

    // Tell the instance to play
    soundEffectInstance.Play();

    // Add it to the active instances for tracking
    _activeSoundEffectInstances.Add(soundEffectInstance);

    return soundEffectInstance;
}
```

```
/// <summary>
/// Plays the given song.
/// </summary>
/// <param name="song">The song to play.</param>
/// <param name="isRepeating">Optionally specify if the song should repeat.  Default
is true.</param>
public void PlaySong(Song song, bool isRepeating = true)
{
    // Check if the media player is already playing, if so, stop it.
    // If we do not stop it, this could cause issues on some platforms
    if (MediaPlayer.State == MediaState.Playing)
    {
        MediaPlayer.Stop();
    }

    MediaPlayer.Play(song);
    MediaPlayer.IsRepeating = isRepeating;
}
```

AudioController State Control

The `AudioController` provides methods to control the state of audio playback including pausing and resuming audio as well as muting and unmuting.

Add the following methods:

```
/// <summary>
/// Pauses all audio.
/// </summary>
public void PauseAudio()
{
    // Pause any active songs playing.
    MediaPlayer.Pause();

    // Pause any active sound effects.
    foreach (SoundEffectInstance soundEffectInstance in _activeSoundEffectInstances)
    {
        soundEffectInstance.Pause();
    }
}

/// <summary>
/// Resumes play of all previous paused audio.
/// </summary>
public void ResumeAudio()
{
    // Resume paused music
    MediaPlayer.Resume();

    // Resume any active sound effects.
    foreach (SoundEffectInstance soundEffectInstance in _activeSoundEffectInstances)
    {
        soundEffectInstance.Resume();
    }
}

/// <summary>
/// Mutes all audio.
/// </summary>
public void MuteAudio()
{
    // Store the volume so they can be restored during ResumeAudio
    _previousSongVolume = MediaPlayer.Volume;
    _previousSoundEffectVolume = SoundEffect.MasterVolume;

    // Set all volumes to 0
    MediaPlayer.Volume = 0.0f;
    SoundEffect.MasterVolume = 0.0f;

    IsMuted = true;
}
```

```
/// <summary>
/// Unmutes all audio to the volume level prior to muting.
/// </summary>
public void UnmuteAudio()
{
    // Restore the previous volume values.
    MediaPlayer.Volume = _previousSongVolume;
    SoundEffect.MasterVolume = _previousSoundEffectVolume;

    IsMuted = false;
}

/// <summary>
/// Toggles the current audio mute state.
/// </summary>
public void ToggleMute()
{
    if (IsMuted)
    {
        UnmuteAudio();
    }
    else
    {
        MuteAudio();
    }
}
```

AudioController IDisposable Implementation

Finally, the `AudioController` is required to implement the `IDisposable` interface, to complete this add the following methods:

```
/// <summary>
/// Disposes of this audio controller and cleans up resources.
/// </summary>
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

/// <summary>
/// Disposes this audio controller and cleans up resources.
/// </summary>
/// <param name="disposing">Indicates whether managed resources should be
disposed.</param>
protected void Dispose(bool disposing)
{
    if(IsDisposed)
    {
        return;
    }

    if (disposing)
    {
        foreach (SoundEffectInstance soundEffectInstance
in _activeSoundEffectInstances)
        {
            soundEffectInstance.Dispose();
        }
        _activeSoundEffectInstances.Clear();
    }

    IsDisposed = true;
}
```

Games often use limited system resources like audio channels, when we are done with these resources we need to clean them up properly. In .NET, the standard way to handle resource cleanup is through the `IDisposable` interface.

Think of `IDisposable` like a cleanup checklist that runs when you are finished with something:

1. The interface provides a `Dispose` method that contains all cleanup logic.
2. When called, `Dispose` releases any resources the class was using.
3. Even if you forget to call `Dispose`, the finalizer acts as a backup cleanup mechanism.

For our `AudioController`, implementing `IDisposable` means we can ensure all sound effect instances are properly stopped and disposed when our game ends, preventing resource leaks.

> **NOTE**
>
> Fore more information on `IDisposable` and the `Dispose` method, check out the Implementing a Dispose Method article on Microsoft Learn.

Implementing the AudioController Class

Now that we have the audio controller class complete, we can update the game to use it. We will do this in two steps:

1. First, update the `Core` class to add the `AudioController` globally.
2. Update the `Game1` class to use the global audio controller from `Core`.

Updating the Core Class

The `Core` class serves as our base game class, so we will update it first to add and expose the `AudioController` globally. Open the `Core.cs` file in the *MonoGameLibrary* project and update it to the following:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary.Audio;
using MonoGameLibrary.Input;

namespace MonoGameLibrary;

public class Core : Game
{
    internal static Core s_instance;

    /// <summary>
    /// Gets a reference to the Core instance.
    /// </summary>
    public static Core Instance => s_instance;

    /// <summary>
    /// Gets the graphics device manager to control the presentation of graphics.
    /// </summary>
    public static GraphicsDeviceManager Graphics { get; private set; }

    /// <summary>
    /// Gets the graphics device used to create graphical resources and perform
primitive rendering.
    /// </summary>
    public static new GraphicsDevice GraphicsDevice { get; private set; }

    /// <summary>
    /// Gets the sprite batch used for all 2D rendering.
    /// </summary>
    public static SpriteBatch SpriteBatch { get; private set; }

    /// <summary>
    /// Gets the content manager used to load global assets.
    /// </summary>
    public static new ContentManager Content { get; private set; }

    /// <summary>
    /// Gets a reference to to the input management system.
    /// </summary>
    public static InputManager Input { get; private set; }

    /// <summary>
```

```
    /// Gets or Sets a value that indicates if the game should exit when the esc key
on the keyboard is pressed.
    /// </summary>
    public static bool ExitOnEscape { get; set; }

    /// <summary>
    /// Gets a reference to the audio control system.
    /// </summary>
    public static AudioController Audio { get; private set; }

    /// <summary>
    /// Creates a new Core instance.
    /// </summary>
    /// <param name="title">The title to display in the title bar of the game
window.</param>
    /// <param name="width">The initial width, in pixels, of the game
window.</param>
    /// <param name="height">The initial height, in pixels, of the game
window.</param>
    /// <param name="fullScreen">Indicates if the game should start in fullscreen
mode.</param>
    public Core(string title, int width, int height, bool fullScreen)
    {
        // Ensure that multiple cores are not created.
        if (s_instance != null)
        {
            throw new InvalidOperationException($"Only a single Core instance can
be created");
        }

        // Store reference to engine for global member access.
        s_instance = this;

        // Create a new graphics device manager.
        Graphics = new GraphicsDeviceManager(this);

        // Set the graphics defaults.
        Graphics.PreferredBackBufferWidth = width;
        Graphics.PreferredBackBufferHeight = height;
        Graphics.IsFullScreen = fullScreen;

        // Apply the graphic presentation changes.
        Graphics.ApplyChanges();

        // Set the window title.
        Window.Title = title;
```

```
        // Set the core's content manager to a reference of the base Game's
        // content manager.
        Content = base.Content;

        // Set the root directory for content.
        Content.RootDirectory = "Content";

        // Mouse is visible by default.
        IsMouseVisible = true;

        // Exit on escape is true by default
        ExitOnEscape = true;
    }

    protected override void Initialize()
    {
        base.Initialize();

        // Set the core's graphics device to a reference of the base Game's
        // graphics device.
        GraphicsDevice = base.GraphicsDevice;

        // Create the sprite batch instance.
        SpriteBatch = new SpriteBatch(GraphicsDevice);

        // Create a new input manager.
        Input = new InputManager();

        // Create a new audio controller.
        Audio = new AudioController();
    }

    protected override void UnloadContent()
    {
        // Dispose of the audio controller.
        Audio.Dispose();

        base.UnloadContent();
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the input manager.
        Input.Update(gameTime);
```

```
            // Update the audio controller.
            Audio.Update();

            if (ExitOnEscape && Input.Keyboard.IsKeyDown(Keys.Escape))
            {
                Exit();
            }

            base.Update(gameTime);
        }
    }
```

The key changes made here are:

1. Added the `using MonoGameLibrary.Audio;` directive to access the `AudioController` class.
2. Added a static `Audio` property to provide global access to the audio controller.
3. Created the new audio controller instance in the `Initialize` method.
4. Added an override for the `UnloadContent` method where we dispose of the audio controller.
5. The audio controller is updated in the `Update` method.

Updating the Game1 Class

Next, update the `Game1` class to use the audio controller for audio playback. Open `Game1.cs` and make the following updates:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    // Tracks the position of the bat.
    private Vector2 _batPosition;

    // Tracks the velocity of the bat.
    private Vector2 _batVelocity;

    // Defines the tilemap to draw.
    private Tilemap _tilemap;

    // Defines the bounds of the room that the slime and bat are contained within.
    private Rectangle _roomBounds;

    // The sound effect to play when the bat bounces off the edge of the screen.
    private SoundEffect _bounceSoundEffect;

    // The sound effect to play when the slime eats a bat.
    private SoundEffect _collectSoundEffect;

    // The background theme song
    private Song _themeSong;
```

```
    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        base.Initialize();

        Rectangle screenBounds = GraphicsDevice.PresentationParameters.Bounds;

        _roomBounds = new Rectangle(
            (int)_tilemap.TileWidth,
            (int)_tilemap.TileHeight,
            screenBounds.Width - (int)_tilemap.TileWidth * 2,
            screenBounds.Height - (int)_tilemap.TileHeight * 2
        );

        // Initial slime position will be the center tile of the tile map.
        int centerRow = _tilemap.Rows / 2;
        int centerColumn = _tilemap.Columns / 2;
        _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

        // Initial bat position will the in the top left corner of the room.
        _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

        // Assign the initial random velocity to the bat.
        AssignRandomBatVelocity();

        // Start playing the background music.
        Audio.PlaySong(_themeSong);
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
```

```
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);

        // Create the tilemap from the XML configuration file.
        _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
        _tilemap.Scale = new Vector2(4.0f, 4.0f);

        // Load the bounce sound effect.
        _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

        // Load the collect sound effect.
        _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

        // Load the background theme music.
        _themeSong = Content.Load<Song>("audio/theme");
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);

        // Update the bat animated sprite.
        _bat.Update(gameTime);

        // Check for keyboard input and handle it.
        CheckKeyboardInput();

        // Check for gamepad input and handle it.
        CheckGamePadInput();

        // Creating a bounding circle for the slime.
        Circle slimeBounds = new Circle(
            (int)(_slimePosition.X + (_slime.Width * 0.5f)),
            (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
            (int)(_slime.Width * 0.5f)
        );

        // Use distance based checks to determine if the slime is within the
        // bounds of the game screen, and if it is outside that screen edge,
        // move it back inside.
        if (slimeBounds.Left < _roomBounds.Left)
        {
            _slimePosition.X = _roomBounds.Left;
        }
        else if (slimeBounds.Right > _roomBounds.Right)
```

```
{
    _slimePosition.X = _roomBounds.Right - _slime.Width;
}

if (slimeBounds.Top < _roomBounds.Top)
{
    _slimePosition.Y = _roomBounds.Top;
}
else if (slimeBounds.Bottom > _roomBounds.Bottom)
{
    _slimePosition.Y = _roomBounds.Bottom - _slime.Height;
}

// Calculate the new position of the bat based on the velocity.
Vector2 newBatPosition = _batPosition + _batVelocity;

// Create a bounding circle for the bat.
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
);

Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal
if (batBounds.Left < _roomBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Left;
}
else if (batBounds.Right > _roomBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Right - _bat.Width;
}

if (batBounds.Top < _roomBounds.Top)
{
    normal.Y = Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Top;
}
else if (batBounds.Bottom > _roomBounds.Bottom)
{
```

```
            normal.Y = -Vector2.UnitY.Y;
            newBatPosition.Y = _roomBounds.Bottom - _bat.Height;
        }

        // If the normal is anything but Vector2.Zero, this means the bat had
        // moved outside the screen edge so we should reflect it about the
        // normal.
        if (normal != Vector2.Zero)
        {
            normal.Normalize();
            _batVelocity = Vector2.Reflect(_batVelocity, normal);

            // Play the bounce sound effect.
            Audio.PlaySoundEffect(_bounceSoundEffect);
        }

        _batPosition = newBatPosition;

        if (slimeBounds.Intersects(batBounds))
        {
            // Choose a random row and column based on the total number of each
            int column = Random.Shared.Next(1, _tilemap.Columns - 1);
            int row = Random.Shared.Next(1, _tilemap.Rows - 1);

            // Change the bat position by setting the x and y values equal to
            // the column and row multiplied by the width and height.
            _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

            // Assign a new random velocity to the bat.
            AssignRandomBatVelocity();

            // Play the collect sound effect.
            Audio.PlaySoundEffect(_collectSoundEffect);
        }

        base.Update(gameTime);
    }

    private void AssignRandomBatVelocity()
    {
        // Generate a random angle.
        float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

        // Convert angle to a direction vector.
        float x = (float)Math.Cos(angle);
        float y = (float)Math.Sin(angle);
```

```
        Vector2 direction = new Vector2(x, y);

        // Multiply the direction vector by the movement speed.
        _batVelocity = direction * MOVEMENT_SPEED;
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }

        // If the M key is pressed, toggle mute state for audio.
        if (Input.Keyboard.WasKeyJustPressed(Keys.M))
        {
            Audio.ToggleMute();
        }
```

```
        // If the + button is pressed, increase the volume.
        if (Input.Keyboard.WasKeyJustPressed(Keys.OemPlus))
        {
            Audio.SongVolume += 0.1f;
            Audio.SoundEffectVolume += 0.1f;
        }

        // If the - button was pressed, decrease the volume.
        if (Input.Keyboard.WasKeyJustPressed(Keys.OemMinus))
        {
            Audio.SongVolume -= 0.1f;
            Audio.SoundEffectVolume -= 0.1f;
        }
    }

    private void CheckGamePadInput()
    {
        GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];

        // If the A button is held down, the movement speed increases by 1.5
        // and the gamepad vibrates as feedback to the player.
        float speed = MOVEMENT_SPEED;
        if (gamePadOne.IsButtonDown(Buttons.A))
        {
            speed *= 1.5f;
            GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
        }
        else
        {
            GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
        }

        // Check thumbstick first since it has priority over which gamepad input
        // is movement.  It has priority since the thumbstick values provide a
        // more granular analog value that can be used for movement.
        if (gamePadOne.LeftThumbStick != Vector2.Zero)
        {
            _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
            _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
        }
        else
        {
            // If DPadUp is down, move the slime up on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadUp))
            {
                _slimePosition.Y -= speed;
```

```
            }

            // If DPadDown is down, move the slime down on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadDown))
            {
                _slimePosition.Y += speed;
            }

            // If DPapLeft is down, move the slime left on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
            {
                _slimePosition.X -= speed;
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the tilemap
        _tilemap.Draw(SpriteBatch);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite.
        _bat.Draw(SpriteBatch, _batPosition);

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

> **NOTE**
>
> Note there were a lot of replacements in the `LoadContent` method, switching from loading and initializing the background Song and replacing it with a call to the new `AudioController` to do all the work managing the Song reference. Much cleaner.

The key changes made here are:

1. The `_themeSong` field is added to store a reference to the background song to play.
2. In **LoadContent**, the background theme song is loaded using the content manager.
3. In **Initialize**, the audio manager is used to play the background theme song.
4. In **Update** the audio manager is used to play the bounce and collect sound effects.
5. In `CheckKeyboardInput` the following checks were added
    1. If the M key on the keyboard is pressed, it will toggle mute for all audio.
    2. If the + key is pressed, the song and sound effect volumes are increased by `0.1f`.
    3. If the - key is pressed, the song and sound effect volumes are decreased by `0.1f`.

Running the game now will produce the same result as the previous chapter, only now the lifetime of sound effects and the state management of audio is done through the new audio controller. You can also mute and unumte the audio with the M key and increase and decrease the volume using the + and - keys.

Figure 15-1: Gameplay with audio.

> **NOTE**
>
> You may note that while we added keybindings to change the audio settings, we did not add any bindings for the GamePad. This is simply because this is not normally how you would adjust these values on a console, on consoles you would have a settings/options screen to update them.
>
> Later in Chapter 20: Implementing UI with GUM we will add an Options screen to adjust all the audio values for the game.

Conclusion

In this chapter, you accomplished the following:

- Created a reusable `AudioController` class to centralize audio management.
- Learned about proper resource management for audio using the `IDisposable` pattern.
- Implemented tracking and cleanup of sound effect instances.
- Added global volume control for both sound effects and music.
- Created methods to toggle audio states (play/pause, mute/unmute).
- Updated the `Core` class to provide global access to the audio controller.
- Added keyboard controls to adjust volume and toggle mute state.

The `AudioController` class you created is a significant improvement over directly using MonoGame's audio APIs. It handles common audio management tasks that would otherwise need to be implemented repeatedly in different parts of your game. By centralizing these functions, you make your code more maintainable and provide a consistent audio experience across your game.

In the next chapter, we will start exploring fonts and adding text to the game.

Chapter 16: Working with SpriteFonts

Learn how to create and use SpriteFonts to render text in your MonoGame project, including loading custom fonts and controlling text appearance.

In Chapter 06, you learned how to load and render textures to display sprites in your game. While images are essential for visual elements, most games also need text for things like scores, player instructions, dialogue, and UI elements. MonoGame provides the **SpriteFont** class to handle text rendering, which works together with the familiar **SpriteBatch** we have been using for drawing textures.

In this chapter, you will:

- Learn how MonoGame handles text rendering with SpriteFonts.
- Create `SpriteFont` description using the MGCB Editor.
- Load custom fonts for use in your game.
- Render text using various parameters to control appearance.
- Implement text rendering in our game.

We will first start by understanding how text rendering works in MonoGame.

Understanding SpriteFonts

MonoGame processes fonts through the content pipeline to create a texture atlas of font characters. MonoGame uses the texture atlas approach rather than directly using system fonts for several important reasons:

- **Cross-platform Compatibility**: System fonts cannot be guaranteed to exist on all platforms.
- **Consistency**: Ensures that the text appears the same across all platforms.
- **GPU Rendering**: Graphics cards do not understand font formats directly; they can only render textures.
- **Performance**: Pre-rendering the glyphs to a texture atlas allow for faster rendering at runtime with no texture swapping.

A **SpriteFont** in MonoGame consists of:

1. A texture atlas containing pre-rendered glyphs (characters).
2. Data that tracks the position, size, and spacing of each character.
3. Kerning information for adjusting spacing between specific character pairs.

The texture atlas approach means fonts are rendered as sprites, using the same **SpriteBatch** system you learned about for drawing textures (Chapter 7: Optimized Texture Rendering). When you draw text, MonoGame is actually drawing small portions of the texture atlas for each character assembled together to form complete words and sentences.

Creating a SpriteFont Description

To use text in your game, you first need to create a SpriteFont Description file and process it through the Content Pipeline, thankfully the MGCB Editor makes this process straightforward.

For example, in the MGCB Editor:

1. Right-click the content project node where the SpriteFont Description will be created and choose *Add > New Item...*.
2. Select `SpriteFont Description (.spritefont)` from the options.
3. Specify a name for the SpriteFont Description file and click `Create`.

This will create a default SpriteFont Description file that look something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file contains an xml description of a font, and will be read by the XNA
Framework Content Pipeline. Follow the comments to customize the appearance
of the font in your game, and to change the characters which are available to draw
with.
-->
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">

    <!--
    Modify this string to change the font that will be imported.
    -->
    <FontName>Arial</FontName>

    <!--
    Size is a float value, measured in points. Modify this value to change
    the size of the font.
    -->
    <Size>12</Size>

    <!--
    Spacing is a float value, measured in pixels. Modify this value to change
    the amount of spacing in between characters.
    -->
    <Spacing>0</Spacing>

    <!--
    UseKerning controls the layout of the font. If this value is true,
kerning information
    will be used when placing characters.
    -->
    <UseKerning>true</UseKerning>

    <!--
    Style controls the style of the font. Valid entries are "Regular",
"Bold", "Italic",
    and "Bold, Italic", and are case sensitive.
    -->
    <Style>Regular</Style>

    <!--
    If you uncomment this line, the default character will be substituted if
you draw
    or measure text that contains characters which were not included in the font.
    -->
```

```
    <!-- <DefaultCharacter>*</DefaultCharacter> -->

    <!--
    CharacterRegions control what letters are available in the font. Every
    character from Start to End will be built and made available for drawing. The
    default range is from 32, (ASCII space), to 126, ('~'), covering the basic Latin
    character set. The characters are ordered according to the Unicode standard.
    See the documentation for more information.
    -->
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

When creating a SpriteFont Description for your game, you will need to make several important decisions about font selection, size, formatting, and licensing. The following sections will guide you through customizing the SpriteFont Description using these considerations.

Customizing the SpriteFont

The SpriteFont Description file allows you to customize various aspects of how the font will be processed and appear in your game. Here are the key elements you can modify:

FontName

The `<FontName>` element specifies which font to use. By default, it references "Arial". When a font name is specified just by name like this, it is required that the font be installed on the system where the content is built.

> **IMPORTANT**
>
> MonoGame recommends changing the default Arial font if you are targeting any platforms other than Windows. Arial is a legacy from XNA and is only guaranteed to be available in Windows builds. As an alternative, MonoGame currently recommends using Roboto.

Alternatively, for better portability across development environments, it is recommended instead to directly reference a TrueType (.ttf) or OpenType (.otf) font file. To do this

1. Download or locate a TTF or OTF font file.
2. Place it in the **same folder** as the `.spritefont` file.

> **IMPORTANT**
>
> You place the font file in the **same folder** as the `.spritefont` file directly, not through the MGCB Editor.

3. Update the `<FontName>` element to include the exact filename with extension.

> **TIP**
>
> Use fonts with permissive licenses (like SIL Open Font License) to ensure you can **legally** use them in your game.
>
> **Always check the license of any font you use!**

Size

The `<Size>` element controls the font size in points. While it might seem straightforward, font sizing requires consideration and can be dependent on several factors. When choosing a font size, consider:

- **Resolution impact**: Fonts that look good at 1080p may appear too small at 4K or too large at 720p.
- **Font style**: Pixel fonts look best with small sizes to preserve crispness.
- **Use case**: Different UI elements may require different sizes for proper hierarchy.

You may want to create multiple SpriteFont Description files for different use cases in your game such as:

- A larger font for headings and titles.
- A medium-sized font for standard UI elements.
- A smaller font for detailed information.

> **TIP**
>
> Creating multiple SpriteFont Description files, however, can remove some of the benefits of fonts being a texture atlas since you will now have multiple atlases for each size. You will also now have multiple assets to manage both as asset files and references in code.
>
> An alternative approach is to create a single SpriteFont Description with a larger than needed size font, then scale it down during runtime in the game. This approach allows you to maintain the single SpriteFont Description file and single texture atlas, however, the size of the texture atlas will now be larger.
>
> There are tradeoffs to each approach and you should choose the one that works best for your game.

Spacing

The `<Spacing>` element adjusts the space between characters. The default value of 0 uses the font's built-in spacing. Positive values increase spacing, while negative values (though rarely used) can decrease it.

UseKerning

The `<UseKerning>` element determines whether to use kerning information from the font. Kerning adjusts the spacing between specific pairs of characters for more visually pleasing results. For most fonts, you will want to leave this as `true`.

> **NOTE**
>
> While kerning typically improves text appearance, some fonts (including Arial) may not respond optimally to kerning adjustments. If you notice unusual character spacing with a particular font, try setting this value to `false`.

Style

The `<Style>` element sets the font style. Valid options are "Regular", "Bold", "Italic", or "Bold, Italic". Note that not all fonts have all styles available, and using a style that does not exist will fall back to Regular.

DefaultCharacter

The `<DefaultCharacter>` element (commented out by default) specifies what character to use as a fallback when trying to render a character that is not included in the font. This is useful for handling special characters or international text.

CharacterRegions

The `<CharacterRegions>` element defines which Unicode character ranges to include in the font. The default range (32-126) covers basic Latin characters, which is sufficient for English text. Including more characters increases the font texture size but allows support for more languages or special symbols.

For most games, the default range is sufficient.

> **NOTE**
>
> Although for fun, TRY using the Wingdings font :D

Loading a SpriteFont Description

To load a SpriteFont Description, we use the **ContentManager.Load** method with the **SpriteFont** type:

```
// Loading a SpriteFont Description using the content pipeline
SpriteFont font = Content.Load<SpriteFont>("font");
```

Drawing Text with SpriteBatch

MonoGame's **SpriteBatch** class provides several overloads of the **DrawString** method to render text. The basic approach is similar to drawing textures, with a few unique parameters specific to text rendering.

The most basic DrawString overload looks like this:

```
_spriteBatch.DrawString(
    font,                   // font
    "Hello, MonoGame!",     // text
    Vector2.Zero,           // position
    Color.White             // color
);
```

This overload contains the following parameters:

1. **font**: The **SpriteFont** to use for rendering.
2. **text**: The text to display (as a string or `StringBuilder`).
3. **position**: A **Vector2** defining where to draw the text.
4. **color**: The **Color** to tint the text.

Just like with texture rendering, there are more advanced overloads that give you additional control:

```
_spriteBatch.DrawString(
    font,                   // font
    "Hello, MonoGame!",     // text
    Vector2.Zero,           // position
    Color.White,            // color
    0.0f,                   // rotation
    Vector2.Zero,           // origin
    Vector2.One,            // scale
    SpriteEffects.None,     // effects
    0.0f                    // layerDepth
);
```

> **NOTE**
>
> Many of these parameters (`rotation`, `origin`, `scale`, `effects`, and `layerDepth`) work exactly the same way as they do for texture rendering, as explained in Chapter 06. If you need a refresher on how these parameters affect rendering, refer back to that chapter.

Calculating Text Dimensions

One common task when working with text is determining how much space it will occupy on screen. This is important for:

- Centering text
- Creating UI layouts
- Checking if text fits within a designated area
- Implementing text wrapping

MonoGame's **SpriteFont** class provides the **MeasureString** method that returns the dimensions of rendered text:

```
// The text to measure.
string message = "Hello, MonoGame!";

// Measure the size of the message to get the text dimensions.
Vector2 textSize = font.MeasureString(message);
```

The returned **Vector2** contains:

- `X`: The width of the text in pixels
- `Y`: The height of the text in pixels

Below is an example of centering text on the screen by drawing the text at the center of the screen and using **MeasureString** to calculate the center origin or the text

```
// The text to draw.
string message = "Hello, MonoGame!";

// Measure the size of the message to get the text dimensions.
Vector2 textSize = font.MeasureString(message);

// Set the origin to the center of the text dimensions.
Vector2 origin = textSize * 0.5f;

// Position will be the center of the screen.
Vector2 position = new Vector2(
    GraphicsDevice.PresentationParameters.BackBufferWidth,
    GraphicsDevice.PresentationParameters.BackBufferHeight
) * 0.5f;

// Draw centered text
_spriteBatch.DrawString(
    font,                   // font
    message,                // text
    position,               // position
    Color.White,            // color
    0.0f,                   // rotation
    origin,                 // origin
    1.0f,                   // scale
    SpriteEffects.None,     // effects
    0.0f                    // layerDepth
);
```

Implementing Text in Our Game

To explore implementing text, we will add text to our game to display information to the player. We will add a score counter that increases when the slime eats the bat.

Adding the SpriteFont Description

First, we will need to create a SpriteFont Definition. Open the *Content.mgcb* content project file in the MGCB Editor and perform the following:

1. Create a new folder called `fonts` (right-click *Content* > *Add* > *New Folder*).
2. Right-click the new `fonts` folder and choose `Add > New Item...`.
3. Select `SpriteFont Description (.spritefont)` from the options.
4. Name the file `04B_30` and click `Create`.

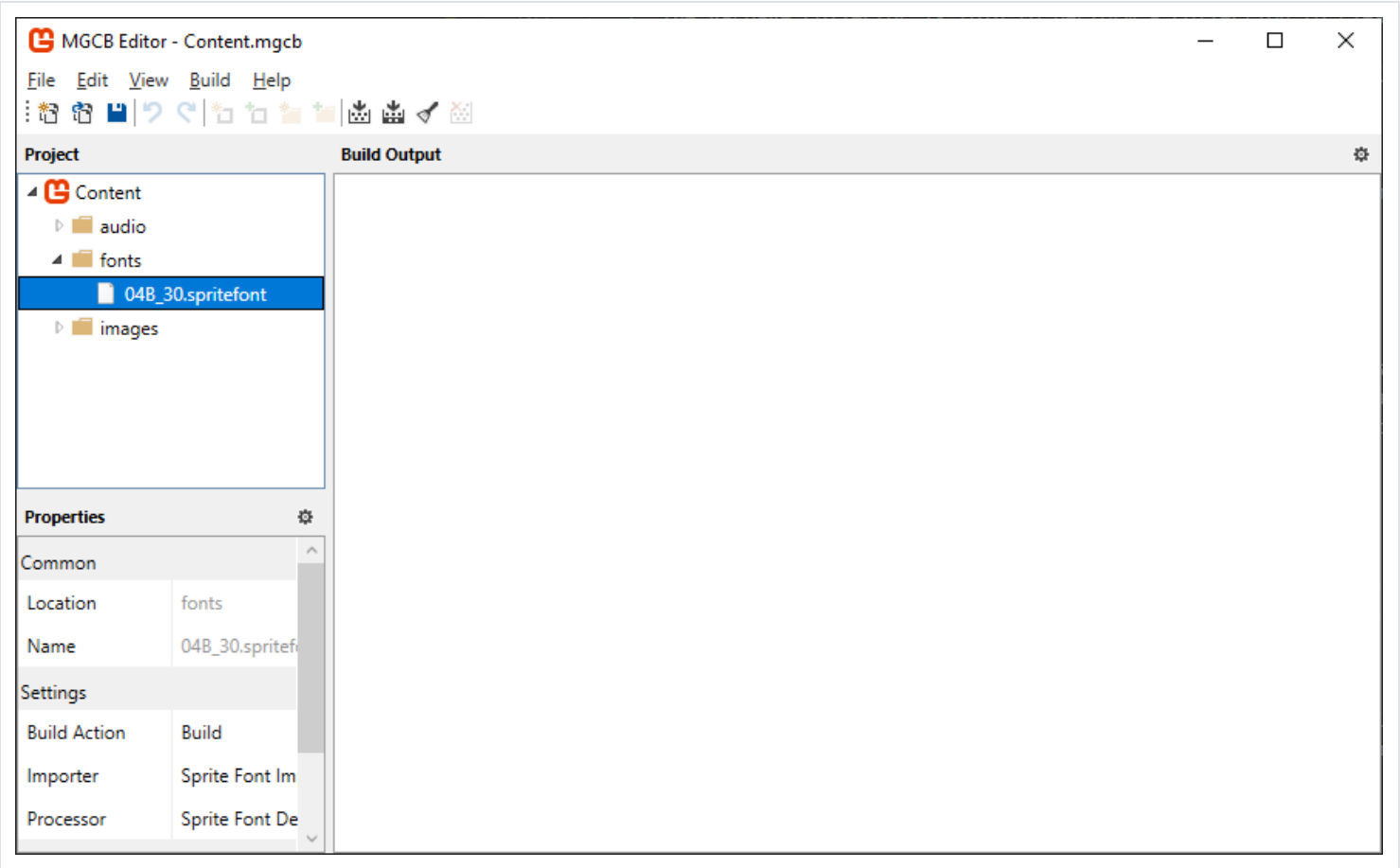


Figure 16-1: The 04B_30.spritefont file created in the MGCB Editor

> **NOTE**
>
> We have named the SpriteFont Description file with the same name as the font we will be using. This makes it easier to remember when updating the code later.

Download the Font File

Next, right-click the following TTF font and choose "Save Link as...". Save the file in the same folder where you created the *04B_30.spriteFont* file using the MGCB Editor. This ensures the font is correctly linked and available for your SpriteFont configuration.

- 04B_30.ttf

Update the SpriteFont Description

Next, open the *04B_30.spritefont* file in your code editor and make the following changes:

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">
    <FontName>04B_30.ttf</FontName>
    <Size>17.5</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

The key changes here are:

1. The `<FontName>` element was updated to `04B_30.ttf`, the exact filename with extension of the TTF font we just downloaded.
2. The `<Size>` element was updated to be `17.5`.

Updating the Game

Finally, open the `Game1.cs` file and make the following changes:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;

namespace DungeonSlime;

public class Game1 : Core
{
    // Defines the slime animated sprite.
    private AnimatedSprite _slime;

    // Defines the bat animated sprite.
    private AnimatedSprite _bat;

    // Tracks the position of the slime.
    private Vector2 _slimePosition;

    // Speed multiplier when moving.
    private const float MOVEMENT_SPEED = 5.0f;

    // Tracks the position of the bat.
    private Vector2 _batPosition;

    // Tracks the velocity of the bat.
    private Vector2 _batVelocity;

    // Defines the tilemap to draw.
    private Tilemap _tilemap;

    // Defines the bounds of the room that the slime and bat are contained within.
    private Rectangle _roomBounds;

    // The sound effect to play when the bat bounces off the edge of the screen.
    private SoundEffect _bounceSoundEffect;

    // The sound effect to play when the slime eats a bat.
    private SoundEffect _collectSoundEffect;

    // The background theme song.
    private Song _themeSong;
```

```
    // The SpriteFont Description used to draw text.
    private SpriteFont _font;

    // Tracks the players score.
    private int _score;

    // Defines the position to draw the score text at.
    private Vector2 _scoreTextPosition;

    // Defines the origin used when drawing the score text.
    private Vector2 _scoreTextOrigin;


    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        base.Initialize();

        Rectangle screenBounds = GraphicsDevice.PresentationParameters.Bounds;

        _roomBounds = new Rectangle(
            (int)_tilemap.TileWidth,
            (int)_tilemap.TileHeight,
            screenBounds.Width - (int)_tilemap.TileWidth * 2,
            screenBounds.Height - (int)_tilemap.TileHeight * 2
        );

        // Initial slime position will be the center tile of the tile map.
        int centerRow = _tilemap.Rows / 2;
        int centerColumn = _tilemap.Columns / 2;
        _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

        // Initial bat position will the in the top left corner of the room.
        _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

        // Assign the initial random velocity to the bat.
        AssignRandomBatVelocity();

        // Start playing the background music.
        Audio.PlaySong(_themeSong);
```

```
        // Set the position of the score text to align to the left edge of the
        // room bounds, and to vertically be at the center of the first tile.
        _scoreTextPosition = new Vector2(_roomBounds.Left, _tilemap.TileHeight
* 0.5f);

        // Set the origin of the text so it is left-centered.
        float scoreTextYOrigin = _font.MeasureString("Score").Y * 0.5f;
        _scoreTextOrigin = new Vector2(0, scoreTextYOrigin);
    }

    protected override void LoadContent()
    {
        // Create the texture atlas from the XML configuration file.
        TextureAtlas atlas = TextureAtlas.FromFile(Content,
"images/atlas-definition.xml");

        // Create the slime animated sprite from the atlas.
        _slime = atlas.CreateAnimatedSprite("slime-animation");
        _slime.Scale = new Vector2(4.0f, 4.0f);

        // Create the bat animated sprite from the atlas.
        _bat = atlas.CreateAnimatedSprite("bat-animation");
        _bat.Scale = new Vector2(4.0f, 4.0f);

        // Create the tilemap from the XML configuration file.
        _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
        _tilemap.Scale = new Vector2(4.0f, 4.0f);

        // Load the bounce sound effect.
        _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

        // Load the collect sound effect.
        _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

        // Load the background theme music.
        _themeSong = Content.Load<Song>("audio/theme");

        // Load the font
        _font = Content.Load<SpriteFont>("fonts/04B_30");
    }

    protected override void Update(GameTime gameTime)
    {
        // Update the slime animated sprite.
        _slime.Update(gameTime);
```

```
// Update the bat animated sprite.
_bat.Update(gameTime);

// Check for keyboard input and handle it.
CheckKeyboardInput();

// Check for gamepad input and handle it.
CheckGamePadInput();

// Creating a bounding circle for the slime.
Circle slimeBounds = new Circle(
    (int)(_slimePosition.X + (_slime.Width * 0.5f)),
    (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
    (int)(_slime.Width * 0.5f)
);

// Use distance based checks to determine if the slime is within the
// bounds of the game screen, and if it is outside that screen edge,
// move it back inside.
if (slimeBounds.Left < _roomBounds.Left)
{
    _slimePosition.X = _roomBounds.Left;
}
else if (slimeBounds.Right > _roomBounds.Right)
{
    _slimePosition.X = _roomBounds.Right - _slime.Width;
}

if (slimeBounds.Top < _roomBounds.Top)
{
    _slimePosition.Y = _roomBounds.Top;
}
else if (slimeBounds.Bottom > _roomBounds.Bottom)
{
    _slimePosition.Y = _roomBounds.Bottom - _slime.Height;
}

// Calculate the new position of the bat based on the velocity
Vector2 newBatPosition = _batPosition + _batVelocity;

// Create a bounding circle for the bat
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
```

```
);

Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal
if (batBounds.Left < _roomBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Left;
}
else if (batBounds.Right > _roomBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Right - _bat.Width;
}

if (batBounds.Top < _roomBounds.Top)
{
    normal.Y = Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Top;
}
else if (batBounds.Bottom > _roomBounds.Bottom)
{
    normal.Y = -Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Bottom - _bat.Height;
}

// If the normal is anything but Vector2.Zero, this means the bat had
// moved outside the screen edge so we should reflect it about the
// normal.
if (normal != Vector2.Zero)
{
    normal.Normalize();
    _batVelocity = Vector2.Reflect(_batVelocity, normal);

    // Play the bounce sound effect.
    Audio.PlaySoundEffect(_bounceSoundEffect);
}

_batPosition = newBatPosition;

if (slimeBounds.Intersects(batBounds))
{
    // Choose a random row and column based on the total number of each
```

```
            int column = Random.Shared.Next(1, _tilemap.Columns - 1);
            int row = Random.Shared.Next(1, _tilemap.Rows - 1);

            // Change the bat position by setting the x and y values equal to
            // the column and row multiplied by the width and height.
            _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

            // Assign a new random velocity to the bat.
            AssignRandomBatVelocity();

            // Play the collect sound effect.
            Audio.PlaySoundEffect(_collectSoundEffect);

            // Increase the player's score.
            _score += 100;
        }

        base.Update(gameTime);
    }

    private void AssignRandomBatVelocity()
    {
        // Generate a random angle.
        float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

        // Convert angle to a direction vector
        float x = (float)Math.Cos(angle);
        float y = (float)Math.Sin(angle);
        Vector2 direction = new Vector2(x, y);

        // Multiply the direction vector by the movement speed
        _batVelocity = direction * MOVEMENT_SPEED;
    }

    private void CheckKeyboardInput()
    {
        // If the space key is held down, the movement speed increases by 1.5
        float speed = MOVEMENT_SPEED;
        if (Input.Keyboard.IsKeyDown(Keys.Space))
        {
            speed *= 1.5f;
        }

        // If the W or Up keys are down, move the slime up on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.W) || Input.Keyboard.IsKeyDown(Keys.Up))
        {
```

```
            _slimePosition.Y -= speed;
        }

        // if the S or Down keys are down, move the slime down on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.S) || Input.Keyboard.IsKeyDown(Keys.Down))
        {
            _slimePosition.Y += speed;
        }

        // If the A or Left keys are down, move the slime left on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.A) || Input.Keyboard.IsKeyDown(Keys.Left))
        {
            _slimePosition.X -= speed;
        }

        // If the D or Right keys are down, move the slime right on the screen.
        if (Input.Keyboard.IsKeyDown(Keys.D) ||
Input.Keyboard.IsKeyDown(Keys.Right))
        {
            _slimePosition.X += speed;
        }

        // If the M key is pressed, toggle mute state for audio.
        if (Input.Keyboard.WasKeyJustPressed(Keys.M))
        {
            Audio.ToggleMute();
        }

        // If the + button is pressed, increase the volume.
        if (Input.Keyboard.WasKeyJustPressed(Keys.OemPlus))
        {
            Audio.SongVolume += 0.1f;
            Audio.SoundEffectVolume += 0.1f;
        }

        // If the - button was pressed, decrease the volume.
        if (Input.Keyboard.WasKeyJustPressed(Keys.OemMinus))
        {
            Audio.SongVolume -= 0.1f;
            Audio.SoundEffectVolume -= 0.1f;
        }
    }

    private void CheckGamePadInput()
    {
        GamePadInfo gamePadOne = Input.GamePads[(int)PlayerIndex.One];
```

```
// If the A button is held down, the movement speed increases by 1.5
// and the gamepad vibrates as feedback to the player.
float speed = MOVEMENT_SPEED;
if (gamePadOne.IsButtonDown(Buttons.A))
{
    speed *= 1.5f;
    GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
}
else
{
    GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
}

// Check thumbstick first since it has priority over which gamepad input
// is movement.  It has priority since the thumbstick values provide a
// more granular analog value that can be used for movement.
if (gamePadOne.LeftThumbStick != Vector2.Zero)
{
    _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
    _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
}
else
{
    // If DPadUp is down, move the slime up on the screen.
    if (gamePadOne.IsButtonDown(Buttons.DPadUp))
    {
        _slimePosition.Y -= speed;
    }

    // If DPadDown is down, move the slime down on the screen.
    if (gamePadOne.IsButtonDown(Buttons.DPadDown))
    {
        _slimePosition.Y += speed;
    }

    // If DPapLeft is down, move the slime left on the screen.
    if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
    {
        _slimePosition.X -= speed;
    }

    // If DPadRight is down, move the slime right on the screen.
    if (gamePadOne.IsButtonDown(Buttons.DPadRight))
    {
        _slimePosition.X += speed;
```

```
            }
        }
    }

    protected override void Draw(GameTime gameTime)
    {
        // Clear the back buffer.
        GraphicsDevice.Clear(Color.CornflowerBlue);

        // Begin the sprite batch to prepare for rendering.
        SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // Draw the tilemap
        _tilemap.Draw(SpriteBatch);

        // Draw the slime sprite.
        _slime.Draw(SpriteBatch, _slimePosition);

        // Draw the bat sprite.
        _bat.Draw(SpriteBatch, _batPosition);

        // Draw the score
        SpriteBatch.DrawString(
            _font,              // spriteFont
            $"Score: {_score}", // text
            _scoreTextPosition, // position
            Color.White,        // color
            0.0f,               // rotation
            _scoreTextOrigin,   // origin
            1.0f,               // scale
            SpriteEffects.None, // effects
            0.0f                // layerDepth
        );

        // Always end the sprite batch when finished.
        SpriteBatch.End();

        base.Draw(gameTime);
    }
}
```

The key changes made are:

1. The `_font` field was added to store the SpriteFont Description when loaded.
2. The `_score` field was added to track the player's score.
3. The `_scoreTextPosition` field was added to store the precalculated position to draw the score text at.
4. The `_scoreTextOrigin` field was added to store the origin to use when drawing the score text.
5. In **Initialize**
    1. The position to draw the score text is precalculated to align with the left side of the room bounds horizontally and to vertically be at the center of the first tile at the top.
    2. The origin for the score text is precalculated by measuring the string `"Score"` and multiplying the `Y` component to get the vertical center of the text, then the origin is set so that it is at the left-center.
6. In **LoadContent**, the font is loaded using the content manager.
7. In **Update**, the player's score is increased by `100` each time the slime eats the bat.
8. In **Draw**, the score text is drawn at the precalculated position using the precalculated origin so that it aligns in the center of the tile on the blue strip using the sprite batch.

Figure 16-2: The game with score displayed in the top-left corner

Conclusion

In this chapter, you accomplished the following:

- Learned how to create SpriteFont definitions with the MGCB Editor.
- Learned how to reference font files for better cross-platform compatibility.
- Learned how to load SpriteFonts through the content pipeline.
- Learned how to draw text with various parameters to control appearance.
- Learned how to measure text dimensions.
- Implemented a score display in our game.

In the next chapter we will discuss MonoGame's service container and how we can use it to start breaking our monolithic game file into modules for better maintainability.

Test Your Knowledge

1. What are the key components of a SpriteFont in MonoGame?

   A SpriteFont in MonoGame consists of:

   1. A texture atlas containing pre-rendered glyphs (characters)
   2. Data that tracks the position, size, and spacing of each character
   3. Kerning information for adjusting spacing between specific character pairs

2. Why is it recommended to include the font file in your content project rather than referencing system fonts?

   Including the font file (TTF/OTF) directly in your content project and referencing it with the file extension ensures portability across different development environments. This approach does not depend on fonts being installed on the system where the content is built.

3. What method would you use to determine how much space a text string will occupy when rendered, and what does it return?

   The **SpriteFont.MeasureString** method is used to determine text dimensions. It returns a **Vector2** where the X component represents the width and the Y component represents the height of the rendered text in pixels.

Chapter 17: Scene Management

Learn how to implement scene management to handle different game screens like menus, gameplay, and transitions between scenes.

In game development, a scene (sometimes called a screen or state) represents a distinct section of the game. Each scene typically has its own update and draw logic, as well as its own set of game objects. Common examples of scenes include title screens, menus, gameplay screens, game over screens, and more. Scenes help organize the game's code by separating different parts of the game into self-contained modules. This makes the code more manageable as the game grows in complexity and offers several advantages:

1. **Improved organization**: Each scene contains only the code and assets relevant to that part of the game.
2. **Memory management**: Load assets only when needed and unload them when leaving a scene.
3. **Simplified state handling**: Each scene maintains its own state without affecting others.
4. **Code reusability**: Create reusable scene templates for common game screens.

Our game logic is currently contained within the single `Game1` class. Adding more screens to it would make the code harder to manage, so instead we need to start thinking about breaking it down into scenes.

In this chapter, you will:

- Learn the concept of scene management and its benefits
- Create a base Scene class with a consistent lifecycle
- Implement scene transitions using a manager
- Create a title scene and gameplay scene for our game
- Refactor our existing game to use the scene system

We will begin by first defining the lifecycle of a scene that will be followed.

Scene Lifecycle

In Chapter 03, you learned the basic lifecycle of the `Game` class. To be consistent, we can borrow from this lifecycle and adapt it for our scenes. The order of operations for this lifecycle will be:

1. A scene is created and set as the active scene.
2. The first screen is made active and is initialized and content loaded.
3. The active scene is updated and drawn each cycle.
4. When transitioning to a new scene, or when the scene ends:
    1. The current scene is unloaded and disposed of.
    2. The new scene is initialized and content loaded.
    3. The new scene becomes the active scene and the cycle begins again until the game is told to exit.

The Scene Base Class

The base `Scene` class is an abstract class for scenes that provides common functionality for all scenes. In our actual game, we will create concrete implementations of this, like a title scene.

To get started, in the *MonoGameLibrary* project:

1. Create a new folder named `Scenes`.

2. Add a new class file named `Scene.cs` to the `Scenes` folder you just created.

3. Add the following code as the initial structure for the class:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;

namespace MonoGameLibrary.Scenes;

public abstract class Scene : IDisposable
{

}
```

> **NOTE**
>
> Just like with the `AudioController` in Chapter 15, each `Scene` implements the `IDisposable` interface. This provides a standardized in method to release the resources held by a scene when it is no longer needed.

Scene Properties

Add the following properties to the `Scene` class:

```
/// <summary>
/// Gets the ContentManager used for loading scene-specific assets.
/// </summary>
/// <remarks>
/// Assets loaded through this ContentManager will be automatically unloaded when
this scene ends.
/// </remarks>
protected ContentManager Content { get; }

/// <summary>
/// Gets a value that indicates if the scene has been disposed of.
/// </summary>
public bool IsDisposed { get; private set; }
```

- The `Content` property is the scene's personal **ContentManager** that can be used to load scene specific content that will be unloaded when the scene ends. This helps manage memory usage by only loading what is needed for a specific scene.
- The `IsDisposed` property is used to track if the scene has been disposed of since it implements the `IDisposable` interface.

Scene Constructor

Add the following constructor and finalizer to the `Scene` class:

```
/// <summary>
/// Creates a new scene instance.
/// </summary>
public Scene()
{
    // Create a content manager for the scene
    Content = new ContentManager(Core.Content.ServiceProvider);

    // Set the root directory for content to the same as the root directory
    // for the game's content.
    Content.RootDirectory = Core.Content.RootDirectory;
}

// Finalizer, called when object is cleaned up by garbage collector.
~Scene() => Dispose(false);
```

- The constructor initializes the scene's content manager and sets the root folder to match that of the base game's content manager.
- The finalizer is called by the garbage collector automatically when a scene object is collected which just calls the `Dispose` method to ensure resources are disposed of properly.

Scene Methods

Add the following methods to the `Scene` class:

```
/// <summary>
/// Initializes the scene.
/// </summary>
/// <remarks>
/// When overriding this in a derived class, ensure that base.Initialize()
/// still called as this is when LoadContent is called.
/// </remarks>
public virtual void Initialize()
{
    LoadContent();
}

/// <summary>
/// Override to provide logic to load content for the scene.
/// </summary>
public virtual void LoadContent() { }

/// <summary>
/// Unloads scene-specific content.
/// </summary>
public virtual void UnloadContent()
{
    Content.Unload();
}

/// <summary>
/// Updates this scene.
/// </summary>
/// <param name="gameTime">A snapshot of the timing values for the current
frame.</param>
public virtual void Update(GameTime gameTime) { }

/// <summary>
/// Draws this scene.
/// </summary>
/// <param name="gameTime">A snapshot of the timing values for the current
frame.</param>
public virtual void Draw(GameTime gameTime) { }
```

These methods are setup similar to how the `Game` class works to keep the workflow consistent:

- `Initialize` is called only once when the scene becomes the active scene. It can be overridden by the derived class to provide scene specific initialization logic. It also calls the `LoadContent` method the same way the `Game` class is done for consistency.
- `LoadContent` is called only once, at the end of the `Initialize` method. It can be overridden by the derived class to load scene specific content.
- `UnloadContent` is called only once when a scene is ending due to a transition to a new scene. It can be overridden by the derived class to perform unloading of any scene specific content.
- `Update` is called once at the start of every game cycle. It can be overridden to provide the update logic for the scene.
- `Draw` is called once every game cycle, directly after `Update`. It can be overridden to provide the draw logic for the scene.

IDisposable Implementation

Add the following methods to the `Scene` class to complete the implementation of the `IDisposable` interface:

```
/// <summary>
/// Disposes of this scene.
/// </summary>
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

/// <summary>
/// Disposes of this scene.
/// </summary>
/// <param name="disposing">'
/// Indicates whether managed resources should be disposed.  This value is only true
when called from the main
/// Dispose method.  When called from the finalizer, this will be false.
/// </param>
protected virtual void Dispose(bool disposing)
{
    if (IsDisposed)
    {
        return;
    }

    if (disposing)
    {
        UnloadContent();
        Content.Dispose();
    }
    IsDisposed = true;
}
```

With the Base scene implementation complete, we can now use it to create actual scenes for our project. The next step is to implement a manager that organizes and coordinates these scenes within the game.

Scene Management

With the base `Scene` class defined, the `Core` class needs to be updated to handle management of the scenes, including update, drawing, and changing scenes. Open the `Core.cs` file in the *MonoGameLibrary* project and make the following changes:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary.Audio;
using MonoGameLibrary.Input;
using MonoGameLibrary.Scenes;

namespace MonoGameLibrary;

public class Core : Game
{
    internal static Core s_instance;

    /// <summary>
    /// Gets a reference to the Core instance.
    /// </summary>
    public static Core Instance => s_instance;

    // The scene that is currently active.
    private static Scene s_activeScene;

    // The next scene to switch to, if there is one.
    private static Scene s_nextScene;

    /// <summary>
    /// Gets the graphics device manager to control the presentation of graphics.
    /// </summary>
    public static GraphicsDeviceManager Graphics { get; private set; }

    /// <summary>
    /// Gets the graphics device used to create graphical resources and perform
primitive rendering.
    /// </summary>
    public static new GraphicsDevice GraphicsDevice { get; private set; }

    /// <summary>
    /// Gets the sprite batch used for all 2D rendering.
    /// </summary>
    public static SpriteBatch SpriteBatch { get; private set; }

    /// <summary>
    /// Gets the content manager used to load global assets.
    /// </summary>
    public static new ContentManager Content { get; private set; }
```

```
    /// <summary>
    /// Gets a reference to to the input management system.
    /// </summary>
    public static InputManager Input { get; private set; }

    /// <summary>
    /// Gets or Sets a value that indicates if the game should exit when the esc key
on the keyboard is pressed.
    /// </summary>
    public static bool ExitOnEscape { get; set; }

    /// <summary>
    /// Gets a reference to the audio control system.
    /// </summary>
    public static AudioController Audio { get; private set; }

    /// <summary>
    /// Creates a new Core instance.
    /// </summary>
    /// <param name="title">The title to display in the title bar of the game
window.</param>
    /// <param name="width">The initial width, in pixels, of the game
window.</param>
    /// <param name="height">The initial height, in pixels, of the game
window.</param>
    /// <param name="fullScreen">Indicates if the game should start in fullscreen
mode.</param>
    public Core(string title, int width, int height, bool fullScreen)
    {
        // Ensure that multiple cores are not created.
        if (s_instance != null)
        {
            throw new InvalidOperationException($"Only a single Core instance can
be created");
        }

        // Store reference to engine for global member access.
        s_instance = this;

        // Create a new graphics device manager.
        Graphics = new GraphicsDeviceManager(this);

        // Set the graphics defaults
        Graphics.PreferredBackBufferWidth = width;
        Graphics.PreferredBackBufferHeight = height;
```

```
        Graphics.IsFullScreen = fullScreen;

        // Apply the graphic presentation changes.
        Graphics.ApplyChanges();

        // Set the window title
        Window.Title = title;

        // Set the core's content manager to a reference of the base Game's
        // content manager.
        Content = base.Content;

        // Set the root directory for content.
        Content.RootDirectory = "Content";

        // Mouse is visible by default.
        IsMouseVisible = true;

        // Exit on escape is true by default
        ExitOnEscape = true;
    }

    protected override void Initialize()
    {
        base.Initialize();

        // Set the core's graphics device to a reference of the base Game's
        // graphics device.
        GraphicsDevice = base.GraphicsDevice;

        // Create the sprite batch instance.
        SpriteBatch = new SpriteBatch(GraphicsDevice);

        // Create a new input manager.
        Input = new InputManager();

        // Create a new audio controller.
        Audio = new AudioController();
    }

    protected override void UnloadContent()
    {
        // Dispose of the audio controller.
        Audio.Dispose();

        base.UnloadContent();
```

```
}

protected override void Update(GameTime gameTime)
{
    // Update the input manager.
    Input.Update(gameTime);

    // Update the audio controller.
    Audio.Update();

    if (ExitOnEscape && Input.Keyboard.WasKeyJustPressed(Keys.Escape))
    {
        Exit();
    }

    // if there is a next scene waiting to be switch to, then transition
    // to that scene.
    if (s_nextScene != null)
    {
        TransitionScene();
    }

    // If there is an active scene, update it.
    if (s_activeScene != null)
    {
        s_activeScene.Update(gameTime);
    }

    base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    // If there is an active scene, draw it.
    if (s_activeScene != null)
    {
        s_activeScene.Draw(gameTime);
    }

    base.Draw(gameTime);
}

public static void ChangeScene(Scene next)
{
    // Only set the next scene value if it is not the same
    // instance as the currently active scene.
```

```
        if (s_activeScene != next)
        {
            s_nextScene = next;
        }
    }

    private static void TransitionScene()
    {
        // If there is an active scene, dispose of it.
        if (s_activeScene != null)
        {
            s_activeScene.Dispose();
        }

        // Force the garbage collector to collect to ensure memory is cleared.
        GC.Collect();

        // Change the currently active scene to the new scene.
        s_activeScene = s_nextScene;

        // Null out the next scene value so it does not trigger a change over
and over.
        s_nextScene = null;

        // If the active scene now is not null, initialize it.
        // Remember, just like with Game, the Initialize call also calls the
        // Scene.LoadContent
        if (s_activeScene != null)
        {
            s_activeScene.Initialize();
        }
    }
}
```

The key changes here are:

1. The `using MonoGameLibrary.Scenes;` using directive was added so we have access to the `Scene` class.
2. The fields `_activeScene` and `_nextScene` were added to track which scene is currently active and which scene, if any, to switch to.
3. In `Update`:
    1. A check is made to see if there is a next scene, and if so, `TransitionScene` is called to gracefully switch from the current to the next.
    2. A check is made to see if there is an active scene, and if so, updates it.
4. An override for the `Draw` method was added where a check is made to see if there is an active scene, and if so, draws it.
5. The `ChangeScene` method was added which can be called when we want to tell the core to change from one scene to another one.
6. The `TransitionScene` method was added to gracefully switch from the current scene to the next by:
    1. Checking if there is an active scene, and disposing it if so.
    2. Triggering garbage collection to reclaim memory from the disposed scene.
    3. Assigning the next scene as the current scene.
    4. Checking if a current scene now exists, and initializing it if so.

> **TIP**
>
> Notice that we use a two-step process for scene transitions with separate `_activeScene` and `_nextScene` fields. This design allows the current scene to complete its update/draw cycle before the transition occurs, preventing potential issues that could arise from changing scenes in the middle of processing. The actual transition happens at a controlled point in the game loop, ensuring clean disposal of the old scene before initializing the new one.

Updating the Game

With the scene architecture in place, we are now ready to update the game by breaking it into separate scenes. We will create two scenes; a title scene and a gameplay scene. First, however, we need to add an additional SpriteFont Description that will be used during the title scene to display the title of the game. Open the *Content.mgcb* content project file in the MGCB Editor and perform the following:

1. Right-click the `fonts` folder and choose `Add > New Item...`.
2. Select `SpriteFont Description (.spritefont)` from the options.
3. Name the file `04B_30_5x` and click `Create`.

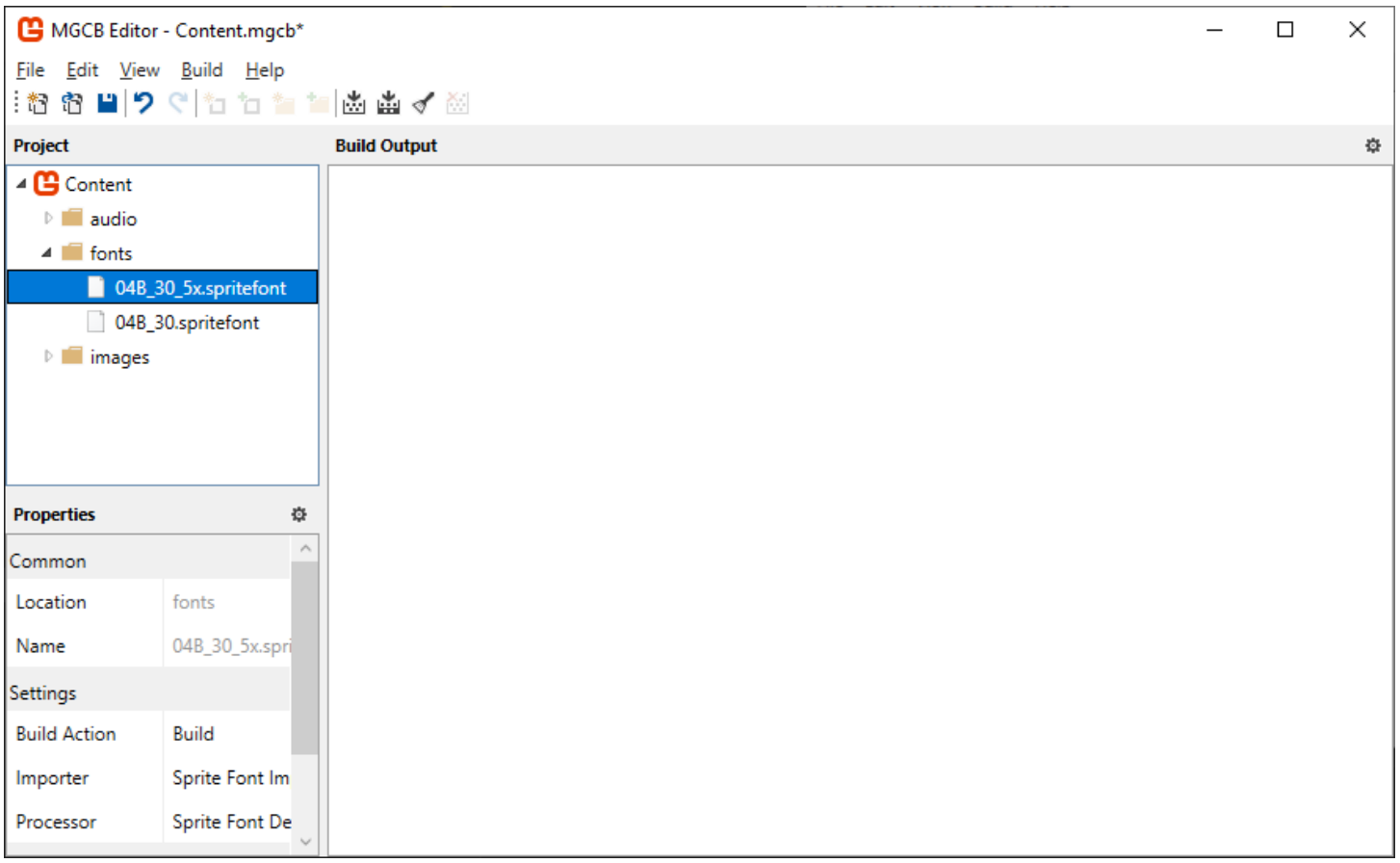


Figure 17-1: The *04B_30_5x.spritefont* file created in the MGCB Editor

Next, open the *04B_30_5x.spritefont* file in your code editor and make the following changes:

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">
    <FontName>04B_30.ttf</FontName>
    <Size>87.5</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

The Title Scene

The title scene serves as the game's initial starting point; the first impression the player gets when they launch the game. For our game, the title scene will display the text for the title of the game and a prompt to inform the player what action to take to start the game. We will use a simple trick for the title text in order to draw it with a drop shadow to add a bit of visual flair.

> **NOTE**
>
> As the following screens are specific to our game and are not reusable bits, these will be added to your game project.
>
> Although, if you do end up making screens that are completely reusable, there is nothing wrong with putting them in your Game Library, it is completely up to you.

To get started, first:

1. In your Game project, create a new folder named `Scenes`. We will put all of our game specific scenes here.
2. Add a new class file named `TitleScene.cs` to the `Scenes` folder you just created.
3. Add the following code as the initial structure for the class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Scenes;

namespace DungeonSlime.Scenes;

public class TitleScene : Scene
{

}
```

Title Scene Fields

Add the following fields to the `TitleScene` class:

```
private const string DUNGEON_TEXT = "Dungeon";
private const string SLIME_TEXT = "Slime";
private const string PRESS_ENTER_TEXT = "Press Enter To Start";

// The font to use to render normal text.
private SpriteFont _font;

// The font used to render the title text.
private SpriteFont _font5x;

// The position to draw the dungeon text at.
private Vector2 _dungeonTextPos;

// The origin to set for the dungeon text.
private Vector2 _dungeonTextOrigin;

// The position to draw the slime text at.
private Vector2 _slimeTextPos;

// The origin to set for the slime text.
private Vector2 _slimeTextOrigin;

// The position to draw the press enter text at.
private Vector2 _pressEnterPos;

// The origin to set for the press enter text when drawing it.
private Vector2 _pressEnterOrigin;
```

- Three `const` fields (`DUNGEON_TEXT`, `SLIME_TEXT`, `PRESS_ENTER_TEXT`) are added for the text that will be displayed on the title screen.
- The `_font` field stores a reference to the **SpriteFont** we will use to draw the press enter prompt with.
- The `_font5x` field stores a reference to the **SpriteFont** we will use to draw the dungeon and slime text with that will make up the title of the game.
- The `_dungeonTextPos` and `_dungeonTextOrigin` fields store the position and origin we will use to draw the "Dungeon" text at.
- The `_slimeTextPos` and `_slimeTextOrigin` fields store the position and origin we will draw the "Slime" text at.
- The `_pressEnterPos` and `_pressEnterOrigin` fields store the position and origin we will draw the "Press Enter To Start" text at.

Title Scene Methods

The `TitleScene` class will override the various methods from the base `Scene` class that it derives from to provide the initialization, content loading, update, and drawing logic.

Title Scene Initialize

Add the following override for the `Initialize` method to the `TitleScene` class:

```
public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // While on the title screen, we can enable exit on escape so the player
    // can close the game by pressing the escape key.
    Core.ExitOnEscape = true;

    // Set the position and origin for the Dungeon text.
    Vector2 size = _font5x.MeasureString(DUNGEON_TEXT);
    _dungeonTextPos = new Vector2(640, 100);
    _dungeonTextOrigin = size * 0.5f;

    // Set the position and origin for the Slime text.
    size = _font5x.MeasureString(SLIME_TEXT);
    _slimeTextPos = new Vector2(757, 207);
    _slimeTextOrigin = size * 0.5f;

    // Set the position and origin for the press enter text.
    size = _font.MeasureString(PRESS_ENTER_TEXT);
    _pressEnterPos = new Vector2(640, 620);
    _pressEnterOrigin = size * 0.5f;
}
```

- We set the `Core.ExitOnEscape` to true to allow players to exit the game when on the title screen by pressing the escape key.
- The position and origin for the "Dungeon", "Slime", and "Press Enter To Start" texts are set.

> **NOTE**
>
> You can see here we are using the **MeasureString** method for the font to work out how long the text to draw is, we then multiply this by `0.5` to work out the middle of the text so that we can properly set the origin of the text to its middle.

Title Scene LoadContent

Add the following override for the `LoadContent` method to the `TitleScene` class:

```
public override void LoadContent()
{
    // Load the font for the standard text.
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

    // Load the font for the title text.
    _font5x = Content.Load<SpriteFont>("fonts/04B_30_5x");
}
```

- The **SpriteFont** used to draw the "Press Enter To Start" text is loaded using the global content manager.
- The **SpriteFont** used to draw the "Dungeon" and "Slime" text is loaded using the scene's content manager.

> **TIP**
>
> Recall from Chapter 05 that when a **ContentManager** loads an asset for the first time, it caches it internally and the subsequent calls to load that asset will return the cached one instead of performing another disk read.
>
> By using a global content manager here to load assets that are used in multiple scenes, when they loaded in a different scene later, the cached version is returned instead of having to do another disk read, making the content loading more efficient.

Title Scene Update

Add the following override for the `Update` method to the `TitleScene` class:

```
public override void Update(GameTime gameTime)
{
    // If the user presses enter, switch to the game scene.
    if (Core.Input.Keyboard.WasKeyJustPressed(Keys.Enter))
    {
        Core.ChangeScene(new GameScene());
    }
}
```

- A check is made to see if the enter key is pressed, and if so, the `Core` is told to change to the game scene.

> **NOTE**
>
> Your editor might show an error here since we have not created the `GameScene` class yet. We will create it in a moment after finishing the title scene.

> **TIP**
>
> You will also notice the Title screen is only checking if the player hits the enter key to start the game, but we are not checking if they hit escape to quit the game, that is because it is already handled in the `Core` class `Update` method, if `ExitOnEscape` is true (as it is here), the game will automatically exit.

Title Scene Draw

For the final act, add the following override for the `Draw` method to the `TitleScene` class:

```
public override void Draw(GameTime gameTime)
{
    Core.GraphicsDevice.Clear(new Color(32, 40, 78, 255));

    // Begin the sprite batch to prepare for rendering.
    Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

    // The color to use for the drop shadow text.
    Color dropShadowColor = Color.Black * 0.5f;

    // Draw the Dungeon text slightly offset from it is original position and
    // with a transparent color to give it a drop shadow.
    Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos + new
Vector2(10, 10), dropShadowColor, 0.0f, _dungeonTextOrigin, 1.0f,
SpriteEffects.None, 1.0f);

    // Draw the Dungeon text on top of that at its original position.
    Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos, Color.White,
0.0f, _dungeonTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

    // Draw the Slime text slightly offset from it is original position and
    // with a transparent color to give it a drop shadow.
    Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos + new Vector2(10,
10), dropShadowColor, 0.0f, _slimeTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

    // Draw the Slime text on top of that at its original position.
    Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos, Color.White,
0.0f, _slimeTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

    // Draw the press enter text.
    Core.SpriteBatch.DrawString(_font, PRESS_ENTER_TEXT, _pressEnterPos,
Color.White, 0.0f, _pressEnterOrigin, 1.0f, SpriteEffects.None, 0.0f);

    // Always end the sprite batch when finished.
    Core.SpriteBatch.End();
}
```

- The back buffer is cleared.
- A `dropShadowColor` is created which is the color black with half transparency.
- The "Dungeon" text is drawn, first 10px down and to the left of the actual position using the drop shadow color, then again at its normal position overtop. Layering this way creates the drop shadow effect.
- The "Slime" text is drawn, again offset from its position first using the drop shadow color and then drawn again at its normal position overtop.
- Finally, the "Press Enter To Start" text is drawn.

With our Title screen in place, it is time to get started with the Game Scene, lets play.

The Game Scene

The Game Scene will contain our actual gameplay logic. This scene will handle updating and rendering the slime that the player controls, the bat the slime can eat, collision detection, score tracking, and input handling. Most of this logic has already been implemented in our `Game1` class in previous chapters, but now we will move it into a dedicated scene class. In the *Scenes* folder:

1. Add a new class file named `GameScene.cs` in the Game projects `Scenes` folder.

2. Add the following code as the initial structure for the class:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;
using MonoGameLibrary.Scenes;

namespace DungeonSlime.Scenes;

public class GameScene : Scene
{

}
```

The following code is effectively replacing the code we have already written in the original `Game1.cs` class, so it should look very familiar. Once complete, we can return to `Game1` and clear out all the redundant code because it is all nicely tidied up in the new `GameScene` class.

Game Scene Fields

Add the following fields to the `GameScene` class:

```
// Defines the slime animated sprite.
private AnimatedSprite _slime;

// Defines the bat animated sprite.
private AnimatedSprite _bat;

// Tracks the position of the slime.
private Vector2 _slimePosition;

// Speed multiplier when moving.
private const float MOVEMENT_SPEED = 5.0f;

// Tracks the position of the bat.
private Vector2 _batPosition;

// Tracks the velocity of the bat.
private Vector2 _batVelocity;

// Defines the tilemap to draw.
private Tilemap _tilemap;

// Defines the bounds of the room that the slime and bat are contained within.
private Rectangle _roomBounds;

// The sound effect to play when the bat bounces off the edge of the screen.
private SoundEffect _bounceSoundEffect;

// The sound effect to play when the slime eats a bat.
private SoundEffect _collectSoundEffect;

// The SpriteFont Description used to draw text
private SpriteFont _font;

// Tracks the players score.
private int _score;

// Defines the position to draw the score text at.
private Vector2 _scoreTextPosition;

// Defines the origin used when drawing the score text.
private Vector2 _scoreTextOrigin;
```

- The `_slime` and `_bat` fields store the animated sprites for the player controlled slime and the bat.
- The `_slimePosition` and `_batPosition` fields track the current position of the slime and bat.
- The `MOVEMENT_SPEED` constant defines the base movement speed for both the slime and bat.
- The `_batVelocity` field tracks the current velocity of the bat as it moves around the screen.
- The `_tilemap` field stores the tilemap that we will load and draw for the level background environment.
- The `_roomBounds` field defines a rectangular boundary that represents the boundary of the room that the slime and bat stays within.
- The `_bounceSoundEffect` and `_collectSoundEffect` fields store the sound effects to play when the bat bounces off a screen edge or is eaten by the slime.
- The `_font` field stores the font used to display the player's score.
- The `_score` field tracks the player's current score, which increases when the slime eats a bat.
- The `_scoreTextPosition` and `_scoreTextOrigin` defines the position and origin to use when drawing the score text.

Game Scene Methods

The `GameScene` class will override the various methods from the base `Scene` class that it derives from to provide the initialization, content loading, update, and drawing logic.

Game Scene Initialize

Add the following override for the `Initialize` method to the `GameScene` class:

```
public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // During the game scene, we want to disable exit on escape. Instead,
    // the escape key will be used to return back to the title screen
    Core.ExitOnEscape = false;

    Rectangle screenBounds = Core.GraphicsDevice.PresentationParameters.Bounds;

    _roomBounds = new Rectangle(
         (int)_tilemap.TileWidth,
         (int)_tilemap.TileHeight,
         screenBounds.Width - (int)_tilemap.TileWidth * 2,
         screenBounds.Height - (int)_tilemap.TileHeight * 2
     );

    // Initial slime position will be the center tile of the tile map.
    int centerRow = _tilemap.Rows / 2;
    int centerColumn = _tilemap.Columns / 2;
    _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

    // Initial bat position will the in the top left corner of the room.
    _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

    // Set the position of the score text to align to the left edge of the
    // room bounds, and to vertically be at the center of the first tile.
    _scoreTextPosition = new Vector2(_roomBounds.Left, _tilemap.TileHeight * 0.5f);

    // Set the origin of the text so it is left-centered.
    float scoreTextYOrigin = _font.MeasureString("Score").Y * 0.5f;
    _scoreTextOrigin = new Vector2(0, scoreTextYOrigin);

    // Assign the initial random velocity to the bat.
    AssignRandomBatVelocity();
}
```

- We set `Core.ExitOnEscape` to false because in the gameplay scene, we want to handle the escape key differently; instead of exiting the game, it will return to the title screen.
- The room bounds is calculated using the bounds of the screen and adjusting that so that it shrinks by one tile width and height on each edge, which will match with the tilemap wall boundary.
- The slime's initial position is set to be the center tile by calculating the center row and column.
- The bat's initial position is placed at the top left of the room bounds.
- The position and origin of the score text is precalculated. The height of the text is measured to properly calculate the center origin for vertical positioning.
- The `AssignRandomBatVelocity` method is called to give the bat its initial velocity.

Game Scene LoadContent

Add the following override for the `LoadContent` method to the `GameScene` class:

```
public override void LoadContent()
{
    // Create the texture atlas from the XML configuration file.
    TextureAtlas atlas = TextureAtlas.FromFile(Core.Content,
"images/atlas-definition.xml");

    // Create the slime animated sprite from the atlas.
    _slime = atlas.CreateAnimatedSprite("slime-animation");
    _slime.Scale = new Vector2(4.0f, 4.0f);

    // Create the bat animated sprite from the atlas.
    _bat = atlas.CreateAnimatedSprite("bat-animation");
    _bat.Scale = new Vector2(4.0f, 4.0f);

    // Create the tilemap from the XML configuration file.
    _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
    _tilemap.Scale = new Vector2(4.0f, 4.0f);

    // Load the bounce sound effect.
    _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

    // Load the collect sound effect.
    _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

    // Load the font.
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");
}
```

- The texture atlas is loaded using the global content manager, and the slime and bat animated sprites are created from it.
- The tilemap is loaded using the scene's content manager since they are specific to the gameplay scene.
- The sound effects are loaded using the scene's content manager since they are specific to the gameplay scene.
- The font is loaded using the global content manager since it is used in multiple scenes.

> **TIP**
>
> Notice how we are following a consistent pattern across scenes: global assets are loaded with `Core.Instance.Content` while scene-specific assets are loaded with the scene's `Content` property.

Game Scene Update

Add the following override for the `Update` method to the `GameScene` class:

```
public override void Update(GameTime gameTime)
{
    // Update the slime animated sprite.
    _slime.Update(gameTime);

    // Update the bat animated sprite.
    _bat.Update(gameTime);

    // Check for keyboard input and handle it.
    CheckKeyboardInput();

    // Check for gamepad input and handle it.
    CheckGamePadInput();

    // Creating a bounding circle for the slime.
    Circle slimeBounds = new Circle(
        (int)(_slimePosition.X + (_slime.Width * 0.5f)),
        (int)(_slimePosition.Y + (_slime.Height * 0.5f)),
        (int)(_slime.Width * 0.5f)
    );

    // Use distance based checks to determine if the slime is within the
    // bounds of the game screen, and if it is outside that screen edge,
    // move it back inside.
    if (slimeBounds.Left < _roomBounds.Left)
    {
        _slimePosition.X = _roomBounds.Left;
    }
    else if (slimeBounds.Right > _roomBounds.Right)
    {
        _slimePosition.X = _roomBounds.Right - _slime.Width;
    }

    if (slimeBounds.Top < _roomBounds.Top)
    {
        _slimePosition.Y = _roomBounds.Top;
    }
    else if (slimeBounds.Bottom > _roomBounds.Bottom)
    {
        _slimePosition.Y = _roomBounds.Bottom - _slime.Height;
    }

    // Calculate the new position of the bat based on the velocity.
    Vector2 newBatPosition = _batPosition + _batVelocity;

    // Create a bounding circle for the bat.
```

```
Circle batBounds = new Circle(
    (int)(newBatPosition.X + (_bat.Width * 0.5f)),
    (int)(newBatPosition.Y + (_bat.Height * 0.5f)),
    (int)(_bat.Width * 0.5f)
);

Vector2 normal = Vector2.Zero;

// Use distance based checks to determine if the bat is within the
// bounds of the game screen, and if it is outside that screen edge,
// reflect it about the screen edge normal.
if (batBounds.Left < _roomBounds.Left)
{
    normal.X = Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Left;
}
else if (batBounds.Right > _roomBounds.Right)
{
    normal.X = -Vector2.UnitX.X;
    newBatPosition.X = _roomBounds.Right - _bat.Width;
}

if (batBounds.Top < _roomBounds.Top)
{
    normal.Y = Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Top;
}
else if (batBounds.Bottom > _roomBounds.Bottom)
{
    normal.Y = -Vector2.UnitY.Y;
    newBatPosition.Y = _roomBounds.Bottom - _bat.Height;
}

// If the normal is anything but Vector2.Zero, this means the bat had
// moved outside the screen edge so we should reflect it about the
// normal.
if (normal != Vector2.Zero)
{
    normal.Normalize();
    _batVelocity = Vector2.Reflect(_batVelocity, normal);

    // Play the bounce sound effect.
    Core.Audio.PlaySoundEffect(_bounceSoundEffect);
}

_batPosition = newBatPosition;
```

```
    if (slimeBounds.Intersects(batBounds))
    {
        // Choose a random row and column based on the total number of each
        int column = Random.Shared.Next(1, _tilemap.Columns - 1);
        int row = Random.Shared.Next(1, _tilemap.Rows - 1);

        // Change the bat position by setting the x and y values equal to
        // the column and row multiplied by the width and height.
        _batPosition = new Vector2(column * _bat.Width, row * _bat.Height);

        // Assign a new random velocity to the bat.
        AssignRandomBatVelocity();

        // Play the collect sound effect.
        Core.Audio.PlaySoundEffect(_collectSoundEffect);

        // Increase the player's score.
        _score += 100;
    }
}
```

- The animated sprites for the slime and bat are updated.
- Input from keyboard and gamepad is checked with dedicated methods `CheckKeyboardInput` and `CheckGamePadInput`.
- Collision detection is performed to:
  - Keep the slime within the room bounds.
  - Make the bat bounce off edges of the room bounds.
  - Detect when the slime eats the bat.
- When the slime eats the bat, the bat respawns in a random location within the room bounds, given a new velocity, the collect sound is played, and the score is increased.

Game Scene Helper Methods

Next, add these helper methods to the `GameScene` class:

```
private void AssignRandomBatVelocity()
{
    // Generate a random angle.
    float angle = (float)(Random.Shared.NextDouble() * Math.PI * 2);

    // Convert angle to a direction vector.
    float x = (float)Math.Cos(angle);
    float y = (float)Math.Sin(angle);
    Vector2 direction = new Vector2(x, y);

    // Multiply the direction vector by the movement speed
    _batVelocity = direction * MOVEMENT_SPEED;
}

private void CheckKeyboardInput()
{
    // Get a reference to the keyboard inof
    KeyboardInfo keyboard = Core.Input.Keyboard;

    // If the escape key is pressed, return to the title screen.
    if (Core.Input.Keyboard.WasKeyJustPressed(Keys.Escape))
    {
        Core.ChangeScene(new TitleScene());
    }

    // If the space key is held down, the movement speed increases by 1.5
    float speed = MOVEMENT_SPEED;
    if (keyboard.IsKeyDown(Keys.Space))
    {
        speed *= 1.5f;
    }

    // If the W or Up keys are down, move the slime up on the screen.
    if (keyboard.IsKeyDown(Keys.W) || keyboard.IsKeyDown(Keys.Up))
    {
        _slimePosition.Y -= speed;
    }

    // if the S or Down keys are down, move the slime down on the screen.
    if (keyboard.IsKeyDown(Keys.S) || keyboard.IsKeyDown(Keys.Down))
    {
        _slimePosition.Y += speed;
    }

    // If the A or Left keys are down, move the slime left on the screen.
    if (keyboard.IsKeyDown(Keys.A) || keyboard.IsKeyDown(Keys.Left))
```

```
    {
        _slimePosition.X -= speed;
    }

    // If the D or Right keys are down, move the slime right on the screen.
    if (keyboard.IsKeyDown(Keys.D) || keyboard.IsKeyDown(Keys.Right))
    {
        _slimePosition.X += speed;
    }

    // If the M key is pressed, toggle mute state for audio.
    if (keyboard.WasKeyJustPressed(Keys.M))
    {
        Core.Audio.ToggleMute();
    }

    // If the + button is pressed, increase the volume.
    if (keyboard.WasKeyJustPressed(Keys.OemPlus))
    {
        Core.Audio.SongVolume += 0.1f;
        Core.Audio.SoundEffectVolume += 0.1f;
    }

    // If the - button was pressed, decrease the volume.
    if (keyboard.WasKeyJustPressed(Keys.OemMinus))
    {
        Core.Audio.SongVolume -= 0.1f;
        Core.Audio.SoundEffectVolume -= 0.1f;
    }
}

private void CheckGamePadInput()
{
    // Get the gamepad info for gamepad one.
    GamePadInfo gamePadOne = Core.Input.GamePads[(int)PlayerIndex.One];

    // If the A button is held down, the movement speed increases by 1.5
    // and the gamepad vibrates as feedback to the player.
    float speed = MOVEMENT_SPEED;
    if (gamePadOne.IsButtonDown(Buttons.A))
    {
        speed *= 1.5f;
        GamePad.SetVibration(PlayerIndex.One, 1.0f, 1.0f);
    }
    else
    {
```

```
            GamePad.SetVibration(PlayerIndex.One, 0.0f, 0.0f);
        }

        // Check thumbstick first since it has priority over which gamepad input
        // is movement.  It has priority since the thumbstick values provide a
        // more granular analog value that can be used for movement.
        if (gamePadOne.LeftThumbStick != Vector2.Zero)
        {
            _slimePosition.X += gamePadOne.LeftThumbStick.X * speed;
            _slimePosition.Y -= gamePadOne.LeftThumbStick.Y * speed;
        }
        else
        {
            // If DPadUp is down, move the slime up on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadUp))
            {
                _slimePosition.Y -= speed;
            }

            // If DPadDown is down, move the slime down on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadDown))
            {
                _slimePosition.Y += speed;
            }

            // If DPapLeft is down, move the slime left on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadLeft))
            {
                _slimePosition.X -= speed;
            }

            // If DPadRight is down, move the slime right on the screen.
            if (gamePadOne.IsButtonDown(Buttons.DPadRight))
            {
                _slimePosition.X += speed;
            }
        }
    }
```

- `AssignRandomBatVelocity`: Calculates a random direction and applies it to the bat's velocity.
- `CheckKeyboardInput`: Handles keyboard controls for moving the slime, toggling audio settings, and returning to the title screen.
- `CheckGamePadInput`: Handles gamepad controls for moving the slime.

Game Scene Draw

Finally, add the following override for the `Draw` method to the `GameScene` class:

```
public override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    Core.GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

    // Draw the tilemap
    _tilemap.Draw(Core.SpriteBatch);

    // Draw the slime sprite.
    _slime.Draw(Core.SpriteBatch, _slimePosition);

    // Draw the bat sprite.
    _bat.Draw(Core.SpriteBatch, _batPosition);

    // Draw the score.
    Core.SpriteBatch.DrawString(
        _font,              // spriteFont
        $"Score: {_score}", // text
        _scoreTextPosition, // position
        Color.White,        // color
        0.0f,               // rotation
        _scoreTextOrigin,   // origin
        1.0f,               // scale
        SpriteEffects.None, // effects
        0.0f                // layerDepth
    );

    // Always end the sprite batch when finished.
    Core.SpriteBatch.End();
}
```

- The back buffer is cleared.
- The tilemap is drawn.
- The slime and bat animated sprites are drawn at their current positions.
- The player's score is drawn at using its precalculated position and origin so that it is in the top left of the room bounds centered on the wall sprite.

This concludes the `GameScene` class. With all the logic for our actual gameplay now housed in a single place, we can clean up our project to use it.

Updating the Game1 Class

With our scene system and scene classes in place, we can now simplify our main `Game1` class to just initialize the game and start with the title scene. Open the `Game1.cs` file and replace its content with the following:

```
using DungeonSlime.Scenes;
using Microsoft.Xna.Framework.Media;
using MonoGameLibrary;

namespace DungeonSlime;

public class Game1 : Core
{
    // The background theme song.
    private Song _themeSong;

    public Game1() : base("Dungeon Slime", 1280, 720, false)
    {

    }

    protected override void Initialize()
    {
        base.Initialize();

        // Start playing the background music.
        Audio.PlaySong(_themeSong);

        // Start the game with the title scene.
        ChangeScene(new TitleScene());
    }

    protected override void LoadContent()
    {
        // Load the background theme music.
        _themeSong = Content.Load<Song>("audio/theme");
    }
}
```

> **NOTE**
>
> Feel free to check your homework and compare the original `Game1` class with the updated version, as well as checking the `GameScene` class did not lose any functionality (it has not, but you have to be sure!). Refactoring code to be cleaner and more organised is a careful task.

The `Game1` class is now much simpler as most of the game logic has been moved to the appropriate scene classes.

The updates include:

1. Sets up the game window with the constructor parameters.
2. Overrides the `Initialize` method to set the title scene as the starting scene.
3. Overrides the `LoadContent` method to load the background theme song and start playing it.

Running the game now, we can see that once the game screen comes up, the title scene is displayed with the animated slime and the press enter prompt. The background music starts playing on this scene as well. Pressing enter from here will switch to the game scene where the game starts and we can play the game implemented thus far.

Figure 17-2: The game launching with the title screen first, then transitioning to the game play screen when enter is pressed

Conclusion

In this chapter, you accomplished the following:

- Learned about scene management and why it is important for organizing game code.
- Created an abstract `Scene` base class that provides a consistent lifecycle similar to the MonoGame **Game** class.
- Implemented the `IDisposable` interface to properly handle resource cleanup.
- Extended the `Core` class to handle scene transitions and management.
- Created a `TitleScene` for the main menu with text prompts and animations.
- Created a `GameScene` that encapsulates the gameplay mechanics.
- Refactored the main `Game1` class to be much simpler by using the scene system.

The approach we have taken follows a common pattern in game development, where each scene has control over its own lifecycle and resources. This pattern simplify state management by isolating different game states from one another. As your game grows in complexity, you could easily extend this system to include additional scenes like a pause menu or a game over screen.

In the next chapter, we will explore **RenderTarget2D** and how we can use it to add different types of transitions when switching scenes.

Test Your Knowledge

1. What are the main benefits of implementing a scene management system in a game?

> The main benefits include:
>
> - Improved organization by separating different parts of the game into self-contained modules.
> - Better memory management by loading assets only when needed and unloading them when leaving a scene.
> - Simplified state handling as each scene maintains its own state without affecting others.
> - Increased code reusability through the ability to create reusable scene templates.

2. How does the scene lifecycle in our implementation mirror the MonoGame Game class lifecycle?

> The scene lifecycle mirrors the MonoGame Game class lifecycle by implementing similar methods in the same order:
>
> - `Initialize` is called once when the scene becomes active.
> - `LoadContent` is called at the end of the `Initialize` method.
> - `Update` is called every frame to update game logic.
> - `Draw` is called every frame to render the scene.
> - `UnloadContent` is called when transitioning away from the scene.

3. What is the purpose of having a separate **ContentManager** for each scene?

> Having a separate **ContentManager** for each scene:
>
> - Allows scene-specific content to be automatically unloaded when the scene is disposed.
> - Provides better organization of which assets belong to which scenes.
> - Improves memory efficiency by only loading assets that are currently needed.
> - Makes it clear which assets are meant to be used globally versus locally to a scene.

4. When implementing scene transitions, why do we use a two-step process with `_nextScene` and `_activeScene`?

   The two-step process with `_nextScene` and `_activeScene` is used because:

   - It allows the current scene to complete its update/draw cycle before the transition occurs.
   - It provides a clean way to handle the disposal of the current scene before initializing the new one.
   - It ensures that scene transitions happen at a safe point in the game loop.
   - It prevents potential issues that could occur from immediately changing scenes in the middle of an update or draw operation.

Chapter 18: Texture Sampling and Tiling Backgrounds

Learn how to use texture sampling states and add a scrolling background effect to the game.

In previous chapters, we have drawn individual sprites and textures with the sprite batch, but for creating repeating background patterns, we need a more efficient approach than manually drawing the same texture multiple times. We could reuse the tilemap system that was created to make repeated background patterns, but this has a limitation in that the tiles are stationary and would require constantly updating the tiles and positions if we wanted to animate it. Instead, this chapter introduces texture sampling states, specifically focusing on how to create and animate tiled backgrounds using **SamplerState.PointWrap**.

In this chapter, you will:

- Learn about texture sampling and sampler states in MonoGame.
- Understand how wrap mode allows efficient texture tiling.
- Implement a scrolling tiled background for the title screen.

Understanding Texture Sampling

When a texture is drawn to the screen, MonoGame uses a process called "sampling" to determine which pixels from the texture should be displayed. Sampling is the process by which a graphics pipeline determines what color value to use from a texture when mapping it onto a surface. Think of it like placing a grid over an image and selecting which pixels to use when that image needs to be transformed in some way. When textures are drawn at their exact pixel size and position, with no rotation, sampling is straightforward, a direct 1:1 mapping for each pixel. However, when a texture is scaled, rotated, or only partially visible, the graphics hardware needs to decide how to interpret the texture data.

For example, if you draw a texture twice its size, there are not enough pixels to fill the new larger space, so the graphics hardware must determine how to fill those gaps. Similarly, if you were to scale down a texture, multiple source pixels might map to a single output pixel, requiring the hardware to decide which ones to use or how to blend them. The rules that govern these decisions are defined by sampler states.

Texture Coordinates

In graphics programming, textures are addressed using a normalized coordinate system ranging from 0.0 to 1.0, regardless of the texture's actual pixel dimensions:

- The top-left corner of a texture is (0.0, 0.0)
- The bottom-right corner is (1.0, 1.0)
- The center is (0.5, 0.5)

This normalized system means that regardless of whether your texture is 32×32 pixels or 2048×2048 pixels, the coordinates to access the entire texture always range from 0.0 to 1.0. The graphics hardware automatically converts these normalized coordinates to the actual pixel locations within the texture.

When you use **SpriteBatch** to draw a texture with a source rectangle specified in pixels, MonoGame internally converts those pixel coordinates to normalized texture coordinates before sending them to the GPU. Similarly, when you specify a destination rectangle, MonoGame determines how the normalized texture coordinates should map to screen coordinates.

What is a SamplerState

A SamplerState controls how textures are sampled during rendering. It determines several aspects of texture rendering:

- How textures are filtered when scaled (point/linear/anisotropic filtering).
- How texture coordinates outside the 0.0 to 1.0 range are handled (wrap/clamp/mirror).
- How mipmap levels are selected and blended.

In MonoGame, these sampler states are represented by the **SamplerState** class, which provides several predefined states for common scenarios.

Filtering Modes

One aspect of sampler states if the filtering mode. Filtering in computer graphics refers to how the graphics hardware decides to blend or select pixels when a texture is displayed at a different size than its original dimensions. The filter mode determines how pixels are interpolated (calculated and combined) when a texture is scaled up or down.

Think of filtering as the graphics hardware's strategy for filling in missing information when a texture is transformed. When you enlarge a texture, the system needs to create new pixels that did not exist in the original. When you shrink a texture, multiple original pixels must be combined into fewer output pixels. The filtering mode controls how this process happens.

There are three filtering modes available in MonoGame:

- Point
- Linear
- Anisotropic.

Each mode offers a different balance between performance and visual quality.

Point Filtering Mode

Point mode uses what is called nearest neighbor sampling. This means that when a texture is scaled, the closest pixel is selected resulting in a pixelated appearance when scaled up. This is typically the ideal mode to use for pixel-art games when you want to preserve the exact pixel appearance of a scaled texture. Point filtering is the least computationally expensive filtering mode of the three since it only samples a single pixel without any blending calculations. This makes it the fastest option, especially on lower-end hardware or when rendering many textures simultaneously.

Figure 18-1: Illustration of using Point filtering mode. Left: MonoGame logo at 32x32 pixels. Right: MonoGame logo at 128x128 pixels

Linear Filtering Mode

Linear filtering mode blends neighboring pixels when the texture is scaled. This creates a smoother, but potentially blurrier appearance. This is better for realistic or high-resolution textures. Linear filtering requires more processing power than point filtering since it needs to sample multiple pixels and calculated weighted averages between them. However, on modern hardware, this performance difference is usually negligible for 2D games, making it a good balance between quality and performance for most non-pixel art games or assets.

Figure 18-2: Illustration of using Linear filtering mode. Left: MonoGame logo at 32x32 pixels. Right: MonoGame logo at 128x128 pixels

Anisotropic Filtering Mode

Anisotropic filtering mode provides higher-quality filter for textures viewed from oblique angles. This is primarily used in 3D rendering. It helps textures look more detailed by reducing blur and aliasing that occurs when a surface is angled away from the viewer. Anisotropic filtering is the most computationally intensive option, as it samples many more pixels and performs complex calculations to determine the appropriate blending. The performance cost increases with the anisotropic level (typically 2x, 4x, 8x, or 16x), which determines how many samples are taken. This can significantly impact frame rates in complex 3D scenes, especially on mobile or lower-end devices, so it should be used selectively where visual quality at angles is most important.

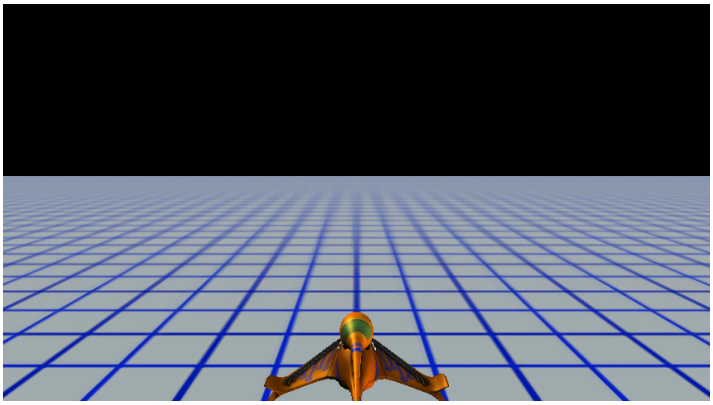


Figure 18-3: Illustration of the MonoGame Fuel cell demo using Linear filtering

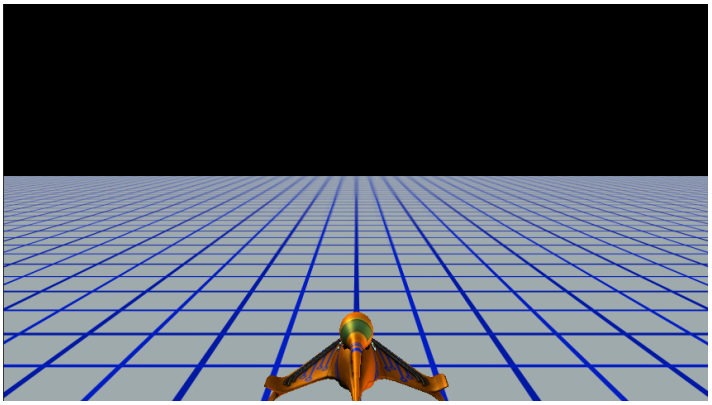


Figure 18-4: Illustration of the MonoGame Fuel cell demo using Anisotropic filtering

Addressing Modes

The other aspect is the addressing mode, which determines what happens when texture coordinates fall outside the normal 0.0 to 1.0 range. When drawing textures, sometimes the calculated texture coordinates end up being less than 0.0 or greater than 1.0. The addressing mode tells the graphics hardware what to do in these situations; whether to repeat the texture, mirror it, stretch the edge pixels, or use a specific border color. Think of it as instructions for what to display in areas where the texture does not naturally exist. These modes are particularly important for creating effects like seamless tiling backgrounds, scrolling texture, or handling the edges of transformed sprites properly. There are four addressing modes available; Wrap, Mirror, Clamp, and Border Color:

Wrap Mode

When using Wrap mode, at every whole integer of the texture coordinates (0.0 and 1.0), the texture coordinate is wrapped to stay within the 0.0 to 1.0 range (i.e. if the texture coordinate is 1.2, then that wraps to become 0.2). This creates a tiled pattern.

For example, if we were to take the MonoGame logo at 128x128 pixels and draw it to a destination rectangle that was three times the size at 384x384 pixels, then the texture coordinates of the destination rectangle become (0.0, 0.0) (top-left), (3.0, 0.0) (top-right), (0.0) (bottom-left), and (3.0, 3.0) bottom-right. The MonoGame logo texture would repeat three times horizontally and vertically within the destination.

Figure 18-5: Illustration of the MonoGame logo drawn using wrapped addressing mode

Mirror Mode

Mirror mode is similar to Wrap mode. However instead of repeating the texture at every whole integer of the texture coordinates (0.0 and 1.0), the texture is flipped, creating a mirror effect.

Using the same example as above, taking the MonoGame logo at 128x128 pixels and drawing it to a destination rectangle three times the size with Mirror mode would produce the following:

Figure 18-6: Illustration of the MonoGame logo drawn using mirror addressing mode

Clamp Mode

When using Clamp mode, the texture coordinates are clamped to the 0.0 and 1.0 range. Texture coordinates that would go beyond this (edge pixels) are smeared.

The simplest demonstration of this is to use a checkerboard pattern. If we were to take a texture that was a checkerboard pattern at 128x128 pixels and draw it to a destination rectangle three times the size with Clamped mode, then any pixels that extend outside the clamped range would smeared, producing the following:

Figure 18-7: Illustration of a checkerboard pattern drawn using clamped addressing mode

Border Color

When using Border Color mode, similar to Clamped mode, the texture coordinates are clamped to the 0.0 and 1.0f range. However, in Border Color mode, texture coordinates that would go beyond this (edge pixels) are instead drawn using the color set as the border color for the sampler state.

For example, if we use the checkerboard pattern again, using Border Color mode with a border color of green, then it would produce the following:

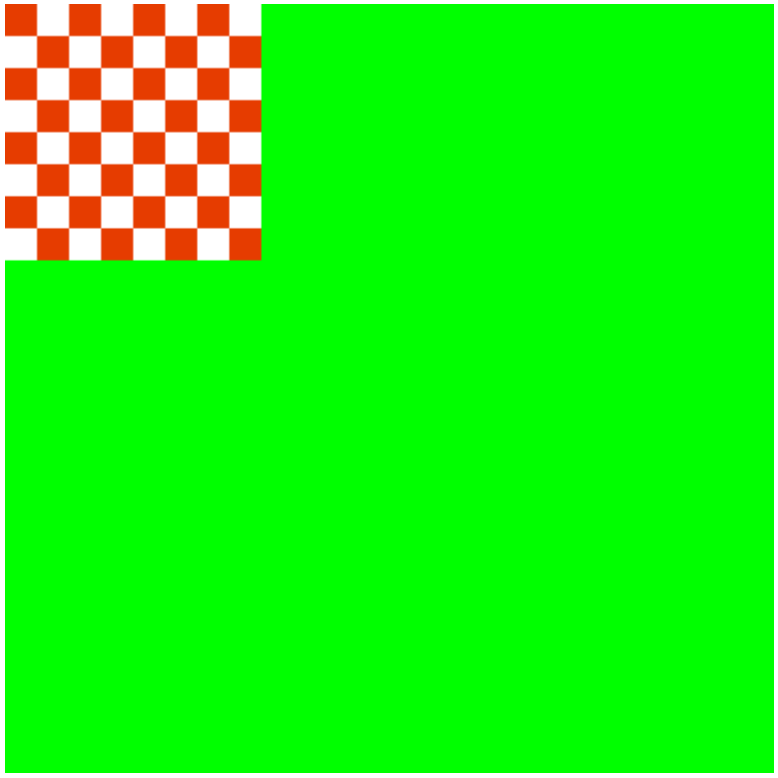


Figure 18-8: Illustration of a checkerboard pattern drawn using border addressing mode with the border color set to green

Using SamplerStates

MonoGame offers several predefined sampler states as part of the **SamplerState** class that cover common scenarios:

SamplerState	Description	Common Use Case
AnisotropicClamp	Combines Anisotropic filter mode with the Clamp addressing mode.	3D textures viewed at oblique angles, like ground textures in a 3D world where you want high-quality filtering but no repeating patterns.
AnisotropicWrap	Combines Anisotropic filter mode with the Wrap addressing mode.	Terrain textures in 3D games where you need high-quality filtering and repeating patterns over large surfaces.
LinearClamp	Combines Linear filter mode with the Clamp addressing mode.	UI elements and single sprites where you want smooth scaling but no repeating patterns. Good for realistic graphics that need to scale.
LinearWrap	Combines Linear filter mode with the Wrap addressing mode.	Scrolling backgrounds with smooth transitions, like water or cloud textures that need to tile seamlessly with blended edges.
PointClamp	Combines Point filter mode with the Clamp addressing mode.	Pixel art sprites and UI elements where you want to preserve crisp pixel edges without any blurring when scaled. Default for most 2D games.
PointWrap	Combines Point filter mode with the Wrap addressing mode.	Tiled pixel art backgrounds and patterns where you want crisp pixels and repeating patterns

When using the SpriteBatch, you specify which sampler state you want to use as the `samplerState` parameter for the SpriteBatch.Begin method

```
// Example of using the Point Clamp sampler state
spriteBatch.Begin(samplerState: SamplerState.PointClamp);
```

> **NOTE**
>
> The default sampler state for SpriteBatch is SamplerState.LinearClamp in MonoGame, though SamplerState.PointClamp is often preferred for pixel art games to prevent blurring.

Adding a Scrolling Background to the Title Scene

We will now update the title scene of our game by adding a scrolling background pattern using **SamplerState.PointWrap**. By using the Wrap addressing mode, we can create a large scrolling background using only a small texture. When the texture is drawn with a destination rectangle larger than the texture itself, the Wrap mode will automatically tile the texture to fill the space. By adjusting the source rectangle over time, we can create a scrolling effect with minimal effort.

First, download the following image of a repeatable background pattern by right-clicking it and saving it as `background-pattern.png` in the `Content/images` folder of the game project:

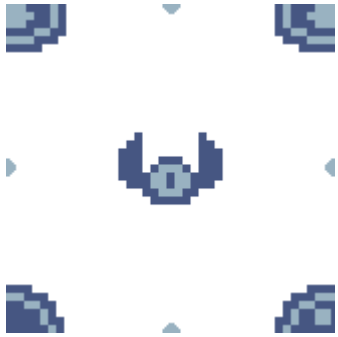


Figure 18-9: The repeatable background pattern we will use for the title screen

Next, add this texture to your content project using the MGCB Editor:

1. Open the `Content.mgcb` content project file in the MGCB Editor.
2. Right-click the images folder and choose `Add > Existing item...`.
3. Navigate to and select the `background-pattern.png` file.
4. Save the changes and close the MGCB Editor.

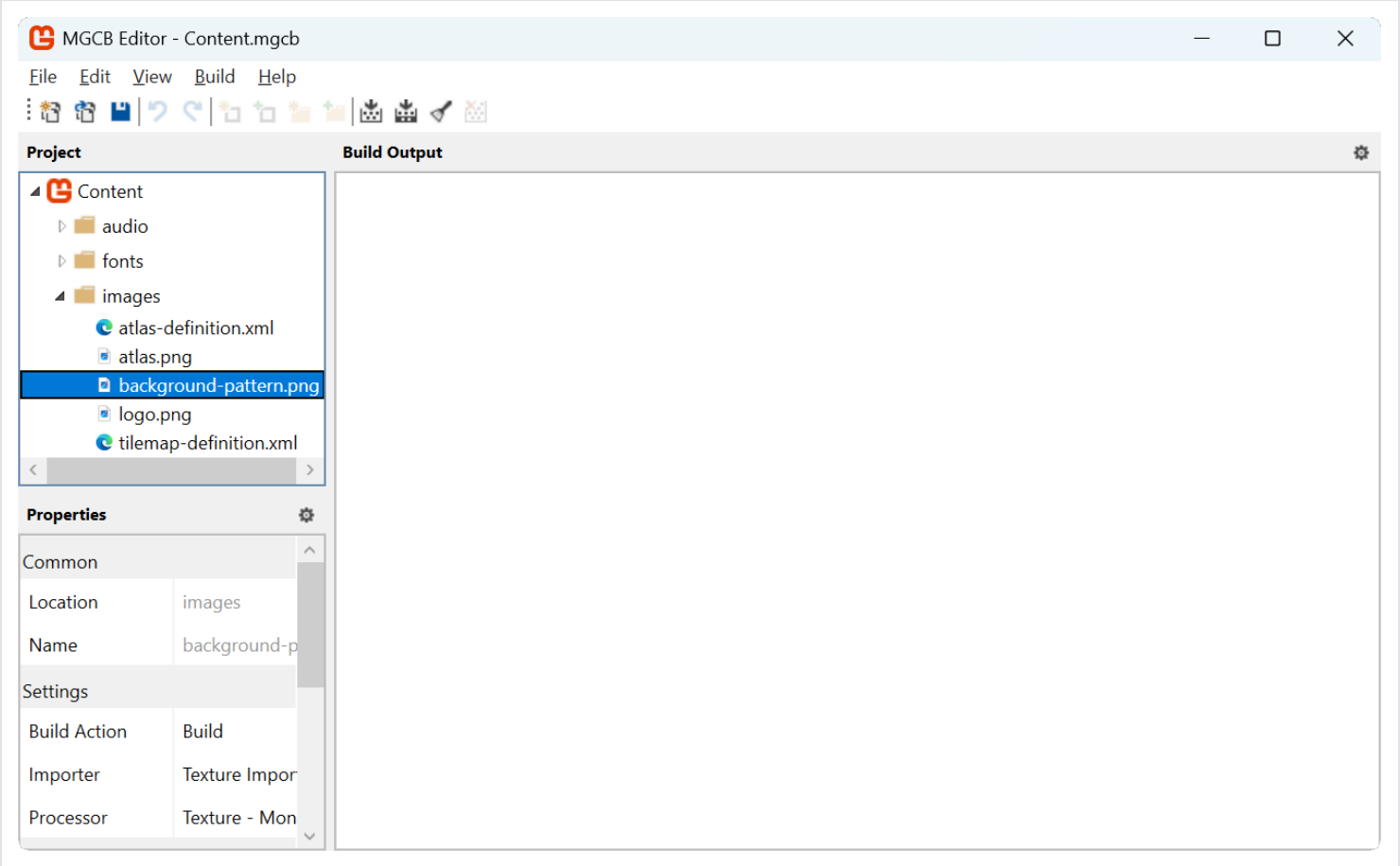


Figure 18-10: The MGCB Editor with the *background-pattern* image added

Updating the Title Scene

Now that we have the background pattern texture added, we can update the `TitleScene` class to implement the scrolling background. Open the `TitleScene.cs` file in the game project and update it to the following

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Scenes;

namespace DungeonSlime.Scenes;

public class TitleScene : Scene
{
    private const string DUNGEON_TEXT = "Dungeon";
    private const string SLIME_TEXT = "Slime";
    private const string PRESS_ENTER_TEXT = "Press Enter To Start";

    // The font to use to render normal text.
    private SpriteFont _font;

    // The font used to render the title text.
    private SpriteFont _font5x;

    // The position to draw the dungeon text at.
    private Vector2 _dungeonTextPos;

    // The origin to set for the dungeon text.
    private Vector2 _dungeonTextOrigin;

    // The position to draw the slime text at.
    private Vector2 _slimeTextPos;

    // The origin to set for the slime text.
    private Vector2 _slimeTextOrigin;

    // The position to draw the press enter text at.
    private Vector2 _pressEnterPos;

    // The origin to set for the press enter text when drawing it.
    private Vector2 _pressEnterOrigin;

    // The texture used for the background pattern.
    private Texture2D _backgroundPattern;

    // The destination rectangle for the background pattern to fill.
    private Rectangle _backgroundDestination;

    // The offset to apply when drawing the background pattern so it appears to
    // be scrolling.
```

```
private Vector2 _backgroundOffset;

// The speed that the background pattern scrolls.
private float _scrollSpeed = 50.0f;

public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // While on the title screen, we can enable exit on escape so the player
    // can close the game by pressing the escape key.
    Core.ExitOnEscape = true;

    // Set the position and origin for the Dungeon text.
    Vector2 size = _font5x.MeasureString(DUNGEON_TEXT);
    _dungeonTextPos = new Vector2(640, 100);
    _dungeonTextOrigin = size * 0.5f;

    // Set the position and origin for the Slime text.
    size = _font5x.MeasureString(SLIME_TEXT);
    _slimeTextPos = new Vector2(757, 207);
    _slimeTextOrigin = size * 0.5f;

    // Set the position and origin for the press enter text.
    size = _font.MeasureString(PRESS_ENTER_TEXT);
    _pressEnterPos = new Vector2(640, 620);
    _pressEnterOrigin = size * 0.5f;

    // Initialize the offset of the background pattern at zero.
    _backgroundOffset = Vector2.Zero;

    // Set the background pattern destination rectangle to fill the entire
    // screen background.
    _backgroundDestination = Core.GraphicsDevice.PresentationParameters.Bounds;
}

public override void LoadContent()
{
    // Load the font for the standard text.
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

    // Load the font for the title text.
    _font5x = Content.Load<SpriteFont>("fonts/04B_30_5x");

    // Load the background pattern texture.
```

```
        _backgroundPattern = Content.Load<Texture2D>("images/background-pattern");
    }

    public override void Update(GameTime gameTime)
    {
        // If the user presses enter, switch to the game scene.
        if (Core.Input.Keyboard.WasKeyJustPressed(Keys.Enter))
        {
            Core.ChangeScene(new GameScene());
        }

        // Update the offsets for the background pattern wrapping so that it
        // scrolls down and to the right.
        float offset = _scrollSpeed * (float)gameTime.ElapsedGameTime.TotalSeconds;
        _backgroundOffset.X -= offset;
        _backgroundOffset.Y -= offset;

        // Ensure that the offsets do not go beyond the texture bounds so it is
        // a seamless wrap.
        _backgroundOffset.X %= _backgroundPattern.Width;
        _backgroundOffset.Y %= _backgroundPattern.Height;
    }

    public override void Draw(GameTime gameTime)
    {
        Core.GraphicsDevice.Clear(new Color(32, 40, 78, 255));

        // Draw the background pattern first using the PointWrap sampler state.
        Core.SpriteBatch.Begin(samplerState: SamplerState.PointWrap);
        Core.SpriteBatch.Draw(_backgroundPattern, _backgroundDestination, new
Rectangle(_backgroundOffset.ToPoint(), _backgroundDestination.Size), Color.White
* 0.5f);
        Core.SpriteBatch.End();

        // Begin the sprite batch to prepare for rendering.
        Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // The color to use for the drop shadow text.
        Color dropShadowColor = Color.Black * 0.5f;

        // Draw the Dungeon text slightly offset from it is original position and
        // with a transparent color to give it a drop shadow.
        Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos + new
Vector2(10, 10), dropShadowColor, 0.0f, _dungeonTextOrigin, 1.0f,
SpriteEffects.None, 1.0f);
```

```
        // Draw the Dungeon text on top of that at its original position.
        Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos,
Color.White, 0.0f, _dungeonTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

        // Draw the Slime text slightly offset from it is original position and
        // with a transparent color to give it a drop shadow.
        Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos
+ new Vector2(10, 10), dropShadowColor, 0.0f, _slimeTextOrigin, 1.0f,
SpriteEffects.None, 1.0f);

        // Draw the Slime text on top of that at its original position.
        Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos, Color.White,
0.0f, _slimeTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

        // Draw the press enter text.
        Core.SpriteBatch.DrawString(_font, PRESS_ENTER_TEXT, _pressEnterPos,
Color.White, 0.0f, _pressEnterOrigin, 1.0f, SpriteEffects.None, 0.0f);

        // Always end the sprite batch when finished.
        Core.SpriteBatch.End();
    }
}
```

The key changes here are

- The `_backgroundPattern` field was added to store a reference to the texture of the background pattern once its loaded.
- The `_backgroundDestination` field was added to define the destination rectangle to draw the background pattern to.
- The `_backgroundOffset` field was added to apply an offset to the source rectangle when rendering the background pattern to give it the appearance that it is scrolling.
- The `_scrollSpeed` field was added to set the speed at which the background pattern scrolls.
- In `Initialize`, the initial offset of the background is set to **Vector2.Zero** and the background destination rectangle is set to the bounds of the screen.
- In `LoadContent`, the *background-pattern* texture is loaded and stored in `_backgroundPattern`.
- In `Update`, the X and Y offset for the background source rectangle is calculated by adjusting the based on the scroll speed multiplied by the delta time. Modulo division is then used to ensure that the new offset calculations remain within the width and height bounds of the background texture so that the wrap is seamless.
- In `Draw`, a new sprite batch begin/end block is added that uses **SamplerState.PointWrap** and draws the background pattern to the destination rectangle using a source rectangle with the offset calculations.

> **NOTE**
>
> We use two separate sprite batch begin/end blocks for this. The first uses **SamplerState.PointWrap** to draw the background and the second uses **SamplerState.PointClamp** to draw the rest of the scene.
>
> This separation is necessary because changing the sampler state requires ending the current sprite batch and beginning a new one.

Running the game now with these changes, the title screen now has a scroll background that adds more visual depth and interest to it than just the plain colored background we had before.

Figure 18-11: The title screen now with the repeating background texture of the slime and bat scrolling diagonally down and to the right

Conclusion

In this chapter, you accomplished the following:

- Learned about texture coordinates and how they map from normalized 0.0 to 1.0 space to actual pixel locations.
- Understood the difference between various filtering modes (Point, Linear, Anisotropic) and their visual impact.
- Explored different addressing modes (Wrap, Mirror, Clamp, Border) and when to use each.
- Discovered how to use predefined sampler states to simplify common rendering tasks.
- Implemented a scrolling background pattern using **SamplerState.PointWrap**.

Test Your Knowledge

1. What is the difference between normalized texture coordinates and pixel coordinates?

> Normalized texture coordinates always range from 0.0 to 1.0 regardless of the texture's actual pixel dimensions. The top-left corner is (0.0, 0.0) and the bottom-right is (1.0, 1.0). Pixel coordinates, on the other hand, directly reference specific pixel locations within the texture using integer values based on the actual texture dimensions. MonoGame automatically converts between these coordinate systems when drawing textures.

2. Which filtering mode would be most appropriate for a pixel art game, and why?

> Point filtering (also called nearest neighbor) is most appropriate for pixel art games. It selects the closest pixel when scaling rather than blending neighboring pixels, which preserves the crisp, pixelated aesthetic that defines pixel art. Linear or Anisotropic filtering would blur the intentionally sharp edges of pixel art graphics.

3. Why do we use modulo (%) operation on the background offset values when implementing the scrolling background?

> The modulo operation ensures that the offset values always remain within the bounds of the original texture dimensions. This prevents potential graphical artifacts that could appear when the offset exceeds the texture size, and it guarantees seamless wrapping as the background continuously scrolls. Without this, the background pattern might show visible seams or discontinuities when it repeats.

4. Why do we need to use two separate sprite batch begin/end blocks when drawing the background and the other elements in the title scene?

> We need separate blocks because changing the sampler state requires ending the current batch and beginning a new one. Since we want to use **SamplerState.PointWrap** for the tiling background but **SamplerState.PointClamp** for the UI elements, we must use two distinct sprite batch blocks with different sampler state settings. Using a single batch would apply the same sampler state to all drawn elements.

5. How does using a tiled background with **SamplerState.PointWrap** compare to manually drawing multiple copies of a texture to create a background?

   Using **SamplerState.PointWrap** offers several advantages:

   - It requires only a single draw call instead of multiple calls for each tile.
   - No need to calculate positions for each individual tile
   - Manually drawing multiple copies would be more code-intensive, less performant, and harder to maintain, especially for animations.

Chapter 19: User Interface Fundamentals

Learn the core principles of game user interface design.

A game's user interface (UI) allows players to interact with the game beyond just controlling the character. UI elements include menus, buttons, panels, labels, and various other interactive components that provide information and control options to the player.

In this chapter you will:

- Learn the basics of user interface design in games.
- Understand different UI types and their purposes.
- Explore UI layout approaches and positioning strategies.
- Understand the parent-child relationship for UI elements.
- Learn about accessibility considerations in game UI design.

We will first start by understanding what a user interface is and how it functions in game development.

Understanding Game User Interfaces

A user interface in games serves as a bridge between the player and the game's systems. Well designed UIs help players navigate the game's mechanics, understand their current status, and make informed decisions. For new game developers, understanding UI principles is crucial because even the most mechanically sound game can fail if players can not effectively interact with it.

Game UIs consist of various visual elements that serve different purposes:

1. **Information Display**: Elements like health bars, score counters, or minimap displays provide players with game state information. These elements help players understand their progress, resources, and current status without interrupting gameplay.
2. **Interactive Controls**: Buttons, sliders, checkboxes, and other interactive elements allow players to make choices, adjust settings, or navigate through different sections of the game. These elements should provide clear visual feedback when interacted with to confirm the player's actions.
3. **Feedback Mechanisms**: Visual effects like highlighting, color changes, or animations that respond to player actions help confirm that input was received. This feedback loop creates an intuitive and responsive feel for the UI in your game.

User interfaces for games can be categorized into two main types, each with their own design considerations:

- **Diegetic UI**: These elements exist within the game world itself and are often part of the narrative. Examples include a health meter integrated into a character's suit, ammunition displayed on a weapon's holographic sight, or the dashboard instruments in the cockpit of a racing game. A Diegetic UI can enhance immersion by making interface elements feel like natural parts of the game world.
- **Non-diegetic UI**: These elements exist outside the game world, overlaid on top of the gameplay. Traditional menus, health bars in the corner of the screen, and score displays are common examples. While less immersive than a diegetic UI, non-diegetic elements are often clearer and easier to read.

UI Layout Systems

When designing and implementing game UI systems, developers must decide how UI elements will be positioned on the screen. Two primary approaches exist, each with distinct advantages and trade-offs:

1. **Absolute Positioning**: In this approach, each UI element is placed at specific coordinates on the screen. Elements are positioned using exact locations, which gives precise control over the layout. This approach is straightforward to implement and works well for static layouts where elements do not need to adjust based on screen size or content changes. The main disadvantage of absolute positioning is its lack of flexibility, as iterating on design can be more difficult since one change may have cascading effects on other elements. If the screen resolution changes or if an element's size changes, manual adjustments to positions are often necessary to maintain the desired layout.

2. **Layout engines**: These system position UI elements relative to one another using rules and constraints. Elements might be positioned using concepts like "center", "align to parent", or "flow horizontally with spacing". Layout engines add complexity but provide flexibility. The advantage of layout engines is adaptability to different screen sizes and content changes. However, they require more initial setup and can be more complex to implement from scratch.

Parent-Child Relationships

Parent-child relationships are a part of many UI system. This relationship is implemented with UI elements containing other UI elements, creating a tree-like structure. This hierarchial approach mirrors how interface elements naturally group together in designs.

For example, a settings panel might contain multiple buttons, labels, and sliders. By making these elements children of the panel, they can be managed as a cohesive unit. This organizational structure provides several significant advantages:

- **Inheritance of Properties**: Child elements can automatically inherit certain properties from their parents. For instance, if a parent element is hidden or disabled, all its children can be hidden or disabled as well. This cascading behavior simplifies state management across complex interfaces.
- **Relative Positioning**: Child elements can be positioned relative to their parents rather than relative to the screen. This means you can place elements within a container and then move the entire container as a unit without having to update each child's position individually.
- **Simplified State Management**: Actions on parent elements can automatically propagate to their children. For example, disabling a menu panel can automatically disable all buttons within it, preventing interaction with elements that should not be active.
- **Batch Operations**: Operations like drawing and updating can be performed on a parent element and automatically cascade to all children, reducing the need for repetitive code.
- **Logical Grouping**: The hierarchy naturally models the conceptual grouping of UI elements, making the code structure more intuitive and easier to maintain.

Anchoring and Docking

In UI systems, two important concepts help with positioning elements: anchoring and docking.

Anchoring

Anchoring allows you to position UI elements relative to specific reference points on their parents. The following diagram demonstrates common anchor points:

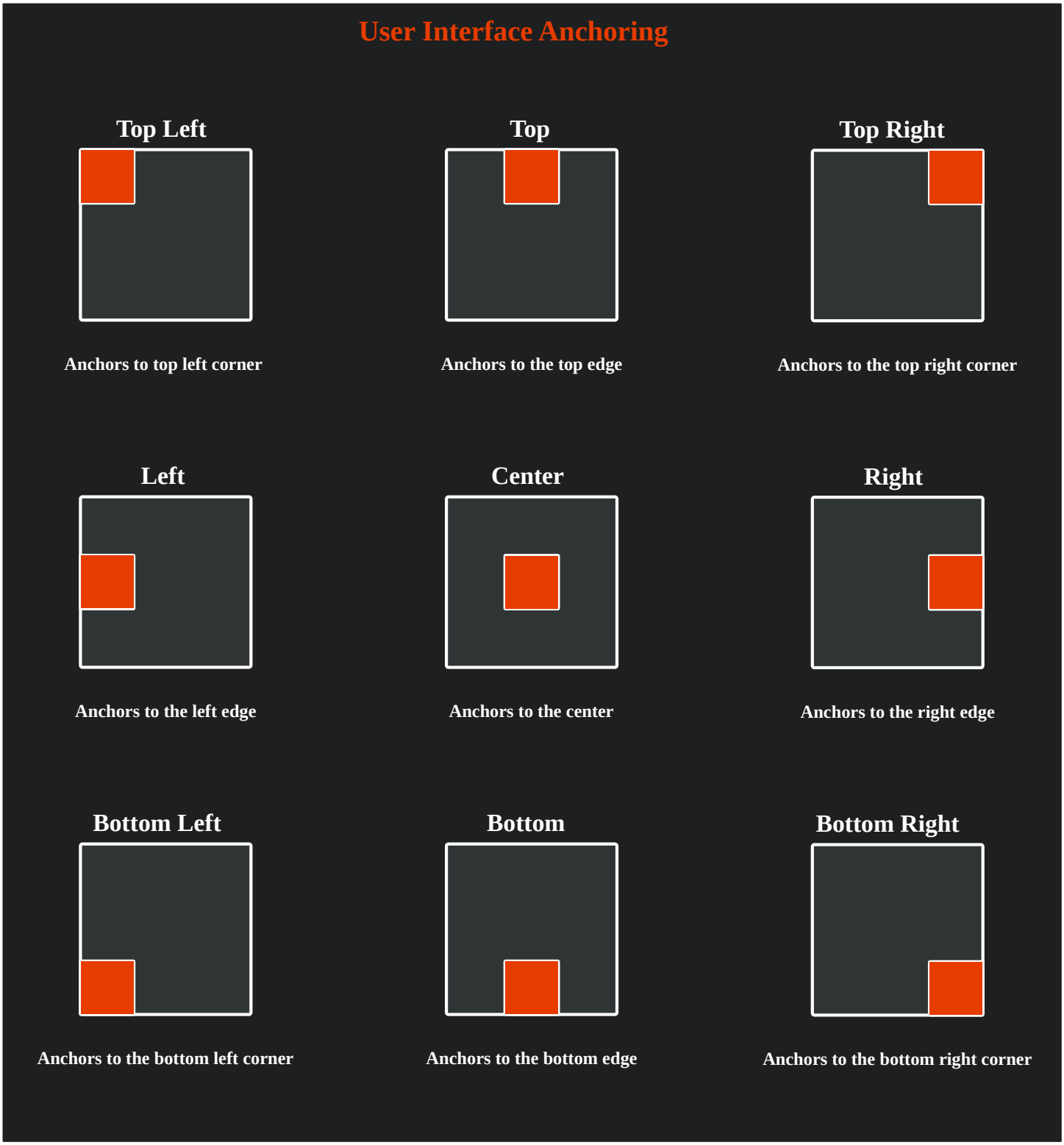


Figure 19-1: Diagram showing common anchor points

When you set an anchor point, the elements' position coordinates become relative to that anchor point. For example with a "Right" anchor and an X value of *-5*, you element would position itself *5 pixels* to the left of the parent's right edge, creating a consistent margin regardless of the parent's size, as demonstrated in the following diagram:

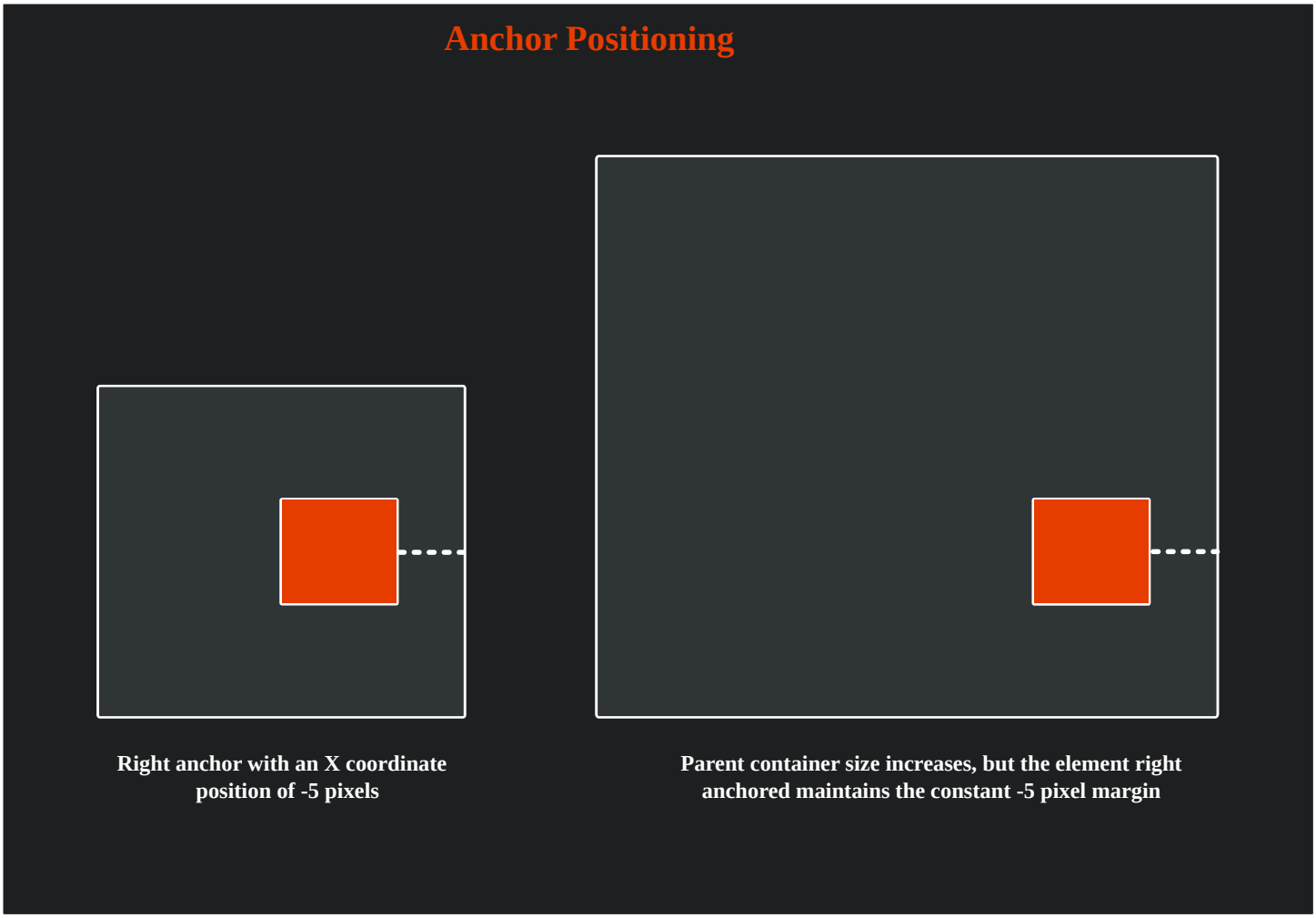


Figure 19-2: Diagram showing anchored element positioning relative to anchor regardless of parent size

Docking

Docking takes anchoring a step further by also adjusting an element's size to fill available space. The following diagram demonstrates common docking options:

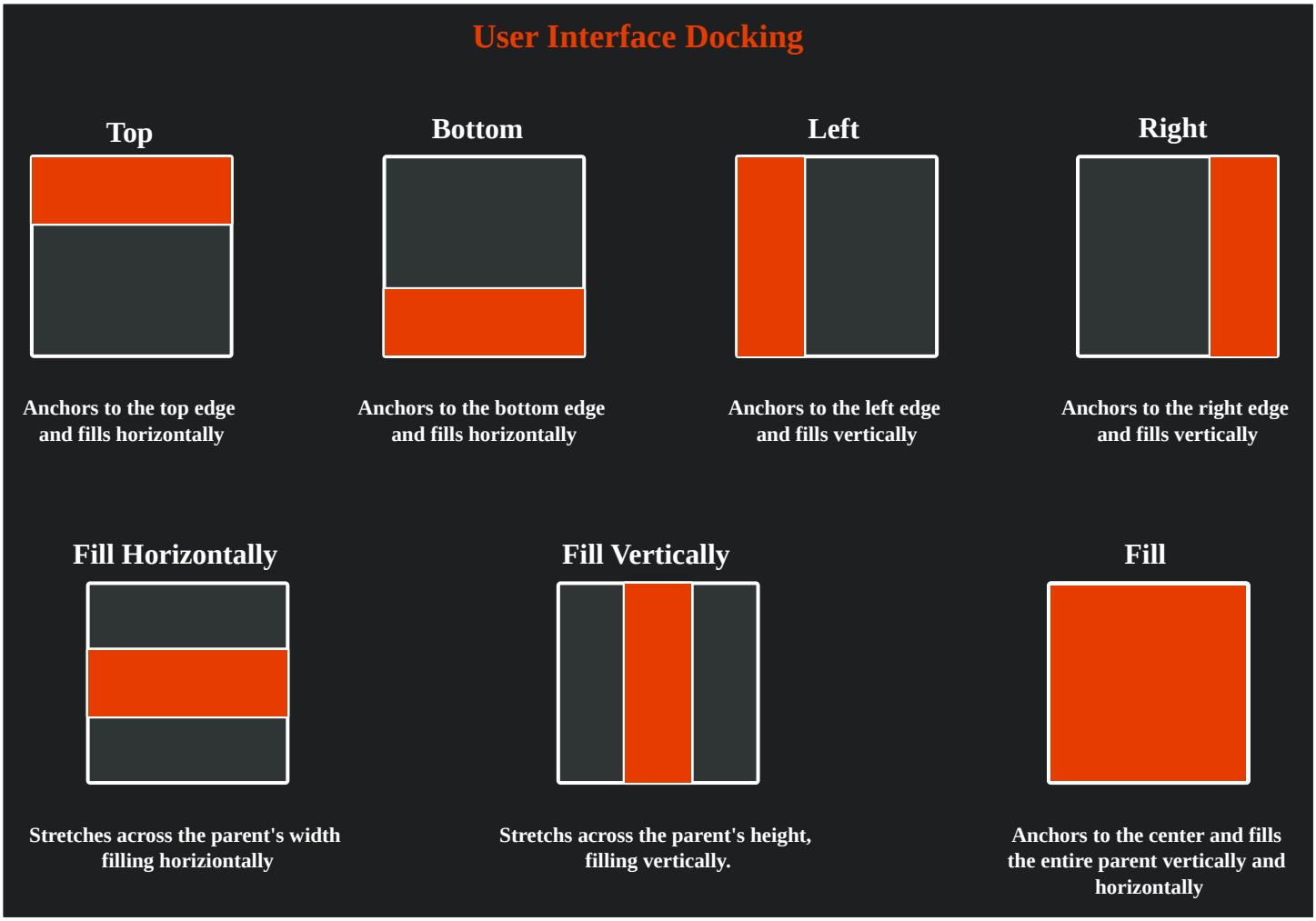


Figure 19-3: Diagram showing the common docking options

These positioning strategies enable you to create layouts that can maintain their visual relationships even when parent elements change size or position.

Accessibility in Game UI

Creating accessible user interfaces is an essential aspect of inclusive game design. Accessibility ensures that your game can be played by a broader audience, including players with visual acuity or other specific needs. When designing your UI system, consider some of these key accessibility principles:

Visual Accessibility

- **Color contrast**: Ensure sufficient contrast between text and backgrounds.
- **Use shapes**: Do not rely solely on color to convey important information; add shapes, patterns, or text labels as well. For example, if displaying warning text, also use something such as the common warning sign ⚠️.
- **Text size and scaling**: Allow players to adjust text size or implement a UI scaling option.
- **Internationalization (i18n)**: Consider how your UI might be interpreted across different cultures and regions. Number formatting can vary significantly - some regions use periods for thousands separators (1.000.000) while others use commas (1,000,000). Control symbolism also differs culturally; for example, on console controllers, the Cross button typically means "Select" in Western regions but "Cancel" in Japan, with Circle having the opposite meaning.

Input Accessibility

- **Input redundancy**: Support multiple input methods for the same action. This ensures players can interact with UI elements using their preferred input devices.
- **Reduce input precision requirements**: Implement generous hitboxes for clickable UI elements to help players with motor control difficulties.

Testing for Accessibility

The most effect way to ensure accessibility is through testing under different circumstances and with diverse users:

- Test your UI using only keyboard navigation.
- Try playing without sound.
- Check your UI with a color blindness simulator.
- Adjust the display scale to simulate low vision.
- Get feedback from player with different abilities.

By considering accessibility early in development rather than as an afterthought, you create games that can be enjoyed by more players while also often improving the experience for everyone.

Conclusion

In this chapter, you learned the fundamentals of user interface design for games. You explored the different types of UI elements and their purposes, understood the benefits of parent-child relationships in UI hierarchies, and learned about different positioning strategies like anchoring and docking.

You also discovered the importance of accessible UI design and how to implement practices that make your game playable for a wider audience. These foundational concepts will serve as the building blocks for implementing the UI in our game.

In the next chapter, we will put these concepts into practice by implementing a UI system using Gum, a specialized UI framework that will help us create interactive menus, buttons, and other UI elements for our game.

Test Your Knowledge

1. What are the two main types of game user interfaces, and how do they differ?

The two main types are:

- **Diegetic UI**: Elements that exist within the game world itself and are part of the narrative (like health meters integrated into a character's suit or cockpit instruments in racing games). These enhance immersion by making UI feel like a natural part of the game world.
- **Non-diegetic UI**: Elements that exist outside the game world, overlaid on top of gameplay (like traditional menus, health bars in screen corners, score displays). While less immersive, they are often clearer and easier to read.

2. What are some advantages of using a parent-child relationship in UI systems?

- **Inheritance of properties**: visual states cascade parent to children.
- **Relative positioning**: Child elements are positioned relative to their parents.
- **Simplified state management**: Parent states affect children automatically.
- **Batch operations**: Update and draw calls propagate through the hierarchy.
- **Logical grouping**: Mirrors the conceptual organization of UI elements.

3. How do anchoring and docking differ in UI layout systems?

- **Anchoring**: Positions UI elements relative to specific reference points on their parent (like TopLeft, Center, BottomRight) without changing the element's size. An element's position coordinates become relative to the chosen anchor point.
- **Docking**: Takes anchoring further by also adjusting an element's size to fill available space. For example, docking to the top means the element fills the parent horizontally while staying at the top, while "Fill" docking means the element expands to fill the entire parent area.

4. What are some accessibility considerations that should be implemented in game UI systems?

- **Visual accessibility**: High contrast colors, not relying solely on color for information, adjustable text size and UI scaling, and internationalization support.
- **Input accessibility**: Support for multiple input methods and reduced precision requirements.
- **Testing practices**: Ensure the UI works with keyboard only navigation, without sound, and with simulated visual impairments.

Chapter 20: Implementing UI with Gum

Learn how to integrate and use the Gum UI framework to create functional menus, buttons, and sliders for your MonoGame projects.

In the previous chapter we explored the fundamental concepts of user interface design. Now we are ready to put these principles into practice by implementing a UI system for our game. While it is possible to build a UI system from scratch, we will take advantage of Gum, a specialized UI framework that simplifies many of the complex aspects of UI implementation.

In this chapter you will:

- Install and configure the Gum NuGet package.
- Learn about Gum's core concepts including Forms and Visuals
- Implement UI elements for our game's title scene.
- Create a pause menu for the gameplay scene.
- Handle input from keyboard, mouse, and gamepads
- Integrate the UI system with our existing game architecture.

> **IMPORTANT**
>
> While GUM is used in this tutorial it is only one of many UI libraries available to the MonoGame community, some notable others are EmptyKeys, GeonBit.UI, as well as entire Game Frameworks/Engines like Nez that have their own built in UI systems.
>
> Check out the MonoGame Resources page, as well as awesome-monogame from Alois Deniel for even more community offerings.

What is Gum?

Gum is a powerful UI layout engine and framework. It provides a flexible, efficient system capable of producing virtually any UI layout you might need in your games. While originally developed alongside the FlatRedBall game engine, Gum has evolved to work seamlessly with multiple platforms, including MonoGame, which we will be using in this tutorial.

Why Use Gum?

Creating a UI system from scratch requires solving many complex problems:

1. **Layout Management**: Calculating positions for elements that need to adapt to different screen sizes or content changes.
2. **Input Handling**: Detecting and responding to mouse, keyboard, and gamepad inputs across multiple UI elements.
3. **Visual State Management**: Changing appearances based on user interactions (hovering, focusing, clicking).
4. **Component Hierarchy**: Managing parent-child relationships between elements.

Gum addresses these challenges with ready-made solutions, allowing us to focus on the specific needs of our game rather than reinventing the UI wheel. While MonoGame provides the basic tools for drawing graphics and detecting input, it does not include high-level UI abstractions; this is where tools like Gum fill the gap.

> **IMPORTANT**
>
> This tutorial uses the Gum NuGet package to help with layout and responding to user interactions. This tutorial does not require the use of the Gum editor, we will be doing everything in code.
>
> Keep in mind that while it is possible to build a full UI system without any external dependencies, creating a layout engine is complicated and beyond the scope of this tutorial. Instead, we will be taking advantage of the Gum NuGet package.
>
> Gum is a powerful system enabling the creation of virtually any game UI, and we will be covering some of the basics of its use in this tutorial. The full Gum documentation can be found here: https://docs.flatredball.com/gum/code/about

Gum Concepts

Before we dive into implementation, we will explore the core concepts that Gum provides. Gum simplifies UI development by providing ready-made controls and layout systems that would otherwise require significant effort to build from scratch.

Understanding Gum's Structure

Gum organizes UI elements in a hierarchical tree structure, similar to how HTML organizes web elements or how GUI frameworks like WPF or JavaFX structure their interfaces. This hierarchy consists of:

1. **The Root Element**: The topmost container in the hierarchy that serves as the entry point for all UI elements.
2. **Containers**: Elements that can hold other elements (like panels or screens).
3. **Controls**: Interactive elements that respond to user input (like buttons or sliders).
4. **Visuals**: The actual graphical representations of UI elements (like text, images, or shapes).

When a game using Gum runs, this hierarchy is maintained in memory, with each element knowing its parent and children. The framework automatically handles the flow of events through this hierarchy and manages the drawing of elements according to their positions in the tree.

Gum Root Element

All Gum elements must be directly or indirectly added to Gum's root container. This can be done directly wth the `AddToRoot()` method:

```
// Creating a panel and adding it to the root
Panel mainMenuPanel = new Panel();
mainMenuPanel.AddToRoot();
```

Or it can be done indirectly by adding a control as a child of an element that has been added to Gum's root container:

```
// Creating a panel and adding it to the root
Panel mainMenuPanel = new Panel();
mainMenuPanel.AddToRoot();

// Creating a button and adding it as a child element of the panel
// which indirectly connects it to Gum's root container
Button startButton = new Button();
mainMenuPanel.AddChild(startButton);
```

Gum's root element can also be cleared at any time to remove all UI elements:

```
// Clear all children from Gum's root container.
GumService.Default.Root.Children.Clear();
```

This can be useful when navigating between different scenes to ensure UI elements do not persist from previous scenes:

```
public class GameScene
{
    public override void Initialize()
    {
        // Clear all children from Gum's root container that may have been added
        // during the previous scene
        GumService.Default.Root.Children.Clear();

        // Now that it has been cleared, initialize the UI for this scene
        InitializeUI();
    }
}
```

Anchoring

In the previous chapter we discussed anchoring, a fundamental UI concept that allows you to position elements relative to specific reference points of their parents. Gum supports anchoring of Forms controls through the `Anchor()` method.

```
// Creating a panel and adding it to the root
Panel mainMenuPanel = new Panel();
mainMenuPanel.AddToRoot();

// Creating a button and adding it as a child of the panel
// anchored ot the bottom-left of the panel
Button startButton = new Button();
startButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);
mainMenuPanel.AddChild(startButton);
```

The following anchor types are supported by Gum:

Anchor	Gum Value
TopLeft	`Gum.Wireframe.Anchor.TopLeft`
Top	`Gum.Wireframe.Anchor.Top`
TopRight	`Gum.Wireframe.Anchor.TopRight`
Left	`Gum.Wireframe.Anchor.Left`
Center	`Gum.Wireframe.Anchor.Center`
Right	`Gum.Wireframe.Anchor.Right`
BottomLeft	`Gum.Wireframe.Anchor.BottomLeft`
Bottom	`Gum.Wireframe.Anchor.Bottom`
BottomRight	`Gum.Wireframe.Anchor.BottomRight`

Docking

In the previous chapter, we also discussed docking, a fundamental UI concept that adjusts an element's size to fill the available space. Gum supports the docking of Forms controls through their `Dock()` method.

```
// Creating a panel and adding it to the root
Panel mainMenuPanel = new Panel();
mainMenuPanel.AddToRoot();

// Docking the panel to fill the entire root space
mainMenuPanel.Dock(Gum.Wireframe.Dock.Fill);
```

The following docking modes are supported by Gum:

Anchor	Gum Value	Description
Top	`Gum.Wireframe.Dock.Top`	Anchors to the top edge and fills horizontally.
Left	`Gum.Wireframe.Dock.Left`	Anchors to the left edge and fills vertically.
Right	`Gum.Wireframe.Dock.Right`	Anchors to the ridge edge and fills vertically.
Bottom	`Gum.Wireframe.Dock.Bottom`	Anchors to the bottom edge and fills horizontally.
Fill	`Gum.Wireframe.Dock.Fill`	Anchors to the center and fills the entire parent area vertically and horizontally.
FillHorizontally	`Gum.Wireframe.Dock.FillHorizontally`	Stretches across the parent's width, filling horizontally.
FillVertically	`Gum.Wireframe.Dock.FillVertically`	Stretches across the parent's height, filling vertically.
SizeToChildren	`Gum.Wireframe.Dock.SizeToChildren`	Automatically sizes vertically and horizontally based on contained child element.

Forms and Visuals

Two concepts are important when working with Gum: **Forms** and **Visuals**.

- **Forms** controls are typical interactive UI elements such as buttons, sliders, and text boxes that handle user interaction through mouse, gamepad, and keyboard inputs. These controls come with built-in functionality; a button responds visually when focused, while a slider changes its value when clicked on its *track*. By using these standardized components, you can maintain consistency throughout your UI implementation.
- **Visuals** are the graphical display elements that render the actual appearance of UI components. In Gum, Visual elements have the *Runtime suffix (like TextRuntime, ColoredRectangleRuntime, and NineSliceRuntime) and are used to render graphics. They handle the visual representation but have no built-in interaction behavior on their own.

Forms controls provide customization through their `Visual` property, which serves as a gateway to modifying their appearance and layout. With this property, you can move, resize, restyle, and even completely replace visuals through code. As we will see when building our UI in the next chapter, this separation between functionality and presentation allows us to create consistent behaviors while adapting the visual style to match our game's aesthetic.

For now, we will examine some of the Forms control types we will use in this chapter.

Panel

Panels serve as invisible containers that group related UI elements together. Unlike visible elements that display graphics, panels focus on organization and layout management.

A panel provides several key functions:

- Groups related elements for easier management.
- Controls visibility for entire sections of UI at once.
- Establishes a coordinate system for child elements.
- Provides a foundation for layout management.

Panels are especially useful for creating distinct UI screens, by toggling the visibility of different panels you can implement complete UI state changes with minimal code:

```
// Change the state of the UI by hiding one panel and showing another.
mainMenuPanel.IsVisible = false;
optionsPanel.IsVisible = true;
```

A common pattern is to set a panel's docking to `Fill`, which makes it span the entire available area:

```
// Make the panel fill the entire screen
mainMenuPanel.Dock(Gum.Wireframe.Dock.Fill);
```

This creates a consistent coordinate space for all child elements, allowing them to be positioned relative to the screen.

Button

The `Button` Forms control type is the primary interactive control for triggering actions in your UI.

Buttons provide:

- Responses to clicks from mouse, touch, keyboard, or gamepad input.
- Visual feedback when focused or hovered.
- Raises a `Click` event when activated.

Buttons can be positioned using anchoring to create layouts that adapt to different screen sizes:

```
// Creating a button that is anchored to the bottom left.
Button startButton = new Button;
startButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);

// Set the X and Y position so it is 20px from the left edge
// and 20px from the bottom edge.
startButton.X = 20;
startButton.Y = -20;
```

The `Click` event is raised whenever the button is activated and provides a standard way to respond regardless of input device:

```
startButton.Click += (sender, args) =>
{
    // Handle button click...
    StartGame();
};
```

> **NOTE**
>
> The `Click` event for a button is triggered automatically when it is left-clicked by a mouse or pushed via touch controls. For keyboard and gamepad input, it can only be triggered when the button has focus, which typically happens when the player navigates to it using the tab key or controller.

Slider

The `Slider` Forms control type allows users to select a numeric value from a continuous range. A slider:

- Displays and modifies a `Value` property constrained between a `Minimum` and `Maximum` value.
- Responds to mouse clicks on its track or by dragging its thumb.
- Supports keyboard and gamepad input for incremental adjustments.
- Raises events when its value changes.

Basic slider setup includes defining its range and establishing event handlers:

```
Slider volumeSlider = new Slider();
volumeSlider.Minimum = 0.0f;
volumeSlider.Maximum = 1.0f;
volumeSlider.Value = 0.5f;
volumeSlider.SmallChange = 0.1f;
volumeSlider.LargeChange = 0.2f;
```

The `SmallChange` property sets the increment for keyboard and gamepad adjustments, while the `LargeChange` property determines the increment when clicking directly on the slider *track*.

Sliders provide several events for different interaction scenarios:

- `ValueChanged`: Fires continuously as the value changes (useful for live previews).
- `ValueChangeCompleted`: Fires once when the user finishes adjusting the value (useful for applying final settings).

```
volumeSlider.ValueChanged += (sender, arg) =>
{
    // Handle value changed event...
    UpdateVolume(volumeSlider.Value);
};

volumeSlider.ValueChangedCompleted += (sender, arg) =>
{
    // Handle value change completed event...
    UpdateVolume(volumeSlider.Value);

    // Useful to do things like this here since this fires once
    // the slider value change has completed so it is not constantly
    // triggering ui sound effects.
    PlayUISoundEffect();
};
```

Property Changes vs States

Gum allows you to customize visuals in two ways:

- Direct property assignment
- Using states.

With simple property changes, you can directly assign values in code. For example, the following code example changes the width of a button:

```
startButton.Width = 100;
```

Direct property assignment works well for initial setup, such as positioning elements or setting their dimensions when first creating your UI. However, when you need visual elements to respond to user interactions (like highlighting a button when it is focused), a different approach is required.

For these dynamic changes, Gum uses a system of **states** (implemented as `StateSave` objects), each Forms control maintains a collection of named states that are automatically applied in response to specific user interactions. When a button becomes focused, for instance, Gum looks for and applies a state named "Focused" to alter its appearance.

> **NOTE**
> In the next chapter during the customization pass, we will create states to visually indicate when controls are focused, providing clear feedback to the player.

Updating Our Game To Use Gum

Now that we have covered the core UI concepts and how Gum will help implement them, we can integrate Gum into our game project. We will add the framework, initialize it, and prepare it for use in our scenes.

For now we will use the default styling in Gum to quickly iterate and build the UI and do a customization styling pass in the next chapter.

Adding the Gum NuGet Package

Before we can use Gum in our project, we first need to add it using NuGet. NuGet is a package manager for .NET projects that allows you to add third-party libraries into your project, similar to how we created and added our own class library.

To add the Gum NuGet package to our game project, follow the instructions below based on your development environment:

Visual Studio Code

To add the Gum NuGet package in Visual Studio Code:

1. In the *Solution Explorer* panel, right-click the `DungeonSlime` project.
2. Choose `Add NuGet Package` from the context menu.
3. Enter `Gum.MonoGame` in the `Add NuGet Package` search prompt and press Enter.
4. When the search finishes, select the `Gum.MonoGame` package in the results
5. When prompted for a version choose version `2025.12.9.1`.

To Add the Gum NuGet package in Visual Studio 2022:

1. In the *Solution Explorer* panel, right-click the *DungeonSlime* project.
2. Choose `Manage Nuget Packages...` from the context menu.
3. In the NuGet Package Manager window, select the `Browse` tab if it is not already selected.
4. In the search box, enter `Gum.MonoGame`.
5. Select the "Gum.MonoGame" package from the search results.
6. On the right, in the version dropdown, select version `2025.12.9.1` and click the "Install" button.

To add the Gum NuGet package using the dotnet CLI:

1. Open a Command Prompt or Terminal window in the same folder as the `DungeonSlime.csproj` project file.
2. Enter the following command:

```
dotnet add DungeonSlime.csproj package Gum.MonoGame --version 2025.12.9.1
```

TIP

You can verify the package was successfully added by examining your `DungeonSlime.csproj` file, which should now contain a reference like:

```
<PackageReference Include="Gum.MonoGame" Version="2025.12.9.1" />
```

> **IMPORTANT**
>
> This tutorial uses version `2025.12.9.1` of Gum, which is the latest version of Gum as of this writing. That exact version is specified to use in the section above when installing the NuGet package to ensure compatibility throughout this tutorial. If there are newer versions of Gum available, please consult the Gum documentation before updating in case there are any breaking changes from the code that is presented in this tutorial.

Adding UI Sound Effect

To make our UI more responsive and engaging, we will add audio feedback that plays when players interact with buttons and other UI elements. Sound effects provide immediate confirmation that an input has been recognized, creating a more engaging experience.

First, download the UI sound effect by right-clicking the following link and saving it as `ui.wav` in the game project's `Content/audio` folder:

- ui.wav

Next, add this sound effect to your content project using the MGCB Editor:

1. Open the `Content.mgcb` content project file in the MGCB Editor.
2. Right-click the `audio` folder and choose `Add > Existing Item...`.
3. Navigate to and select the `ui.wav` file you just downloaded.
4. In the Properties panel, verify that the `Processor` is set to `Sound Effect`.
5. Save the changes and close the MGCB Editor.

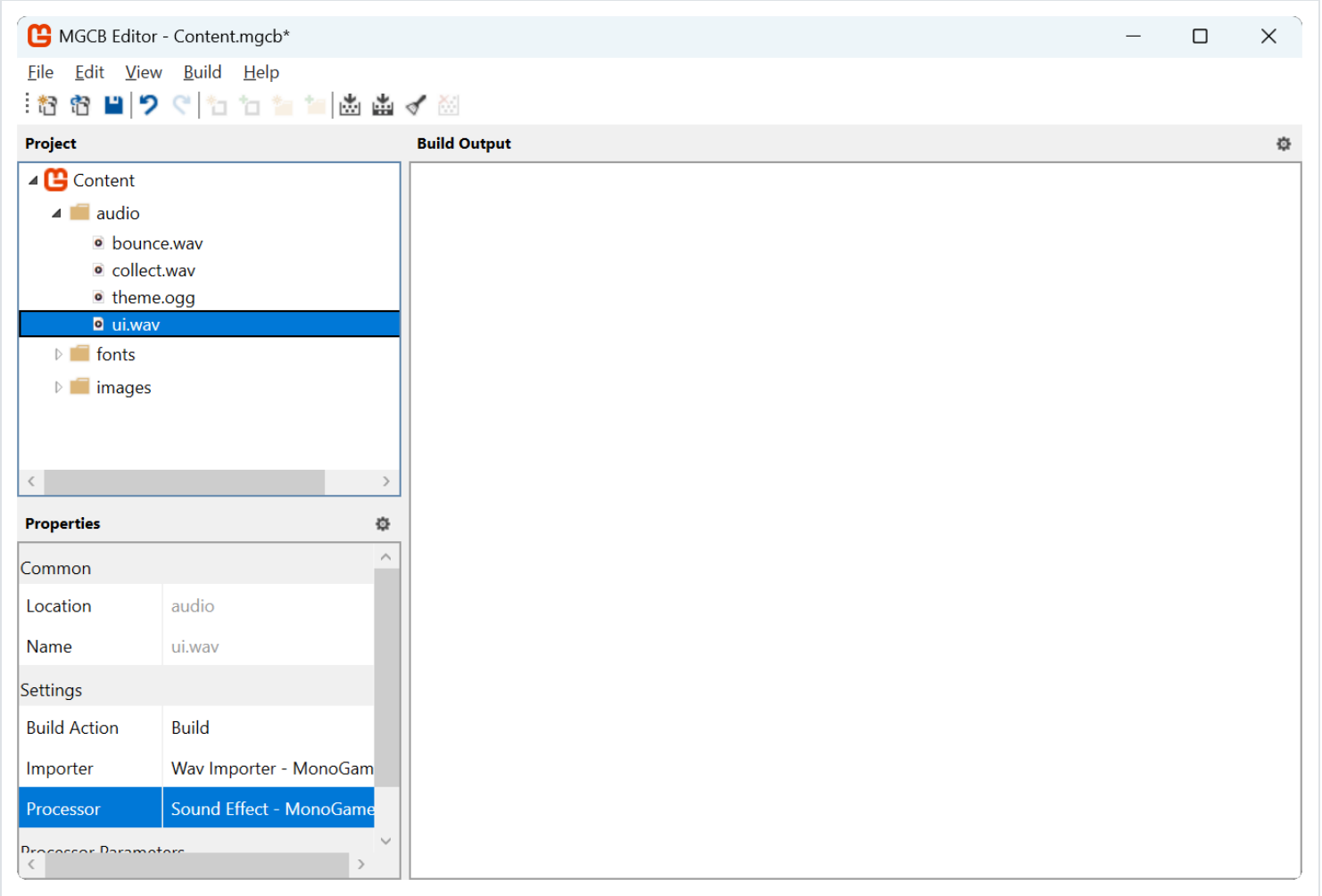


Figure 20-1: The MGCB Editor with ui.wav added to the audio folder

We will load and use this sound effect in our UI implementation to provide auditory feedback when players interact with buttons and sliders.

Initializing Gum

With the Gum NuGet package added to our project, we need to initialize Gum in our game, this will enable the UI system and configure input handling for our controls. Since this is an initialization that only needs to happen once, we can make the necessary changes to the `Game1` class.

First, open the `Game1.cs` file and add the following new using statements to the top:

```
using DungeonSlime.Scenes;
using Gum.Forms;
using Gum.Forms.Controls;
using MonoGameLibrary;
using MonoGameGum;
using Microsoft.Xna.Framework.Media;
```

Next, add the following method to the `Game1` class to encapsulate the initializations of the Gum UI service:

```
private void InitializeGum()
{
    // Initialize the Gum service. The second parameter specifies
    // the version of the default visuals to use. V2 is the latest
    // version.
    GumService.Default.Initialize(this, DefaultVisualsVersion.V3);

    // Tell the Gum service which content manager to use.  We will tell it to
    // use the global content manager from our Core.
    GumService.Default.ContentLoader.XnaContentManager = Core.Content;

    // Register keyboard input for UI control.
    FrameworkElement.KeyboardsForUiControl.Add(GumService.Default.Keyboard);

    // Register gamepad input for Ui control.
    FrameworkElement.GamePadsForUiControl.AddRange(GumService.Default.Gamepads);

    // Customize the tab reverse UI navigation to also trigger when the keyboard
    // Up arrow key is pushed.
    FrameworkElement.TabReverseKeyCombos.Add(
       new KeyCombo() { PushedKey = Microsoft.Xna.Framework.Input.Keys.Up });

    // Customize the tab UI navigation to also trigger when the keyboard
    // Down arrow key is pushed.
    FrameworkElement.TabKeyCombos.Add(
       new KeyCombo() { PushedKey = Microsoft.Xna.Framework.Input.Keys.Down });

    // The assets created for the UI were done so at 1/4th the size to keep the size
of the
    // texture atlas small.  So we will set the default canvas size to be 1/4th the
size of
    // the game's resolution then tell gum to zoom in by a factor of 4.
    GumService.Default.CanvasWidth =
GraphicsDevice.PresentationParameters.BackBufferWidth / 4.0f;
    GumService.Default.CanvasHeight =
GraphicsDevice.PresentationParameters.BackBufferHeight / 4.0f;
    GumService.Default.Renderer.Camera.Zoom = 4.0f;
}
```

Finally, update the **Initialize** method to call the `InitializeGum` method we just created:

```
protected override void Initialize()
{
   base.Initialize();

   // Start playing the background music
   Audio.PlaySong(_themeSong);

   // Initialize the Gum UI service
   InitializeGum();

   // Start the game with the title scene.
   ChangeScene(new TitleScene());
}
```

The following is a breakdown of this initialization process:

1. **Basic Initialization**: `GumService.Default.Initialize(this, DefaultVisualsVersion.V3)` sets up the Gum system with our game instance. This is required for any gum project. The second parameter specifies the default visual styling. V3 is the latest version which makes it easy to style the default controls.

> **NOTE**
>
> We only need to pass our **Game** instance and the visuals version since we are using Gum as a code-first approach. Gum also offers a visual editor that creates Gum project files. When using the editor, you will need to also pass the Gum Project file to `Initialize`. For more information on how to use the Gum visual editor, see the Gum Project Forms Tutorial.

2. **Content Loading**: Gum needs to be made aware of which content manager to use to load assets through the content pipeline. By setting `GumService.Default.ContentLoader.XnaContentManager = Core.Content`, we tell Gum to use our game's content manager when loading assets. By using the game's existing content manager, Gum also gets the benefit of the caching that the content manager performs when loading assets.

3. **Input Configuration**:

    - By default, all Forms controls automatically respond to mouse and touch screen input devices. We need to explicitly register keyboard and gamepad input devices by using th `FrameworkElement.KeyboardsForUiControl` and `Framework.GamePadsForUiControl` properties.
    - By default, Forms controls will automatically respond to tab and shift-tab for navigation. By using the `FrameworkElement.TabKeyCombos` and `FrameworkElement.TabReverseKeyCombos` properties, we can add additional key combinations for tabbing. Here we map the Up arrow for reverse tabbing and the Down arrow for forward tabbing.

> **TIP**
>
> If you prefer different navigation keys, you can remove the built-in Tab/Shift+Tab navigation.
>
> Simply call these methods before adding your custom combinations:

```
FrameworkElement.TabKeyCombos.Clear();
FrameworkElement.TabReverseKeyCombos.Clear();
```

4. **UI Scaling**: Gum allows us to independently scale the UI regardless of the resolution of the game. Our game is set to a resolution of 1280x720, however as we will see during the styling section later, the UI assets created were done at one-fourth the size to reduce the size of the texture atlas. Here, we use the `GumService.Default.CanvasWidth` and `GumService.Default.CanvasHeight` properties to set the canvas size to one-fourth that of our game's resolution. Then using by setting the `GumService.Default.Renderer.Camera.Zoom` property to four, we effectively make it render the UI at full resolution.

Gum is now fully initialized and we can use it in our scenes to add UI to our game.

Adding TitleScene UI

With Gum added and initialized in our game, we can now implement UI elements for our title scene. We will create panels for both the main menu and options menu, implement the necessary event handlers, and integrate everything with our existing title scene.

> **NOTE**
> When adding these sections one by one, you may see compiler errors until all sections are in place. This is normal, as some parts of the code will reference fields or methods that haven't been added yet. Once all sections are complete, these errors will resolve.

First, open the *TitleScene.cs* file in the game project and add the following using declarations to the top of the `TitleScene` class:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameGum;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary;
using MonoGameLibrary.Scenes;
```

Next, add the following fields to the `TitleScene` class:

```
private SoundEffect _uiSoundEffect;
private Panel _titleScreenButtonsPanel;
private Panel _optionsPanel;
private Button _optionsButton;
private Button _optionsBackButton;
```

Creating the Title Panel

First, create a new method that builds our main menu panel with start and options buttons.

Add the following method to the `TitleScene` class:

```
private void CreateTitlePanel()
{
    // Create a container to hold all of our buttons
    _titleScreenButtonsPanel = new Panel();
    _titleScreenButtonsPanel.Dock(Gum.Wireframe.Dock.Fill);
    _titleScreenButtonsPanel.AddToRoot();

    var startButton = new Button();
    startButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);
    startButton.X = 50;
    startButton.Y = -12;
    startButton.Width = 70;
    startButton.Text = "Start";
    startButton.Click += HandleStartClicked;
    _titleScreenButtonsPanel.AddChild(startButton);

    _optionsButton = new Button();
    _optionsButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
    _optionsButton.X = -50;
    _optionsButton.Y = -12;
    _optionsButton.Width = 70;
    _optionsButton.Text = "Options";
    _optionsButton.Click += HandleOptionsClicked;
    _titleScreenButtonsPanel.AddChild(_optionsButton);

    startButton.IsFocused = true;
}
```

Our title panel includes two buttons positioned at the bottom corners of the screen. The "Start" button will allow players to begin the game while the "Options" button will hide the main menu and display the options menu.

> **NOTE**
>
> Notice how we use `Anchor` to position the buttons relative to the panel's edges, with the "Start" button anchored at the bottom left and the "Options" button anchored at the bottom right. Then the positioning of the elements is adjusted relative to its anchor point.

Each button registers a `Click` event handler to respond when the players selects it, we should implement the event handler method for these buttons next. First we will implement the handler for the "Start" button. Add the following method to the `TitleScene` class after the `CreateTitlePanel` method:

```
private void HandleStartClicked(object sender, EventArgs e)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Change to the game scene to start the game.
    Core.ChangeScene(new GameScene());
}
```

When the "Start" button is clicked and this method is called, it will play the UI sound effect for auditory feedback then change the scene to the game scene so the player can start playing the game.

Next is the handler for the "Options" button. Add the following method to the `TitleScene` class after the `HandleStartClicked` method:

```
private void HandleOptionsClicked(object sender, EventArgs e)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Set the title panel to be invisible.
    _titleScreenButtonsPanel.IsVisible = false;

    // Set the options panel to be visible.
    _optionsPanel.IsVisible = true;

    // Give the back button on the options panel focus.
    _optionsBackButton.IsFocused = true;
}
```

When the "Options" button is clicked and this method is called, it will play the UI sound effect for auditory feedback then hide the title panel and show the options panel.

Creating the Options Panel

Next, we will create the options panel with sliders to adjust the volume for music and sound effects.

Add the following method to the `TitleScene` class:

```
private void CreateOptionsPanel() {
    _optionsPanel = new Panel();
    _optionsPanel.Dock(Gum.Wireframe.Dock.Fill);
    _optionsPanel.IsVisible = false;
    _optionsPanel.AddToRoot();

    var optionsText = new TextRuntime();
    optionsText.X = 10;
    optionsText.Y = 10;
    optionsText.Text = "OPTIONS";
    _optionsPanel.AddChild(optionsText);

    var musicLabel = new Label();
    musicLabel.Text = "Music";
    musicLabel.X = 35;
    musicLabel.Y = 35;
    optionsPanel.AddChild(musicLabel);

    var musicSlider = new Slider();
    musicSlider.Anchor(Gum.Wireframe.Anchor.Top);
    musicSlider.Y = 30f;
    musicSlider.Minimum = 0;
    musicSlider.Maximum = 1;
    musicSlider.Value = Core.Audio.SongVolume;
    musicSlider.SmallChange = .1;
    musicSlider.LargeChange = .2;
    musicSlider.ValueChanged += HandleMusicSliderValueChanged;
    musicSlider.ValueChangeCompleted += HandleMusicSliderValueChangeCompleted;
    _optionsPanel.AddChild(musicSlider);

    var sfxLabel = new Label();
    sfxLabel.Text = "SFX";
    sfxLabel.X = 20;
    sfxLabel.Y = 80;
    optionsPanel.AddChild(sfxLabel);

    var sfxSlider = new Slider();
    sfxSlider.Anchor(Gum.Wireframe.Anchor.Top);
    sfxSlider.Y = 93;
    sfxSlider.Minimum = 0;
    sfxSlider.Maximum = 1;
    sfxSlider.Value = Core.Audio.SoundEffectVolume;
    sfxSlider.SmallChange = .1;
    sfxSlider.LargeChange = .2;
    sfxSlider.ValueChanged += HandleSfxSliderChanged;
    sfxSlider.ValueChangeCompleted += HandleSfxSliderChangeCompleted;
```

```
    _optionsPanel.AddChild(sfxSlider);

    _optionsBackButton = new Button();
    _optionsBackButton.Text = "BACK";
    _optionsBackButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
    _optionsBackButton.X = -28f;
    _optionsBackButton.Y = -10f;
    _optionsBackButton.Click += HandleOptionsButtonBack;
    _optionsPanel.AddChild(_optionsBackButton);
}
```

This panel includes a text label, two sliders for adjusting audio volumes, and a back button for returning to the main menu. The panel is initially invisible since we start on the main menu. Both the "Music Volume" slider and the "Sound Effects Volume" slider register events to be called when the value of the sliders change and when the value change has been completed. The "Back" button registers a click event similar to the ones from the main menu.

Now we should implement the event handlers for these controls. First, we will implement the handler for when the value of the sound effect volume slider changes. Add the following method to the `TitleScene` class after the `CreateOptionsPanel` method:

```
private void HandleSfxSliderChanged(object sender, EventArgs args)
{
    // Intentionally not playing the UI sound effect here so that it is not
    // constantly triggered as the user adjusts the slider's thumb on the
    // track.

    // Get a reference to the sender as a Slider.
    var slider = (Slider)sender;

    // Set the global sound effect volume to the value of the slider.;
    Core.Audio.SoundEffectVolume = (float)slider.Value;
}
```

When the value of the "Sound Effects Volume" slider changes and this method is called, a reference to the slider is captured and then the the global sound effect volume is adjusted based on the value of the slider.

Next is the handler when the "Sound Effects Volume" slider has completed a value change. Add the following method to the `TitleScene` class after the `HandleSfxSliderChanged` method:

```
private void HandleSfxSliderChangeCompleted(object sender, EventArgs e)
{
    // Play the UI Sound effect so the player can hear the difference in audio.
    Core.Audio.PlaySoundEffect(_uiSoundEffect);
}
```

When the value of the "Sound Effects Volume" slider has completed a change and this method is called, it plays the UI sound effect to provide auditory feedback so the player can hear the difference in volume.

Next, add the handler for changes to the 'Music Volume' slider. Place the following method in the TitleScene class, directly after the HandleSfxSliderChangeCompleted method:

```
private void HandleMusicSliderValueChanged(object sender, EventArgs args)
{
    // Intentionally not playing the UI sound effect here so that it is not
    // constantly triggered as the user adjusts the slider's thumb on the
    // track.

    // Get a reference to the sender as a Slider.
    var slider = (Slider)sender;

    // Set the global song volume to the value of the slider.
    Core.Audio.SongVolume = (float)slider.Value;
}
```

Similar to how we handled the "Sound Effect Volume" slider value changes, when the "Music Volume" slider value changes and this method is called, a reference to the slider is captured and then the global music volume is adjusted based on the value of the slider.

Next is the handler when the "Music Volume" slider value has completed a value change. Add the following method to the `TitleScene` class after the `HandleMusicSliderValueChanged` method:

```
private void HandleMusicSliderValueChangeCompleted(object sender, EventArgs args)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);
}
```

When the value of the "Music Volume" slider has completed a change, the UI sound effect is played to provide auditory feedback.

Finally, we need to add the handler for when the "Back" button is clicked on the options panel. Add the following method to the `TitleScene` class after the `HandleMusicSliderValueChangeCompleted` method:

```
private void HandleOptionsButtonBack(object sender, EventArgs e)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Set the title panel to be visible.
    _titleScreenButtonsPanel.IsVisible = true;

    // Set the options panel to be invisible.
    _optionsPanel.IsVisible = false;

    // Give the options button on the title panel focus since we are coming
    // back from the options screen.
    _optionsButton.IsFocused = true;
}
```

This method plays the UI sound effect for auditory feedback, then hides the options panel and shows the title panel.

> **TIP**
>
> Notice that for both sliders, we registered a method for the `ValueChangeCompleted` event. This is so we can play the UI sound effect only when the player has finished adjusting the slider value. If we had instead played the UI sound effect in the `ValueChanged` event, then the UI sound effect would trigger constantly while the slider is being adjusted if using a mouse to drag it.

Initializing the UI

Now that we have implemented the methods that will create both the main menu panel and the options menu panel, we need to implement the main UI initializations method that will call them. Add the following method to the `TitleScene` class after the `HandleOptionsButtonBack` method:

```
private void InitializeUI()
{
    // Clear out any previous UI in case we came here from
    // a different screen:
    GumService.Default.Root.Children.Clear();

    CreateTitlePanel();
    CreateOptionsPanel();
}
```

This method first clears any existing UI elements from Gum's root container to prevent duplication, then calls our panel creation methods to build the complete interface.

Integrating with the Game Loop

Finally, we need to integrate our UI initialization, update, and draw with the scene's lifecycle. First, add the call to `InitializeUI()` in the `Initialize` method by updating it to the following:

```
public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // While on the title screen, we can enable exit on escape so the player
    // can close the game by pressing the escape key.
    Core.ExitOnEscape = true;

    // Set the position and origin for the Dungeon text.
    Vector2 size = _font5x.MeasureString(DUNGEON_TEXT);
    _dungeonTextPos = new Vector2(640, 100);
    _dungeonTextOrigin = size * 0.5f;

    // Set the position and origin for the Slime text.
    size = _font5x.MeasureString(SLIME_TEXT);
    _slimeTextPos = new Vector2(757, 207);
    _slimeTextOrigin = size * 0.5f;

    // Initialize the offset of the background pattern at zero
    _backgroundOffset = Vector2.Zero;

    // Set the background pattern destination rectangle to fill the entire
    // screen background
    _backgroundDestination = Core.GraphicsDevice.PresentationParameters.Bounds;

    InitializeUI();
}
```

Next, update the `LoadContent` method to load the sound effect that will be used as auditory feedback for the UI:

```
public override void LoadContent()
{
   // Load the font for the standard text.
   _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

   // Load the font for the title text
   _font5x = Content.Load<SpriteFont>("fonts/04B_30_5x");

   // Load the background pattern texture.
   _backgroundPattern = Content.Load<Texture2D>("images/background-pattern");

   // Load the sound effect to play when ui actions occur.
   _uiSoundEffect = Core.Content.Load<SoundEffect>("audio/ui");
}
```

Next modify the `Update` method to include Gum's update logic:

```
public override void Update(GameTime gameTime)
{
    // Update the offsets for the background pattern wrapping so that it
    // scrolls down and to the right.
    float offset = _scrollSpeed * (float)gameTime.ElapsedGameTime.TotalSeconds;
    _backgroundOffset.X -= offset;
    _backgroundOffset.Y -= offset;

    // Ensure that the offsets do not go beyond the texture bounds so it is
    // a seamless wrap
    _backgroundOffset.X %= _backgroundPattern.Width;
    _backgroundOffset.Y %= _backgroundPattern.Height;

    GumService.Default.Update(gameTime);
}
```

Finally, the `Draw` method needs to be updated to:

1. Only show the text for the game title when the title panel is visible
2. Add Gum's drawing call to draw the user interface

Update the `Draw` method to the following:

```
public override void Draw(GameTime gameTime)
{
    Core.GraphicsDevice.Clear(new Color(32, 40, 78, 255));

    // Draw the background pattern first using the PointWrap sampler state.
    Core.SpriteBatch.Begin(samplerState: SamplerState.PointWrap);
    Core.SpriteBatch.Draw(_backgroundPattern, _backgroundDestination, new
Rectangle(_backgroundOffset.ToPoint(), _backgroundDestination.Size), Color.White
* 0.5f);
    Core.SpriteBatch.End();

    if (_titleScreenButtonsPanel.IsVisible)
    {
        // Begin the sprite batch to prepare for rendering.
        Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

        // The color to use for the drop shadow text.
        Color dropShadowColor = Color.Black * 0.5f;

        // Draw the Dungeon text slightly offset from it is original position and
        // with a transparent color to give it a drop shadow
        Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos + new
Vector2(10, 10), dropShadowColor, 0.0f, _dungeonTextOrigin, 1.0f,
SpriteEffects.None, 1.0f);

        // Draw the Dungeon text on top of that at its original position
        Core.SpriteBatch.DrawString(_font5x, DUNGEON_TEXT, _dungeonTextPos,
Color.White, 0.0f, _dungeonTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

        // Draw the Slime text slightly offset from it is original position and
        // with a transparent color to give it a drop shadow
        Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos
+ new Vector2(10, 10), dropShadowColor, 0.0f, _slimeTextOrigin, 1.0f,
SpriteEffects.None, 1.0f);

        // Draw the Slime text on top of that at its original position
        Core.SpriteBatch.DrawString(_font5x, SLIME_TEXT, _slimeTextPos, Color.White,
0.0f, _slimeTextOrigin, 1.0f, SpriteEffects.None, 1.0f);

        // Always end the sprite batch when finished.
        Core.SpriteBatch.End();
    }

    GumService.Default.Draw();
}
```

With these changes, our UI system is now fully integrated into the scene's game loop. Gum updates its controls in the `Update` method and draws them in the `Draw` method. This produces a fully functional title screen with buttons that allows players to start the game or adjust audio settings.

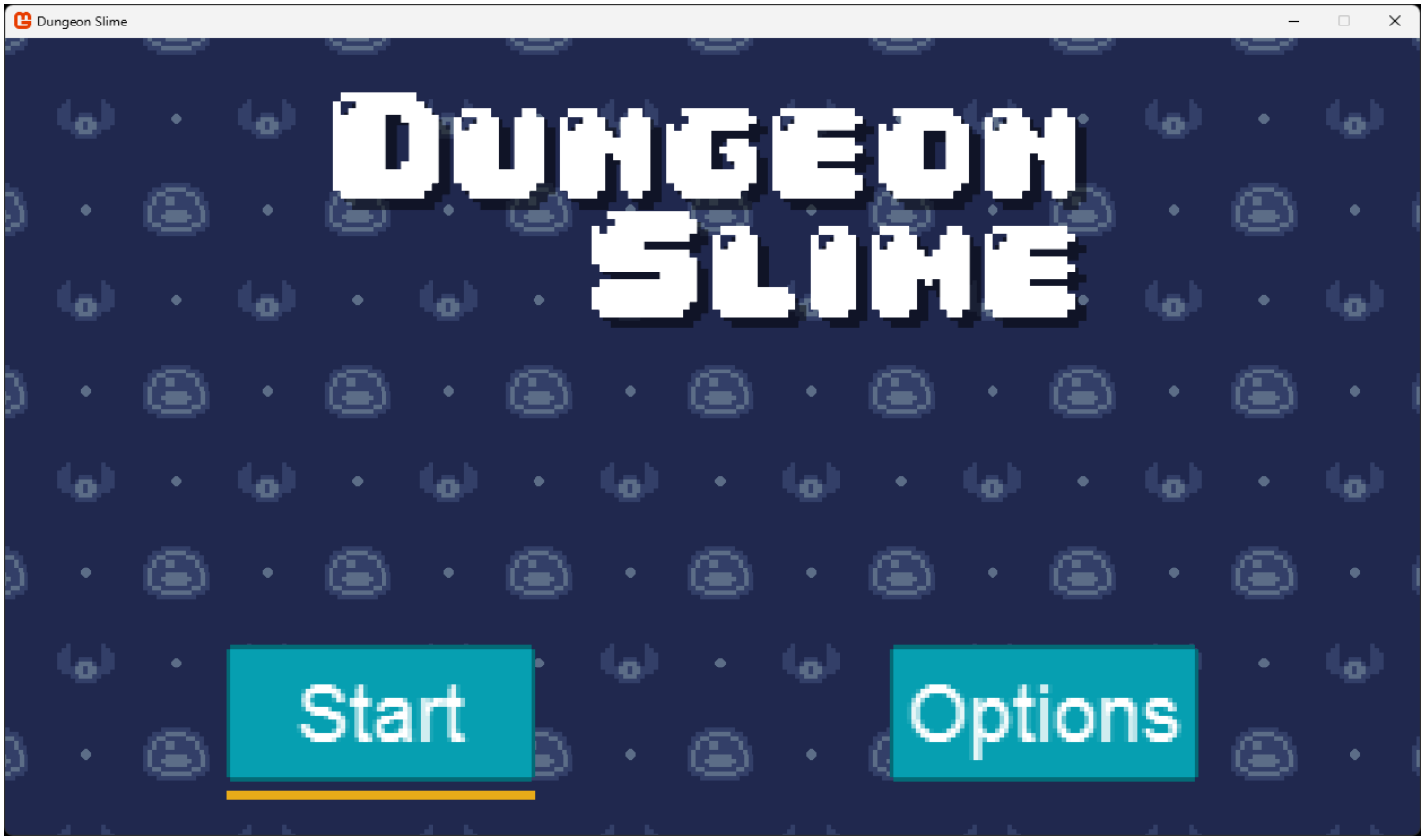


Figure 20-2: Title screen with default Gum buttons

> **NOTE**
>
> You may notice that the UI elements currently use Gum's default styling, which does not match our game's visual theme. We will explore customizing these controls to match our game's visual style in the next chapter.

Adding GameScene UI

Now that we have setup the UI for the title scene, we will add a pause menu to our game scene. This UI will start invisible but will be shown when the player presses the escape key. For consistency, we will implement the UI for the game scene in the same order that we implemented the UI for the title scene.

> **NOTE**
>
> When adding these sections one by one, you may see compiler errors until all sections are in place. This is normal, as some parts of the code will reference fields or methods that haven't been added yet. Once all sections are complete, these errors will resolve.

First, open the *GameScene.cs* file in the game project and add the following using declarations to the top of the `GameScene` class.

```
using System;
using Gum.DataTypes;
using Gum.Wireframe;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameGum;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;
using MonoGameLibrary.Scenes;
```

Next, add the following fields to the `GameScene` class:

```
// A reference to the pause panel UI element so we can set its visibility
// when the game is paused.
private Panel _pausePanel;

// A reference to the resume button UI element so we can focus it
// when the game is paused.
private Button _resumeButton;

// The UI sound effect to play when a UI event is triggered.
private SoundEffect _uiSoundEffect;
```

Pausing the Game

To pause the game, first we will create a method that makes the pause panel visible. Add the following method to the `GameScene` class after the `CheckGamePadInput` method:

```
private void PauseGame()
{
    // Make the pause panel UI element visible.
    _pausePanel.IsVisible = true;

    // Set the resume button to have focus
    _resumeButton.IsFocused = true;
}
```

Next, update the `CheckKeyboardInput` method so that when the escape key is pressed, instead of returning to the title scene, we now pause the game:

```
private void CheckKeyboardInput()
{
    // Get a reference to the keyboard info
    KeyboardInfo keyboard = Core.Input.Keyboard;

    // If the escape key is pressed, pause the game.
    if (Core.Input.Keyboard.WasKeyJustPressed(Keys.Escape))
    {
        PauseGame();
        return;
    }

    // Existing keyboard input code
    // ...
```

Finally, update the `CheckGamePadInput` method so that when the start button is pressed, it pauses the game:

```
private void CheckGamePadInput()
{
    // Get the gamepad info for gamepad one.
    GamePadInfo gamePadOne = Core.Input.GamePads[(int)PlayerIndex.One];

    // If the start button is pressed, pause the game
    if (gamePadOne.WasButtonJustPressed(Buttons.Start))
    {
        PauseGame();
        return;
    }

    // Existing gamepad input code
    // ...
```

Creating the Pause Panel

Next, we will create a method that builds our pause panel with resume and quit buttons. Add the following method to the `GameScene` class after the `LoadContent` method:

```
private void CreatePausePanel()
{
    _pausePanel = new Panel();
    _pausePanel.Anchor(Anchor.Center);
    _pausePanel.WidthUnits = DimensionUnitType.Absolute;
    _pausePanel.HeightUnits = DimensionUnitType.Absolute;
    _pausePanel.Height = 70;
    _pausePanel.Width = 264;
    _pausePanel.IsVisible = false;
    _pausePanel.AddToRoot();

    var background = new ColoredRectangleRuntime();
    background.Dock(Dock.Fill);
    background.Color = Color.DarkBlue;
    _pausePanel.AddChild(background);

    var textInstance = new TextRuntime();
    textInstance.Text = "PAUSED";
    textInstance.X = 10f;
    textInstance.Y = 10f;
    _pausePanel.AddChild(textInstance);

    _resumeButton = new Button();
    _resumeButton.Text = "RESUME";
    _resumeButton.Anchor(Anchor.BottomLeft);
    _resumeButton.X = 9f;
    _resumeButton.Y = -9f;
    _resumeButton.Width = 80;
    _resumeButton.Click += HandleResumeButtonClicked;
    _pausePanel.AddChild(_resumeButton);

    var quitButton = new Button();
    quitButton.Text = "QUIT";
    quitButton.Anchor(Anchor.BottomRight);
    quitButton.X = -9f;
    quitButton.Y = -9f;
    quitButton.Width = 80;
    quitButton.Click += HandleQuitButtonClicked;

    _pausePanel.AddChild(quitButton);
}
```

Now we should implement the event handlers for these controls. First, we will implement the handler for the "Resume" button. Add the following method to the `GameScene` class after the `CreatePausePanel` method:

```
private void HandleResumeButtonClicked(object sender, EventArgs e)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Make the pause panel invisible to resume the game.
    _pausePanel.IsVisible = false;
}
```

This method plays the UI sound effect for auditory feedback and then hides the pause panel so that the game can resume.

Next is the handler for the "Quit" button. Add the following method to the `GameScene` class after the `HandleResumeButtonClicked` method:

```
private void HandleQuitButtonClicked(object sender, EventArgs e)
{
    // A UI interaction occurred, play the sound effect
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Go back to the title scene.
    Core.ChangeScene(new TitleScene());
}
```

This method as well plays the UI sound effect for auditory feedback, then quits the game by changing scenes back to the title scene.

Initializing the Game UI

Now that we have implemented the method to create the pause panel, we can implement the main UI initializations method that will call them. Add the following method to the `GameScene` class after the `HandleQuitButtonClicked` method:

```
private void InitializeUI()
{
    GumService.Default.Root.Children.Clear();

    CreatePausePanel();
}
```

Just like with the `TitleScene`, we first clear any existing UI elements from Gum's root before creating the UI elements for this scene.

Integrating with the Game Loop for the GameScreen

Finally, we need to integrate our UI initialization, update, and draw with the scene's lifecycle. First add the call to `InitializeUI()` in the `Initialize` method by updating it to the following:

```
public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // During the game scene, we want to disable exit on escape. Instead,
    // the escape key will be used to return back to the title screen
    Core.ExitOnEscape = false;

    Rectangle screenBounds = Core.GraphicsDevice.PresentationParameters.Bounds;

    _roomBounds = new Rectangle(
        (int)_tilemap.TileWidth,
        (int)_tilemap.TileHeight,
        screenBounds.Width - (int)_tilemap.TileWidth * 2,
        screenBounds.Height - (int)_tilemap.TileHeight * 2
    );

    // Initial slime position will be the center tile of the tile map.
    int centerRow = _tilemap.Rows / 2;
    int centerColumn = _tilemap.Columns / 2;
    _slimePosition = new Vector2(centerColumn * _tilemap.TileWidth, centerRow
* _tilemap.TileHeight);

    // Initial bat position will the in the top left corner of the room
    _batPosition = new Vector2(_roomBounds.Left, _roomBounds.Top);

    // Set the position of the score text to align to the left edge of the
    // room bounds, and to vertically be at the center of the first tile.
    _scoreTextPosition = new Vector2(_roomBounds.Left, _tilemap.TileHeight * 0.5f);

    // Set the origin of the text so it is left-centered.
    float scoreTextYOrigin = _font.MeasureString("Score").Y * 0.5f;
    _scoreTextOrigin = new Vector2(0, scoreTextYOrigin);

    // Assign the initial random velocity to the bat.
    AssignRandomBatVelocity();

    InitializeUI();
}
```

Next, update the `LoadContent` method to load the sound effect that will be used as auditory feedback for the UI:

```
public override void LoadContent()
{
    // Create the texture atlas from the XML configuration file
    TextureAtlas atlas = TextureAtlas.FromFile(Core.Content,
"images/atlas-definition.xml");

    // Create the slime animated sprite from the atlas.
    _slime = atlas.CreateAnimatedSprite("slime-animation");
    _slime.Scale = new Vector2(4.0f, 4.0f);

    // Create the bat animated sprite from the atlas.
    _bat = atlas.CreateAnimatedSprite("bat-animation");
    _bat.Scale = new Vector2(4.0f, 4.0f);

    // Create the tilemap from the XML configuration file.
    _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
    _tilemap.Scale = new Vector2(4.0f, 4.0f);

    // Load the bounce sound effect
    _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

    // Load the collect sound effect
    _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

    // Load the font
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

    // Load the sound effect to play when ui actions occur.
    _uiSoundEffect = Core.Content.Load<SoundEffect>("audio/ui");
}
```

Next, modify the `Update` method to include Gum's update logic and to only update the game if it is not paused. We will use the visibility of the pause menu to determine if the game is paused or not:

```
public override void Update(GameTime gameTime)
{
    // Ensure the UI is always updated
    GumService.Default.Update(gameTime);

    // If the game is paused, do not continue
    if (_pausePanel.IsVisible)
    {
        return;
    }

    // Existing game update code
    // ...
```

Finally, add Gum's drawing call to the end fo the `Draw` method:

```
public override void Draw(GameTime gameTime)
{
    // Existing game draw code
    // ...

    // Always end the sprite batch when finished
    Core.SpriteBatch.End();

    // Draw the Gum UI
    GumService.Default.Draw();
}
```

With these changes, the pause menu is now fully integrated into the game scene's game loop. Gum updates its controls during the `Update` method and draws them during the `Draw` method. If the game is paused, as determined by the `IsVisible` property of the pause menu, then updating the actual game logic is skipped.

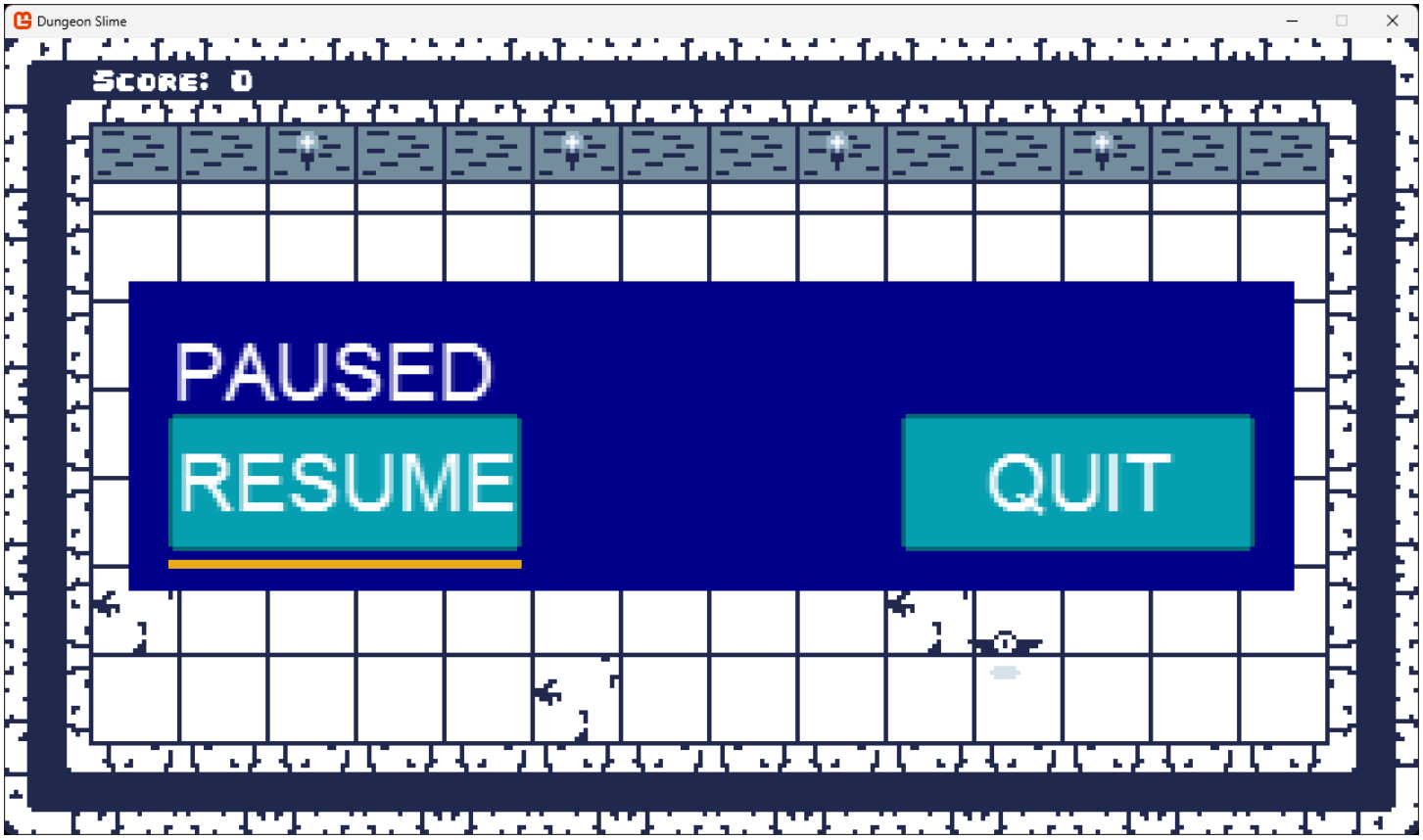


Figure 20-3: The pause menu during the game scene with default Gum buttons

Conclusion

In this chapter, you accomplished the following:

- Add and configure the Gum NuGet package in your project.
- Understand key Gum concepts like Forms controls and Visuals.
- Create and position UI elements using anchoring and docking.
- Implement interactive controls like buttons and sliders.
- Handle user input from various input devices.
- Create transitions between different UI screens.
- Integrate the UI system with the game's scene architecture.

While this UI is now functional, you may have noticed that it uses Gum's default styling which does not match our game's visual theme. In the next chapter, we will learn how to customize the appearance of our UI elements to create a cohesive visual style that complements our game's aesthetic.

Test Your Knowledge

1. What are the two main types of objects in Gum, and how do they differ?

> The two main types are:
>
> - **Forms**: Interactive UI elements like buttons, sliders, and panels that handle user input. They provide built-in functionality for common UI interactions.
> - **Visuals**: Display elements like TextRuntime, ColoredRectangleRuntime, and NineSliceRuntime that are used to render graphics. They have no built-in interaction behavior but can be customized visually.
>
> Forms controls contain Visuals, accessible through the `Visual` property, creating a separation between functionality and presentation.

2. How does Gum handle the parent-child relationship of UI elements, and why is this important?

> Gum implements parent-child relationships through a hierarchical structure where:
>
> - UI elements must be connected to the root container to be visible
> - Children can be added directly to a parent's Visual.Children collection
> - Position coordinates of child elements are relative to their parent
> - Property changes like visibility cascade from parent to children
>
> This relationship is important because it allows for organizing related UI elements as groups, controlling entire sections of UI with a single property change, and positioning elements relative to their container rather than absolute screen coordinates.

3. What are the two ways to customize the appearance of Gum UI elements?

The two ways to customize Gum UI elements are:

1. **Direct property assignment**: Setting properties directly in code (like `MyButton.Width = 100`). This works well for initial setup and static properties.
2. **States**: Using Gum's state system (`StateSave` objects) to define different visual states that can be applied in response to specific conditions or events. States are automatically applied by Forms controls in response to user interactions (like focus or highlighting).

States are useful for dynamic changes that occur during gameplay, as they separate visual response logic from game logic.

4. What steps are necessary to integrate Gum's UI system with MonoGame's game loop?

To integrate Gum with MonoGame's game loop:

1. Initialize Gum in the game's Initialize method with `GumService.Default.Initialize(this)`
2. Configure content loading by setting `GumService.Default.ContentLoader.XnaContentManager`
3. Set up input handling by adding keyboards and gamepads to `FrameworkElement.KeyboardsForUiControl` and `FrameworkElement.GamePadsForUiControl`
4. Call `GumService.Default.Update()` in the game's Update method
5. Call `GumService.Default.Draw()` in the game's Draw method
6. For scene transitions, clear existing UI elements with `GumService.Default.Root.Children.Clear()`

This ensures Gum can update and render UI elements in sync with the game's main loop.

Chapter 21: Customizing Gum UI

Learn how to create custom UI components with animations and visual styling in Gum.

In the previous chapter, we implemented a functional UI system for our game using the Gum framework. While the UI is now fully operational, it uses Gum's default styling. This default styling is good for quickly iterating when building the UI, but it does not match the game's visuals. A well-designed UI should not only be functional but also complement the game's overall visual style to create a cohesive experience.

In this chapter, you will:

- Learn about Gum's visual customization system and component hierarchy.
- Understand how animation chains and visual states work in Gum.
- Create custom styled button and slider components.
- Update the game's texture atlas to include UI graphics.
- Implement responsive visual feedback for player interactions.
- Apply your custom components to the game's UI screens.

Understanding Gum's Customization System

Gum provides a powerful customization system that separates a UI element's functionality from its appearance. This allows you to maintain the built-in behavior of standard controls while completely changing their visual representation.

Container Hierarchy

Every customized UI component in Gum starts with a top-level container that holds all other visual elements. This container is typically of type `ContainerRuntime`, which is similar to the `Panel` type we used earlier, but specifically designed for building custom visuals.

The container hierarchy follows a parent-child relationship:

- The top-level container manages the overall size and positioning of the component.
- Visual elements like backgrounds, text, and icons are added as children.
- Child elements can be positioned relative to their parent container.
- Child elements can also be nested within other children, creating deeper hierarchies.

This hierarchical structure allows you to build complex UI components from simpler parts, with each part playing a specific role in the overall design.

Size Relationships with Width and WidthUnits

One powerful feature of Gum is how it handles size relationships between parent and child elements. By using different `WidthUnits` values, you can create dependencies that flow in different directions:

- **RelativeToChildren**: A parent container can size itself based on its children.
- **PercentageOfParent**: A child element can size itself as a percentage of its parent.
- **Absolute**: An element can have a fixed pixel size.
- **RelativeToParent**: An element can size itself relative to a specific container.

For example:

- A button might use a text element with `WidthUnits` set to `RelativeToChildren`, which means the text will be exactly the size needed to display its content.
- The button's container might use `RelativeToChildren` with some additional padding, allowing the button to automatically resize based on its text content.

Although we have not explicitly assigned WidthUnits and HeightUnits in our code, we have indirectly set these values by calling the Visual's `Dock` method. Specifically, by passing `Dock.Fill` as the parameter, `WidthUnits` and `HeightUnits` are both set to `RelativeToParent`.

> **NOTE**
> These size relationships can create circular dependencies when a child depends on its parent and the parent depends on the child. In such cases, Gum resolves the conflict by making the child depend on the parent, and the parent ignores that particular child when calculating its size.

Visual Elements

Gum provides several visual element types that we can use to build our custom components:

- **ContainerRuntime**: An invisible container for organizing other elements.
- **NineSliceRuntime**: A special graphic that can stretch while preserving its corners and edges.
- **TextRuntime**: An element for displaying text with custom fonts.
- **ColoredRectangleRuntime**: A simple colored rectangle for backgrounds or fills.

The `NineSliceRuntime` is particularly useful for UI elements that need to resize dynamically. It divides a graphic into nine sections (four corners, four edges, and a center), allowing the element to stretch without distorting its borders.

NOTE

A MonoGame and Gum community member Kaltinril also has a video series discussing Gum. With permission, the following video segment is included to demonstrate the advantages of using a *Nineslice* when creating UI elements.

Animation Chains

An `AnimationChain` is a sequence of animation frames that play in order, typically looping after the last frame. Each frame in the chain defines:

- Which part of a texture to display (using texture coordinates).
- How long to display that frame (using a frame length value).
- Which texture to use for the frame.

Texture coordinates in Gum use normalized values (0.0 to 1.0) rather than pixel coordinates, where:

- 0.0 represents the left or top edge of the texture.
- 1.0 represents the right or bottom edge of the texture.

To convert from pixel coordinates to normalized values, you divide the pixel position by the texture's width or height.

Visual States

When UI elements change state (like when a button is focused), Gum uses a state-based system. A control's visual can be casted to a type specific to that control to access its states. Each control-specific visual has a type name matching its control, with the word Visual appended. For example, a button's `Visual` can be casted to type `ButtonVisual` to access button-specific properties and states. Once casted, the visual provides a `States` property containing all available states for that control type. Most control visuals, including `ButtonVisual`, provide the following States properties:

- States.Enabled (the normal, unfocused state).
- States.Focused (when the control has focus).
- States.Highlighted (when the mouse hovers over the control).
- States.Disabled (when the control cannot be interacted with).

Each state contains an `Apply` action that defines what visual changes occur when that state becomes active. For example, when a button becomes focused, its state might change the background color or switch to an animated version.

Input and Focus Handling

Custom UI components can enhance their interactivity by handling specific input events:

- The `KeyDown` event can be used to add custom keyboard navigation.
- The `RollOn` event can detect when the mouse moves over the component.
- The `Click` event can respond to mouse clicks or gamepad button presses.

Gum distinguishes between highlighting (visual response to mouse hover) and focus (ability to receive keyboard/gamepad input). For a seamless experience across input devices, a common pattern is to automatically focus elements when the mouse hovers over them, ensuring that visual highlighting and input focus remain synchronized.

Now that we understand the key concepts behind Gum's customization system, we can apply them to create custom UI components for our game.

Updating the Game Resources

Before we create our custom components, we need to update the game's resources to include UI graphics and fonts.

Update the Texture Atlas

First need to update the *atlas.png* texture atlas file for the game. This new version of the texture atlas includes:

- The characters for the font, generated using Bitmap Font Generator (BMFont)
- The sprites for the UI components we will create

Download the new texture atlas below by right-clicking the following image and saving it as *atlas.png* in the *Content/images* folder of the game project, overwriting the existing one.

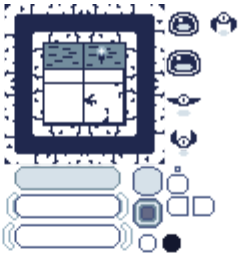


Figure 21-1: The texture atlas for the game updated to include the UI sprites

The slime and bat sprites are no longer in the same position, and we have some new regions to define for our UI sprites. This means we need to update the texture atlas XML configuration file as well. Open the *atlas-definition.xml* configuration file and update it to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TextureAtlas>
    <Texture>images/atlas</Texture>
    <Regions>
        <Region name="slime-1" x="340" y="0" width="20" height="20" />
        <Region name="slime-2" x="340" y="20" width="20" height="20" />
        <Region name="bat-1" x="340" y="40" width="20" height="20" />
        <Region name="bat-2" x="340" y="60" width="20" height="20" />
        <Region name="bat-3" x="360" y="0" width="20" height="20" />
        <Region name="unfocused-button" x="259" y="80" width="65" height="14" />
        <Region name="focused-button-1" x="259" y="94" width="65" height="14" />
        <Region name="focused-button-2" x="259" y="109" width="65" height="14" />
        <Region name="panel-background" x="324" y="97" width="15" height="15" />
        <Region name="slider-off-background" x="341" y="96" width="11"
height="10" />
        <Region name="slider-middle-background" x="345" y="81" width="3"
height="3" />
        <Region name="slider-max-background" x="354" y="96" width="11"
height="10" />
    </Regions>
    <Animations>
        <Animation name="slime-animation" delay="200">
            <Frame region="slime-1" />
            <Frame region="slime-2" />
        </Animation>
        <Animation name="bat-animation" delay="200">
            <Frame region="bat-1" />
            <Frame region="bat-2" />
            <Frame region="bat-1" />
            <Frame region="bat-3" />
        </Animation>
        <Animation name="focused-button-animation" delay="300">
            <Frame region="focused-button-1" />
            <Frame region="focused-button-2" />
        </Animation>
    </Animations>
</TextureAtlas>
```

The same is now true for the tiles in the texture atlas. Since they have been repositioned in the new texture atlas, we need to update the `region` attribute for the tilemap XML configuration file. Open the `tilemap-definition.xml` configuration file and update it to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<Tilemap>
    <Tileset region="260 0 80 80" tileWidth="20"
tileHeight="20">images/atlas</Tileset>
    <Tiles>
        00 01 02 01 02 01 02 01 02 01 02 01 02 01 02 03
        04 05 05 06 05 05 06 05 05 06 05 05 06 05 05 07
        08 09 09 09 09 09 09 09 09 09 09 09 09 09 09 11
        04 09 09 09 09 09 09 09 10 09 09 09 09 10 09 07
        08 09 10 09 09 09 09 09 09 09 09 09 09 09 09 11
        04 09 09 09 09 09 09 09 09 09 09 09 09 09 09 07
        08 10 09 09 09 09 09 09 09 09 10 09 09 09 09 11
        04 09 09 09 09 09 10 09 09 09 09 09 09 09 09 07
        12 13 14 13 14 13 14 13 14 13 14 13 14 13 14 15
    </Tiles>
</Tilemap>
```

Adding Bitmap Fonts

While MonoGame natively uses **SpriteFont** to draw text, Gum uses the AngelCode Bitmap Font (.fnt) font file format. This means we will need to supply Gum with the *.fnt* file that defines our font.

> **NOTE**
> For this tutorial, a pregenerated *.fnt* file is supplied below. For more information on creating *.fnt* files for Gum, see the Create Fonts with BitmapFontGenerator section of the Gum documentation.

Download the *.fnt* file below by right-clicking the following link and saving it as *04b_30.fnt* in the game project's *Content/fonts* folder:

- 04b_30.fnt

Next, add this font file to your content project using the MGCB Editor:

1. Open the `Content.mgcb` content project file in the MGCB Editor.
2. Right-click the `fonts` folder and choose `Add > Existing Item...`.
3. Navigate to and select the `04b_30.fnt` file you just downloaded.
4. In the Properties panel, change the `Build Action` to `Copy`. The MonoGame Content Pipeline cannot process *.fnt* files; we just need it to copy it so we can give it to Gum.
5. **Save the changes and close the MGCB Editor.**

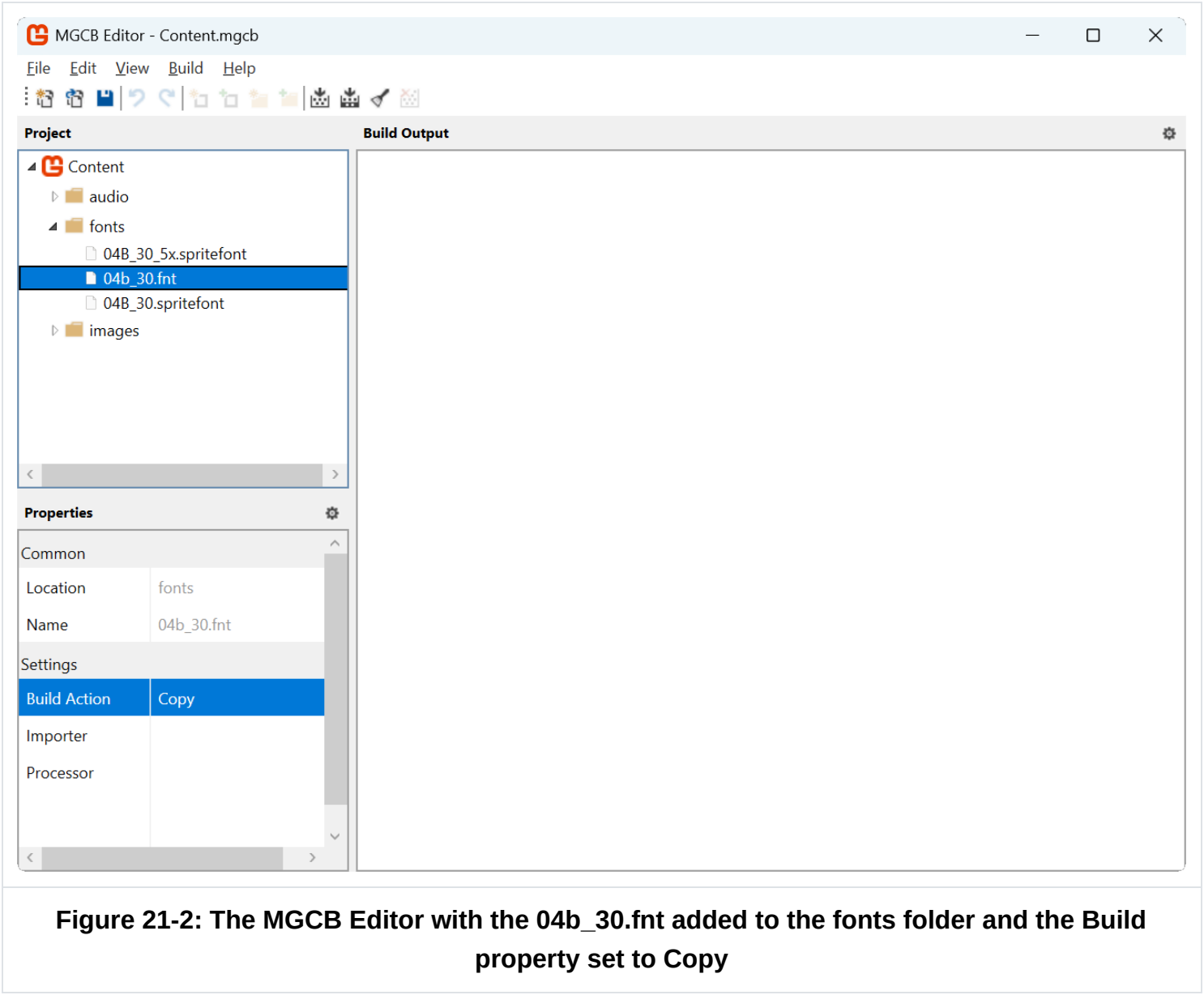


Figure 21-2: The MGCB Editor with the 04b_30.fnt added to the fonts folder and the Build property set to Copy

> **NOTE**
>
> When the *.fnt* font file was generated using the AngelCode Bitmap Font Generator, the graphics from the *.png* file that it produces were copied over into our existing texture atlas. By doing this, it allows Gum to render the visuals for elements and the text from the same atlas, reducing texture swapping.
>
> The font file references our existing texture atlas using a relative path that points to the atlas image.
>
> The best practice when using this method is to ensure that when you copy the graphics from the generated *.png* file to your texture atlas, you place the generated character glyph graphics in the top-left of your texture atlas. This means all of the coordinates in the *.fnt* file will correctly reference the character glyphs without additional manual changes.

Updating the TextureRegion Class

In Chapter 18 we discussed texture coordinates and that graphic devices use a normalized coordinate system between 0.0 and 1.0.

Since Gum also uses this coordinate system, we will update the `TextureRegion` class to easily provide these values for any given region.

Open the `TextureRegion.cs` file in the *MonoGameLibrary* project and add the following properties to the `TextureRegion` class:

```
/// <summary>
/// Gets the top normalized texture coordinate of this region.
/// </summary>
public float TopTextureCoordinate => SourceRectangle.Top / (float)Texture.Height;

/// <summary>
/// Gets the bottom normalized texture coordinate of this region.
/// </summary>
public float BottomTextureCoordinate => SourceRectangle.Bottom
/ (float)Texture.Height;

/// <summary>
///  Gets the left normalized texture coordinate of this region.
/// </summary>
public float LeftTextureCoordinate => SourceRectangle.Left / (float)Texture.Width;

/// <summary>
/// Gets the right normalized texture coordinate of this region.
/// </summary>
public float RightTextureCoordinate => SourceRectangle.Right / (float)Texture.Width;
```

Creating Custom UI Components

Now that we have all our resources prepared, we can create custom versions of the UI controls we are using in our game. We will start with an animated button that uses our game's visual style, then move on to creating a custom slider.

The AnimatedButton Class

Our first custom component is an `AnimatedButton` that inherits from Gum's base `Button` class. This button uses the game's existing texture atlas for its visual appearance and animates when focused.

By default, all Gum components provide a Visual property which can be cast to a type specific to the particular control. By convention, the visual type is named the same as the component with the word `Visual` appended. For example, we will be casting the AnimatedButton's Visual property to `ButtonVisual` to access button-specific properties.

This new `AnimatedButton` class casts the Visual property to `ButtonVisual` and modifies the button-specific properties, such as background and text.

First, in the *DungeonSlime* project (your main game project), create a new folder named `UI` to store our custom UI components. Next, in that `UI` folder, create a new file called `AnimatedButton.cs` and add the following code to it:

```
using System;
using Gum.DataTypes;
using Gum.DataTypes.Variables;
using Gum.Forms.Controls;
using Gum.Forms.DefaultVisuals.V3;
using Gum.Graphics.Animation;
using Gum.Managers;
using Microsoft.Xna.Framework.Input;
using MonoGameGum.GueDeriving;
using MonoGameLibrary.Graphics;

namespace DungeonSlime.UI;

/// <summary>
/// A custom button implementation that inherits from Gum's Button class to provide
/// animated visual feedback when focused.
/// </summary>
internal class AnimatedButton : Button
{
    /// <summary>
    /// Creates a new AnimatedButton instance using graphics from the specified
texture atlas.
    /// </summary>
    /// <param name="atlas">The texture atlas containing button graphics
and animations</param>
    public AnimatedButton(TextureAtlas atlas)
    {
        // Each Forms control has a general Visual property that
        // has properties shared by all control types. This Visual
        // type matches the Forms type. It can be casted to access
        // controls-specific properties.
        ButtonVisual buttonVisual = (ButtonVisual)Visual;
        // Width is relative to children with extra padding, height is fixed
        buttonVisual.Height = 14f;
        buttonVisual.HeightUnits = DimensionUnitType.Absolute;
        buttonVisual.Width = 21f;
        buttonVisual.WidthUnits = DimensionUnitType.RelativeToChildren;

        // Get a reference to the nine-slice background to display the
button graphics
        // A nine-slice allows the button to stretch while preserving
corner appearance
        NineSliceRuntime background = buttonVisual.Background;
        background.Texture = atlas.Texture;
        background.TextureAddress = TextureAddress.Custom;
        background.Color = Microsoft.Xna.Framework.Color.White;
```

```
        // texture coordinates for the background are set down below

        TextRuntime textInstance = buttonVisual.TextInstance;
        textInstance.Text = "START";
        textInstance.Blue = 130;
        textInstance.Green = 86;
        textInstance.Red = 70;
        textInstance.UseCustomFont = true;
        textInstance.CustomFontFile = "fonts/04b_30.fnt";
        textInstance.FontScale = 0.25f;
        textInstance.Anchor(Gum.Wireframe.Anchor.Center);
        textInstance.Width = 0;
        textInstance.WidthUnits = DimensionUnitType.RelativeToChildren;

        // Get the texture region for the unfocused button state from the atlas
        TextureRegion unfocusedTextureRegion = atlas.GetRegion("unfocused-button");

        // Create an animation chain for the unfocused state with a single frame
        AnimationChain unfocusedAnimation = new AnimationChain();
        unfocusedAnimation.Name = nameof(unfocusedAnimation);
        AnimationFrame unfocusedFrame = new AnimationFrame
        {
            TopCoordinate = unfocusedTextureRegion.TopTextureCoordinate,
            BottomCoordinate = unfocusedTextureRegion.BottomTextureCoordinate,
            LeftCoordinate = unfocusedTextureRegion.LeftTextureCoordinate,
            RightCoordinate = unfocusedTextureRegion.RightTextureCoordinate,
            FrameLength = 0.3f,
            Texture = unfocusedTextureRegion.Texture
        };
        unfocusedAnimation.Add(unfocusedFrame);

        // Get the multi-frame animation for the focused button state from the atlas
        Animation focusedAtlasAnimation = atlas.GetAnimation("focused-
button-animation");

        // Create an animation chain for the focused state using all frames from the
atlas animation
        AnimationChain focusedAnimation = new AnimationChain();
        focusedAnimation.Name = nameof(focusedAnimation);
        foreach (TextureRegion region in focusedAtlasAnimation.Frames)
        {
            AnimationFrame frame = new AnimationFrame
            {
                TopCoordinate = region.TopTextureCoordinate,
                BottomCoordinate = region.BottomTextureCoordinate,
                LeftCoordinate = region.LeftTextureCoordinate,
```

```
                RightCoordinate = region.RightTextureCoordinate,
                FrameLength = (float)focusedAtlasAnimation.Delay.TotalSeconds,
                Texture = region.Texture
            };

            focusedAnimation.Add(frame);
        }

        // Assign both animation chains to the nine-slice background
        background.AnimationChains = new AnimationChainList
        {
            unfocusedAnimation,
            focusedAnimation
        };


        // Reset all state to default so we don't have unexpected
variable assignments:
        buttonVisual.ButtonCategory.ResetAllStates();

        // Get the enabled (default/unfocused) state
        StateSave enabledState = buttonVisual.States.Enabled;
        enabledState.Apply = () =>
        {
            // When enabled but not focused, use the unfocused animation
            background.CurrentChainName = unfocusedAnimation.Name;
        };

        // Create the focused state
        StateSave focusedState = buttonVisual.States.Focused;
        focusedState.Apply = () =>
        {
            // When focused, use the focused animation and enable animation playback
            background.CurrentChainName = focusedAnimation.Name;
            background.Animate = true;
        };

        // Create the highlighted+focused state (for mouse hover while focused)
        StateSave highlightedFocused = buttonVisual.States.HighlightedFocused;
        highlightedFocused.Apply = focusedState.Apply;

        // Create the highlighted state (for mouse hover)
        // by cloning the enabled state since they appear the same
        StateSave highlighted = buttonVisual.States.Highlighted;
        highlighted.Apply = enabledState.Apply;
```

```
        // Add event handlers for keyboard input.
        KeyDown += HandleKeyDown;

        // Add event handler for mouse hover focus.
        buttonVisual.RollOn += HandleRollOn;
    }

    /// <summary>
    /// Handles keyboard input for navigation between buttons using left/right keys.
    /// </summary>
    private void HandleKeyDown(object sender, KeyEventArgs e)
    {
        if (e.Key == Keys.Left)
        {
            // Left arrow navigates to previous control
            HandleTab(TabDirection.Up, loop: true);
        }
        if (e.Key == Keys.Right)
        {
            // Right arrow navigates to next control
            HandleTab(TabDirection.Down, loop: true);
        }
    }

    /// <summary>
    /// Automatically focuses the button when the mouse hovers over it.
    /// </summary>
    private void HandleRollOn(object sender, EventArgs e)
    {
        IsFocused = true;
    }
}
```

Next, we will examine the key aspects of this new `AnimatedButton` implementation:

ButtonVisual

As mentioned earlier, we first access the `Visual` object and cast it to a `ButtonVisual`. Doing so gives us access to button-specific properties, including individual elements (such as the text and background visuals) as well as the states that are applied when the button is hovered or pressed.

We can modify the Visual to give it the appropriate size.

```
ButtonVisual buttonVisual = (ButtonVisual)Visual;
// Width is relative to children with extra padding, height is fixed
buttonVisual.Height = 14f;
buttonVisual.HeightUnits = DimensionUnitType.Absolute;
buttonVisual.Width = 21f;
buttonVisual.WidthUnits = DimensionUnitType.RelativeToChildren;
```

The `WidthUnits` property set to `RelativeToChildren` means the container automatically sizes itself based on its child elements, with 21 pixels of additional space. This allows the button to adapt its size depending on the text content.

Nine-slice Background

`ButtonVisual` provides a `Background` which we can modify. This is of type `NineSliceRuntime`, which is a special graphic that can be stretched while preserving its corners and edges:

```
NineSliceRuntime background = buttonVisual.Background;
background.Texture = atlas.Texture;
background.TextureAddress = TextureAddress.Custom;
background.Color = Microsoft.Xna.Framework.Color.White;
```

The `TextureAddress` property is set to `Custom` so we can specify exactly which portion of the atlas texture to use, while `Dock(Dock.Fill)` ensure the background fills the entire button area. The portion of the atlas is assigned using AnimationChains, which are discussed later in this tutorial.

Text

`ButtonVisual` also provides a customizable `Text` property. In this case, we assign the font, color, and size.

```
TextRuntime textInstance = buttonVisual.TextInstance;
textInstance.Text = "START";
textInstance.Blue = 130;
textInstance.Green = 86;
textInstance.Red = 70;
textInstance.UseCustomFont = true;
textInstance.CustomFontFile = "fonts/04b_30.fnt";
textInstance.FontScale = 0.25f;
textInstance.Anchor(Gum.Wireframe.Anchor.Center);
textInstance.Width = 0;
textInstance.WidthUnits = DimensionUnitType.RelativeToChildren;
```

Animated Chains

The most distinctive feature of our animated button is its ability to change appearance when focused. We achieve this by creating two animation chains:

1. An "unfocused" animation with a single static frame.
2. A "focused" animation with two alternating frames that create a visual effect.

Each animation frame specifies the coordinates within our texture atlas to display:

```
TextureRegion unfocusedTextureRegion = atlas.GetRegion("unfocused-button");

// Create an animation chain for the unfocused state with a single frame
AnimationChain unfocusedAnimation = new AnimationChain();
unfocusedAnimation.Name = nameof(unfocusedAnimation);
AnimationFrame unfocusedFrame = new AnimationFrame
{
    TopCoordinate = unfocusedTextureRegion.TopTextureCoordinate,
    BottomCoordinate = unfocusedTextureRegion.BottomTextureCoordinate,
    LeftCoordinate = unfocusedTextureRegion.LeftTextureCoordinate,
    RightCoordinate = unfocusedTextureRegion.RightTextureCoordinate,
    FrameLength = 0.3f,
    Texture = unfocusedTextureRegion.Texture
};
unfocusedAnimation.Add(unfocusedFrame);

// Get the multi-frame animation for the focused button state from the atlas
Animation focusedAtlasAnimation = atlas.GetAnimation("focused-button-animation");

// Create an animation chain for the focused state using all frames from the
atlas animation
AnimationChain focusedAnimation = new AnimationChain();
focusedAnimation.Name = nameof(focusedAnimation);
foreach (TextureRegion region in focusedAtlasAnimation.Frames)
{
    AnimationFrame frame = new AnimationFrame
    {
        TopCoordinate = region.TopTextureCoordinate,
        BottomCoordinate = region.BottomTextureCoordinate,
        LeftCoordinate = region.LeftTextureCoordinate,
        RightCoordinate = region.RightTextureCoordinate,
        FrameLength = (float)focusedAtlasAnimation.Delay.TotalSeconds,
        Texture = region.Texture
    };

    focusedAnimation.Add(frame);
}
```

States and Categories

In Gum, each control type has a specific category name that identifies its state collection. `ButtonVisual` provides access to ready-made states and categories which we can modify. Before we specify how a state should modify the button's appearance, we clear out all existing functionality so that we can fully control the states:

```
buttonVisual.ButtonCategory.ResetAllStates();
```

Each of the button's states can be accessed through `ButtonVisual`. Since the states were cleared previously, the code assigns only the necessary property assignments in the `Apply` delegate. In our case, we switch between animation chains to create the desired visual effect.

```
StateSave enabledState = buttonVisual.States.Enabled;
enabledState.Apply = () =>
{
    // When enabled but not focused, use the unfocused animation
    background.CurrentChainName = unfocusedAnimation.Name;
};

// Create the focused state
StateSave focusedState = buttonVisual.States.Focused;
focusedState.Apply = () =>
{
    // When focused, use the focused animation and enable animation playback
    background.CurrentChainName = focusedAnimation.Name;
    background.Animate = true;
};

// Create the highlighted+focused state (for mouse hover while focused)
StateSave highlightedFocused = buttonVisual.States.HighlightedFocused;
highlightedFocused.Apply = focusedState.Apply;

// Create the highlighted state (for mouse hover)
// by cloning the enabled state since they appear the same
StateSave highlighted = buttonVisual.States.Highlighted;
highlighted.Apply = enabledState.Apply;
```

Custom Input Handling

We add custom keyboard navigation to our button by handling the `KeyDown` event:

```
KeyDown += HandleKeyDown;
```

```
private void HandleKeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Keys.Left)
    {
        // Left arrow navigates to previous control
        HandleTab(TabDirection.Up, loop: true);
    }
    if (e.Key == Keys.Right)
    {
        // Right arrow navigates to next control
        HandleTab(TabDirection.Down, loop: true);
    }
}
```

This allows players to navigate between buttons using the left and right arrow keys, providing additional control options beyond the default tab navigation.

Focus Management

We also add a `RollOn` event handler to ensure the button gets focus when the mouse hovers over it:

```
buttonVisual.RollOn += HandleRollOn;


private void HandleRollOn(object sender, EventArgs e)
{
    IsFocused = true;
}
```

This creates a more responsive interface by immediately focusing elements that the player interacts with using the mouse.

The OptionsSlider Class

Now we will create a custom `OptionsSlider` class to style the volume sliders. This class inherits from Gum's base `Slider` class and provides a styled appearance consistent with the game's visual theme.

Unlike `AnimatedButton`, the `OptionsSlider` creates a Visual completely from scratch. This class provides an example for how to completely customize a Forms control by recreating its Visual object entirely. We do this because the desired appearance and behavior of our `OptionsSlider` is differs enough from the existing Slider that it is easier to replace its `Visual` entirely.

In the `UI` folder of the *DungeonSlime* project (your main game project), create a new file called `OptionsSlider.cs` and add the following code to it:

```
using System;
using Gum.DataTypes;
using Gum.DataTypes.Variables;
using Gum.Managers;
using Microsoft.Xna.Framework;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary.Graphics;

namespace DungeonSlime.UI;

/// <summary>
/// A custom slider control that inherits from Gum's Slider class.
/// </summary>
public class OptionsSlider : Slider
{
    // Reference to the text label that displays the slider's title
    private TextRuntime _textInstance;

    // Reference to the rectangle that visually represents the current value
    private ColoredRectangleRuntime _fillRectangle;

    /// <summary>
    /// Gets or sets the text label for this slider.
    /// </summary>
    public string Text
    {
        get => _textInstance.Text;
        set => _textInstance.Text = value;
    }

    /// <summary>
    /// Creates a new OptionsSlider instance using graphics from the specified
texture atlas.
    /// </summary>
    /// <param name="atlas">The texture atlas containing slider graphics.</param>
    public OptionsSlider(TextureAtlas atlas)
    {
        // Create the top-level container for all visual elements
        ContainerRuntime topLevelContainer = new ContainerRuntime();
        topLevelContainer.Height = 55f;
        topLevelContainer.Width = 264f;

        TextureRegion backgroundRegion = atlas.GetRegion("panel-background");

        // Create the background panel that contains everything
```

```
        NineSliceRuntime background = new NineSliceRuntime();
        background.Texture = atlas.Texture;
        background.TextureAddress = TextureAddress.Custom;
        background.TextureHeight = backgroundRegion.Height;
        background.TextureLeft = backgroundRegion.SourceRectangle.Left;
        background.TextureTop = backgroundRegion.SourceRectangle.Top;
        background.TextureWidth = backgroundRegion.Width;
        background.Dock(Gum.Wireframe.Dock.Fill);
        topLevelContainer.AddChild(background);

        // Create the title text element
        _textInstance = new TextRuntime();
        _textInstance.CustomFontFile = @"fonts/04b_30.fnt";
        _textInstance.UseCustomFont = true;
        _textInstance.FontScale = 0.5f;
        _textInstance.Text = "Replace Me";
        _textInstance.X = 10f;
        _textInstance.Y = 10f;
        _textInstance.WidthUnits = DimensionUnitType.RelativeToChildren;
        topLevelContainer.AddChild(_textInstance);

        // Create the container for the slider track and decorative elements
        ContainerRuntime innerContainer = new ContainerRuntime();
        innerContainer.Height = 13f;
        innerContainer.Width = 241f;
        innerContainer.X = 10f;
        innerContainer.Y = 33f;
        topLevelContainer.AddChild(innerContainer);

        TextureRegion offBackgroundRegion = atlas.GetRegion("slider-
off-background");

        // Create the "OFF" side of the slider (left end)
        NineSliceRuntime offBackground = new NineSliceRuntime();
        offBackground.Dock(Gum.Wireframe.Dock.Left);
        offBackground.Texture = atlas.Texture;
        offBackground.TextureAddress = TextureAddress.Custom;
        offBackground.TextureHeight = offBackgroundRegion.Height;
        offBackground.TextureLeft = offBackgroundRegion.SourceRectangle.Left;
        offBackground.TextureTop = offBackgroundRegion.SourceRectangle.Top;
        offBackground.TextureWidth = offBackgroundRegion.Width;
        offBackground.Width = 28f;
        offBackground.WidthUnits = DimensionUnitType.Absolute;
        innerContainer.AddChild(offBackground);

        TextureRegion middleBackgroundRegion = atlas.GetRegion("slider-
```

```
middle-background");

        // Create the middle track portion of the slider
        NineSliceRuntime middleBackground = new NineSliceRuntime();
        middleBackground.Texture = middleBackgroundRegion.Texture;
        middleBackground.TextureAddress = TextureAddress.Custom;
        middleBackground.TextureHeight = middleBackgroundRegion.Height;
        middleBackground.TextureLeft = middleBackgroundRegion.SourceRectangle.Left;
        middleBackground.TextureTop = middleBackgroundRegion.SourceRectangle.Top;
        middleBackground.TextureWidth = middleBackgroundRegion.Width;
        middleBackground.Width = 179f;
        middleBackground.WidthUnits = DimensionUnitType.Absolute;
        middleBackground.Dock(Gum.Wireframe.Dock.Left);
        middleBackground.X = 27f;
        innerContainer.AddChild(middleBackground);

        TextureRegion maxBackgroundRegion = atlas.GetRegion("slider-
max-background");

        // Create the "MAX" side of the slider (right end)
        NineSliceRuntime maxBackground = new NineSliceRuntime();
        maxBackground.Texture = maxBackgroundRegion.Texture;
        maxBackground.TextureAddress = TextureAddress.Custom;
        maxBackground.TextureHeight = maxBackgroundRegion.Height;
        maxBackground.TextureLeft = maxBackgroundRegion.SourceRectangle.Left;
        maxBackground.TextureTop = maxBackgroundRegion.SourceRectangle.Top;
        maxBackground.TextureWidth = maxBackgroundRegion.Width;
        maxBackground.Width = 36f;
        maxBackground.WidthUnits = DimensionUnitType.Absolute;
        maxBackground.Dock(Gum.Wireframe.Dock.Right);
        innerContainer.AddChild(maxBackground);

        // Create the interactive track that responds to clicks
        // The special name "TrackInstance" is required for Slider functionality
        ContainerRuntime trackInstance = new ContainerRuntime();
        trackInstance.Name = "TrackInstance";
        trackInstance.Dock(Gum.Wireframe.Dock.Fill);
        trackInstance.Height = -2f;
        trackInstance.Width = -2f;
        middleBackground.AddChild(trackInstance);

        // Create the fill rectangle that visually displays the current value
        _fillRectangle = new ColoredRectangleRuntime();
        _fillRectangle.Dock(Gum.Wireframe.Dock.Left);
        _fillRectangle.Width = 90f; // Default to 90% - will be updated by
value changes
```

```
        _fillRectangle.WidthUnits = DimensionUnitType.PercentageOfParent;
        trackInstance.AddChild(_fillRectangle);

        // Add "OFF" text to the left end
        TextRuntime offText = new TextRuntime();
        offText.Red = 70;
        offText.Green = 86;
        offText.Blue = 130;
        offText.CustomFontFile = @"fonts/04b_30.fnt";
        offText.FontScale = 0.25f;
        offText.UseCustomFont = true;
        offText.Text = "OFF";
        offText.Anchor(Gum.Wireframe.Anchor.Center);
        offBackground.AddChild(offText);

        // Add "MAX" text to the right end
        TextRuntime maxText = new TextRuntime();
        maxText.Red = 70;
        maxText.Green = 86;
        maxText.Blue = 130;
        maxText.CustomFontFile = @"fonts/04b_30.fnt";
        maxText.FontScale = 0.25f;
        maxText.UseCustomFont = true;
        maxText.Text = "MAX";
        maxText.Anchor(Gum.Wireframe.Anchor.Center);
        maxBackground.AddChild(maxText);

        // Define colors for focused and unfocused states
        Color focusedColor = Color.White;
        Color unfocusedColor = Color.Gray;

        // Create slider state category - Slider.SliderCategoryName is the
required name
        StateSaveCategory sliderCategory = new StateSaveCategory();
        sliderCategory.Name = Slider.SliderCategoryName;
        topLevelContainer.AddCategory(sliderCategory);

        // Create the enabled (default/unfocused) state
        StateSave enabled = new StateSave();
        enabled.Name = FrameworkElement.EnabledStateName;
        enabled.Apply = () =>
        {
            // When enabled but not focused, use gray coloring for all elements
            background.Color = unfocusedColor;
            _textInstance.Color = unfocusedColor;
            offBackground.Color = unfocusedColor;
```

```
            middleBackground.Color = unfocusedColor;
            maxBackground.Color = unfocusedColor;
            _fillRectangle.Color = unfocusedColor;
        };
        sliderCategory.States.Add(enabled);

        // Create the focused state
        StateSave focused = new StateSave();
        focused.Name = FrameworkElement.FocusedStateName;
        focused.Apply = () =>
        {
            // When focused, use white coloring for all elements
            background.Color = focusedColor;
            _textInstance.Color = focusedColor;
            offBackground.Color = focusedColor;
            middleBackground.Color = focusedColor;
            maxBackground.Color = focusedColor;
            _fillRectangle.Color = focusedColor;
        };
        sliderCategory.States.Add(focused);

        // Create the highlighted+focused state by cloning the focused state
        StateSave highlightedFocused = focused.Clone();
        highlightedFocused.Name = FrameworkElement.HighlightedFocusedStateName;
        sliderCategory.States.Add(highlightedFocused);

        // Create the highlighted state by cloning the enabled state
        StateSave highlighted = enabled.Clone();
        highlighted.Name = FrameworkElement.HighlightedStateName;
        sliderCategory.States.Add(highlighted);

        // Assign the configured container as this slider's visual
        Visual = topLevelContainer;

        // Enable click-to-point functionality for the slider
        // This allows users to click anywhere on the track to jump to that value
        IsMoveToPointEnabled = true;

        // Add event handlers
        Visual.RollOn += HandleRollOn;
        ValueChanged += HandleValueChanged;
        ValueChangedByUi += HandleValueChangedByUi;
    }

    /// <summary>
    /// Automatically focuses the slider when the user interacts with it
```

```
        /// </summary>
        private void HandleValueChangedByUi(object sender, EventArgs e)
        {
            IsFocused = true;
        }

        /// <summary>
        /// Automatically focuses the slider when the mouse hovers over it
        /// </summary>
        private void HandleRollOn(object sender, EventArgs e)
        {
            IsFocused = true;
        }

        /// <summary>
        /// Updates the fill rectangle width to visually represent the current value
        /// </summary>
        private void HandleValueChanged(object sender, EventArgs e)
        {
            // Calculate the ratio of the current value within its range
            double ratio = (Value - Minimum) / (Maximum - Minimum);

            // Update the fill rectangle width as a percentage
            // _fillRectangle uses percentage width units, so we multiply by 100
            _fillRectangle.Width = 100 * (float)ratio;
        }
    }
```

The `OptionsSlider` is more complex than then `AnimatedButton` because it contains more visual elements. Below are the key aspects of this implementation:

Slider Components

Walking through the `OptionsSlider` implementation, it consists of several components

1. A background container with a label for the slider.
2. An inner container that holds the slider track.
3. "OFF" and "MAX" section at each end of the slider.
4. A track where the thumb moves.
5. A fill rectangle that shows the current value visually.

Each of these elements is styled to match the game's visual theme using sprites from our atlas.

Custom Text Property

We add a custom `Text` property to set the slider's label:

```
/// Gets or sets the text label for this slider.
/// </summary>
public string Text
{
    get => _textInstance.Text;
    set => _textInstance.Text = value;
}
```

This allows us to easily customize the label for each slider instance we create.

Visual Feedback

The slider uses color changes to provide visual feedback:

```
enabled.Apply = () =>
{
    // When enabled but not focused, use gray coloring for all elements
    background.Color = unfocusedColor;
    _textInstance.Color = unfocusedColor;
    offBackground.Color = unfocusedColor;
    middleBackground.Color = unfocusedColor;
    maxBackground.Color = unfocusedColor;
    _fillRectangle.Color = unfocusedColor;
};
sliderCategory.States.Add(enabled);

// Create the focused state
StateSave focused = new StateSave();
```

```
focused.Apply = () =>
{
    // When focused, use white coloring for all elements
    background.Color = focusedColor;
    _textInstance.Color = focusedColor;
    offBackground.Color = focusedColor;
    middleBackground.Color = focusedColor;
    maxBackground.Color = focusedColor;
    _fillRectangle.Color = focusedColor;
};
sliderCategory.States.Add(focused);

// Create the highlighted+focused state by cloning the focused state
StateSave highlightedFocused = focused.Clone();
```

When the slider is focused, all its elements change from gray to white, making it clear to the player which UI element currently has focus.

Notice that unlike the button which used existing states, we create the slider's states from scratch. These states must use specific names so that Gum knows to use them when the slider gains or loses focus. These names can be obtained through the `FrameworkElement` class. Our slider only needs to handle focused and unfocused (which returns to the `enabled` state), bug Gum provides additional states if needed.

Fill Visualization

One of the most important aspects of a slider is the visual representation of its value. We achieve this by updating the width of the `_fillRectangle` element:

```
    private void HandleValueChanged(object sender, EventArgs e)
    {
        // Calculate the ratio of the current value within its range
        double ratio = (Value - Minimum) / (Maximum - Minimum);

        // Update the fill rectangle width as a percentage
        // _fillRectangle uses percentage width units, so we multiply by 100
        _fillRectangle.Width = 100 * (float)ratio;
    }
}
```

This method converts the slider's current value to a percentage and applies it to the fill rectangle's width, creating a visual indicator of the current setting.

Updating the Scenes to Use Custom Controls

Now that we have created our custom controls, we need to update our game scenes to use them instead of the default Gum controls.

Updating the TitleScene

First, open the `TitleScene.cs` file in the game project and add the following using declaration to the top of the `TitleScene` class:

```
using System;
using DungeonSlime.UI;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameGum;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Scenes;
```

Next, update both the `_optionsButton` and the `_optionsBackButton` fields to be of our new `AnimatedButton` type, and add a new field to store a reference to the texture atlas in.

```
// Existing fields...

// The options button used to open the options menu.
private AnimatedButton _optionsButton;

// The back button used to exit the options menu back to the title menu.
private AnimatedButton _optionsBackButton;

// Reference to the texture atlas that we can pass to UI elements when they
// are created.
private TextureAtlas _atlas;
```

Next, in the `LoadContent` method, we need to update it so that it loads the texture atlas from the XML configuration file and stores it in the new `_atlas` field:

```
public override void LoadContent()
{
    // Load the font for the standard text.
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

    // Load the font for the title text
    _font5x = Content.Load<SpriteFont>("fonts/04B_30_5x");

    // Load the background pattern texture.
    _backgroundPattern = Content.Load<Texture2D>("images/background-pattern");

    // Load the sound effect to play when ui actions occur.
    _uiSoundEffect = Core.Content.Load<SoundEffect>("audio/ui");

    // Load the texture atlas from the xml configuration file.
    _atlas = TextureAtlas.FromFile(Core.Content, "images/atlas-definition.xml");
}
```

Next, update the `CreateTitlePanel` method so that instead of using the default Gum `Button` Forms controls it now uses our custom `AnimatedButton` control and remove the explicit setting of the `Visual.Width` property since this is managed by the `AnimatedButton` now:

```
private void CreateTitlePanel()
{
    // Create a container to hold all of our buttons
    _titleScreenButtonsPanel = new Panel();
    _titleScreenButtonsPanel.Dock(Gum.Wireframe.Dock.Fill);
    _titleScreenButtonsPanel.AddToRoot();

    AnimatedButton startButton = new AnimatedButton(_atlas);
    startButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);
    startButton.X = 50;
    startButton.Y = -12;
    startButton.Text = "Start";
    startButton.Click += HandleStartClicked;
    _titleScreenButtonsPanel.AddChild(startButton);

    _optionsButton = new AnimatedButton(_atlas);
    _optionsButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
    _optionsButton.X = -50;
    _optionsButton.Y = -12;
    _optionsButton.Text = "Options";
    _optionsButton.Click += HandleOptionsClicked;
    _titleScreenButtonsPanel.AddChild(_optionsButton);

    startButton.IsFocused = true;
}
```

Finally, update the `CreateOptionsPanel` method so that:

- It uses a `TextRuntime` to display the text "OPTIONS" using the bitmap font
- Instead of using the default Gum `Button` and `Slider` Forms controls, it now uses our custom `AnimatedButton` and `OptionsSlider` controls.
- Both the `musicSlider` and `sfxSlider` have been given `Name` and `Text` properties.

```
private void CreateOptionsPanel()
{
    _optionsPanel = new Panel();
    _optionsPanel.Dock(Gum.Wireframe.Dock.Fill);
    _optionsPanel.IsVisible = false;
    _optionsPanel.AddToRoot();

    TextRuntime optionsText = new TextRuntime();
    optionsText.X = 10;
    optionsText.Y = 10;
    optionsText.Text = "OPTIONS";
    optionsText.UseCustomFont = true;
    optionsText.FontScale = 0.5f;
    optionsText.CustomFontFile = @"fonts/04b_30.fnt";
    _optionsPanel.AddChild(optionsText);

    OptionsSlider musicSlider = new OptionsSlider(_atlas);
    musicSlider.Name = "MusicSlider";
    musicSlider.Text = "MUSIC";
    musicSlider.Anchor(Gum.Wireframe.Anchor.Top);
    musicSlider.Y = 30f;
    musicSlider.Minimum = 0;
    musicSlider.Maximum = 1;
    musicSlider.Value = Core.Audio.SongVolume;
    musicSlider.SmallChange = .1;
    musicSlider.LargeChange = .2;
    musicSlider.ValueChanged += HandleMusicSliderValueChanged;
    musicSlider.ValueChangeCompleted += HandleMusicSliderValueChangeCompleted;
    _optionsPanel.AddChild(musicSlider);

    OptionsSlider sfxSlider = new OptionsSlider(_atlas);
    sfxSlider.Name = "SfxSlider";
    sfxSlider.Text = "SFX";
    sfxSlider.Anchor(Gum.Wireframe.Anchor.Top);
    sfxSlider.Y = 93;
    sfxSlider.Minimum = 0;
    sfxSlider.Maximum = 1;
    sfxSlider.Value = Core.Audio.SoundEffectVolume;
    sfxSlider.SmallChange = .1;
    sfxSlider.LargeChange = .2;
    sfxSlider.ValueChanged += HandleSfxSliderChanged;
    sfxSlider.ValueChangeCompleted += HandleSfxSliderChangeCompleted;
    _optionsPanel.AddChild(sfxSlider);

    _optionsBackButton = new AnimatedButton(_atlas);
    _optionsBackButton.Text = "BACK";
```

```
    _optionsBackButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
    _optionsBackButton.X = -28f;
    _optionsBackButton.Y = -10f;
    _optionsBackButton.Click += HandleOptionsButtonBack;
    _optionsPanel.AddChild(_optionsBackButton);
}
```

Updating the GameScene

Next, open the `GameScene.cs` file in the game project and add the following using declaration to the top of the `GameScene` class:

```
using System;
using DungeonSlime.UI;
using Gum.DataTypes;
using Gum.Managers;
using Gum.Wireframe;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MonoGameGum;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Input;
using MonoGameLibrary.Scenes;
```

Next, update the `_resumeButton` field to be of our new `AnimatedButton` type and add a field to store a reference to the texture atlas in.

```
// Existing fields...

// A reference to the resume button UI element so we can focus it
// when the game is paused.
private AnimatedButton _resumeButton;

// The UI sound effect to play when a UI event is triggered.
private SoundEffect _uiSoundEffect;

// Reference to the texture atlas that we can pass to UI elements when they
// are created.
private TextureAtlas _atlas;
```

Next, in the `LoadContent` method, we need to update it so that it stores the texture atlas once loaded in the new `_atlas` field.

```
public override void LoadContent()
{
    // Create the texture atlas from the XML configuration file
    _atlas = TextureAtlas.FromFile(Core.Content, "images/atlas-definition.xml");

    // Create the slime animated sprite from the atlas.
    _slime = _atlas.CreateAnimatedSprite("slime-animation");
    _slime.Scale = new Vector2(4.0f, 4.0f);

    // Create the bat animated sprite from the atlas.
    _bat = _atlas.CreateAnimatedSprite("bat-animation");
    _bat.Scale = new Vector2(4.0f, 4.0f);

    // Create the tilemap from the XML configuration file.
    _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
    _tilemap.Scale = new Vector2(4.0f, 4.0f);

    // Load the bounce sound effect
    _bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

    // Load the collect sound effect
    _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

    // Load the font
    _font = Core.Content.Load<SpriteFont>("fonts/04B_30");

    // Load the sound effect to play when ui actions occur.
    _uiSoundEffect = Core.Content.Load<SoundEffect>("audio/ui");
}
```

Finally, update the `CreatePausePanel` method so that

1. Instead of using a `ColoredRectangleRuntime` for the background of the pause panel, it now uses a `NineSliceRuntime` that uses the sprite from the texture atlas.
2. The `textInstance` is updated so that it uses the custom bitmap font file.
3. The `_resumeButton` and `quiteButton` are updated to use our custom `AnimatedButton` control instead of the default Gum `Button` Forms control.

```
private void CreatePausePanel()
{
    _pausePanel = new Panel();
    _pausePanel.Anchor(Anchor.Center);
    _pausePanel.WidthUnits = DimensionUnitType.Absolute;
    _pausePanel.HeightUnits = DimensionUnitType.Absolute;
    _pausePanel.Height = 70;
    _pausePanel.Width = 264;
    _pausePanel.IsVisible = false;
    _pausePanel.AddToRoot();

    TextureRegion backgroundRegion = _atlas.GetRegion("panel-background");

    NineSliceRuntime background = new NineSliceRuntime();
    background.Dock(Dock.Fill);
    background.Texture = backgroundRegion.Texture;
    background.TextureAddress = TextureAddress.Custom;
    background.TextureHeight = backgroundRegion.Height;
    background.TextureLeft = backgroundRegion.SourceRectangle.Left;
    background.TextureTop = backgroundRegion.SourceRectangle.Top;
    background.TextureWidth = backgroundRegion.Width;
    _pausePanel.AddChild(background);

    TextRuntime textInstance = new TextRuntime();
    textInstance.Text = "PAUSED";
    textInstance.CustomFontFile = @"fonts/04b_30.fnt";
    textInstance.UseCustomFont = true;
    textInstance.FontScale = 0.5f;
    textInstance.X = 10f;
    textInstance.Y = 10f;
    _pausePanel.AddChild(textInstance);

    _resumeButton = new AnimatedButton(_atlas);
    _resumeButton.Text = "RESUME";
    _resumeButton.Anchor(Anchor.BottomLeft);
    _resumeButton.X = 9f;
    _resumeButton.Y = -9f;
    _resumeButton.Click += HandleResumeButtonClicked;
    _pausePanel.AddChild(_resumeButton);

    AnimatedButton quitButton = new AnimatedButton(_atlas);
    quitButton.Text = "QUIT";
    quitButton.Anchor(Anchor.BottomRight);
    quitButton.X = -9f;
    quitButton.Y = -9f;
    quitButton.Click += HandleQuitButtonClicked;
```

```
    _pausePanel.AddChild(quitButton);
}
```

Testing the Styled UI

When you run the game now, you will see a dramatic improvement in the visual appearance of the UI:

1. The buttons now use our custom animated background that pulses when focused.
2. The sliders have a cleaner, mores stylized appearance with the OFF and MAX labels.
3. All text uses our custom bitmap font.
4. Visual feedback clearly indicates which element has focus.

Figure 21-3: The game using Gum now with custom styled UI components

The entire UI now has a cohesive style that matches the rest of the game.

Conclusion

In this chapter, you learned how to transform basic UI components into custom, styled elements that match the game's visual theme. You explored several key aspects of UI customization:

- How container hierarchies and size relationships work in Gum.
- Creating animation chains for visual feedback.
- Using the state system to respond to user interactions.
- Building complex custom controls by extending base classes.
- Integrating custom fonts and graphics from a texture atlas.

By creating reusable custom controls, you have not only improved the look of your game, but you have also developed components that can be used in future projects. This approach of separating functionality from appearance allows you to maintain consistent behavior while completely changing the visual style to match different games.

The principles you have learned in this chapter extend beyond the specific components we created. You can apply the same techniques to create other custom UI elements like checkboxes, radio buttons, scroll panels, and more. By understanding how to build on Gum's foundation, you have the tools to create any UI component your game might need.

Test Your Knowledge

1. What are the two main approaches to customizing visuals in Gum, and when would you use each one?

   The two main approaches are:

   - **Direct property assignment**: Setting properties directly in code (like `button.Width = 100`). This approach is best for initial setup of UI elements and static properties that do not change during gameplay.
   - **States (StateSave objects)**: Defining different visual states that are applied automatically in response to interactions. This approach is best for dynamic changes that happen during gameplay, like highlighting a button when it is focused or changing colors when a slider is adjusted.

2. What is the purpose of using a top-level container in a custom Gum control?

   A top-level container in a custom Gum control serves several purposes:

   - It provides a single parent element that holds all visual components of the control.
   - It establishes the coordinate system for positioning child elements.
   - It can manage the overall size of the control (often using `RelativeToChildren` sizing).
   - It serves as the attachment point for states and categories.
   - It creates a clear separation between the control's visuals and its functionality.

3. How do animation chains work in Gum, and what are the key components needed to create one?

Animation chains in Gum work by displaying a sequence of frames in order, typically looping after the last frame. The key components needed to create an animation chain are:

1. An `AnimationChain` object to hold the sequence of frames
2. Multiple `AnimationFrame` objects, each with:
    - Texture coordinates (left, right, top, bottom) defining which part of the texture to display
    - A frame length value determining how long to display the frame
    - A reference to the texture where the frame appears
3. A method to add the animation to a visual element (like assigning to a NineSliceRuntime's CurrentChainName)

The animation system uses normalized texture coordinates (0.0 to 1.0) rather than pixel coordinates.

4. What is the relationship between Gum's state system and Forms controls, and why is it important?

Gum's state system links with Forms controls through specifically named categories and states:

- Each Forms control type has states specific to that type which are accessed through a casted visual (e.g., `buttonVisual.States.Enabled`)
- When the control's state changes (like gaining focus), it automatically applies the corresponding visual state

This relationship is important because it:

- Separates the control's functionality from its appearance
- Enables consistent behavior while allowing complete visual customization
- Provides automatic visual feedback in response to user interactions without requiring manual state management
- Makes it easier to create controls that work with mouse, keyboard, and gamepad input.

Chapter 22: Snake Game Mechanics

Learn how to implement classic snake-like game mechanics and organize game objects into reusable components.

In the previous chapters, we have built all the fundamental systems needed for our game: graphics, input, collision detection, audio, scene management, and a user interface. Now it is time to transform our demo into a complete experience by implementing classic snake-like game mechanics. Before we do that, we first need to define what mechanics make a snake game.

In this chapter, you will:

- Understand the core mechanics that define a classic snake-like game.
- Learn how to implement grid-based movement with timed intervals.
- Create a segmented character that grows when collecting objects.
- Implement a unified input controller for game actions.
- Build the `SlimeSegment` struct for storing segment data.
- Create the `Slime` class to manage all snake-like behavior for the slime.
- Create the `Bat` class as the collectable object.

> **NOTE**
>
> This chapter will not focus much on MonoGame itself, but rather our implementation of the mechanics to transform our current game into a snake-like game.

Understanding Snake Game Mechanics

In a classic snake-like game, the mechanics follow a set of simple but engaging rules:

1. The player controls a snake by telling it to move in one of four cardinal directions (up, down, left, and right).
2. The snake cannot reverse into itself, only moving forward or perpendicular to its current direction.
3. The actual movement of the snake occurs at regular timed intervals, creating a grid-based movement pattern.
4. When the snake eats food, it grows longer by adding a new segment to its tail.
5. If the snake collides with a wall or its own body, the game ends.

The mechanics create an increasingly challenging experience as the snake grows longer, requiring planning and reflexes to avoid collision.

Directions

In snake, players input a cardinal direction (up, down, left, and right), to indicate which direction the snake will move during the next movement cycle. When direction input occurs, it must be checked against the current direction to determine if the move is valid.

For example, if the snake is moving to the right, an invalid input would allow a player to move it to the left. Doing so would cause the head of the snake to reverse direction and immediately collide with the first body segment. This means the only valid inputs are those where the next direction would be the same as the current direction or perpendicular to the current direction.

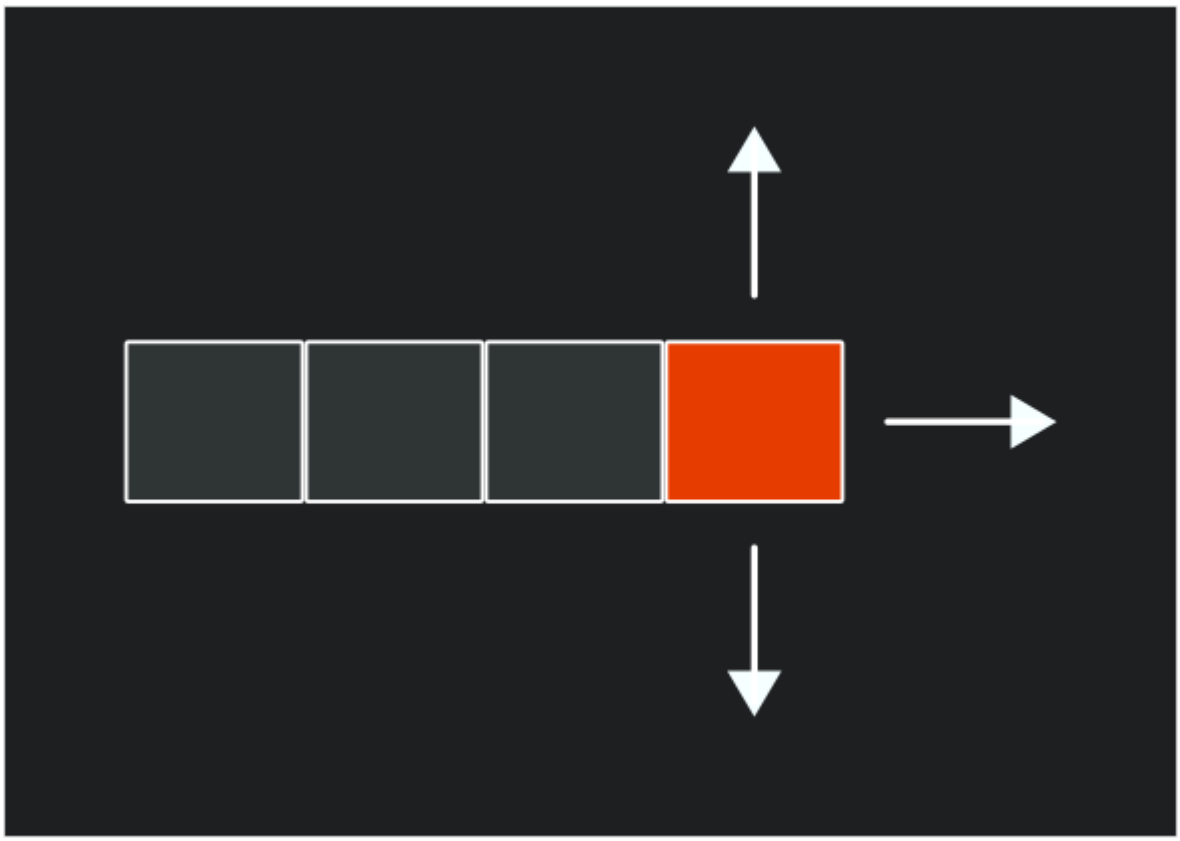


Figure 22-1: An example snake with four segments, the head segment highlighted in orange, moving to the right. Arrows show that the only valid movements for the head segment are up or down (perpendicular) or to continue to the right.

Movement Cycle

Instead of moving every update frame while a directional input is being pressed, the snake moves only at fixed time intervals. A timer is used to determine how much time has passed since the last movement cycle, and when it reaches a set threshold, the next movement cycle occurs. During this movement cycle, the snake should move forward in the direction that was input by the player between the last and current movement cycles. This creates the grid-based movement system typically found in snake-like games.

There are various methods for handling the movement, such as iterating through each segment of the snake and updating the position of that segment to move forward. Methods such as this though are wasteful, since visually the only parts of the snake that move on the screen are the head and the tail.

Instead, a more common approach is to:

1. Make a copy of the head segment.

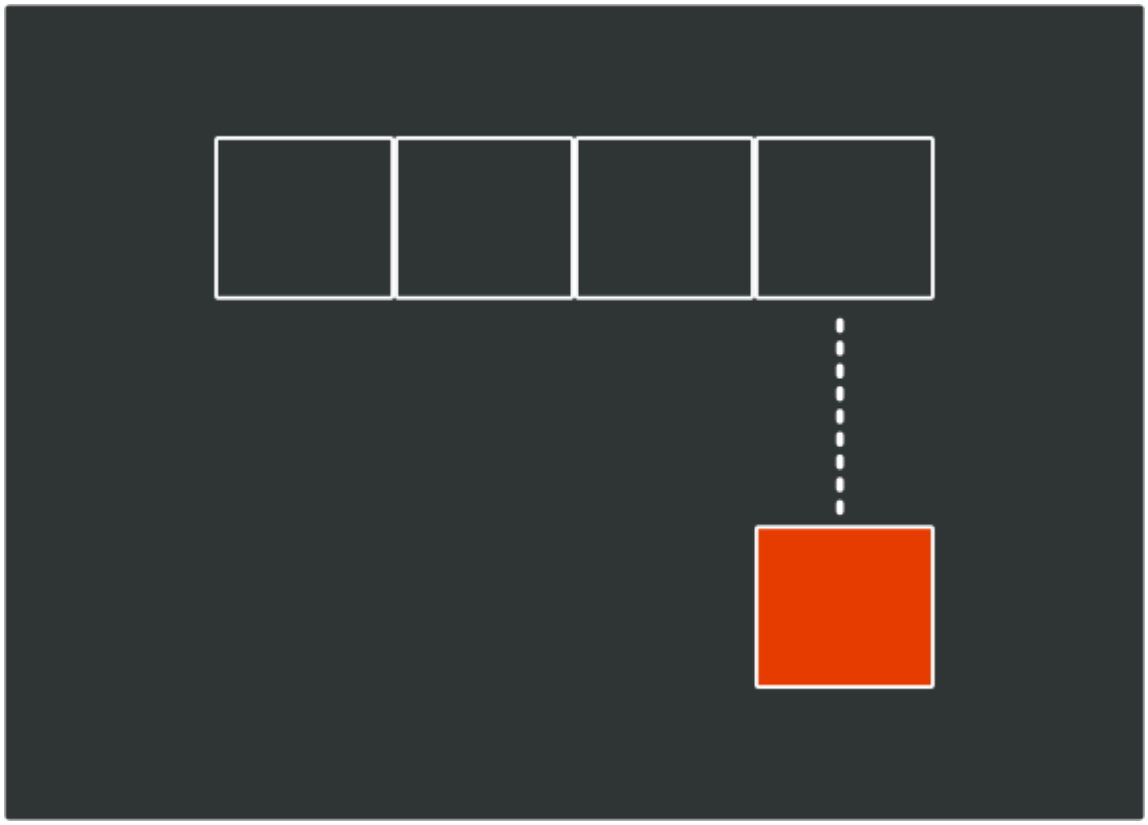


Figure 22-2 From a snake with four segments, a copy of the head segment is made, represented by the orange block

2. Update the properties of the copy so that it is positioned where the original head segment would have moved to.

3. Insert the copy at the front of the segment collection.

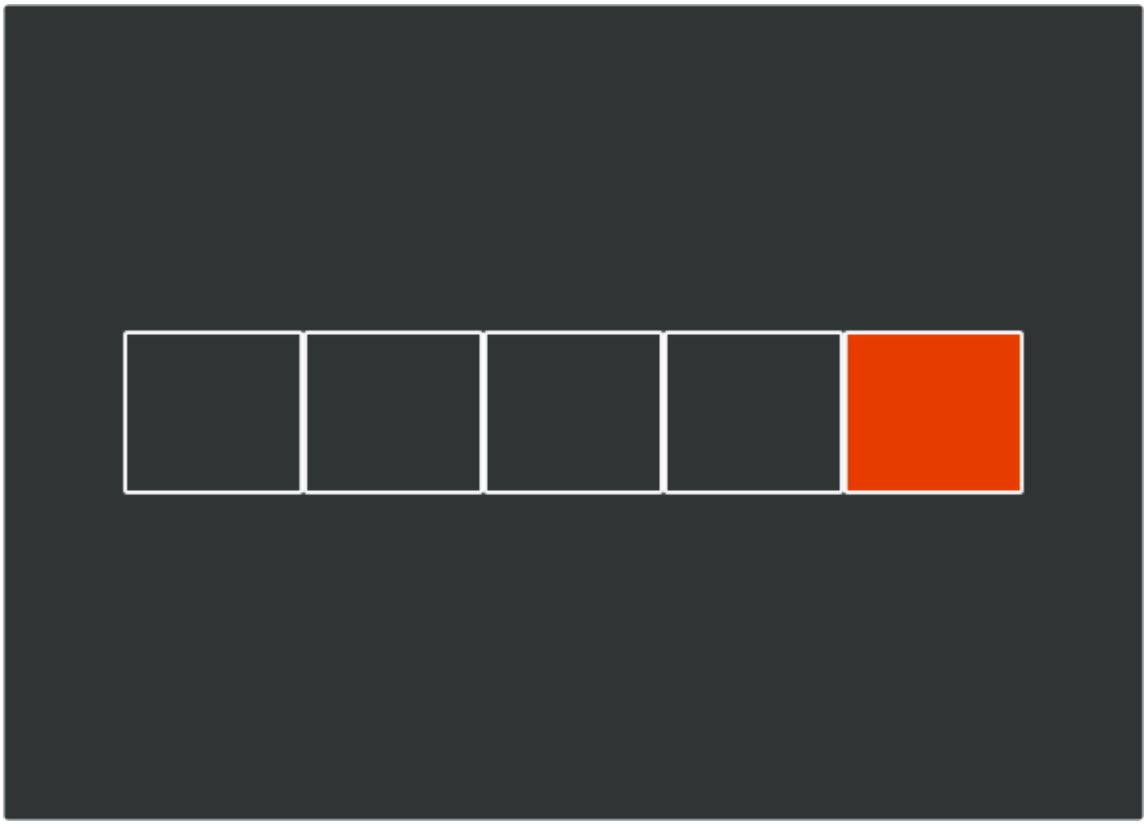


Figure 22-3: The copy of the head segment, represented by the orange block, is inserted at the front of the segment collection as the new head, which now makes it five segments (one too many)

4. Remove the tail segment.

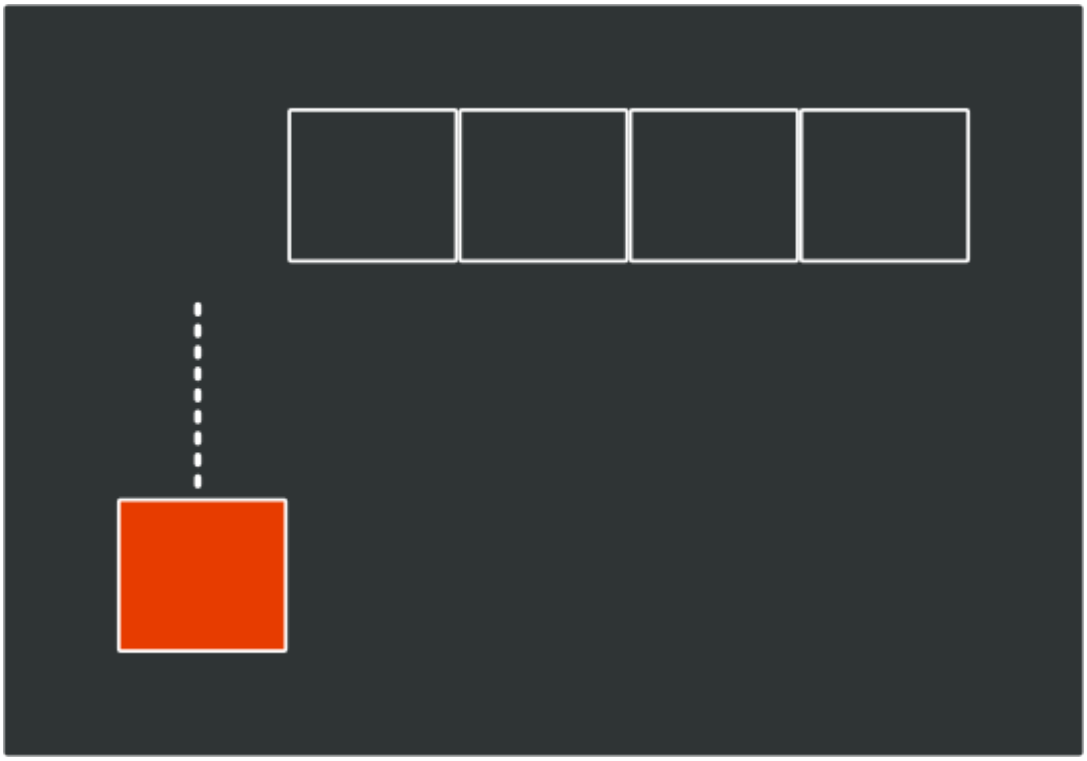


Figure 22-4: The tail segment of the snake is removed, bringing it back to the original four segments, giving the illusion that the entire snake moved forward

By doing this, no matter how many segments the snake body has, we only ever need to update two of them, the head and the tail.

Growth

The next core mechanic for a snake-like game is the growth of the snake. Typically, somewhere in the game is an objective for the snake to eat, such as an apple. When the head of the snake reaches this objective, a new segment is added to the snake's body, making it longer, the player's score is increased, and a new objective is spawned at a random location within the game.

This mechanic also acts as an artificial difficulty curve for the game. As the body of the snake grows longer, it requires planning to avoid creating situations where the snake becomes trapped by its own body making it impossible to avoid a collision.

Game Over

The challenge in a snake-like game is to avoid colliding with either a wall or another segment of the snake body. Doing so will cause a game over condition as the snake can no longer continue moving forward.

Implementing Snake-Like Mechanics In Our Game

Now that we have a foundational set of rules in place for the mechanics of a snake-like game, we will implement them into the current demo we have been building up. Our game will adapt these mechanics to fit them into our existing game structure:

- Instead of a snake, we will use the slime and create a chain of slime segments that follow the slime at the front.
- The player will control the direction of the slime segment
- The bat will serve as the objective to acquire to grow the slime segment.
- Collisions with either the walls of the room or a slime segment will result in a game over state.

As we implement these mechanics, we are also going to be creating classes that encapsulate the properties and functions of existing implementations in the game scene, such as the slime and the bat. For example, currently, the game scene tracks fields for the `AnimatedSprite` and the position of the slime, as well as updating, input handling, and drawing the slime. These can be moved into their dedicated classes encapsulating the functionality and also cleaning up the code in the game scene that has grown quite large.

The GameController Class

Currently, we have two methods dedicated to handling input in the game scene, `CheckKeyboardInput` and `CheckGamePadInput`, both of these methods essentially perform the same logic across different input devices. This presents an opportunity to improve our code.

To simplify input handling for the game, we can create a dedicated class that consolidates the input methods, providing a unified input profile for the game. This pattern is widely used in game development to separate the "what" (game actions) from the "how" (specific input devices and buttons).

Create a new file named `GameController.cs` in the root of the `DungeonSlime` project (your main game project) and add the following code:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
using MonoGameLibrary;
using MonoGameLibrary.Input;

namespace DungeonSlime;

/// <summary>
/// Provides a game-specific input abstraction that maps physical inputs
/// to game actions, bridging our input system with game-specific functionality.
/// </summary>
public static class GameController
{
    private static KeyboardInfo s_keyboard => Core.Input.Keyboard;
    private static GamePadInfo s_gamePad =>
Core.Input.GamePads[(int)PlayerIndex.One];

    /// <summary>
    /// Returns true if the player has triggered the "move up" action.
    /// </summary>
    public static bool MoveUp()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Up) ||
               s_keyboard.WasKeyJustPressed(Keys.W) ||
               s_gamePad.WasButtonJustPressed(Buttons.DPadUp) ||
               s_gamePad.WasButtonJustPressed(Buttons.LeftThumbstickUp);
    }

    /// <summary>
    /// Returns true if the player has triggered the "move down" action.
    /// </summary>
    public static bool MoveDown()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Down) ||
               s_keyboard.WasKeyJustPressed(Keys.S) ||
               s_gamePad.WasButtonJustPressed(Buttons.DPadDown) ||
               s_gamePad.WasButtonJustPressed(Buttons.LeftThumbstickDown);
    }

    /// <summary>
    /// Returns true if the player has triggered the "move left" action.
    /// </summary>
    public static bool MoveLeft()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Left) ||
               s_keyboard.WasKeyJustPressed(Keys.A) ||
```

```
                s_gamePad.WasButtonJustPressed(Buttons.DPadLeft) ||
                s_gamePad.WasButtonJustPressed(Buttons.LeftThumbstickLeft);
    }

    /// <summary>
    /// Returns true if the player has triggered the "move right" action.
    /// </summary>
    public static bool MoveRight()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Right) ||
               s_keyboard.WasKeyJustPressed(Keys.D) ||
               s_gamePad.WasButtonJustPressed(Buttons.DPadRight) ||
               s_gamePad.WasButtonJustPressed(Buttons.LeftThumbstickRight);
    }

    /// <summary>
    /// Returns true if the player has triggered the "pause" action.
    /// </summary>
    public static bool Pause()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Escape) ||
               s_gamePad.WasButtonJustPressed(Buttons.Start);
    }

    /// <summary>
    /// Returns true if the player has triggered the "action" button,
    /// typically used for menu confirmation.
    /// </summary>
    public static bool Action()
    {
        return s_keyboard.WasKeyJustPressed(Keys.Enter) ||
               s_gamePad.WasButtonJustPressed(Buttons.A);
    }
}
```

The `GameController` class implements an important design pattern in game development known as **"Input Abstraction"** or the **"Command"** pattern. This pattern separates what happens in the game (the actions) from how players trigger those actions (the inputs).

This separation provides several benefits, including:

1. **Input Device Independence**: The game logic does not need to know which input device the player is using. Whether they are playing with a keyboard, gamepad, or touch screen, the game only cares that a "move up" action was triggered, not which specific button or key caused it.
2. **Simplified Input Handling**: Instead of checking multiple input combinations throughout the codebase, game objects can simply ask "Should I move up?" through a clean API call.
3. **Easy Rebinding**: If you want to add key rebinding features, you only need to modify the `GameController` class, not every piece of code that uses input.
4. **Consistent Input Logic**: The rules for determining if an action occurred (like checking if a button was just pressed versus being held down) are defined in one place.
5. **Cross-Platform Compatibility**: When porting to different platforms with different input methods, you only need to update the `GameController` class to map the new input devices to your existing game actions.

By implementing this pattern in our game, we are not only making our current input handling cleaner, but we are also establishing a foundation that would make it easier to add features like input customization or support for new input devices in the future.

With our input handling system in place, now we can turn our attention to implementing the core mechanics of our snake-like game. First, we need to create a structure that will represent each segment of the slime's body.

The SlimeSegment Struct

We will need to implement a structure that can represent each segment of the slime, this structure will store the position and movement data for each segment.

In the *DungeonSlime* project (your main game project), create a new folder named `GameObjects`. We will be putting all of our code related to the objects within the game here. Then create a new file named `SlimeSegment.cs` inside the `GameObjects` folder you just created and add the following code:

```
using Microsoft.Xna.Framework;

namespace DungeonSlime.GameObjects;

public struct SlimeSegment
{
    /// <summary>
    /// The position this slime segment is at before the movement cycle occurs.
    /// </summary>
    public Vector2 At;

    /// <summary>
    /// The position this slime segment should move to during the next
movement cycle.
    /// </summary>
    public Vector2 To;

    /// <summary>
    /// The direction this slime segment is moving.
    /// </summary>
    public Vector2 Direction;

    /// <summary>
    /// The opposite direction this slime segment is moving.
    /// </summary>
    public Vector2 ReverseDirection => new Vector2(-Direction.X, -Direction.Y);
}
```

This structure contains fields to track:

- `At`: The current position of the segment.
- `To`: The position the segment will move to during the next movement cycle if it is the head segment.
- `Direction`: A normalized vector representing the direction the segment is moving in.
- `ReverseDirection`: A computed property that returns the opposite of the `Direction` property.

> **NOTE**
>
> We are implementing this as a struct rather than a class because SlimeSegment is a small, simple data container with value semantics. Structs are more efficient for small data structures since they are allocated on the stack rather than the heap, reducing garbage collection overhead. Since our game will potentially create many segments as the snake grows, using a struct can provide better performance, especially when we will be copying segment data during movement operations.

> **IMPORTANT**
>
> Structs work best with value types (like int, float, **Vector2**); using reference types in structs can cause boxing operations that negate the performance benefits. For more information on structs, refer to the Structure Types - C# Reference documentation on Microsoft Learn.

By tracking both the current (`At`) and target (`To`) positions, we can implement smooth visual movement between grid positions, creating more fluid animations than the traditional stutter step movement seen in classic snake games. We will explore this concept a little further in this chapter.

The Slime Class

Next, we can implement a class to encapsulate the properties and functionality of our snake-like slime.

In the `GameObjects` folder of the *DungeonSlime* project (your main game project), create a new file named `Slime.cs` and add the following initial code:

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime.GameObjects;

public class Slime
{

}
```

This code sets up the basic structure for our `Slime` class. We have added the necessary using statements to access MonoGame's framework components and placed the class in the `DungeonSlime.GameObjects` namespace to keep our code organized. The empty class will serve as our foundation, and we will build it up piece by piece in the following sections.

Each section below should be added to the `Slime` class in the order presented. As we go through each part, the class will gradually take shape to handle all the snake-like behavior we need.

> **NOTE**
> When adding these sections one by one, you may see compiler errors until all sections are in place. This is normal, as some parts of the code will reference fields or methods that have not been added yet. Once all sections are complete, these errors will resolve.

Slime Fields

Add the following fields to the `Slime` class:

```
// A constant value that represents the amount of time to wait between
// movement updates.
private static readonly TimeSpan s_movementTime = TimeSpan.FromMilliseconds(200);

// The amount of time that has elapsed since the last movement update.
private TimeSpan _movementTimer;

// Normalized value (0-1) representing progress between movement ticks for
visual interpolation
private float _movementProgress;

// The next direction to apply to the head of the slime chain during the
// next movement update.
private Vector2 _nextDirection;

// The number of pixels to move the head segment during the movement cycle.
private float _stride;

// Tracks the segments of the slime chain.
private List<SlimeSegment> _segments;

// The AnimatedSprite used when drawing each slime segment
private AnimatedSprite _sprite;
```

Each of these fields is responsible for:

- `s_movementTime`: This constant represents how long the slime waits between movement cycles (300ms). This creates the classic snake game's grid-based movement feel, where the snake moves at regular intervals rather than continuously.
- `_movementTime`: This field accumulates elapsed time until it reaches the movement threshold. When it does, the slime moves one grid cell and the timer resets.
- `_movementProgress`: This normalized value (0-1) represents progress between movement ticks and is used for visual interpolation. It allows us to smoothly animate the slime's movement between grid positions.
- `_nextDirection`: This stores the direction that will be applied to the head segment during the next movement cycle.
- `_stride`: This represents the total number of pixels the head segment should move during movement cycles.
- `_segments`: This collection holds all the `SlimeSegment` structures that make up the slime's body. The first segment is the head, and the rest form the trailing body.
- `_sprite`: This stores the `AnimatedSprite` that is used to draw each segment of the slime.

These fields implement core snake-like mechanics - the timed interval movement, direction control, and the segmented body that forms the snake.

Slime Events

Next, add the following event to the `Slime` class after the fields:

```
/// <summary>
/// Event that is raised if it is detected that the head segment of the slime
/// has collided with a body segment.
/// </summary>
public event EventHandler BodyCollision;
```

This event will allow the `Slime` class to notify the game scene when the head of the slime collides with another segment, triggering a game over.

Slime Constructor

After the event, add the following constructor to the `Slime` class:

```
/// <summary>
/// Creates a new Slime using the specified animated sprite.
/// </summary>
/// <param name="sprite">The AnimatedSprite to use when drawing the slime.</param>
public Slime(AnimatedSprite sprite)
{
    _sprite = sprite;
}
```

This is a simple constructor that requires the slime to be given the `AnimatedSprite` that will be used to draw each of the slime segments.

Slime Initialization

Add the following `Initialization` method to the `Slime` class after the constructor:

```
/// <summary>
/// Initializes the slime, can be used to reset it back to an initial state.
/// </summary>
/// <param name="startingPosition">The position the slime should start at.</param>
/// <param name="stride">The total number of pixels to move the head segment during
each movement cycle.</param>
public void Initialize(Vector2 startingPosition, float stride)
{
    // Initialize the segment collection.
    _segments = new List<SlimeSegment>();

    // Set the stride
    _stride = stride;

    // Create the initial head of the slime chain.
    SlimeSegment head = new SlimeSegment();
    head.At = startingPosition;
    head.To = startingPosition + new Vector2(_stride, 0);
    head.Direction = Vector2.UnitX;

    // Add it to the segment collection.
    _segments.Add(head);

    // Set the initial next direction as the same direction the head is
    // moving.
    _nextDirection = head.Direction;

    // Zero out the movement timer.
    _movementTimer = TimeSpan.Zero;
}
```

With this method, we can initialize, or reset the state of slime. It:

- Instantiates a new segment collection.
- Creates the initial head segment and positions it at the specific `startingPosition`.
- Sets the initial direction to be to the right.
- Initialize the movement timer to zero.

Slime Input Handling

Next, add the `HandleInput` method to process player input after the `Initialize` method:

```
private void HandleInput()
{
    Vector2 potentialNextDirection = _nextDirection;

    if (GameController.MoveUp())
    {
        potentialNextDirection = -Vector2.UnitY;
    }
    else if (GameController.MoveDown())
    {
        potentialNextDirection = Vector2.UnitY;
    }
    else if (GameController.MoveLeft())
    {
        potentialNextDirection = -Vector2.UnitX;
    }
    else if (GameController.MoveRight())
    {
        potentialNextDirection = Vector2.UnitX;
    }

    // Only allow direction change if it is not reversing the current
    // direction.  This prevents the slime from backing into itself.
    float dot = Vector2.Dot(potentialNextDirection, _segments[0].Direction);
    if (dot >= 0)
    {
        _nextDirection = potentialNextDirection;
    }
}
```

This method implements the following:

1. Determine if the player is attempting to change directions instead of directly moving the slime. This direction change will be applied later during the movement cycle update.
2. Uses **Vector2.Dot** to prevent the slime from reversing into itself, causing an immediate collision and game over state.
3. Updates the `_nextDirection` value only if the direction input is valid.

> **NOTE**
>
> The dot product measures how much two vectors point in the same direction. It is:
>
> - Positive if they are pointing in the same direction.
> - Negative if they are pointing in opposite directions.
> - Zero when they are perpendicular.
>
> By using dot product here, this effectively implements the classic snake-like game rule that you cannot turn 180° into yourself.

Slime Movement Cycle

To handle the snake-like movement cycle of the slime, we will create a method called `Move`.

Add the following method to the `Slime` class after the `HandleInput` method:

```
private void Move()
{
    // Capture the value of the head segment
    SlimeSegment head = _segments[0];

    // Update the direction the head is supposed to move in to the
    // next direction cached.
    head.Direction = _nextDirection;

    // Update the head's "at" position to be where it was moving "to"
    head.At = head.To;

    // Update the head's "to" position to the next tile in the direction
    // it is moving.
    head.To = head.At + head.Direction * _stride;

    // Insert the new adjusted value for the head at the front of the
    // segments and remove the tail segment. This effectively moves
    // the entire chain forward without needing to loop through every
    // segment and update its "at" and "to" positions.
    _segments.Insert(0, head);
    _segments.RemoveAt(_segments.Count - 1);

    // Iterate through all of the segments except the head and check
    // if they are at the same position as the head. If they are, then
    // the head is colliding with a body segment and a body collision
    // has occurred.
    for (int i = 1; i < _segments.Count; i++)
    {
        SlimeSegment segment = _segments[i];

        if (head.At == segment.At)
        {
            if(BodyCollision != null)
            {
                BodyCollision.Invoke(this, EventArgs.Empty);
            }

            return;
        }
    }
}
```

This method performs the core snake-like movement cycle logic by:

1. Copying the value of the current head segment.
2. Updating the copy's position (`At`) to where the head was moving to and updating the position it is moving to (`To`).
3. Insert the copy into the front of the segment collection and remove the tail.
4. Check if the head is now in the same position as any body segments, which would cause a collision and trigger a game over.

> **NOTE**
>
> By inserting a new head segment at the front of the chain and removing the last segment, this creates the illusion of the entire chain moving forward as one, even though we are only actually moving the head forward and removing the tail.
>
> This follows the common snake movement pattern as discussed in the Understanding Snake Game Mechanics: Movement Cycle section above.

Slime Growth

To handle the snake-like growth of the slime, we will create a new method called `Grow`.

Add the following method to the `Slime` class after the `Move` method:

```
/// <summary>
/// Informs the slime to grow by one segment.
/// </summary>
public void Grow()
{
    // Capture the value of the tail segment
    SlimeSegment tail = _segments[_segments.Count - 1];

    // Create a new tail segment that is positioned a grid cell in the
    // reverse direction from the tail moving to the tail.
    SlimeSegment newTail = new SlimeSegment();
    newTail.At = tail.To + tail.ReverseDirection * _stride;
    newTail.To = tail.At;
    newTail.Direction = Vector2.Normalize(tail.At - newTail.At);

    // Add the new tail segment
    _segments.Add(newTail);
}
```

The `Grow` method works as follows:

1. First it creates a copy of the current tail value.
2. It then adjusts the values of the copy so that it is now positioned behind the current tail by using the `ReverseDirection` value of the tail.
3. Finally, it inserts this new tail into the segments collection.

Slime Update

With most of the core snake-like mechanics now added to the `Slime` class within their own methods we can now work on what happens while the slime is operating.

Add the following `Update` method to the `Slime` class after the `Grow` method:

```
/// <summary>
/// Updates the slime.
/// </summary>
/// <param name="gameTime">A snapshot of the timing values for the current update
cycle.</param>
public void Update(GameTime gameTime)
{
    // Update the animated sprite.
    _sprite.Update(gameTime);

    // Handle any player input
    HandleInput();

    // Increment the movement timer by the frame elapsed time.
    _movementTimer += gameTime.ElapsedGameTime;

    // If the movement timer has accumulated enough time to be greater than
    // the movement time threshold, then perform a full movement.
    if (_movementTimer >= s_movementTime)
    {
        _movementTimer -= s_movementTime;
        Move();
    }

    // Update the movement lerp offset amount
    _movementProgress = (float)(_movementTimer.TotalSeconds /
s_movementTime.TotalSeconds);
}
```

This update method:

1. Updates the slime's `AnimatedSprite` to ensure the sprite animations occur.
2. Calls `HandleInput` to check for player input.
3. Increments the movement timer by the amount of time that has elapsed between the game's update cycles.
4. Performs a check to see if the movement timer has accumulated more time than the threshold to perform a movement cycle update. If it has then:
    1. The movement timer is reduced by the threshold time.
    2. The `Move` method is called to perform a movement cycle update.
5. Finally, the movement progress amount is calculated by dividing the number of seconds accumulated for the movement timer by the number of seconds for the threshold. This gives us a normalized value between 0.0 and 1.0 that we can use for visual interpolation for fluid movement.

> **TIP**
>
> In games, frame rates can vary based on system performance, causing inconsistent update intervals. If we simply reset the movement timer to zero after each movement cycle, we would lose any excess time that accumulated beyond the movement threshold.
>
> For example:
>
> - Our movement threshold is 200ms.
> - The game runs at 60fps (16.67ms per frame).
> - After 12 frames, we have accumulated 200.04ms.
> - If we reset to zero, we lose 0.04ms.
> - Over time, these small losses can add up and cause inconsistent movement.
>
> By subtracting the threshold instead of resetting to zero, we "bank" the excess time (0.06ms in this example) for the next movement cycle. This ensures that:
>
> 1. Movement happens exactly at the intended frequency, maintaining consistent game speed.
> 2. The visual smoothness of movement remains intact even if the game occasionally drops frames.
> 3. Players experience the same game timing regardless of their hardware's performance.
>
> This technique is standard practice in game development, especially for timing-sensitive mechanics like rhythmic games, animations, and movement systems. It is a simple solution that significantly improves gameplay consistency.

Slime Draw

We also need a method to handle drawing the slime and all of its segments.

Add the following `Draw` method after the `Update` method to the `Slime` class:

```
/// <summary>
/// Draws the slime.
/// </summary>
public void Draw()
{
    // Iterate through each segment and draw it
    foreach (SlimeSegment segment in _segments)
    {
        // Calculate the visual position of the segment at the moment by
        // lerping between its "at" and "to" position by the movement
        // offset lerp amount
        Vector2 pos = Vector2.Lerp(segment.At, segment.To, _movementProgress);

        // Draw the slime sprite at the calculated visual position of this
        // segment
        _sprite.Draw(Core.SpriteBatch, pos);
    }
}
```

This draw method iterates each segment of the slime and calculates the visual position to draw each segment at by performing linear interpolation (lerp) to determine the position of the segment between its current position (`At`) and the position it is moving to (`To`) based on the `_movementProgress` calculation.

> **NOTE**
>
> **Vector2.Lerp** performs a linear interpolation between two vectors, creating a smooth transition from start to end based on an amount parameter. The formula is:
>
> $$\mathbf{result} = \mathbf{start} + (\mathbf{end} - \mathbf{start}) \cdot \mathbf{amount}$$
>
> Where the amount parameter ranges from 0.0 (returns the start position) to 1.0 (returns the end position). Values between 0.0 and 1.0 give positions along the straight line between start and end.
>
> In our game, this is used to create a visual fluid movement illusion by interpolating between a segment's current position (`At`) and its target position (`To`) based on the elapsed time, rather than having the segments abruptly jump from one grid position to the next after each movement update.

Slime Bounds

For the game scene to detect collisions between the slime and other elements (walls or the bat), we need a method to calculate the current collision bounds.

Add the following method to the `Slime` class after the `Draw` method:

```
/// <summary>
/// Returns a Circle value that represents collision bounds of the slime.
/// </summary>
/// <returns>A Circle value.</returns>
public Circle GetBounds()
{
    SlimeSegment head = _segments[0];

    // Calculate the visual position of the head at the moment of this
    // method call by lerping between the "at" and "to" position by the
    // movement offset lerp amount
    Vector2 pos = Vector2.Lerp(head.At, head.To, _movementProgress);

    // Create the bounds using the calculated visual position of the head.
    Circle bounds = new Circle(
        (int)(pos.X + (_sprite.Width * 0.5f)),
        (int)(pos.Y + (_sprite.Height * 0.5f)),
        (int)(_sprite.Width * 0.5f)
    );

    return bounds;
}
```

This method takes the current head segment (the first segment in our collection), calculates its visual position using linear interpolation, and then creates a `Circle` value to represent its collision boundary. Using the interpolated position ensures that collision detection aligns with what the player sees on screen.

> **NOTE**
>
> We only need collision bounds for the slime's head for interactions with the bat and walls, as this matches the classic snake game mechanic where only the head's collisions matter for gameplay. For detecting collisions between the head and body segments, we use a simpler position-based check in the `Move` method since those positions are always aligned to the grid.

With all these methods in place, our Slime class now fully implements the snake-like mechanics we need.

It handles:

- Movement on a grid.
- Prevents invalid direction changes
- Detects self-collisions
- Provides smooth visual movement between grid positions.

This encapsulation allows us to manage all slime-related behavior in one place while exposing only the necessary interfaces to the game scene.

Now that we have our player-controlled character implemented, we can create the object that the slime will try to collect; the bat.

The Bat Class

In the `GameObjects` folder of the *DungeonSlime* project (your main game project), create a new file named `Bat.cs` and add the following initial code:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime.GameObjects;

public class Bat
{

}
```

This code establishes the foundation for our `Bat` class. We have included the necessary using statements for MonoGame components, audio functionality, and our library references. The class is placed in the same `DungeonSlime.GameObjects` namespace as our Slime class to maintain a consistent organization.

Now we will build this class step by step, adding all the functionality needed for the bat to serve as the collectible object in our game. Add each of the following sections to the `Bat` class in the order they are presented.

> **NOTE**
>
> As with the Slime class, you may encounter compiler errors until all sections are in place. These errors will be resolved once all components of the class have been added.

Bat Fields

Add the following fields to the `Bat` class:

```
private const float MOVEMENT_SPEED = 5.0f;

// The velocity of the bat that defines the direction and how much in that
// direction to update the bats position each update cycle.
private Vector2 _velocity;

// The AnimatedSprite used when drawing the bat.
private AnimatedSprite _sprite;

// The sound effect to play when the bat bounces off the edge of the room.
private SoundEffect _bounceSoundEffect;
```

Each of these fields is responsible for:

- `MOVEMENT_SPEED`: This constant represents the factor to multiply the velocity vector by to determine how fast the bat is moving.
- `_velocity`: A vector that defines the direction and how much in that direction to update the position of the bat each update cycle.
- `_sprite`: This stores the `AnimatedSprite` that is used to draw the bat.
- `_bounceSoundEffect`: This store the **SoundEffect** to play when the bat is told to bounce.

Bat Properties

Next, add the following property to the `Bat` class after the fields:

```
/// <summary>
/// Gets or Sets the position of the bat.
/// </summary>
public Vector2 Position { get; set;}
```

This property exposes the position of the bat so it can be used for calculations in the game scene when determining where to place the bat after the slime eats it.

Bat Constructor

After the property, add the following constructor to the `Bat` class:

```
/// <summary>
/// Creates a new Bat using the specified animated sprite and sound effect.
/// </summary>
/// <param name="sprite">The AnimatedSprite ot use when drawing the bat.</param>
/// <param name="bounceSoundEffect">The sound effect to play when the bat bounces
off a wall.</param>
public Bat(AnimatedSprite sprite, SoundEffect bounceSoundEffect)
{
    _sprite = sprite;
    _bounceSoundEffect = bounceSoundEffect;
}
```

This is a simple constructor that requires the bat to be given the `AnimatedSprite` that will be used to draw the bat and the **SoundEffect** to be played when the bat bounces off a wall.

Bat Randomize Velocity

Currently, we have the `AssignRandomVelocity` method in the `GameScene` that we call to randomize the velocity of the bat after it has been eaten by the slime. We can take this method out of the `GameScene` class and put it directly into the `Bat` class itself.

Add the following method to the `Bat` class after the constructor:

```
/// <summary>
/// Randomizes the velocity of the bat.
/// </summary>
public void RandomizeVelocity()
{
    // Generate a random angle
    float angle = (float)(Random.Shared.NextDouble() * MathHelper.TwoPi);

    // Convert the angle to a direction vector
    float x = (float)Math.Cos(angle);
    float y = (float)Math.Sin(angle);
    Vector2 direction = new Vector2(x, y);

    // Multiply the direction vector by the movement speed to get the
    // final velocity
    _velocity = direction * MOVEMENT_SPEED;
}
```

Bat Bounce

We are also going to take the logic from the `GameScene` class that bounces the bat off the walls and move it into a dedicated method in the `Bat` class.

Add the following method to the `Bat` class after the `RandomizeVelocity` method:

```
/// <summary>
/// Handles a bounce event when the bat collides with a wall or boundary.
/// </summary>
/// <param name="normal">The normal vector of the surface the bat is bouncing
against.</param>
public void Bounce(Vector2 normal)
{
    Vector2 newPosition = Position;

    // Adjust the position based on the normal to prevent sticking to walls.
    if(normal.X != 0)
    {
        // We are bouncing off a vertical wall (left/right).
        // Move slightly away from the wall in the direction of the normal.
        newPosition.X += normal.X * (_sprite.Width * 0.1f);
    }

    if(normal.Y != 0)
    {
        // We are bouncing off a horizontal wall (top/bottom).
        // Move slightly way from the wall in the direction of the normal.
        newPosition.Y += normal.Y * (_sprite.Height * 0.1f);
    }

    // Apply the new position
    Position = newPosition;

    // Normalize before reflecting
    normal.Normalize();

    // Apply reflection based on the normal.
    _velocity = Vector2.Reflect(_velocity, normal);

    // Play the bounce sound effect.
    Core.Audio.PlaySoundEffect(_bounceSoundEffect);
}
```

This method only takes a single parameter, the normal vector of the surface the bat is bouncing against. Based on the X and Y components of the normal vector, we can determine which wall the bat bounced against and adjust the position of the bat so that it does not stick to the wall.

Bat Bounds

Similar to the `Slime` class, for the game scene to detect collision between the bat and other elements, we need a method to calculate the current collision bounds of the bat.

Add the following method to the `Bat` class after the `Bounce` method:

```
/// <summary>
/// Returns a Circle value that represents collision bounds of the bat.
/// </summary>
/// <returns>A Circle value.</returns>
public Circle GetBounds()
{
    int x = (int)(Position.X + _sprite.Width * 0.5f);
    int y = (int)(Position.Y + _sprite.Height * 0.5f);
    int radius = (int)(_sprite.Width * 0.25f);

    return new Circle(x, y, radius);
}
```

Bat Update

The `Bat` class will also need to be updated.

Add the following `Update` method to the `Bat` class after the `GetBounds` method:

```
/// <summary>
/// Updates the bat.
/// </summary>
/// <param name="gameTime">A snapshot of the timing values for the current update
cycle.</param>
public void Update(GameTime gameTime)
{
    // Update the animated sprite
    _sprite.Update(gameTime);

    // Update the position of the bat based on the velocity.
    Position += _velocity;
}
```

This method simply updates the bat's `AnimatedSprite` to ensure animations occur and adjusts the position of the bat based on the current velocity.

> **NOTE**
> The continuous movement of the bat contrasts with the grid-based interval movement of the slime, creating different gameplay dynamics for the player to consider. This makes catching the bat challenging without requiring any complex behaviors.

Bat Draw

Finally, we need a method to draw the bat.

Add the following `Draw` method to the `Bat` class after the `Update` method:

```
/// <summary>
/// Draws the bat.
/// </summary>
public void Draw()
{
    _sprite.Draw(Core.SpriteBatch, Position);
}
```

This method simply draws the bat's `AnimatedSprite` at the bat's current position.

With the `Bat` class complete, we have now encapsulated all the behavior needed for the collectible element in our game. The bat moves continuously around the screen and can bounce off walls, adding a twist on the classic snake-like mechanic by creating a target for the player to chase.

Conclusion

> **NOTE**
> To the observant, you should notice that the main game screen has not been updated and therefore nothing has changed if we run the game at this point. In the next chapter we will finalize the gameplay.

In this chapter, we have learned about and implemented the core mechanics of a class snake-like game. We created:

- A `GameController` class that provides a unified input interface, separating game actions from specific input devices.
- A `SlimeSegment` struct to efficiently store and manage individual segments of our snake-like character.
- A `Slime` class that implements grid-based movement, segment management and self-collision detection.
- A `Bat` class that serves as the collectible object with continuous movement and wall bouncing.

These implementations encapsulate the core gameplay mechanics into reusable, maintainable objects.

In the next chapter, we will build on these mechanics by updating the `GameScene` to implement game state management and a new UI element for the game over state to create a complete game experience.

Test Your Knowledge

1. Why must a snake-like game prevent the player from reversing direction?

   Preventing reverse movement is necessary because it would cause an immediate collision between the snake's head and the first body segment, resulting in an unfair game over.

2. How does the movement cycle for a snake work, and why is it more efficient than updating each segment individually?

   The snake movement cycle works by:

   1. Creating a copy of the head segment.
   2. Positioning the copy one grid cell ahead in the current direction
   3. Inserting this copy at the front of the segment collection
   4. Removing the last segment.

   This approach is more efficient because it only requires manipulating two segments (adding a new head and removing the tail) regardless of how long the snake becomes, rather than iterating through and updating every segment individually.

3. What are the benefits of using the Input Abstraction pattern implemented in the `GameController` class?

   The Input Abstraction pattern provides several benefits:

   - Input device independence, allowing the game to handle keyboard, gamepad, or other inputs through a unified interface
   - Simplified input handling through clean API calls rather than checking multiple input combinations
   - Easier implementation of key rebinding features by only needing to modify the GameController class
   - Consistent input logic defined in a single location
   - Better cross-platform compatibility by centralizing platform-specific input handling

4. How does the implementation use **Vector2.Lerp** to create smooth visual movement, and why is this important?

> The implementation uses **Vector2.Lerp** to interpolate between a segment's current position (`At`) and its target position (`To`) based on a normalized movement progress value. This creates smooth visual movement by drawing the segments at intermediate positions between grid points rather than abruptly jumping from one grid position to the next.
>
> This is important because it provides more fluid animation while maintaining the logical grid-based movement, enhancing the visual quality of the game without changing the core mechanics.

Chapter 23: Completing the Game

Finalize game mechanics by updating our current demo into a snake-like inspired game.

In Chapter 22 we implemented the core mechanics of a snake-like game by creating the `Slime`, `Bat` and `GameController` classes. While these classes handle the foundational gameplay, a complete game needs additional elements to provide player feedback, manage game states, and create a polished experience.

In this chapter, you will:

- Create a dedicated UI class to manage the UI for the game scene.
- Implement pause and game over screens with appropriate controls.
- Refactor the `GameScene` class to coordinate all game elements.
- Add game state management to handle playing, paused, and game over conditions
- Implement input buffering to improve control responsiveness
- Connect all elements to create a complete, playable game.

The GameSceneUI Class

Currently, the `GameScene` class contains the methods for initializing and creating the pause menu. However, now that we have a defined condition for game over, we need to create a game-over menu as well. To do this, we will take the opportunity to refactor the current code and pull the UI-specific code into its own class.

In the `UI` folder of the *DungeonSlime* project, create a new file named `GameSceneUI.cs` and add the following initial code:

```
using System;
using Gum.DataTypes;
using Gum.Managers;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using MonoGameGum;
using Gum.Forms.Controls;
using MonoGameGum.GueDeriving;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;

namespace DungeonSlime.UI;

public class GameSceneUI : ContainerRuntime
{

}
```

This code establishes the foundation for our `GameSceneUI` class, which inherits from Gum's `ContainerRuntime` class. This inheritance means our UI class is itself a UI container that can hold and manage other UI elements. We have included all necessary using statements for MonoGame, Gum UI components, and our library references.

You will build out this class by adding each section in sequence. Follow the order below to create the complete UI management system for our game scene.

> **NOTE**
>
> You may see compiler errors as you add these sections one by one. This is expected because some parts of the code will reference fields, properties, or methods that we haven't added yet. Once all sections are in place, these errors will resolve.

GameSceneUI Fields

Add the following fields to the `GameSceneUI` class:

```
// The string format to use when updating the text for the score display.
private static readonly string s_scoreFormat = "SCORE: {0:D6}";

// The sound effect to play for auditory feedback of the user interface.
private SoundEffect _uiSoundEffect;

// The pause panel
private Panel _pausePanel;

// The resume button on the pause panel. Field is used to track reference so
// focus can be set when the pause panel is shown.
private AnimatedButton _resumeButton;

// The game over panel.
private Panel _gameOverPanel;

// The retry button on the game over panel. Field is used to track reference
// so focus can be set when the game over panel is shown.
private AnimatedButton _retryButton;

// The text runtime used to display the players score on the game screen.
private TextRuntime _scoreText;
```

Here is a break down what each of these fields is responsible for:

- `s_scoreFormat`: A string format template used to display the player's score with leading zeros.
- `_uiSoundEffect`: Stores the sound effect played for UI interactions like button clicks and focus changes.
- `_pausePanel`: The panel containing the UI elements shown when the game is paused.
- `_resumeButton`: A reference to the resume button, allowing us to set focus on it when the pause panel is shown.
- `_gameOverPanel`: The panel containing the UI elements shown when a game over occurs.
- `_retryButton`: A reference to the retry button, allowing us to set focus to it when the game over panel is shown.
- `_scoreText`: The text display showing the player's current score.

GameSceneUI Events

After the fields, add the following events to the `GameSceneUI` class:

```
/// <summary>
/// Event invoked when the Resume button on the Pause panel is clicked.
/// </summary>
public event EventHandler ResumeButtonClick;

/// <summary>
/// Event invoked when the Quit button on either the Pause panel or the
/// Game Over panel is clicked.
/// </summary>
public event EventHandler QuitButtonClick;

/// <summary>
/// Event invoked when the Retry button on the Game Over panel is clicked.
/// </summary>
public event EventHandler RetryButtonClick;
```

These events allow the `GameSceneUI` class to notify the `GameScene` when important UI actions occur:

- `ResumeButtonClick`: Triggered when the player clicks the Resume button on the pause panel.
- `QuitButtonClick`: Triggered when the player clicks the Quit button on either panel.
- `RetryButtonClick`: Triggered when the player clicks the Retry button on the game over panel.

GameSceneUI Constructor

Add the following constructor to the `GameSceneUI` class after the events:

```
public GameSceneUI()
{
    // The game scene UI inherits from ContainerRuntime, so we set its
    // doc to fill so it fills the entire screen.
    Dock(Gum.Wireframe.Dock.Fill);

    // Add it to the root element.
    this.AddToRoot();

    // Get a reference to the content manager that was registered with the
    // GumService when it was original initialized.
    ContentManager content = GumService.Default.ContentLoader.XnaContentManager;

    // Use that content manager to load the sound effect and atlas for the
    // user interface elements
    _uiSoundEffect = content.Load<SoundEffect>("audio/ui");
    TextureAtlas atlas = TextureAtlas.FromFile(content,
"images/atlas-definition.xml");

    // Create the text that will display the players score and add it as
    // a child to this container.
    _scoreText = CreateScoreText();
    AddChild(_scoreText);

    // Create the Pause panel that is displayed when the game is paused and
    // add it as a child to this container
    _pausePanel = CreatePausePanel(atlas);
    AddChild(_pausePanel.Visual);

    // Create the Game Over panel that is displayed when a game over occurs
    // and add it as a child to this container
    _gameOverPanel = CreateGameOverPanel(atlas);
    AddChild(_gameOverPanel.Visual);
}
```

This constructor initializes all UI components:

1. Set the container to fill the entire screen.
2. Adds itself to Gum's root element.
3. Loads necessary assets (sound effect and texture atlas).
4. Creates and adds child elements in the correct order.

GameSceneUI UI Creation Methods

To keep the code more organized, we will create separate functions to build the individual UI elements that will be managed by the `GameSceneUI` class.

Creating the Score Text

To display the player's score, we will begin by adding a method to create a `TextRuntime` element.

Add the following method to the `GameSceneUI` after the constructor:

```
private TextRuntime CreateScoreText()
{
    TextRuntime text = new TextRuntime();
    text.Anchor(Gum.Wireframe.Anchor.TopLeft);
    text.WidthUnits = DimensionUnitType.RelativeToChildren;
    text.X = 20.0f;
    text.Y = 5.0f;
    text.UseCustomFont = true;
    text.CustomFontFile = @"fonts/04b_30.fnt";
    text.FontScale = 0.25f;
    text.Text = string.Format(s_scoreFormat, 0);

    return text;
}
```

Creating the Pause Panel

Next, we will add a method to create a `Panel` element that is shown when the game is paused, including the "Resume" and "Quit" buttons.

Add the following method to the `GameSceneUI` class after the `CreateScoreText` method:

```
private Panel CreatePausePanel(TextureAtlas atlas)
{
    Panel panel = new Panel();
    panel.Anchor(Gum.Wireframe.Anchor.Center);
    panel.WidthUnits = DimensionUnitType.Absolute;
    panel.HeightUnits = DimensionUnitType.Absolute;
    panel.Width = 264.0f;
    panel.Height = 70.0f;
    panel.IsVisible = false;

    TextureRegion backgroundRegion = atlas.GetRegion("panel-background");

    NineSliceRuntime background = new NineSliceRuntime();
    background.Dock(Gum.Wireframe.Dock.Fill);
    background.Texture = backgroundRegion.Texture;
    background.TextureAddress = TextureAddress.Custom;
    background.TextureHeight = backgroundRegion.Height;
    background.TextureWidth = backgroundRegion.Width;
    background.TextureTop = backgroundRegion.SourceRectangle.Top;
    background.TextureLeft = backgroundRegion.SourceRectangle.Left;
    panel.AddChild(background);

    TextRuntime text = new TextRuntime();
    text.Text = "PAUSED";
    text.UseCustomFont = true;
    text.CustomFontFile = "fonts/04b_30.fnt";
    text.FontScale = 0.5f;
    text.X = 10.0f;
    text.Y = 10.0f;
    panel.AddChild(text);

    _resumeButton = new AnimatedButton(atlas);
    _resumeButton.Text = "RESUME";
    _resumeButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);
    _resumeButton.X = 9.0f;
    _resumeButton.Y = -9.0f;

    _resumeButton.Click += OnResumeButtonClicked;
    _resumeButton.GotFocus += OnElementGotFocus;

    panel.AddChild(_resumeButton);

    AnimatedButton quitButton = new AnimatedButton(atlas);
    quitButton.Text = "QUIT";
    quitButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
    quitButton.X = -9.0f;
```

```
        quitButton.Y = -9.0f;

        quitButton.Click += OnQuitButtonClicked;
        quitButton.GotFocus += OnElementGotFocus;

        panel.AddChild(quitButton);

        return panel;
    }
```

Creating the Game Over Panel

Finally, we will add a method to create a `Panel` element that is shown when a game over occurs, including the "Retry" and "Quit" buttons.

Add the following method to the `GameSceneUI` class after the `CreatePausePanel` method:

```
private Panel CreateGameOverPanel(TextureAtlas atlas)
{
    Panel panel = new Panel();
    panel.Anchor(Gum.Wireframe.Anchor.Center);
    panel.WidthUnits = DimensionUnitType.Absolute;
    panel.HeightUnits = DimensionUnitType.Absolute;
    panel.Width = 264.0f;
    panel.Height = 70.0f;
    panel.IsVisible = false;

    TextureRegion backgroundRegion = atlas.GetRegion("panel-background");

    NineSliceRuntime background = new NineSliceRuntime();
    background.Dock(Gum.Wireframe.Dock.Fill);
    background.Texture = backgroundRegion.Texture;
    background.TextureAddress = TextureAddress.Custom;
    background.TextureHeight = backgroundRegion.Height;
    background.TextureWidth = backgroundRegion.Width;
    background.TextureTop = backgroundRegion.SourceRectangle.Top;
    background.TextureLeft = backgroundRegion.SourceRectangle.Left;
    panel.AddChild(background);

    TextRuntime text = new TextRuntime();
    text.Text = "GAME OVER";
    text.WidthUnits = DimensionUnitType.RelativeToChildren;
    text.UseCustomFont = true;
    text.CustomFontFile = "fonts/04b_30.fnt";
    text.FontScale = 0.5f;
    text.X = 10.0f;
    text.Y = 10.0f;
    panel.AddChild(text);

    _retryButton = new AnimatedButton(atlas);
    _retryButton.Text = "RETRY";
    _retryButton.Anchor(Gum.Wireframe.Anchor.BottomLeft);
    _retryButton.X = 9.0f;
    _retryButton.Y = -9.0f;

    _retryButton.Click += OnRetryButtonClicked;
    _retryButton.GotFocus += OnElementGotFocus;

    panel.AddChild(_retryButton);

    AnimatedButton quitButton = new AnimatedButton(atlas);
    quitButton.Text = "QUIT";
    quitButton.Anchor(Gum.Wireframe.Anchor.BottomRight);
```

```
        quitButton.X = -9.0f;
        quitButton.Y = -9.0f;

        quitButton.Click += OnQuitButtonClicked;
        quitButton.GotFocus += OnElementGotFocus;

        panel.AddChild(quitButton);

        return panel;
    }
```

Both the pause panel and the game over panel use event handlers for their buttons. We will add those next.

GameSceneUI Event Handlers

After the `CreateGameOverPanel` method, add the following method to the `GameSceneUI` class:

```
private void OnResumeButtonClicked(object sender, EventArgs args)
{
    // Button was clicked, play the ui sound effect for auditory feedback.
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Since the resume button was clicked, we need to hide the pause panel.
    HidePausePanel();

    // Invoke the ResumeButtonClick event
    if(ResumeButtonClick != null)
    {
        ResumeButtonClick(sender, args);
    }
}

private void OnRetryButtonClicked(object sender, EventArgs args)
{
    // Button was clicked, play the ui sound effect for auditory feedback.
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Since the retry button was clicked, we need to hide the game over panel.
    HideGameOverPanel();

    // Invoke the RetryButtonClick event.
    if(RetryButtonClick != null)
    {
        RetryButtonClick(sender, args);
    }
}

private void OnQuitButtonClicked(object sender, EventArgs args)
{
    // Button was clicked, play the ui sound effect for auditory feedback.
    Core.Audio.PlaySoundEffect(_uiSoundEffect);

    // Both panels have a quit button, so hide both panels
    HidePausePanel();
    HideGameOverPanel();

    // Invoke the QuitButtonClick event.
    if(QuitButtonClick != null)
    {
        QuitButtonClick(sender, args);
    }
}
```

```
private void OnElementGotFocus(object sender, EventArgs args)
{
    // A ui element that can receive focus has received focus, play the
    // ui sound effect for auditory feedback.
    Core.Audio.PlaySoundEffect(_uiSoundEffect);
}
```

These event handlers provide audio feedback and appropriate UI updates when buttons are clicked or UI elements receive focus.

GameSceneUI Public Methods

Finally, add the following public methods to the `GameSceneUI` class after the `OnElementGotFocus` method:

```
/// <summary>
/// Updates the text on the score display.
/// </summary>
/// <param name="score">The score to display.</param>
public void UpdateScoreText(int score)
{
    _scoreText.Text = string.Format(s_scoreFormat, score);
}

/// <summary>
/// Tells the game scene ui to show the pause panel.
/// </summary>
public void ShowPausePanel()
{
    _pausePanel.IsVisible = true;

    // Give the resume button focus for keyboard/gamepad input.
    _resumeButton.IsFocused = true;

    // Ensure the game over panel isn't visible.
    _gameOverPanel.IsVisible = false;
}

/// <summary>
/// Tells the game scene ui to hide the pause panel.
/// </summary>
public void HidePausePanel()
{
    _pausePanel.IsVisible = false;
}

/// <summary>
/// Tells the game scene ui to show the game over panel.
/// </summary>
public void ShowGameOverPanel()
{
    _gameOverPanel.IsVisible = true;

    // Give the retry button focus for keyboard/gamepad input.
    _retryButton.IsFocused =true;

    // Ensure the pause panel isn't visible.
    _pausePanel.IsVisible = false;
}

/// <summary>
```

```
    /// Tells the game scene ui to hide the game over panel.
    /// </summary>
    public void HideGameOverPanel()
    {
        _gameOverPanel.IsVisible = false;
    }

    /// <summary>
    /// Updates the game scene ui.
    /// </summary>
    /// <param name="gameTime">A snapshot of the timing values for the current update
    cycle.</param>
    public void Update(GameTime gameTime)
    {
        GumService.Default.Update(gameTime);
    }

    /// <summary>
    /// Draws the game scene ui.
    /// </summary>
    public void Draw()
    {
        GumService.Default.Draw();
    }
```

These public methods provide the interface for the `GameScene` to implement:

- Update the score display.
- Show or hide the pause menu.
- Show or hide the game over menu.
- Update and draw the UI components.

With the `GameSceneUI` class complete, we now have a fully encapsulated UI system that can handle displaying game information (score), providing feedback for game states (pause, game over), and processing user interactions (button clicks). This separation of UI logic from game logic will make our codebase much easier to maintain and extend.

Now that we have all our specialized components ready, we can refactor the `GameScene` class to coordinate between them and manage the overall game flow.

Refactoring The GameScene Class

Now that we have created the encapsulated `Slime`, `Bat`, and `GameSceneUI` classes, we can refactor the `GameScene` class to leverage these new components. This will make our code more maintainable and allow us to focus on the game logic within the scene itself.

We will rebuild/replace the existing `GameScene` class to coordinate the interactions between the components.

In the `Scenes` folder of the *DungeonSlime* project (your main game project), open the `GameScene.cs` file and replace **ALL** of the code with the following replacement code (starting fresh):

```
using System;
using DungeonSlime.GameObjects;
using DungeonSlime.UI;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using MonoGameGum;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
using MonoGameLibrary.Scenes;

namespace DungeonSlime.Scenes;

public class GameScene : Scene
{
    private enum GameState
    {
        Playing,
        Paused,
        GameOver
    }

    // Reference to the slime.
    private Slime _slime;

    // Reference to the bat.
    private Bat _bat;

    // Defines the tilemap to draw.
    private Tilemap _tilemap;

    // Defines the bounds of the room that the slime and bat are contained within.
    private Rectangle _roomBounds;

    // The sound effect to play when the slime eats a bat.
    private SoundEffect _collectSoundEffect;

    // Tracks the players score.
    private int _score;

    private GameSceneUI _ui;

    private GameState _state;
}
```

This code provides the foundation for our refactored `GameScene` class. We have included all the necessary using statements to reference our new game object classes and UI components. The class will now focus on managing the game state and coordinating between our specialized component classes rather than implementing all the functionality directly.

The `GameScene` class now contains the following key fields:

- `GameState`: An enum that defines the different states that the game can be in (playing, paused, or game over).
- `_slime`: A reference to the slime (snake-like player character) instance.
- `_bat`: A reference to the bat (food) instance.
- `_tilemap`: The tilemap that defines the level layout.
- `_roomBounds`: A rectangle defining the playable area within the walls.
- `_collectSoundEffect`: The sound effect played when the slime eats a bat.
- `_score`: Tracks the player's current score.
- `_ui`: A reference to the game scene UI component.
- `_state`: The current state of the game represented by the `GameState` enum.

Next we will add the various methods needed to complete the `GameScene` class and finalize the game logic.

Add each section in the sequence presented below. This will build up the scene's functionality step by step.

> **NOTE**
> As with previous classes, you might encounter compiler errors until all sections are in place. These errors will be resolved once all components of the class have been added.

GameScene Initialize Method

To set up the scene, add the following `Initialize` method after the fields in te `GameScene` class:

```
public override void Initialize()
{
    // LoadContent is called during base.Initialize().
    base.Initialize();

    // During the game scene, we want to disable exit on escape. Instead,
    // the escape key will be used to return back to the title screen.
    Core.ExitOnEscape = false;

    // Create the room bounds by getting the bounds of the screen then
    // using the Inflate method to "Deflate" the bounds by the width and
    // height of a tile so that the bounds only covers the inside room of
    // the dungeon tilemap.
    _roomBounds = Core.GraphicsDevice.PresentationParameters.Bounds;
    _roomBounds.Inflate(-_tilemap.TileWidth, -_tilemap.TileHeight);

    // Subscribe to the slime's BodyCollision event so that a game over
    // can be triggered when this event is raised.
    _slime.BodyCollision += OnSlimeBodyCollision;

    // Create any UI elements from the root element created in previous
    // scenes.
    GumService.Default.Root.Children.Clear();

    // Initialize the user interface for the game scene.
    InitializeUI();

    // Initialize a new game to be played.
    InitializeNewGame();
}
```

This method sets up the initial state of the game scene:

1. Disables the "exit on escape" behavior so we can use the escape key for pausing.
2. Calculate the playable area within the tilemap walls.
3. Subscribes to the slime's body collision event to detect when the player collides with itself triggering a game over state.
4. Initialize the UI components.
5. Set up a new game.

GameScene InitializeUI Method

The `Initialize` method we just added calls a method to initialize the user interface for the scene, we can add that method now.

Add the following method after the `Initialize` method in the `GameScene` class:

```
private void InitializeUI()
{
    // Clear out any previous UI element incase we came here
    // from a different scene.
    GumService.Default.Root.Children.Clear();

    // Create the game scene ui instance.
    _ui = new GameSceneUI();

    // Subscribe to the events from the game scene ui.
    _ui.ResumeButtonClick += OnResumeButtonClicked;
    _ui.RetryButtonClick += OnRetryButtonClicked;
    _ui.QuitButtonClick += OnQuitButtonClicked;
}
```

This method creates the UI components and subscribes to its events to respond to button clicks.

GameScene UI Event Handlers

In the `InitializeUI` method we just added, we subscribe to the events from the `GameSceneUI` class that are triggered when buttons are clicked. Now we need to add those methods that would be called when the events are triggered.

Add the following methods to the `GameScene` class after the `InitializeUI` method:

```
private void OnResumeButtonClicked(object sender, EventArgs args)
{
    // Change the game state back to playing.
    _state = GameState.Playing;
}

private void OnRetryButtonClicked(object sender, EventArgs args)
{
    // Player has chosen to retry, so initialize a new game.
    InitializeNewGame();
}

private void OnQuitButtonClicked(object sender, EventArgs args)
{
    // Player has chosen to quit, so return back to the title scene.
    Core.ChangeScene(new TitleScene());
}
```

These methods respond to the UI events:

- `OnResumeButtonClicked`: Resumes the game from a paused state.
- `OnRetryButtonClicked`: Restarts the game after a game over.
- `OnQuitButtonClicked`: Quits the game by returning to the title scene.

GameScene InitializeNewGame Method

In the `Initialize` method we added above, it also makes a call to an `InitializeNewGame` method which we will add next.

Add the following method to the `GameScene` class after the `OnQuitButtonClicked` method:

```
private void InitializeNewGame()
{
    // Calculate the position for the slime, which will be at the center
    // tile of the tile map.
    Vector2 slimePos = new Vector2();
    slimePos.X = (_tilemap.Columns / 2) * _tilemap.TileWidth;
    slimePos.Y = (_tilemap.Rows / 2) * _tilemap.TileHeight;

    // Initialize the slime.
    _slime.Initialize(slimePos, _tilemap.TileWidth);

    // Initialize the bat.
    _bat.RandomizeVelocity();
    PositionBatAwayFromSlime();

    // Reset the score.
    _score = 0;

    // Set the game state to playing.
    _state = GameState.Playing;
}
```

This method will:

1. Position the slime in the center of the map.
2. Initialize the slime with its starting position and movement stride.
3. Randomize the bat's velocity and position it away from the slime.
4. Reset the player's score.
5. Set the game state to "Playing".

GameScene LoadContent Method

Next, we need to add the method to load game assets for the scene.

Add the following method to the `GameScene` class after the `InitializeNewGame` method:

```
public override void LoadContent()
{
    // Create the texture atlas from the XML configuration file.
    TextureAtlas atlas = TextureAtlas.FromFile(Core.Content,
"images/atlas-definition.xml");

    // Create the tilemap from the XML configuration file.
    _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
    _tilemap.Scale = new Vector2(4.0f, 4.0f);

    // Create the animated sprite for the slime from the atlas.
    AnimatedSprite slimeAnimation = atlas.CreateAnimatedSprite("slime-animation");
    slimeAnimation.Scale = new Vector2(4.0f, 4.0f);

    // Create the slime.
    _slime = new Slime(slimeAnimation);

    // Create the animated sprite for the bat from the atlas.
    AnimatedSprite batAnimation = atlas.CreateAnimatedSprite("bat-animation");
    batAnimation.Scale = new Vector2(4.0f, 4.0f);

    // Load the bounce sound effect for the bat.
    SoundEffect bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

    // Create the bat.
    _bat = new Bat(batAnimation, bounceSoundEffect);

    // Load the collect sound effect.
    _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");
}
```

This method loads all necessary assets for the game scene:

1. The texture atlas containing the sprite graphics.
2. The tilemap that defines the level layout.
3. The animated sprites for the slime and bat.
4. Sound effects for the bat bouncing and collecting.

GameScene Update Method

Next, to update the scene, add the following method to the `GameScene` class after the `LoadContent` method:

```
public override void Update(GameTime gameTime)
{
    // Ensure the UI is always updated.
    _ui.Update(gameTime);

    // If the game is in a game over state, immediately return back
    // here.
    if (_state == GameState.GameOver)
    {
        return;
    }

    // If the pause button is pressed, toggle the pause state.
    if (GameController.Pause())
    {
        TogglePause();
    }

    // At this point, if the game is paused, just return back early.
    if (_state == GameState.Paused)
    {
        return;
    }

    // Update the slime.
    _slime.Update(gameTime);

    // Update the bat.
    _bat.Update(gameTime);

    // Perform collision checks.
    CollisionChecks();
}
```

This method updates the scene in each frame to:

1. Always update the UI, regardless of game state.
2. Return early if the game is over.
3. Check for pause input and toggle the pause state if needed.
4. Return early if the game is paused.
5. Update the slime and bat.
6. Check for collisions between the game objects.

GameScene CollisionChecks Method

In the `Update` method we just added, it makes a call to a `CollisionChecks` method to handle the collision detection and response so we will add that now.

Add the following method to the `GameScene` class after the `Update` method:

```
private void CollisionChecks()
{
    // Capture the current bounds of the slime and bat.
    Circle slimeBounds = _slime.GetBounds();
    Circle batBounds = _bat.GetBounds();

    // FIrst perform a collision check to see if the slime is colliding with
    // the bat, which means the slime eats the bat.
    if (slimeBounds.Intersects(batBounds))
    {
        // Move the bat to a new position away from the slime.
        PositionBatAwayFromSlime();

        // Randomize the velocity of the bat.
        _bat.RandomizeVelocity();

        // Tell the slime to grow.
        _slime.Grow();

        // Increment the score.
        _score += 100;

        // Update the score display on the UI.
        _ui.UpdateScoreText(_score);

        // Play the collect sound effect.
        Core.Audio.PlaySoundEffect(_collectSoundEffect);
    }

    // Next check if the slime is colliding with the wall by validating if
    // it is within the bounds of the room.  If it is outside the room
    // bounds, then it collided with a wall which triggers a game over.
    if (slimeBounds.Top < _roomBounds.Top ||
       slimeBounds.Bottom > _roomBounds.Bottom ||
       slimeBounds.Left < _roomBounds.Left ||
       slimeBounds.Right > _roomBounds.Right)
    {
        GameOver();
        return;
    }

    // Finally, check if the bat is colliding with a wall by validating if
    // it is within the bounds of the room.  If it is outside the room
    // bounds, then it collided with a wall, and the bat should bounce
    // off of that wall.
    if (batBounds.Top < _roomBounds.Top)
```

```
        {
            _bat.Bounce(Vector2.UnitY);
        }
        else if (batBounds.Bottom > _roomBounds.Bottom)
        {
            _bat.Bounce(-Vector2.UnitY);
        }

        if (batBounds.Left < _roomBounds.Left)
        {
            _bat.Bounce(Vector2.UnitX);
        }
        else if (batBounds.Right > _roomBounds.Right)
        {
            _bat.Bounce(-Vector2.UnitX);
        }
    }
```

This method checks for three types of collisions:

1. Slime-Bat collision: The slime "eats" the bat, gains points, grows, and the bat respawns.
2. Slime-Wall collision: Triggers a game over if the slime hits a wall.
3. Bat-Wall collision: Causes the bat to bounce off the walls.

GameScene PositionBatAwayFromSlime Method

The `CollisionCheck` method makes a call to `PositionBatAwayFromSlime`, previously, when we needed to set the position of the bat when it respawns, we simply chose a random tile within the tilemap to move it to. However, by choosing a completely random location it could be on top fo the head segment of the slime, forcing an instant collision, or it could spawn very close to the head segment, which is not challenging for the player. To ensure the bat appears in a random, but strategic location, we can instead set it to position away from the slime on the opposite side of the room.

Add the following method to the `GameScene` class after the `CollisionCheck` method:

```
private void PositionBatAwayFromSlime()
{
    // Calculate the position that is in the center of the bounds
    // of the room.
    float roomCenterX = _roomBounds.X + _roomBounds.Width * 0.5f;
    float roomCenterY = _roomBounds.Y + _roomBounds.Height * 0.5f;
    Vector2 roomCenter = new Vector2(roomCenterX, roomCenterY);

    // Get the bounds of the slime and calculate the center position.
    Circle slimeBounds = _slime.GetBounds();
    Vector2 slimeCenter = new Vector2(slimeBounds.X, slimeBounds.Y);

    // Calculate the distance vector from the center of the room to the
    // center of the slime.
    Vector2 centerToSlime = slimeCenter - roomCenter;

    // Get the bounds of the bat.
    Circle batBounds =_bat.GetBounds();

    // Calculate the amount of padding we will add to the new position of
    // the bat to ensure it is not sticking to walls
    int padding = batBounds.Radius * 2;

    // Calculate the new position of the bat by finding which component of
    // the center to slime vector (X or Y) is larger and in which direction.
    Vector2 newBatPosition = Vector2.Zero;
    if (Math.Abs(centerToSlime.X) > Math.Abs(centerToSlime.Y))
    {
        // The slime is closer to either the left or right wall, so the Y
        // position will be a random position between the top and bottom
        // walls.
        newBatPosition.Y = Random.Shared.Next(
            _roomBounds.Top + padding,
            _roomBounds.Bottom - padding
        );

        if (centerToSlime.X > 0)
        {
            // The slime is closer to the right side wall, so place the
            // bat on the left side wall.
            newBatPosition.X = _roomBounds.Left + padding;
        }
        else
        {
            // The slime is closer ot the left side wall, so place the
            // bat on the right side wall.
```

```
                newBatPosition.X = _roomBounds.Right - padding * 2;
            }
        }
        else
        {
            // The slime is closer to either the top or bottom wall, so the X
            // position will be a random position between the left and right
            // walls.
            newBatPosition.X = Random.Shared.Next(
                _roomBounds.Left + padding,
                _roomBounds.Right - padding
            );

            if (centerToSlime.Y > 0)
            {
                // The slime is closer to the top wall, so place the bat on the
                // bottom wall.
                newBatPosition.Y = _roomBounds.Top + padding;
            }
            else
            {
                // The slime is closer to the bottom wall, so place the bat on
                // the top wall.
                newBatPosition.Y = _roomBounds.Bottom - padding * 2;
            }
        }

        // Assign the new bat position.
        _bat.Position = newBatPosition;
    }
```

This method positions the bat after it has been eaten:

1. Determines which wall (top, bottom, left, or right) is furthest from the slime.
2. Places the bat near that wall, making it more challenging for the player to reach.

GameScene Event Handler and Game State Methods

Next, we will add some of the missing methods being called from above that handle game events and state changes.

Add the following methods to the `GameScene` class after the `PositionBatAwayFromSlime` method:

```
private void OnSlimeBodyCollision(object sender, EventArgs args)
{
    GameOver();
}

private void TogglePause()
{
    if (_state == GameState.Paused)
    {
        // We're now unpausing the game, so hide the pause panel.
        _ui.HidePausePanel();

        // And set the state back to playing.
        _state = GameState.Playing;
    }
    else
    {
        // We're now pausing the game, so show the pause panel.
        _ui.ShowPausePanel();

        // And set the state to paused.
        _state = GameState.Paused;
    }
}

private void GameOver()
{
    // Show the game over panel.
    _ui.ShowGameOverPanel();

    // Set the game state to game over.
    _state = GameState.GameOver;
}
```

These methods handle specific game events:

- `OnSlimeBodyCollision`: Called when the slime collides with itself, triggering a game over.
- `TogglePause`: Switches between paused and playing states.
- `GameOver`: Called when a game over condition is met, showing the game over UI.

GameScene Draw Method

Finally, we need a method to draw the scene.

Add the following method to the `GameScene` class after the `GameOver` method.

```
public override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    Core.GraphicsDevice.Clear(Color.CornflowerBlue);

    // Begin the sprite batch to prepare for rendering.
    Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);

    // Draw the tilemap
    _tilemap.Draw(Core.SpriteBatch);

    // Draw the slime.
    _slime.Draw();

    // Draw the bat.
    _bat.Draw();

    // Always end the sprite batch when finished.
    Core.SpriteBatch.End();

    // Draw the UI.
    _ui.Draw();
}
```

This method handles drawing the scene by:

1. Clearing the screen.
2. Drawing the tilemap as the background.
3. Drawing the slime and bat sprites.
4. Drawing the UI elements on top.

By refactoring our game into these encapsulated components, we have created a more maintainable codebase with a clear separation of concerns:

- The `Slime` class handles snake-like movement and growth.
- The `Bat` class manages its movement and bouncing.
- The `GameSceneUI` class manages all UI components.
- The `GameScene` class coordinates between these components and manages the game state.

This architecture makes it easier to add new features or fix bugs, as changes to one component are less likely to affect others.

Adding Input Buffering to the Slime Class

The game at this point is now playable. If you test it out though, you may notice a small issue with inputs. As we discussed in Chapter 10, in games where movement updates happen at fixed intervals, inputs can sometimes feel unresponsive, especially when trying to make multiple inputs in succession.

For instance, if a player wants to navigate a tight corner by pressing up and then immediately left, pressing these keys in rapid succession often results in only the second input being registered. When this happens, the slime will only continue left without first moving upward, missing the intended two-part movement completely. This occurs because the second input overwrites the first one before the game has a chance to process it, leading to frustrating gameplay.

Implementing the input buffering technique we introduced in Chapter 10 aims to solve this problem in our `Slime` class.

Implementing Input Buffering in the Slime Class

For the `Slime` class, we will implement input buffering based on the example given using a `Queue<T>` in Chapter 10. In the `GameObject` folder of the *DungeonSlime* project (your main game project), open the `Slime.cs` file so we can make the changes.

First, update the using statements at the top of the `Slime` class to add the `System.Linq` using statement:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using MonoGameLibrary;
using MonoGameLibrary.Graphics;
```

Next, add the following fields to the `Slime` class after the `_sprite` field:

```
// Buffer to queue inputs input by player during input polling.
private Queue<Vector2> _inputBuffer;

// The maximum size of the buffer queue.
private const int MAX_BUFFER_SIZE = 2;
```

The queue will store the directional vectors (up, down, left, right) that we will apply to the slime's movement in the order they were received. Next, we need to initialize the queue.

In the `Slime` class, locate the `Initialize` method and and update it to the following:

```
/// <summary>
/// Initializes the slime, can be used to reset it back to an initial state.
/// </summary>
/// <param name="startingPosition">The position the slime should start at.</param>
/// <param name="stride">The total number of pixels to move the head segment during
each movement cycle.</param>
public void Initialize(Vector2 startingPosition, float stride)
{
    // Initialize the segment collection.
    _segments = new List<SlimeSegment>();

    // Set the stride
    _stride = stride;

    // Create the initial head of the slime chain.
    SlimeSegment head = new SlimeSegment();
    head.At = startingPosition;
    head.To = startingPosition + new Vector2(_stride, 0);
    head.Direction = Vector2.UnitX;

    // Add it to the segment collection.
    _segments.Add(head);

    // Set the initial next direction as the same direction the head is
    // moving.
    _nextDirection = head.Direction;

    // Zero out the movement timer.
    _movementTimer = TimeSpan.Zero;

    // initialize the input buffer.
    _inputBuffer = new Queue<Vector2>(MAX_BUFFER_SIZE);
}
```

Next we need to update the input handling method to store the inputs in the queue instead of immediately overwriting the `_nextDirection` field.

In the `Slime` class, locate the `HandleInput` method and update it to the following:

```
private void HandleInput()
{
    Vector2 potentialNextDirection = Vector2.Zero;

    if (GameController.MoveUp())
    {
        potentialNextDirection = -Vector2.UnitY;
    }
    else if (GameController.MoveDown())
    {
        potentialNextDirection = Vector2.UnitY;
    }
    else if (GameController.MoveLeft())
    {
        potentialNextDirection = -Vector2.UnitX;
    }
    else if (GameController.MoveRight())
    {
        potentialNextDirection = Vector2.UnitX;
    }

    // If a new direction was input, consider adding it to the buffer
    if (potentialNextDirection != Vector2.Zero && _inputBuffer.Count
< MAX_BUFFER_SIZE)
    {
        // If the buffer is empty, validate against the current direction;
        // otherwise, validate against the last buffered direction
        Vector2 validateAgainst = _inputBuffer.Count > 0 ?
                                  _inputBuffer.Last() :
                                  _segments[0].Direction;

        // Only allow direction change if it is not reversing the current
        // direction.  This prevents th slime from backing into itself
        float dot = Vector2.Dot(potentialNextDirection, validateAgainst);
        if (dot >= 0)
        {
            _inputBuffer.Enqueue(potentialNextDirection);
        }
    }
}
```

1. The `potentialNewDirection` is now given the initial value of **Vector2.Zero**.
2. A check is made to see if the player has pressed a direction key and if the input buffer is not already at maximum capacity.
3. If a new direction key is pressed and the buffer has space:
    1. The validation is made using **Vector2.Dot** just like before to ensure it is a valid direction
    2. If it is a valid direction, then it is added to the queue.

Finally, we need to modify how we apply the movement direction during the movement update cycle.

In the `Slime` class, locate the `Move` method and update it to the following:

```
private void Move()
{
    // Get the next direction from the input buffer if one is available
    if (_inputBuffer.Count > 0)
    {
        _nextDirection = _inputBuffer.Dequeue();
    }

    // Capture the value of the head segment
    SlimeSegment head = _segments[0];

    // Update the direction the head is supposed to move in to the
    // next direction cached.
    head.Direction = _nextDirection;

    // Update the head's "at" position to be where it was moving "to"
    head.At = head.To;

    // Update the head's "to" position to the next tile in the direction
    // it is moving.
    head.To = head.At + head.Direction * _stride;

    // Insert the new adjusted value for the head at the front of the
    // segments and remove the tail segment. This effectively moves
    // the entire chain forward without needing to loop through every
    // segment and update its "at" and "to" positions.
    _segments.Insert(0, head);
    _segments.RemoveAt(_segments.Count - 1);

    // Iterate through all of the segments except the head and check
    // if they are at the same position as the head. If they are, then
    // the head is colliding with a body segment and a body collision
    // has occurred.
    for (int i = 1; i < _segments.Count; i++)
    {
        SlimeSegment segment = _segments[i];

        if (head.At == segment.At)
        {
            if (BodyCollision != null)
            {
                BodyCollision.Invoke(this, EventArgs.Empty);
            }

            return;
        }
```

```
        }
    }
```

The key change here is that we now dequeue a direction from the input buffer rather than directly using the `_nextDirection` value. This ensures we process inputs in the order they were received, preserving the player's intent.

With these changes in place, our game now supports input buffering. This small enhancement improves how the game feels to play, particularly when making rapid directional changes.

Players will notice:

- When navigating a corner, they can quickly press up followed by left (or any other valid combination), and both inputs will be respected
- The game feels more responsive since it remembers inputs between movement update cycles
- Complex maneuvers are easier to execute since timing is more forgiving

The difference might seem subtle, but it significantly reduces frustration during gameplay.

Putting It All Together

With all of these components now in place, our Dungeon Slime game has transformed from a simple demo built on learning MonoGame concepts into a complete snake-like game experience. The player controls the slime that moves through the dungeon, consuming bats to grow longer. If the slime collides with the wall or its own body, the game ends.

Now we can see how it all looks and plays:

Figure 23-1: Gameplay demonstration of the completed Dungeon Slime game showing the snake-like slime growing as it eats bats and a game over when colliding with the wall

1. The game starts with a single slime segment in the center of the room.
2. The player controls the direction of the slime by using the keyboard (arrow keys or WASD) or by using a game pad (DPad or left thumbstick).
3. The slime moves at regular intervals, creating a grid-based movement pattern.
4. When the slime eats a bat, it grows longer by adding a new segment to its tail.
5. The bat respawns at a strategic location after being eaten.
6. The player's score increases with each bat consumed.
7. If the slime collides with a wall or its own body, the game over panel appears.
8. On the game over panel, the player can choose to retry or return to the title scene.

With these mechanics implemented, Dungeon Slime is now a complete game with clear objectives, escalating difficulty, and a game feedback loop.

Conclusion

In this chapter, we have transformed our technical demo into a complete game by integrating UI systems with game mechanics. We have accomplished several important goals:

- Created a dedicated `GameSceneUI` class to manage the game's user interface.
- Implemented pause and game over screens that provide clear feedback to the player.
- Refactored the `GameScene` class to coordinate all game components.
- Added game state management to handle different gameplay conditions.
- Enhanced player control through input buffering for more responsive gameplay.
- Connected all of the elements to create a complete playable game.

The refactoring process we undertook demonstrates an important game development principle: separating concerns into specialized components makes code more maintainable and easier to extend. The `Slime` class manages snake-like behavior, the `Bat` class handles movement and collision response, and the `GameSceneUI` class encapsulates all UI-related functionality.

Test Your Knowledge

1. How does the game handle different states (playing, paused, game over), and why is this state management important?

   The game uses an enum (`GameState`) to track its current state and implements different behavior based on that state:

   - During the `Playing` state, the game updates all objects and checks for collisions
   - During the `Paused` state, the game shows the pause menu and stops updating game objects
   - During the `GameOver` state, the game shows the game over menu and prevents further gameplay

   This state management is important because it:

   - Prevents inappropriate updates during non-gameplay states
   - Creates a clear flow between different game conditions
   - Simplifies conditional logic by using explicit states rather than multiple boolean flags
   - Makes the game's behavior more predictable and easier to debug

2. Why is it important to position the bat away from the slime after it has been eaten rather than at a completely random location?

   Positioning the bat away from the slime after it has been eaten rather than at a completely random location is important because:

   - It prevents unfair situations where the bat might spawn right on top of the slime causing an immediate collision
   - It creates a more strategic gameplay experience by forcing the player to navigate toward the bat
   - It ensures the player faces an appropriate level of challenge that increases as the slime grows longer
   - It prevents potential frustration from random spawns that might be either too easy or too difficult to reach
   - It creates a more balanced and predictable game experience while still maintaining variety

3. What problem does input buffering solve and how does our implementation address it?

Input buffering solves the timing disconnect between when players press buttons and when the game can actually process those inputs in games with fixed movement cycles. Without buffering, inputs that occur between movement cycles are lost, especially when players make rapid sequential inputs like navigating corners.

Our implementation addresses this by:

- Using a queue data structure to store up to two directional inputs
- Processing inputs in First-In-First-Out order to preserve the player's intended sequence
- Validating each input against the previous one to prevent impossible movements

Introduction to Shaders

Learn how to create custom visual effects using shaders in MonoGame

In the previous chapters, we have built a complete snake-style game with animations, collision detection, audio, and scene management. While our game is fully functional, we can enhance the visual experience by implementing special effects to provide additional feedback to players. One powerful way to create these effects is through shaders.

In this chapter, you will:

- Understand what shaders are and how they function in MonoGame.
- Learn about different types of shaders including vertex and pixel shaders.
- Explore shader models and cross-platform considerations.
- Analyze the default shader template used in MonoGame.
- Create a custom grayscale shader for visual feedback.

> **IMPORTANT**
>
> This chapter is an introduction to shaders in MonoGame and will focus on the basic foundation of understanding how to create shader effect (*.fx*) files, loading them through the content pipeline, and using them in your game.
>
> If you want to learn more about the shader language itself, a good place to start would be the High-level shader language (HLSL) documentation on Microsoft Learn.
>
> For inspiration on what can be achieved with shaders, check out ShaderToy, which showcases real-time shader effects created by others. Note that ShaderToy uses OpenGL Shading Language (GLSL) which has some syntactic differences from HLSL, but the underlying concepts and mathematics are very similar.

We can begin by understanding what shaders are and how they work in MonoGame.

Understanding Shaders

Shaders are small programs that run directly on your graphics card (GPU) instead of your computer's main processor (CPU). While traditional game code runs on the CPU and handles things like game logic and physics, shaders run on the GPU and focus specifically on how things are drawn to the screen. This separation allows for efficient processing and enables visual effects that would be too performance-intensive to calculate on the CPU.

Think of shaders as special instructions that tell your GPU exactly how to display each pixel or transform each vertex in your game. By customizing these instructions, you can create a wide variety of visual effects without having to change your original artwork.

Types of Shaders

There are two main types of shaders that you will work with in MonoGame:

Vertex Shaders

Vertex shaders process the corners (vertices) of the shapes that make up your game objects. They determine where these points should be positioned on the screen and can manipulate their positions to create effects like:

- Waves or ripples in water
- Swaying grass or trees
- Character deformation for animations

Vertex shaders are especially important in 3D games, but they can also be used in 2D games for special effects that involve moving or distorting sprite positions.

> **NOTE**
>
> Even in 2D games like ours, MonoGame actually draws sprites by mapping textures onto simple 3D shapes called quads (rectangles made of two triangles). Each quad has four vertices, one at each corner. While we do not normally think about these vertices in 2D development, they are still there behind the scenes, and vertex shaders process them before rendering.

Pixel Shaders

Pixel shaders (also sometimes called fragment shaders) determine the actual color of each pixel that gets drawn to the screen. After the GPU figures out which pixels need to be drawn based on your geometry, the pixel shader calculates the final color for each of those pixels.

Pixel shaders are useful in 2D games for creating effects like:

- Color adjustments (brightness, contrast, saturation)
- Visual filters (grayscale, sepia, negative)
- Special transitions (fades, dissolves, color shifts)
- Dynamic lighting effects

For our Dungeon Slime game, we will focus primarily on pixel shaders since we want to create a color effect for our game over state.

> **NOTE**
>
> There are other types of shaders beyond vertex and pixel shaders, such as compute shaders, geometry shaders, and hull/domain shaders. These more advanced shader types enable powerful features like physics simulations, procedural geometry, and complex post-processing effects. However, they are not currently supported in the standard MonoGame implementation and are beyond the scope of this beginner tutorial. As the MonoGame graphics pipeline evolves, support for these advanced shader types may be added in future versions.

The Shader Pipeline

To understand how shaders work, it helps to visualize how data flows through the rendering pipeline.

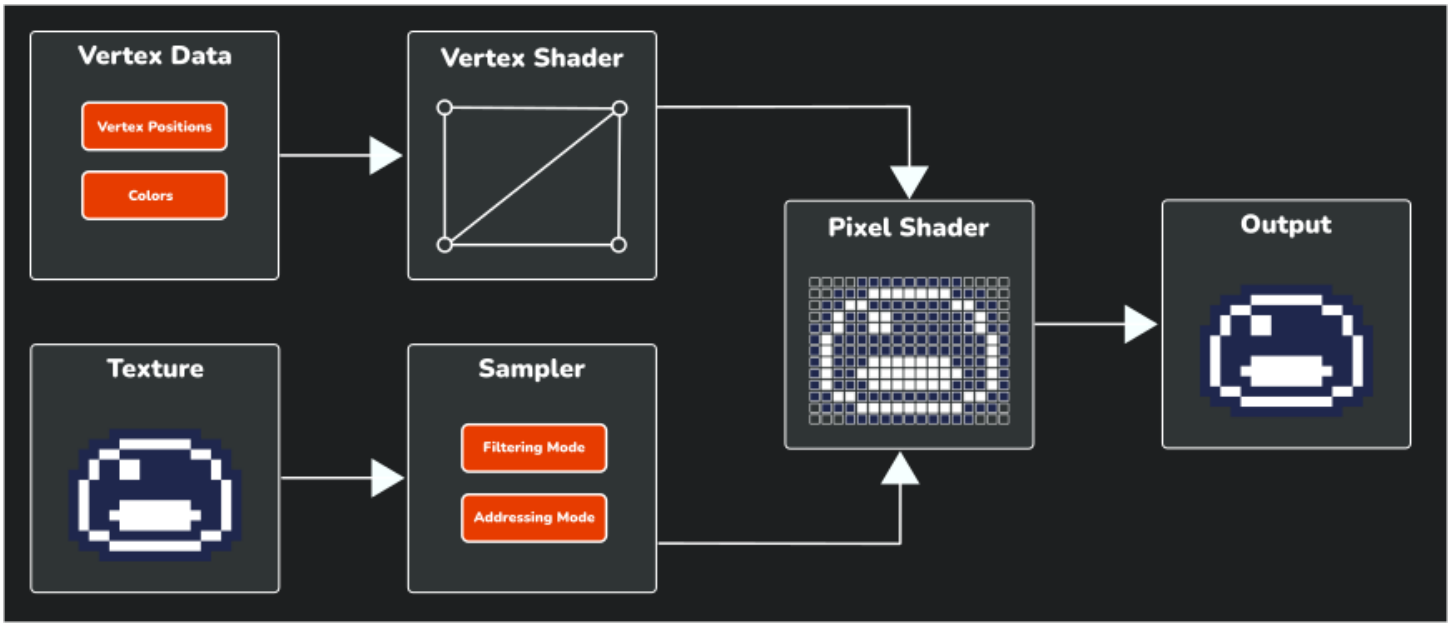


Figure 24-1: Basic shader pipeline showing how data flows through the rendering process

This diagram illustrates the fundamental steps of the shader pipeline:

1. **Input Data**: The process begins with input data that is sent to the GPU to render:
    1. **Vertex Data**: The vertex data (positions, colors, etc.) that define the geometry of what is being drawn.
    2. **Texture**: The image data that will be applied to the geometry.
2. **Vertex Shader**: Processes each vertex, calculating its final position on the screen and passing the data to the next stage.
3. **Sampler**: Controls how texture data is accessed, applying filtering (how pixels blend when scaled) and addressing (what happens at texture edges).
4. **Pixel Shader**: Takes the transformed vertices and sampled texture data to calculate the final color of each pixel.
5. **Output**: The final rendered image that appears on your screen.

> **NOTE**
>
> When working with **SpriteBatch** (2D) in MonoGame, the framework handles most of the vertex shader work automatically, which is why we will focus primarily on writing pixel shaders for visual effects.

Shader Languages and Cross-Platform Considerations

MonoGame uses the High-Level Shader Language (HLSL) for writing shader effects. **HLSL** is a C-like programming language developed by Microsoft for DirectX. As MonoGame also supports OpenGL which uses the OpenGL Shading Language (GLSL) instead of DirectX as it needs a way to make shaders work everywhere.

This is where MojoShader comes in. MojoShader is a library that automatically translates your **HLSL** shader code into whatever format the target platform requires (like **GLSL** for OpenGL platforms), this translation happens during the content build process when you compile your game.

Shader Models and Compatibility

Different platforms support different shader capabilities, known as "shader profiles." (Also known as "Shader Models" in some circles). When writing shaders for MonoGame, you need to consider compatibility across platforms.

MonoGame supports the following shader models when targeting DirectX platforms:

Vertex Shader Profile	Pixel Shader Profile
`vs_4_0_level_9_1`	`ps_4_0_level_9_1`
`vs_4_0_level_9_3`	`ps_4_0_level_9_3`
`vs_4_0`	`ps_4_0`
`vs_4_1`	`ps_4_1`
`vs_5_0`	`ps_5_0`

When targeting OpenGL platforms, MonoGame supports:

Vertex Shader Profile	Pixel Shader Profile
`vs_2_0`	`ps_2_0`
`vs_3_0`	`ps_3_0`

For maximum compatibility, it is best to target the following shader models:

- For DirectX platforms: `vs_4_0_level_9_1` (vertex) and `ps_4_0_level_9_1` (pixel)
- For OpenGL platforms: `vs_3_0` (vertex) and `ps_3_0` (pixel)

> **NOTE**
> In shader model notation, "vs" stands for "vertex shader" and "ps" stands for "pixel shader". The numbers represent the version and feature level of the shader model, with higher numbers indicating more advanced capabilities.

> **NOTE**
> MonoGame is currently planning to upgrade its graphics pipeline to support Vulkan and DirectX 12, which will significantly enhance graphical capabilities and shader support across platforms, enabling more advanced visual effects and better performance in future versions.

Understanding the Default Shader Template

When you create a new `Sprite Effect (.fx)` file in MonoGame using the MGCB Editor, it generates a default template file. To get a feel for the language used in shaders we can use this template to understand the foundation that we will build upon:

```
#if OPENGL
    #define SV_POSITION POSITION
    #define VS_SHADERMODEL vs_3_0
    #define PS_SHADERMODEL ps_3_0
#else
    #define VS_SHADERMODEL vs_4_0_level_9_1
    #define PS_SHADERMODEL ps_4_0_level_9_1
#endif

Texture2D SpriteTexture;

sampler2D SpriteTextureSampler = sampler_state
{
    Texture = <SpriteTexture>;
};

struct VertexShaderOutput
{
    float4 Position : SV_POSITION;
    float4 Color : COLOR0;
    float2 TextureCoordinates : TEXCOORD0;
};

float4 MainPS(VertexShaderOutput input) : COLOR
{
    return tex2D(SpriteTextureSampler,input.TextureCoordinates) * input.Color;
}

technique SpriteDrawing
{
    pass P0
    {
        PixelShader = compile PS_SHADERMODEL MainPS();
    }
};
```

Breaking down what each section of this template we can see:

1. **Platform compatibility defines**: The first block created defines for different shader model versions based on the target platform the shader gets compiled for (OpenGL vs DirectX).

```
#if OPENGL
    #define SV_POSITION POSITION
    #define VS_SHADERMODEL vs_3_0
    #define PS_SHADERMODEL ps_3_0
#else
    #define VS_SHADERMODEL vs_4_0_level_9_1
    #define PS_SHADERMODEL ps_4_0_level_9_1
#endif
```

> ⊗ **IMPORTANT**
>
> These preprocessor directives ensure your shader works across different platforms by defining appropriate shader models and semantics. When MonoGame compiles your shader:
>
> - On OpenGL platforms (macOS, Linux, etc.), it uses the `OPENGL` definition, setting shader models to `vs_3_0` and `ps_3_0`.
> - On DirectX platforms (Windows, Xbox), it uses the DirectX path, setting shader models to `vs_4_0_level_9_1` and `ps_4_0_level_9_1`.
>
> This compatibility block directly impacts which **HLSL** features you can use in your shader. To maintain cross-platform compatibility, you should restrict yourself to features available at the lowest shader model you target. Using advanced features only available in higher shader models will require additional conditional compilation blocks and platform-specific code paths.

2. **Texture declaration**: This declares a `Texture2D` variable called `SpriteTexture` that will receive the texture being drawn by the **SpriteBatch**.

```
Texture2D SpriteTexture;
```

3. **Sampler state**: This creates a `sampler2D` called `SpriteTextureSampler` that controls how the shader reads pixel data from the texture received from the **SpriteBatch**. It defines properties like filtering and addressing modes for texture sampling, similar to the **SamplerState** we discussed in Chapter 18.

```
sampler2D SpriteTextureSampler = sampler_state
{
    Texture = <SpriteTexture>;
};
```

4. **Vertex shader output structure**: This defines a struct called `VertexShaderOutput` with three fields; `Position`, `Color`, and `TextureCoordinates`. This struct represents the data that is passed from the **Vertex Shader** function to the **Pixel Shader** function.

```
struct VertexShaderOutput
{
    float4 Position : SV_POSITION;
    float4 Color : COLOR0;
    float2 TextureCoordinates : TEXCOORD0;
};
```

This struct defines the passing of:

- The position of the vertex in homogeneous coordinates (x, y, z, w).
- The RGBA color of the vertex.
- The UV coordinates for texture mapping.

> **TIP**
>
> Notice the unusual syntax like `float4 Position : SV_POSITION` where there is a colon followed by something after each variable declaration. These are called *semantics* in HLSL and they are special labels that define how the data should be used by the graphics hardware or passed between shader stages (like from vertex shader to pixel shader).
>
> For example, `SV_POSITION` tells the system "this contains the final screen position," `COLOR0` means "this contains color data," and `TEXCOORD0` means "this contains texture coordinates."
>
> Semantics are required in HLSL to connect your shader variables with the graphics pipeline. Without them, the GPU would not know what each piece of data represents or how to use it correctly.

5. **Pixel shader function**: `MainPS` is the main **Pixel Shader** function that determines the color of each pixel. The default implementation simply samples the texture at the current texture coordinates and multiplies it by the vertex color, which is the color value supplied in the **SpriteBatch.Draw** method call.

```
float4 MainPS(VertexShaderOutput input) : COLOR
{
    return tex2D(SpriteTextureSampler,input.TextureCoordinates) * input.Color;
}
```

6. **Technique and pass**: This block defines a technique called `SpriteDrawing` with a single pass called `P0`. The pass itself defines which shader functions to use.

```
technique SpriteDrawing
{
    pass P0
    {
        PixelShader = compile PS_SHADERMODEL MainPS();
    }
};
```

> **NOTE**
>
> You may have noticed that the default shader does not define a vertex shader method that would execute and return back the `VertexShaderOutput` value that is used as the input for the **Pixel Shader** function.
>
> When you are using **SpriteBatch**, MonoGame uses a built-in vertex shader under the hood that executes and creates this value to pass to the pixel shader function.

Understanding Techniques and Passes

The `technique` and `pass` sections might seem a bit confusing at first, but they are actually quite straightforward:

- A **technique** is like a recipe for rendering something. Each technique has a name (in this case `SpriteDrawing`) and contains one or more passes.
- A **pass** is a single step in that recipe. For simple effects, you often need just one pass (which is why the template only has `P0`), but more complex effects might use multiple passes to build up a final result.

Think of it like baking a cake:

- The technique is the overall cake recipe.
- Each pass is a step in that recipe (mix ingredients, bake, add frosting, etc).

In simple shaders, such as a grayscale shader, you would only need one technique with one pass. For more complex effects like blur, you might use multiple passes: one to blur horizontally and another to blur vertically.

> **TIP**
>
> For a real-world example of a blur shader with multiple techniques and passes, take a look at the Blur shader from the MonoGme Ship Game sample. This shader demonstrates how visual effects can be built by combining multiple rendering passes, with separate horizontal and vertical blur passes that work together to create a final blur effect.

The line `PixelShader = compile PS_SHADERMODEL MainPS();` simply tells the GPU which pixel shader function to use for this pass (in this case the `MainPS` function) and compiles it using the appropriate shader model defined earlier.

Using Shaders in MonoGame

Now that we understand what shaders are, we can explore how to integrate them into a MonoGame project. Before implementing a shader in our game, we will first take a look at the process of loading and using shaders with **SpriteBatch**.

To understand, we will start with a basic example (not for the game itself):

Loading Shader Effects

Like other game assets such as textures and sounds, shader effects are loaded through the content pipeline using the **ContentManager**. When loading a shader, we specify **Effect** as the target type.

```
// Example of loading an effect
Effect exampleEffect = Content.Load<Effect>("exampleEffect");
```

You should typically load shader effects during your game's **LoadContent** method along with other game assets, and store them in class fields so they can be accessed during the **Draw** method.

Using Effects With SpriteBatch

Once you have loaded a shader effect, applying it to your game's visuals requires integrating it with the **SpriteBatch**. The effect is specified during the `Begin` call, **but is actually applied during drawing operations or when `End` is called (depending on the SpriteSortMode)**.

```
// Specify the effect during Begin
spriteBatch.Begin(effect: exampleEffect);

// The effect will be applied when these draw calls are processed
spriteBatch.Draw(exampleTexture, texturePosition, Color.White);
spriteBatch.DrawString(exampleFont, "Hello World", textPosition, Color.White);

// For most SpriteSortMode values, actual drawing with the effect happens here
spriteBatch.End();
```

Setting Effect Parameters

Most shader effects have parameters that you can adjust to control their behavior. For example, the shader we will create for our game will have a `Saturation` parameter. You should set these parameters before the actual drawing occurs:

```
// Update the parameter value
exampleEffect.Parameters["Saturation"].SetValue(0.5f);

// Specify the effect during Begin
spriteBatch.Begin(effect: exampleEffect);

// Draw calls will use the effect with Saturation = 0.5f when processed
```

Implementing a Shader In Our Game

Now it is time to implement a shader for our game. The shader we will create is a simple grayscale effect that can be applied when the game is paused or there is a game over to provide visual feedback to the player that the game is inactive, the background will become grey offsetting the color of the Pause menu or other prompt.

Creating the Shader File

First, we need to create a new shader effect file and add it to our content project.

1. In the *DungeonSlime* project (your main game project), open the `/Content/Content.mgcb` content project file in the **MGCB** Editor.
2. In the MGCB Editor, right-click the `Content` now and choose `Add > New Folder...`.
3. Give the new folder the name `effects` and click the `Ok` button.
4. Right-click on the new `effects` folder in the MGCB Editor and choose `Add > New Item...`.
5. Choose `Sprite Effect (.fx)` from the type list and name the file `grayscaleEffect`, then click the `Ok` button.
6. **Save the changes in the MGCB Editor then close it.**

The steps above will create a new shader effect (*.fx*) file with the default template we discussed earlier. Now, we need to modify this template to create our grayscale effect.

Writing the Grayscale Shader

In the *DungeonSlime* project (your main game project), open the `Content/effects/grayscaleEffect.fx` file that we just created in your code editor and modify it as follows:

```
#if OPENGL
    #define SV_POSITION POSITION
    #define VS_SHADERMODEL vs_3_0
    #define PS_SHADERMODEL ps_3_0
#else
    #define VS_SHADERMODEL vs_4_0_level_9_1
    #define PS_SHADERMODEL ps_4_0_level_9_1
#endif

Texture2D SpriteTexture;

// A value between 0 and 1 that controls the intensity of the grayscale effect.
// 0 = full color, 1 = full grayscale.
float Saturation = 1.0;

sampler2D SpriteTextureSampler = sampler_state
{
    Texture = <SpriteTexture>;
};

struct VertexShaderOutput
{
    float4 Position : SV_POSITION;
    float4 Color : COLOR0;
    float2 TextureCoordinates : TEXCOORD0;
};

float4 MainPS(VertexShaderOutput input) : COLOR
{
    // Sample the texture
    float4 color = tex2D(SpriteTextureSampler, input.TextureCoordinates)
* input.Color;

    // Calculate the grayscale value based on human perception of colors.
    float grayscale = dot(color.rgb, float3(0.3, 0.59, 0.11));

    // create a grayscale color vector (same value for R, G, and B)
    float3 grayscaleColor = float3(grayscale, grayscale, grayscale);

    // Linear interpolation between he grayscale color and the original color's
    // rgb values based on the saturation parameter.
    float3 finalColor = lerp(grayscale, color.rgb, Saturation);

    // Return the final color with the original alpha value.
    return float4(finalColor, color.a);
}
```

```
technique SpriteDrawing
{
    pass P0
    {
        PixelShader = compile PS_SHADERMODEL MainPS();
    }
};
```

The key modifications made to create this grayscale effect include:

1. **Added a Parameter**: A `Saturation` parameter was added that controls the intensity of the grayscale effect.
    - When set to **0**, the image will be fully grayscale.
    - When set to **1**, the image will be its original color.
    - Values in between create a partial grayscale effect.
2. **Modified the Pixel Shader**: The `MainPS` function has been updated to:
    - Sample the original color from the texture.
    - Calculate a grayscale value by taking a weighted average of the RGB components.
    - Create a grayscale color vector from this single brightness value.
    - Use `lerp` (linear interpolation) to blend between the grayscale and original color's rgb values based on the `Saturation` parameter.
    - Preserve the original alpha (transparency) value.
    - Output the new color value with the grayscale effect for the pixel.

Understanding the Shader Code

The heart of the grayscale effect is this line:

```
// Calculate the grayscale value based on human perception of colors.
float grayscale = dot(color.rgb, float3(0.3, 0.59, 0.11));
```

This uses the `dot` function to calculate the dot product between the color's RGB values and the vector $(0.3, 0.59, 0.11)$. This effectively calculates a weighted average where:

- Red contributes 30%.
- Green contributes 59%.
- Blue contributes 11%.

> **NOTE**
>
> These specific weights are based on how the human eye perceives brightness in different colors. Green appears brighter to us than red, which appears brighter than blue.
>
> The weighted values themselves are based on the formula that represents the luma component from the ITU-R BT.601 standard, which is commonly used for converting RGB images to grayscale based on human perception where:
>
> $$Y'_{601} = 0.299R' + 0.587G' + 0.114B'$$
>
> By using these weights, we get a natural looking grayscale conversion.

Then, we use linear interpolation with the `lerp` function to blend between this grayscale value and the original color's RGB values:

```
// Linear interpolation between he grayscale color and the original color's
// rgb values based on the saturation parameter.
float3 finalColor = lerp(grayscale, color.rgb, Saturation);
```

The `lerp` function blends between the first two parameters based on the third parameter (`Saturation`). When `Saturation` is **0**, we get full grayscale, when it is **1**, we get the original color.

Implementing the Grayscale Shader

Now that we have our grayscale shader, we can implement it in our game when the game is paused or a game over state occurs. In the *DungeonSlime* project (your main game project), open the `Scenes/GameScene.cs` file and perform the following:

1. First, add these fields to the `GameScene` class after the `private GameState _state` property:

```
// The grayscale shader effect.
private Effect _grayscaleEffect;

// The amount of saturation to provide the grayscale shader effect.
private float _saturation = 1.0f;

// The speed of the fade to grayscale effect.
private const float FADE_SPEED = 0.02f;
```

2. Next, update the `LoadContent` method to load the grayscale shader:

```
public override void LoadContent()
{
    // Create the texture atlas from the XML configuration file.
    TextureAtlas atlas = TextureAtlas.FromFile(Core.Content,
"images/atlas-definition.xml");

    // Create the tilemap from the XML configuration file.
    _tilemap = Tilemap.FromFile(Content, "images/tilemap-definition.xml");
    _tilemap.Scale = new Vector2(4.0f, 4.0f);

    // Create the animated sprite for the slime from the atlas.
    AnimatedSprite slimeAnimation = atlas.CreateAnimatedSprite("slime-
animation");
    slimeAnimation.Scale = new Vector2(4.0f, 4.0f);

    // Create the slime.
    _slime = new Slime(slimeAnimation);

    // Create the animated sprite for the bat from the atlas.
    AnimatedSprite batAnimation = atlas.CreateAnimatedSprite("bat-animation");
    batAnimation.Scale = new Vector2(4.0f, 4.0f);

    // Load the bounce sound effect for the bat.
    SoundEffect bounceSoundEffect = Content.Load<SoundEffect>("audio/bounce");

    // Create the bat.
    _bat = new Bat(batAnimation, bounceSoundEffect);

    // Load the collect sound effect.
    _collectSoundEffect = Content.Load<SoundEffect>("audio/collect");

    // Load the grayscale effect.
    _grayscaleEffect = Content.Load<Effect>("effects/grayscaleEffect");
}
```

3. Next, update the `TogglePause` method so that when the game is paused, it sets the saturation value to **1.0f**:

```
private void TogglePause()
{
    if (_state == GameState.Paused)
    {
        // We're now unpausing the game, so hide the pause panel.
        _ui.HidePausePanel();

        // And set the state back to playing.
        _state = GameState.Playing;
    }
    else
    {
        // We're now pausing the game, so show the pause panel.
        _ui.ShowPausePanel();

        // And set the state to paused.
        _state = GameState.Paused;

        // Set the grayscale effect saturation to 1.0f
        _saturation = 1.0f;
    }
}
```

4. We also need to update the `GameOver` method so that when a game over state occurs, it sets the saturation value to **1.0f**:

```
private void GameOver()
{
    // Show the game over panel.
    _ui.ShowGameOverPanel();

    // Set the game state to game over.
    _state = GameState.GameOver;

    // Set the grayscale effect saturation to 1.0f
    _saturation = 1.0f;
}
```

5. Next, modify the `Update` method to handle the grayscale transition:

```
public override void Update(GameTime gameTime)
{
    // Ensure the UI is always updated.
    _ui.Update(gameTime);

    if (_state != GameState.Playing)
    {
        // The game is in either a paused or game over state, so
        // gradually decrease the saturation to create the fading grayscale.
        _saturation = Math.Max(0.0f, _saturation - FADE_SPEED);

        // If its just a game over state, return back.
        if (_state == GameState.GameOver)
        {
            return;
        }
    }

    // If the pause button is pressed, toggle the pause state.
    if (GameController.Pause())
    {
        TogglePause();
    }

    // At this point, if the game is paused, just return back early.
    if (_state == GameState.Paused)
    {
        return;
    }

    // Update the slime.
    _slime.Update(gameTime);

    // Update the bat.
    _bat.Update(gameTime);

    // Perform collision checks.
    CollisionChecks();
}
```

6. Finally, update the `Draw` method to apply the shader when the game is paused or in a game over state:

```
public override void Draw(GameTime gameTime)
{
    // Clear the back buffer.
    Core.GraphicsDevice.Clear(Color.CornflowerBlue);

    if (_state != GameState.Playing)
    {
        // We are in a game over state, so apply the saturation parameter.
        _grayscaleEffect.Parameters["Saturation"].SetValue(_saturation);

        // And begin the sprite batch using the grayscale effect.
        Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp,
effect: _grayscaleEffect);
    }
    else
    {
        // Otherwise, just begin the sprite batch as normal.
        Core.SpriteBatch.Begin(samplerState: SamplerState.PointClamp);
    }

    // Draw the tilemap
    _tilemap.Draw(Core.SpriteBatch);

    // Draw the slime.
    _slime.Draw();

    // Draw the bat.
    _bat.Draw();

    // Always end the sprite batch when finished.
    Core.SpriteBatch.End();

    // Draw the UI.
    _ui.Draw();
}
```

> **NOTE**
>
> Notice how we set the shader parameters with the current saturation value every frame before beginning the sprite batch. This is because shaders are stateless; they do not remember any values from the previous draw cycle. Each time the GPU processes a shader, it only works with the parameters provided in that specific frame. Even if the saturation value has not changed since the last frame, we still need to send it to the shader again to apply it. This is why we constantly update the shader parameters in the `Draw` method rather than only when the value is changed.

With these changes, when the game enters a paused or game over state, the screen will gradually fade to gray using the grayscale shader effect. This provides a clear indication that the game is inactive during these states.

Figure 24-2: The game, now using a grayscale effect when paused or a game over state occurs to visually indicate that the game is inactive

Important Considerations

When working with effects in **SpriteBatch**, there are some important points to keep in mind:

1. Only one effect can be applied to a **SpriteBatch.Begin**/**SpriteBatch.End** at a time. If you need to use multiple effects for different sprites, you will need multiple **SpriteBatch.Begin**/**SpriteBatch.End** blocks.

```
// Begin sprite batch with effect.  All draw calls made within this begin/end
block will have the effect applied.
spriteBatch.Begin(effect: exampleEffect1)
spriteBatch.Draw(texture, position, color);
spriteBatch.End();

// Begins sprite batch with a different effect.  All draw calls made within this
begin/end block will have the specified effect applied.
spriteBatch.Begin(effect: exampleEffect2)
spriteBatch.Draw(texture, position, color);
spriteBatch.End();
```

2. Along with #1 above, if you want to be selective and only have the effect apply to some sprites and not others, you will still need to use multiple **SpriteBatch.Begin**/**SpriteBatch.End** blocks

```
// Begin sprite batch with effect.  All draw calls made within this begin/end
block will have the specified effect applied.
spriteBatch.Begin(effect: exampleEffect)
spriteBatch.Draw(texture, position, color);
spriteBatch.End();

// Begin sprite batch without effect so that draw calls made here have no
effect applied.
spriteBatch.Begin();
spriteBatch.Draw(texture, position, color);
spriteBatch.End();
```

3. Even though the effect to use is specified during the **SpriteBatch.Begin** call, the actual effect is not applied until all batched items are processed when **SpriteBatch.End** is called. This means if you adjust parameter values of the effect between draw calls, **only the last parameter value set** is what is applied to all draw calls within the **SpriteBatch.Begin**.

```
// Begin sprite batch with effect.
// Specifying the effect here is only specifying what effect to apply when
batching ends
spriteBatch.Begin(effect: exampleEffect);

// Change a parameter and draw something
exampleEffect.Parameters["ExampleParameter"].SetValue(1.0f);
spriteBatch.Draw(texture1, position, color);

// Change the parameter to a different value and draw something else.
exampleEffect.Parameters["ExampleParameter"].SetValue(0.5f);
spriteBatch.Draw(texture2, position, color);

// The actual effect is applied to the draw calls here, when End is called.
// This means the value of the parameter that will be used in the effect
// is the last parameter value set (0.5f) which will be applied to both
// draw calls instead of each of them having different parameter values.
spriteBatch.End();
```

4. There is an exception to #2 above. In Chapter 06: Working with Textures, we discussed the different **SpriteSortMode** values that can be used when rendering. From this chapter, we learned that when using **SpriteSortMode.Immediate** that when a draw call is made, it is immediately flushed to the GPU and rendered to the screen, ignoring batching. This means that if you are using **SpriteSortMode.Immediate** then changing parameters between draw calls will apply the parameter change after it is made for the next draw call.

> ⊗ **IMPORTANT**
>
> As mentioned in Chapter 06, **SpriteSortMode.Immediate** can cause performance issues and should only be used when absolutely necessary, as `Immediate` mode effectively disables batching.

```
// Begins sprite batch with the effect AND intentionally
specifying SpriteSortMode.Immediate
spriteBatch.Begin(effect: exampleEffect, sortMode: SpriteSortMode.Immediate);

// Change a parameter and draw something.  Since we are in immediate mode, the
value of the parameter is used because the shader is immediately applied to the
draw call.
exampleEffect.Parameters["ExampleParameter"].SetValue(1.0f);
spriteBatch.Draw(texture1, position, color);

// Change the parameter to something else.  Since we are in immediate mode, the
new value of the parameter is used because the shader is immediately applied to
the draw call.
exampleEffect.Parameters["ExampleParameter"].SetValue(0.5f);
spriteBatch.Draw(texture2, position, color);

// Since immediate mode is used, batching is not performed so the effect was
applied immediately on the draw calls above and not here during the end call.
spriteBatch.End();
```

Conclusion

This chapter explored the fundamentals of creating and adding a shader to MonoGame by implementing a grayscale shader effect. We covered several important concepts:

- Shaders are specialized programs that run directly on the GPU, allowing for efficient visual effects that would be too performance-intensive to calculate on the CPU.
- Effect parameters allow runtime control of shader behavior.
- In MonoGame, shaders are written in High-Level Shader Language (HLSL) and can be loaded through the content pipeline like other game assets.
- Pixel shaders determine the color of each rendered pixel, making them ideal for effects like our grayscale shader.
- The timing of the shader application depends on the **SpriteSortMode** used with **SpriteBatch**, with most effects being applied during the **SpriteBatch.End** call.

While we only focused on a simple grayscale effect, the principles learned here can be used to start learning more about shaders and creating more complex visual effects.

Test Your Knowledge

1. What is the primary difference between pixel shaders and vertex shaders in the context of 2D games?

   Pixel shaders determine the color of each pixel being rendered, making them ideal for visual effects like grayscale, color tinting, and image filters. Vertex shaders manipulate the position of vertices, which can be used for effects like distortion, waves, or character deformation. In 2D games, pixel shaders are more commonly used for visual effects while vertex shaders are less frequently needed.

2. What function is used in the shader to blend between grayscale and the original color, and how does it work?

   The `lerp` function (linear interpolation) is used to blend between grayscale and original colors. It takes three parameters: the first is the grayscale color vector, the second is the original color's RGB values, and the third is the `Saturation` parameter (between 0 and 1). When Saturation is 0, the output is fully grayscale; when it is 1, the output is the original color; values in between create a partial blend of both.

3. Why can changing effect parameters between **SpriteBatch.Draw** calls sometimes not work as expected, and what is the exception to this?

   Changing effect parameters between draw calls typically does not work as expected because the effect is actually applied during **SpriteBatch.End**, not during the draw calls. The last parameter value set before end is applied to all draw calls in the batch. The exception is when using **SpriteSortMode.Immediate**, which causes each draw call to be processed immediately rather than batched, allowing parameter changes to be applied between individual draw calls.

Chapter 25: Packaging Your Game for Distribution

Learn how to package your game for distribution across Windows, macOS, and Linux platforms.

After all of our work creating Dungeon Slime, we need to prepare the game for distribution to players. Properly packaging your game ensures it runs correctly on different platforms without requiring players to have development tools installed.

In this chapter you will:

- Learn how to prepare your game for release.
- Package your game for Windows, macOS, and Linux platforms.
- Create platform-specific distributions with appropriate configurations.
- Understand important publishing parameters and their impact on game performance.
- Address common cross-platform distribution challenges.
- Learn about third-party tools that can automate the packaging process.

Understanding Game Packaging

When developing with MonoGame, you are working in a .NET environment that abstracts away many platform-specific details. However, when distributing your game to players, you need to ensure they can run it without installing the .NET runtime or other development dependencies.

Self-Contained Deployments

The recommended approach for distributing MonoGame games is to use self-contained deployments. This approach packages your game with all necessary .NET dependencies, resulting in a larger distribution but ensuring your game runs without requiring players install additional runtimes.

A self-contained deployment offers several advantages:

- Players can run your game without installing the .NET runtime.
- Your game will always use the exact version of the runtime it was developed with.
- Distribution is simplified with fewer external dependencies.

The main trade-off is a larger distribution size compared to framework-dependent deployments, but this is usually worth it for the improved player experience.

Preparing Your Game for Release

Before packaging your game for distribution, you should take some preparatory steps:

1. **Set Release Configuration**: Ensure your build configuration is set to "Release" rather than "Debug" for better performance and smaller executable size.
2. **Update Game Information**: Verify your game's title, version, and other information in the project's properties file (`.csproj`).
3. **Final Testing**: Perform thorough testing in Release mode to catch any issues that might not appear in Debug mode.
4. **Asset Optimization**: Consider optimizing larger content files to reduce the final package size.

Platform-Specific Packaging

Now that we understand the general packaging concepts, we will explore how to create distributions for Windows, macOS, and Linux. Each platform has specific requirements and tooling that we will need to navigate. Choose the instructions below based on the platform you are using.

> **IMPORTANT**
>
> The packaging instructions for each platform are designed to be executed on that same platform. This is because each operating system provides specific tools needed for proper packaging (like `lipo` on macOS or `permission settings` on Unix-based systems). When building on Windows for macOS or Linux, the executable permissions cannot be set since Windows lacks these concepts.
>
> While you will need access to each platform for the steps below, do not worry if you do not have all these systems available. At the end of this chapter, third party libraries provided by MonoGame community members are included that can automate these processes for you without requiring you to own each type of machine.

Windows

Windows is the most straightforward platform to target since MonoGame development typically happens on Windows machines.

Building for Windows

To create a self-contained application for Windows, open a new command prompt window in the same folder as the main game project (in our case the folder with the `DungeonSlime.csproj` file) and execute the following .NET CLI command:

```
dotnet publish -c Release -r win-x64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

This command specifies:

- `-c Release`: Builds in Release configuration for better performance.
- `-r win-x64`: Targets 64-bit Windows platforms.
- `-p:PublishReadyToRun=false`: Disables ReadyToRun compilation (explained later).
- `-p:TieredCompilation=false`: Disables Tiered Compilation (explained later).
- `--self-contained`: Includes the .NET runtime in the package.

The output will be placed in a directory like `bin/Release/net8.0/win-x64/publish/`, relative to the game's `.csproj` file. This directory will contain the executable and all necessary files to run your game.

> **NOTE**
> If your base game project is created with **dotnet 9.0** (which at the time of writing is the default), the above folder will be more like `bin/Release/net9.0/win-x64/publish/`, just so you are aware. Noting the change in folder from `net8.0` to `net9.0`.

Creating a Windows Distribution

Once you have created a build for Windows, to create a Windows distribution, you can simply:

1. Zip the entire contents of the publish folder.
2. Distribute the ZIP file to your players.
3. Players can extract the ZIP and run the executable directly.

> **NOTE**
> If you are using the WindowsDX platform target, players may need to install the DirectX June 2010 Runtime for audio and gamepad support. If you are targeting this platform, consider including this information in your game's documentation.

Packaging for macOS requires creating an **Application Bundle** (`.app`), which is a directory structure that macOS recognizes as an application.

Building for macOS

For macOS, you will need to build for both the Intel (x64) and Apple Silicon (arm64) to support all modern Mac computers. Open a new terminal window in the same folder as the `DungeonSlime.csproj` file (the main game project).

> **TIP**
>
> The following sections will guide you through several terminal commands that build on each other. It is best to use a single terminal window located in your projects root directory (where the `DungeonSlime.csproj` file is) for all of these steps to ensure paths remain consistent.

First, to create the Intel (x64) self contained application, execute the following .NET CLI command in the terminal:

```
dotnet publish -c Release -r osx-x64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

This command specifies:

- `-c Release`: Builds in Release configuration for better performance.
- `-r osx-x64`: Targets Intel (x64) macOS platforms.
- `-p:PublishReadyToRun=false`: Disables ReadyToRun compilation (explained later).
- `-p:TieredCompilation=false`: Disables Tiered Compilation (explained later).
- `--self-contained`: Includes the .NET runtime in the package.

The output from this command will be placed in a directory like `bin/Release/net8.0/osx-x64/publish/`, relative to the game's `.csproj` file.

> **NOTE**
>
> If your base game project is created with **dotnet 9.0** (which at the time of writing is the default), the above folder will be more like `bin/Release/net9.0/osx-x64/publish/`, just so you are aware. Noting the change in folder from `net8.0` to `net9.0`.

Next, to create the Apple Silicon (arm64) self contained application for macOS, in the same terminal window, execute the following .NET CLI command:

```
dotnet publish -c Release -r osx-arm64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

The only difference in this command is the use of `-r osx-arm64` which specifies to target the Apple Silicon (arm64) macOS platform.

The output from this command will be placed in a directory like `bin/Release/net8.0/osx-arm64/publish/`, relative to the game's `.csproj` file.

Creating a macOS Application Bundle

With the Intel (x64) and Apple Silicon (arm64) builds completed, we can now create the macOS **Application Bundle**. macOS applications follow a very specific directory structure:

```
YourGame.app/
├── Contents/
│   ├── Info.plist
│   ├── MacOS/
│   │   └── YourGame
│   └── Resources/
│       ├── Content/
│       └── YourGame.icns
```

To create this structure, from the same terminal window:

1. First, create the folder structure by executing the following commands:

```
mkdir -p bin/Release/DungeonSlime.app/Contents/MacOS/
mkdir -p bin/Release/DungeonSlime.app/Contents/Resources/Content
```

> **NOTE**
>
> The `mkdir -p` command creates directories including any necessary parent directories. The `-p` flag ensures all intermediate directories are created without error if they do not exist yet.

2. Copy all files from the Intel (x64) build to the MacOS directory. This ensures all the required dependencies are included. To do this, execute the following command:

```
cp -R bin/Release/net8.0/osx-x64/publish/*
bin/Release/DungeonSlime.app/Contents/MacOS/
```

> **NOTE**
>
> This copies all files from the publish directory, including the executable, all dependent `.dll` files, and the `Content` directory that contains your game assets.

3. Replace the executable with a universal binary that works on both Intel and Apple Silicon Macs. To do this, execute the following command:

```
lipo -create bin/Release/net8.0/osx-arm64/publish/DungeonSlime
bin/Release/net8.0/osx-x64/publish/DungeonSlime -output
bin/Release/DungeonSlime.app/Contents/MacOS/DungeonSlime
```

> **NOTE**
>
> The `lipo` command is a macOS utility that works with multi-architecture binaries. Here, it combines the Intel (x64) and Apple Silicon (arm64) executables into a single "universal binary" that can run natively on both Apple Silicon and Intel processor based Macs.

4. Move the Content directory from the MacOS directory to the Resources directory, following macOS application bundle conventions. To do this, execute the following command:

```
mv bin/Release/DungeonSlime.app/Contents/MacOS/Content
bin/Release/DungeonSlime.app/Contents/Resources/
```

> **NOTE**
>
> This moves the `Content` directory to the expected location for resources in a macOS application bundles.

5. Create a new file called `Info.plist` in the `Contents` directory of the application bundle with the following command:

```
touch bin/Release/DungeonSlime.app/Contents/Info.plist
```

> **NOTE**
>
> The `touch` command creates an empty file if it does not exist or updates the modification time if it does exist. We are using it here to create a blank file that we will populate with content in the next step.

6. Open the `Info.plist` file you just created in a text editor and add the following content to the file and save it.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>DungeonSlime</string>
    <key>CFBundleIconFile</key>
    <string>DungeonSlime</string>
    <key>CFBundleIdentifier</key>
    <string>com.yourdomain.dungeonslime</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>DungeonSlime</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>FONV</string>
    <key>CFBundleVersion</key>
    <string>1</string>
    <key>LSApplicationCategoryType</key>
    <string>public.app-category.games</string>
    <key>LSMinimumSystemVersion</key>
    <string>10.15</string>
    <key>NSHumanReadableCopyright</key>
    <string>Copyright © 2025</string>
    <key>NSPrincipalClass</key>
    <string>NSApplication</string>
    <key>LSRequiresNativeExecution</key>
    <true/>
    <key>LSArchitecturePriority</key>
    <array>
        <string>arm64</string>
    </array>
</dict>
</plist>
```

NOTE

The `Info.plist` file is a critical component of any macOS application bundle. It contains essential metadata that macOS uses to:

- Identify the application (bundle identifier, name, version).
- Display the application correctly in Finder and the Dock.
- Associate the correct icon with the application.
- Define the application's capabilities and requirements.
- Provide copyright and developer information.

Without a properly formatted `Info.plist` file, macOS would not recognize your game as a valid application, and users would be unable to launch it through normal means.

TIP

The highlighted sections in the `Info.plist` file need to be customized for your game:

- Replace all instances of "DungeonSlime" with your game's name.
- The `CFBundleIconFile` value (line 10) must exactly match the name of your `.icns` file that we will create in the next step.
- Update the bundle identifier on line 12 with your domain.
- Modify the copyright information on line 30 as needed.

Getting these values right, especially the icon filename, ensures your game appears correctly on macOS.

For more information on the `Info.plist` manifest file, refer to the About Info.plist Keys and Values Apple Developer documentation.

7. Next, create the application bundle `.icns` icon file. To do this, perform the following:

1. First, you will need a `.png` file that can be used to create the icon set for the final `.icns` output. If you already have a `.png` icon for your game, ensure it is in the root of the main project directory and is named `Icon.png`. If you do not have one already prepared, you can use the `Icon.bmp` that was generated in the root of the main project directory when you initially created the project. However, it will need to be converted to a `.png` first. To do this, execute the following command:

```
sips -s format png Icon.bmp --out Icon.png
```

> **NOTE**
>
> `sips` (Scriptable Image Processing System) is a command line tool in macOS for image manipulation. Here we are using it to convert a `.bmp` to a `.png`. In a moment, we will also use it to resize the `.png` into different icon sizes required for the application bundle.

2. Next, create a directory that we can output each of the generated `.png` icon files to for the icon set. Execute the following command:

```
mkdir -p bin/Release/DungeonSlime.iconset
```

3. Now we use the `sips` command to generate the icon for each size required for a mac app bundle. Each size generated is necessary for different display scenarios in macOS (Dock, Finder, etc.). To do this, execute the following commands:

```
 sips -z 16 16 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_16x16.png
 sips -z 32 32 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_16x16@2x.png
 sips -z 32 32 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_32x32.png
 sips -z 64 64 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_32x32@2x.png
 sips -z 128 128 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_128x128.png
 sips -z 256 256 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_128x128@2x.png
 sips -z 256 256 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_256x256.png
 sips -z 512 512 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_256x256@2x.png
 sips -z 512 512 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_512x512.png
 sips -z 1024 1024 Icon.png --out
bin/Release/DungeonSlime.iconset/icon_512x512@2x.png
```

4. Finally, combine all of the generated icons for the icon set into a `.icns` file. To do this, execute the following:

```
iconutil -c icns bin/Release/DungeonSlime.iconset --output
bin/Release/DungeonSlime.app/Contents/Resources/DungeonSlime.icns
```

> **NOTE**
>
> `iconutil` is a command line tool in macOS used to convert icon sets into a single high-resolution `.icns` file.

> **TIP**
>
> After creating the `.icns` file using the above command, if you open the folder in Finder with `DungeonSlime.app` and it shows a blank square as the icon instead of the one you just created, right-click on `DungeonSlime.app` and choose `Get Info` from the context menu. This will force it to do a refresh and show the icon properly. After doing this, if the icon still does not show, then you need to double check that the `CFBundleIconFile` value in the `Info.plist` is named **exactly** the same as the `.icns` file that was created (minus the extension).

8. Set executable permissions for the game executable. To do this, execute the following command:

```
chmod +x bin/Release/DungeonSlime.app/Contents/MacOS/DungeonSlime
```

> **NOTE**
>
> the `chmod +x` command changes the file permissions to make it executable. Without this step, macOS would not be able to run the application.

Distributing for macOS

For macOS distribution:

1. Archive the application bundle using the `tar.gz` archive format to preserve the executable permissions that were set. To do this, execute the following command in the same terminal window:

```
tar -czf DungeonSlime-osx.tar.gz -C bin/Release/DungeonSlime.app
```

> **NOTE**
>
> The `tar` command creates an archive file:
>
> - `-c` creates a new archive.
> - `-z` compresses the archive using gzip.
> - `-f` specifies the filename to create
> - `-C` changes to the specified directory before adding files.
>
> Unlike **ZIP** files, the `tar.gz` format preserves Unix file permissions, which is crucial for maintaining the executable permission we set in the previous steps.

2. Distribute the `tar.gz` archive file to players.

3. Players can extract the `tar.gz` archive file and run the application bundle to play the game.

Linux

Linux packaging is relatively straightforward, but requires attention to ensure executable permission are set.

Building for Linux

To create a self-contained application for Linux, open a new Terminal window in the same folder as the `DungeonSlime.csproj` file (your main game project folder) and execute the following .NET CLI command:

```
dotnet publish -c Release -r linux-x64 -p:PublishReadyToRun=false -
p:TieredCompilation=false --self-contained
```

- `-c Release`: Builds in Release configuration for better performance.
- `-r linux-x64`: Targets 64-bit Linux platforms.
- `-p:PublishReadyToRun=false`: Disables ReadyToRun compilation (explained later).
- `-p:TieredCompilation=false`: Disables Tiered Compilation (explained later).
- `--self-contained`: Includes the .NET runtime in the package.

The output will be placed in a directory like `bin/Release/net8.0/linux-x64/publish`, relative to the `DungeonSlime.csproj` file. This folder will contain the executable and all necessary files to run the game.

> **NOTE**
>
> If your base game project is created with **dotnet 9.0** (which at the time of writing is the default), the above folder will be more like `bin/Release/net9.0/linux-x64/publish/`, just so you are aware. Noting the change in folder from `net8.0` to `net9.0`.

Creating a Linux Distribution

Once you have created a build for Linux, to create a distributable archive:

1. Ensure the main executable has proper execution permissions by executing the following command in the same terminal window:

```
chmod +x bin/Release/net8.0/linux-x64/publish/DungeonSlime
```

> **NOTE**
>
> the `chmod +x` command changes the file permissions to make it executable. Without this step, Linux would not be able to run the application.

2. Package the game using the `tar.gz` archive format to preserve executable permissions by executing the following command:

```
tar -czf DungeonSlime-linux-x64.tar.gz -C bin/Release/net8.0/linux-x64/publish/
```

> **NOTE**
>
> The `tar` command creates an archive file:
>
> - `-c` creates a new archive.
> - `-z` compresses the archive using gzip.
> - `-f` specifies the filename to create
> - `-C` changes to the specified directory before adding files.
>
> Unlike **ZIP** files, the `tar.gz` format preserves Unix file permissions, which is crucial for maintaining the executable permission we set in the previous step.

Important .NET Publishing Parameters

When publishing your game, there are several .NET parameters that can significantly impact the performance of the game. In the above sections, these are all set to the recommended values, however, we will examine them in detail below.

ReadyToRun (R2R)

ReadyToRun is a feature in .NET that pre-compiles code to improve startup time. This sounds like a good thing on paper, however, for games, it can lead to micro-stutters during gameplay.

This happens because ReadyToRun-compiled code is initially of lower quality, and the Just-In-Time (JIT) compiler will trigger periodically to optimize the code further. These optimization passes can cause visible stutters in the game.

For games, it is recommended to disable ReadyToRun by setting `-p:PublishReadyToRun=false` in your publish command, which we have already included in our examples.

For more information on ReadyToRun, refer to the ReadyToRun deployment overview documentation on Microsoft Learn

Tiered Compilation

Tiered compilation is another .NET feature that works similarly to ReadyToRun. It initially compiles code quickly at a lower optimization level, then recompiles frequently-used methods with higher optimization.

While this improves application startup time, it can also cause stutters during gameplay as methods are recompiled. It is recommended to disable tiered compilation by setting `-p:TieredCompilation=false` in your publish command, which we have already included in our examples.

For more information on Tiered Compilation, refer to the Tiered compilation section on Microsoft Learn.

Native AOT (Ahead-of-Time) Compilation

Native AOT compilation (specified with `-p:PublishAot=true`) compiles your entire application to native code at build time, eliminating the need for JIT compilation during runtime. This can provide better performance and a smaller distribution size.

However, AOT has limitations:

1. No support for runtime reflection.
2. No runtime code generation.
3. Some third-party libraries your game uses may not be compatible.

For MonoGame game, AOT can work well if you avoid these limitations.

> **NOTE**
>
> Native AoT is recommended for mobile platforms due to its performance benefits and smaller binary size, which are important for mobile devices with limited resources. Additionally, it is mandatory when targeting console platforms (Xbox, PlayStation, Switch) as these platforms typically do not support JIT compilation for security and performance reasons.

For more information on Native AOT, refer to the Native AOT deployment overview documentation on Microsoft Learn.

Trimming

Trimming (specified with `-p:Trimming:true`) removes unused code from your distribution to reduce size. It is automatically enabled when using AOT.

While trimming can significantly reduce your game's size, it may remove types that appear unused but are accessed indirectly through reflection or generics causing runtime errors.

> **IMPORTANT**
>
> Trimming can cause issues with content pipeline extensions that are used at runtime. When the compiler cannot detect that certain types are used (especially with reflection or generic collections), they might be trimmed away, resulting in "type not found" exceptions when loading content.
>
> If you encounter runtime exceptions about missing types when loading content with trimming enabled, you can resolve this by ensuring the compiler recognizes the types being used at runtime by making the following call:
>
> ```
> ContentTypeReaderManager.AddTypeCreator(typeof(ReflectiveReader<ReaderType>).Ful
> lName, () => new ReflectiveReader<ReaderType>())
> ```
>
> Where `ReaderType` is the `ContentTypeReader` of the content pipeline extension to be preserved. This call should be made somewhere in your code before loading content that uses these types.

For more information on Trimming, refer to the Trim self-contained applications documentation on Microsoft Learn.

Single File Publishing

Single file publishing packages your entire application into a single executable. While this sounds convenient, it is essentially a self-extracting archive that extracts to a temporary directory at runtime.

This can significantly increase startup time for larger games and may fail on system with restricted permissions or limited storage. For this reason, it is not recommended to use this option for games.

For more information on Single File Publishing, refer to the Create a single file for application deployment documentation on Microsoft Learn.

Cross-Platform Considerations

When distributing your games across multiple platforms, be aware of these additional considerations:

File Paths

Different operating systems use different path separators (Windows uses backslashes, macOS and Linux use forward slashes). Always use `Path.Combine` in your code rather than hardcoding path separators.

```
// Incorrect approach - will fail on some platforms
string path = "Content\\images\\atlas-definition.xml";

// Correct approach, works on all platforms
string path = Path.Combine("Content", "images", "atlas-definition.xml");
```

Case Sensitivity

Windows is case-insensitive for filenames, but macOS and Linux are case-sensitive. Ensure your asset references use the exact case that matches your files for maximum compatibility.

```
// If the content path on disk is:
// images/Atlas.xnb

// On Windows, this would work fine since windows is case-insensitive.
// On macOS and Linux, this would fail since they are case-sensitive.
Texture2D text = Content.Load<Texture2D>("images/atlas");
```

External Dependencies

Try to minimize external dependencies. If your game requires additional libraries or runtimes, document these requirements clearly for players.

> **NOTE**
>
> When publishing to distribution platforms and app stores (such as Steam, Epic Game Store, App Store, or Google Play), you are typically required to disclose all external dependencies in your privacy policy or a dedicated dependencies section. This includes third-party libraries, analytics tools, and any software components that your game depends on.
>
> Check specific requirements for each distribution platform you plant to target, as well as requirements by third-party libraries for using them, as disclosure requirements may vary.

Asset Security and Protection

When distributing your game, you may have concerns about protecting your assets and code from unauthorized access or reverse engineering. It is important to understand the practical limitations and trade-offs involved in various security approaches.

XNB Asset Protection

MonoGame's content pipeline compiles assets into compressed XNB format, which provides a basic level of protection for your game content. For the majority of indie game projects, this compression is sufficient protection for several reasons:

- **XNB files are not standard formats**: Unlike raw images or audio files, XNB files require specific knowledge and tools to extract. While the XNB format is documented and part of the MonoGame open source code, this still creates a barrier for the casual user.
- **Practical protection**: While not cryptographically secure, XNB compression deters casual attempts at asset extraction.
- **Performance benefits**: The primary purpose of XNB compilation is optimization, with content protection being a secondary benefit.
- **Cross-platform consistency**: The same XNB format works across all MonoGame platforms without additional configuration.

> **TIP**
>
> Unless you are working with highly valuable or sensitive assets (such as unreleased music from major artists or proprietary artwork), the standard XNB compression provides adequate protection for most games.

Code Obfuscation

For protecting your game's source code logic, obfuscation tools can make reverse engineering more difficult by renaming variables, restructuring code flow, and adding dummy logic paths. However, code obfuscation comes with significant trade-offs:

- **Performance impact**: Obfuscated code often runs slower than clean, optimized code due to additional indirection and complexity.
- **Debugging complexity**: Stack traces become unreadable, making it nearly impossible to diagnose issues reported by players.
- **Build process overhead**: Additional build steps, tools, and integration are required in your development workflow.
- **Platform limitations**: Some obfuscation techniques may not work correctly across all target platforms or may interfere with .NET features that MonoGame uses internally.
- **Compatibility issues**: Obfuscation can break reflection-based code or third-party libraries.

> **IMPORTANT**
>
> Consider whether the performance cost of obfuscation is worth the potential security benefits for your specific project. For most indie game, the impact on player experience may outweigh the security advantages.

The Reality of Modern Society

It is crucial to understand that in the modern digital landscape, no security measure is truly impenetrable:

- **Corporate security investments**: Major corporations invest millions of dollars annually in security research and implementation, yet breaches still occur regularly. This demonstrates the fundamental challenge of client-side protection.
- **Advanced tools**: Sophisticated reverse engineering tools are readily available and constantly improving, making traditional protection less effective.
- **AI-assisted analysis**: Artificial intelligence can now assist in code analysis and pattern recognition, making traditional obfuscation techniques less reliable.
- **Determined attackers**: If someone is sufficiently motived to extract your assets or reverse engineer your code, they will likely succeed regardless of protection measures.
- **Diminished returns**: For indie developers, time spent on extensive security measures often exceeds the value of the content being protected and could be better invested in core development.

Practical Security Recommendations

For most MonoGame projects, consider these practical approaches to content protection:

1. **Accept standard protection**: The built-in XNB compression is sufficient for typical use cases and provides the good balance of protection and performance.
2. **Focus on gameplay**: Invest development time in creating compelling gameplay rather than extensive security measures.
3. **Legal protection**: Consider proper licensing, terms of service, and copyright notices as your primary protection—intellectual property law provides stronger protection than technical measures.
4. **Contractual compliance**: If using licensed assets with specific protection requirements, work with the licensor to understand what constitutes "reasonable protection."
5. **Threat assessment**: Realistically evaluate whether your game is likely to be a target for asset extraction or reverse engineering.

> **NOTE**
>
> Remember that the goal of asset protection should be to deter the casual extraction, not to create an impenetrable fortress. The time and resources spent on extensive security measures are often better invested in improving the game itself.

Mobile Platforms

While this tutorial series has focused on creating a 2D game for desktop platforms, MonoGame also offers support for mobile development on Android and iOS. The game we have built throughout this series could be adapted for touch controls and distributed through mobile app stores with additional work.

Mobile deployment involves several considerations beyond those of desktop platforms:

- App store submission process and platform-specific requirements.
- Platform-specific signing and certification procedures.
- Extensive device compatibility testing across various screen sizes and hardware.
- Optimization of touch input controls (replacing our keyboard and gamepad input).
- Power consumption management and performance optimization for mobile hardware.

For the Dungeon Slime game, adapting to mobile would require:

- Implementing touch controls to replace the keyboard/gamepad movement.
- Potentially rethinking game mechanics to suit mobile play patterns.

> **NOTE**
> Mobile deployment for MonoGame games is significantly more complex than desktop deployment and typically requires platform-specific development environments (Android Studio for Android and Xcode for iOS). A comprehensive guide to mobile deployment will be covered in a future tutorial.

If you are interested in extending the Dungeon Slime game, or future games, to mobile platforms after completing this tutorial series, these resources provide a good starting point:

- Android Deployment Guide
- iOS App Store Distribution

Third-Party Packaging Tools

While the platform-specific packaging steps outlined in this chapter give you complete control over the distribution process, they require multiple commands and potentially access to different operating systems. Fortunately, the MonoGame community has developed several tools that can automate these packaging steps across platforms.

GameBundle

GameBundle is a .NET command-line tool created by Ellpeck that simplifies packaging MonoGame and other .NET applications into several distributable formats. This tool can automatically bundle your game for Windows, Linux, and macOS platforms, create applications bundles for macOS, and handle various packaging configurations with a single command.

For more information about GameBundle, including installation and usage instructions, visit the official repository on GitHub

MonoPack

MonoPack is a .NET command-line tool created by ShyFox Studio designed specifically for MonoGame projects. According to its documentation, "MonoPack is a dotnet tool used for MonoGame projects to package the game for Windows, Linux, and/or macOS".

Key features include:

- Cross-platform packaging capabilities (build for any OS from any OS).
- Automatic creation of macOS application bundles.
- Appropriate compression formats for each target platform for distribution.

For more information about MonoPack, including installation and usage instructions, visit the official repository on GitHub

Conclusion

In this chapter, you learned how to package your MonoGame project for distribution across Windows, macOS, and Linux platforms. You now understand how to create self-contained deployments for each target platform, the impact of various .NET publishing options on game performance, and important cross-platform considerations.

Whether you choose to use the manual platform-specific packaging steps or automate the process with tools like GameBundle or MonoPack, you now have the knowledge to ensure your game runs smoothly for players across different platforms without requiring them to install additional dependencies.

Test Your Knowledge

1. Why is it recommended to use self-contained deployments for distributing MonoGame games?

> Self-contained deployments package your game with all necessary .NET dependencies, ensuring players can run the game without installing the .NET runtime. This simplifies distribution, guarantees your game uses the exact runtime version it was developed with, and provides a better player experience despite the larger package size.

2. Why should ReadyToRun and Tiered Compilation be disabled when publishing games?

> ReadyToRun and Tiered Compilation both initially produce lower-quality code that gets optimized during runtime. This dynamic optimization process causes micro-stutters during gameplay as the Just-In-Time compiler triggers to improve code quality. Disabling these features results in slightly longer startup times but provides smoother gameplay without performance hitches.

3. What is the purpose of the `Info.plist` file when creating a macOS application bundle?

> The `Info.plist` file contains essential metadata about the macOS application, including the bundle identifier, application name, version, copyright information, minimum system requirements, and other configuration details. macOS requires this file to properly recognize and display the application in the Finder and Dock, and to associate the correct icon and file types with the application.

4. What is the advantage of using a tar.gz archive over zip file when distributing for macOS and Linux?

> A tar.gz archive preserves Unix file permissions, which is crucial for maintaining the executable permissions set on game files. Without these permissions, users would need to manually set execute permissions before running the game. ZIP files do not reliably preserve these Unix-specific permissions, which could prevent the game from running directly after extraction on macOS and Linux platforms.

5. What is the purpose of creating a universal binary for macOS distributions?

A universal binary combines executables for multiple CPU architectures (Intel x64 and Apple Silicon arm64) into a single file. This allows the game to run natively on both older Intel-based Macs and newer Apple Silicon Macs without requiring separate distributions.

Chapter 26: Publishing Your Game to itch.io

Learn how to deploy your MonoGame project to itch.io and configure it for players across different platforms.

After packaging your game for various platforms in Chapter 25, the next step is to make it available for other players. While there are many distribution platforms available (Steam, Epic Games store, Good Old Games (GOG)), itch.io has become a popular choice for indie developers due to its simple publishing processes, developer-friendly revenue model, and strong community focus.

In this chapter, you will:

- Learn about itch.io and why it is beneficial for indie game developers.
- Create and configure an itch.io developer account with appropriate security measures.
- Create a project page that effectively showcases your game.

Understanding itch.io as a Distribution Platform

Itch.io is an open marketplace for independent digital creators with a focus on indie games. Unlike larger platforms with strict approval processes, itch.io allows any developer to publish their games instantly. This makes it an ideal starting point for new developers looking to build an audience.

Key benefits of itch.io for developers include:

1. **Simple Publishing Process**: No approval waiting periods or complex requirements.
2. **Pay-What-You-Want Pricing**: Flexibility to set minimum prices, suggested prices, or make your game completely free.
3. **Cross-Platform Support**: Easy distribution for Windows, macOS, and Linux builds.
4. **Community Focused**: Built-in tools for building a community around you and your games, including devlogs and comments.
5. **Developer-Friendly Revenue Split**: The platform lets developers choose their revenue share (the default 90% to developers, 10% to itch.io).
6. **Analytics and Insights**: Basic analytics to track downloads, views, and purchases.
7. **Game Jams**: Integrated tools for hosting and participating in game development competitions.

Setting Up an itch.io Account

Before you can publish your game, you need to create an itch.io account and set up your developer profile.

To do this:

1. Navigate to itch.io and click "Register" in the top-right corner.
2. Fill in the registration form with your desired username, password, and email address.

> **TIP**
>
> Your itch.io username will appear in the URL of your games (e.g. `https://yourusername.itch.io/dungeon-slime`). Choose a username that represents you or your studio professionally and is easy to remember.
>
> The username can be changed later in your account settings if needed.

3. After registering, you will automatically be logged in. However, for some additional configurations that need to be made, you must validate your email address. Check the email address you entered when you registered for a new email from itch.io to validate the email address.
4. After validating your email address, go to your account settings by clicking your username in the top-right corner, and then choosing "Settings".
5. The settings page opens with your developer profile configuration. Complete your developer profile with:
    - A profile picture.
    - Links to your social media platforms or portfolio.
    - A short biography for your profile page.
6. Next, click "Two factor auth" under the "Basics" section on the left to go through the process of configuring two factor authentication for your account.
7. If you plan to sell your game, you will need to configure your publisher information. To do this, click the "Get Started" section under "Publisher" on the left and follow the steps presented to configure your account to accept payments.

Creating a New Project

To put your game on itch.io, you first need to create a new project. When creating a new project, you will be filling out information about the game to generate a project page that contains all the information about the game including the title, description, screenshots, and download files.

To create a new project, click the arrow beside your username in the top-right corner of the page, and choose "Upload new project".

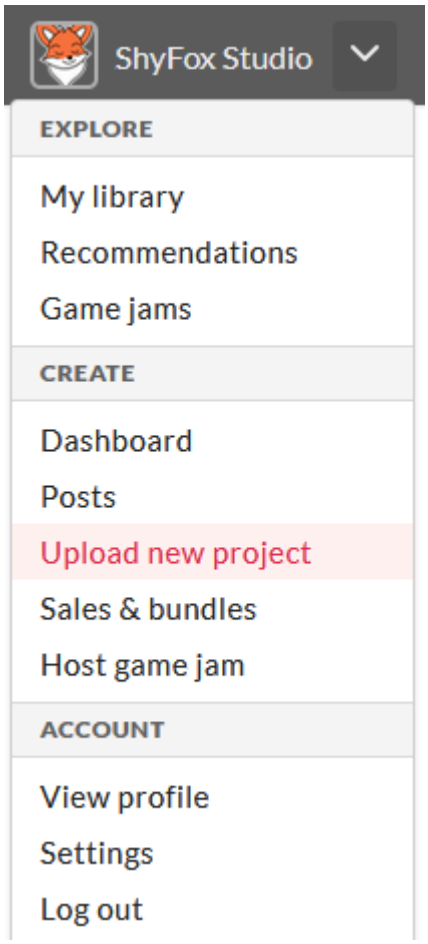


Figure 26-1: Drop-down menu after clicking the arrow beside user name with "Upload new project" highlighted

This will open the "Create a new project page", which presents you with a form to enter information about the game, upload the game files, and add screenshots.

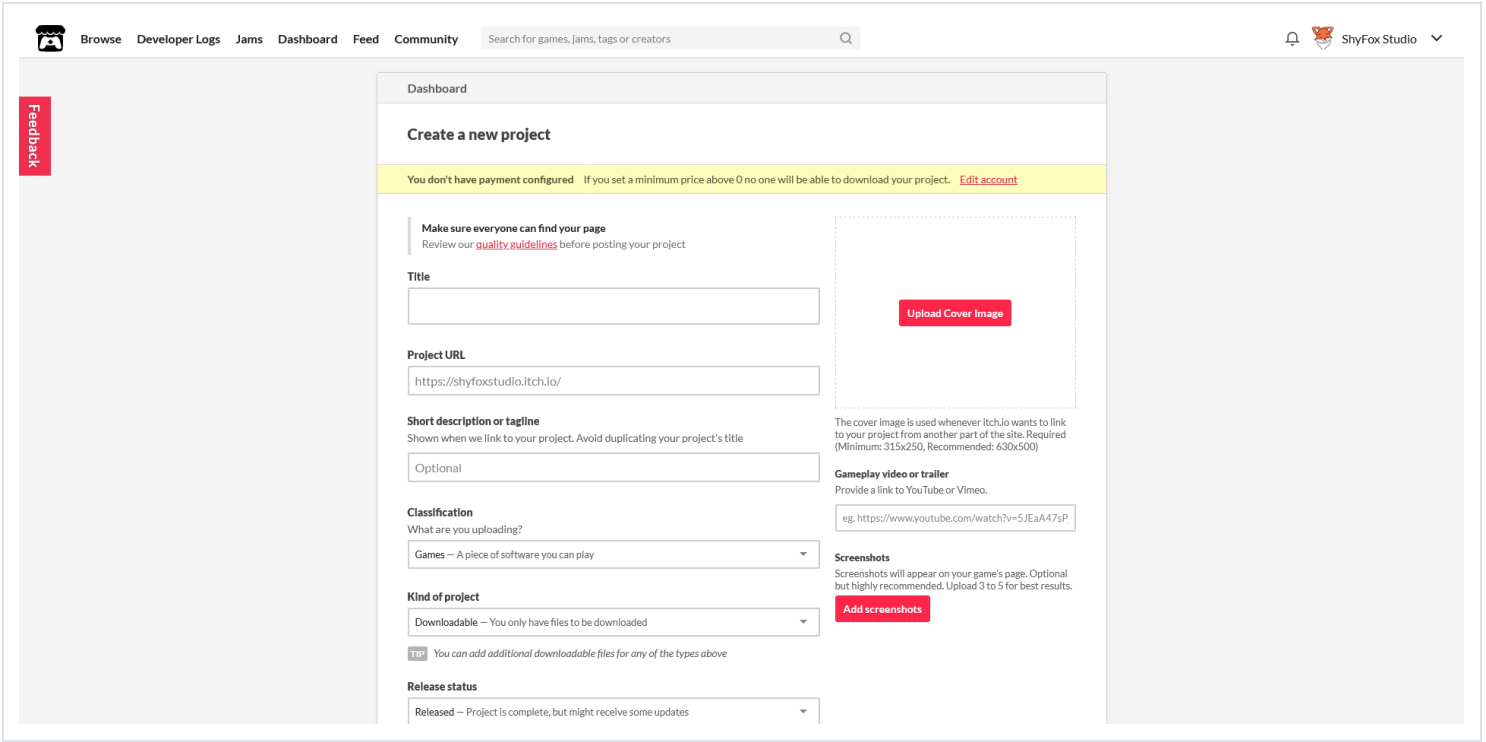


Figure 26-2: The "Create a new project" page on itch.io

> **TIP**
>
> You can also create a new project by going to your dashboard by clicking the "Dashboard" navigation link at the top of the page, then clicking the "Create new project" button on the dashboard page.

Basic Project Information

The first section of the form contains fields for the basic information about the project.

1. **Title**: The title of the game. This is how it will be displayed and shown throughout the itch.io website.
2. **Project URL**: The URL of the project. This is the address you will share on social media platforms and other communities to point others to the project page for your game.

> **NOTE**
>
> When you enter the title of your game, the project URL will automatically generate based on the title entered (e.g. `https://yourusername.itch.io/project-title`). The autogenerated URL is usually fine, but you can customize it if you are not happy with it.

3. **Short description or tagline**: Here you can enter a short description that is shown in embedded link previews for your game when it is shared on different social media platforms.
4. **Kind of project**: The type of project you are uploading. The available options are:
    - **Downloadable**: For projects that require users to download and run files locally. This is the appropriate choice for MonoGame projects since they are distributed as executable packages.
    - **HTML**: For web-based games or applications that run directly in the browser, typically using HTML5, JavaScript, and CSS.
    - **Flash**: For legacy Flash applications that require the Flash Player plugin. This format is largely obsolete since Adobe discontinued Flash support.
    - **Java applet**: For Java-based applications that run in a browser. Like Flash, this technology has been largely phased out of modern web development.
    - **Unity <= 5.3**: For older Unity Web Player games that run in the browser. Newer Unity games typically use WebGL, which would fall under the HTML category.

> **TIP**
>
> Since MonoGame games are distributed by packaging them into an archived bundle, you should choose the "Downloadable" option.

5. **Classification**: What type of project are you uploading. The available options are:

    - **Games**: Interactive software designed primarily for entertainment or educational purposes.
    - **Game assets**: Resources used in game development, such as graphics, audio, fonts, or code modules.
    - **Game mods**: Modifications that alter or enhance an existing game's content or mechanics.
    - **Physical games**: Tabletop, card, or other non-digital games that can be printed or manufactured.
    - **Albums & soundtracks**: Musical compositions, including game soundtracks or independent music projects.
    - **Tools**: Utility software that assists in game development or other creative processes.
    - **Comics**: Sequential art narratives in digital format.
    - **Books**: Literary works including novels, guides, or educational texts.
    - **Other**: Projects that do not fit into the predefined categories.
6. **Release Status**: Set the development status of the project. The available options are:

    - **Released**: The project is complete and ready for public distribution, though it may receive updates.
    - **In development**: The project is actively being worked on and may be available as an early access version.
    - **On hold**: Development has been temporarily paused but is expected to resume in the future.
    - **Canceled**: Development has been permanently discontinued.
    - **Prototype**: An early experimental version meant for concept testing rather than complete gameplay.

> **NOTE**
> You can change the release status of the project at any time after creating the project page.

The following is an example of the basic project information that would be filled in for the Dungeon Slime game we developed throughout this tutorial series:

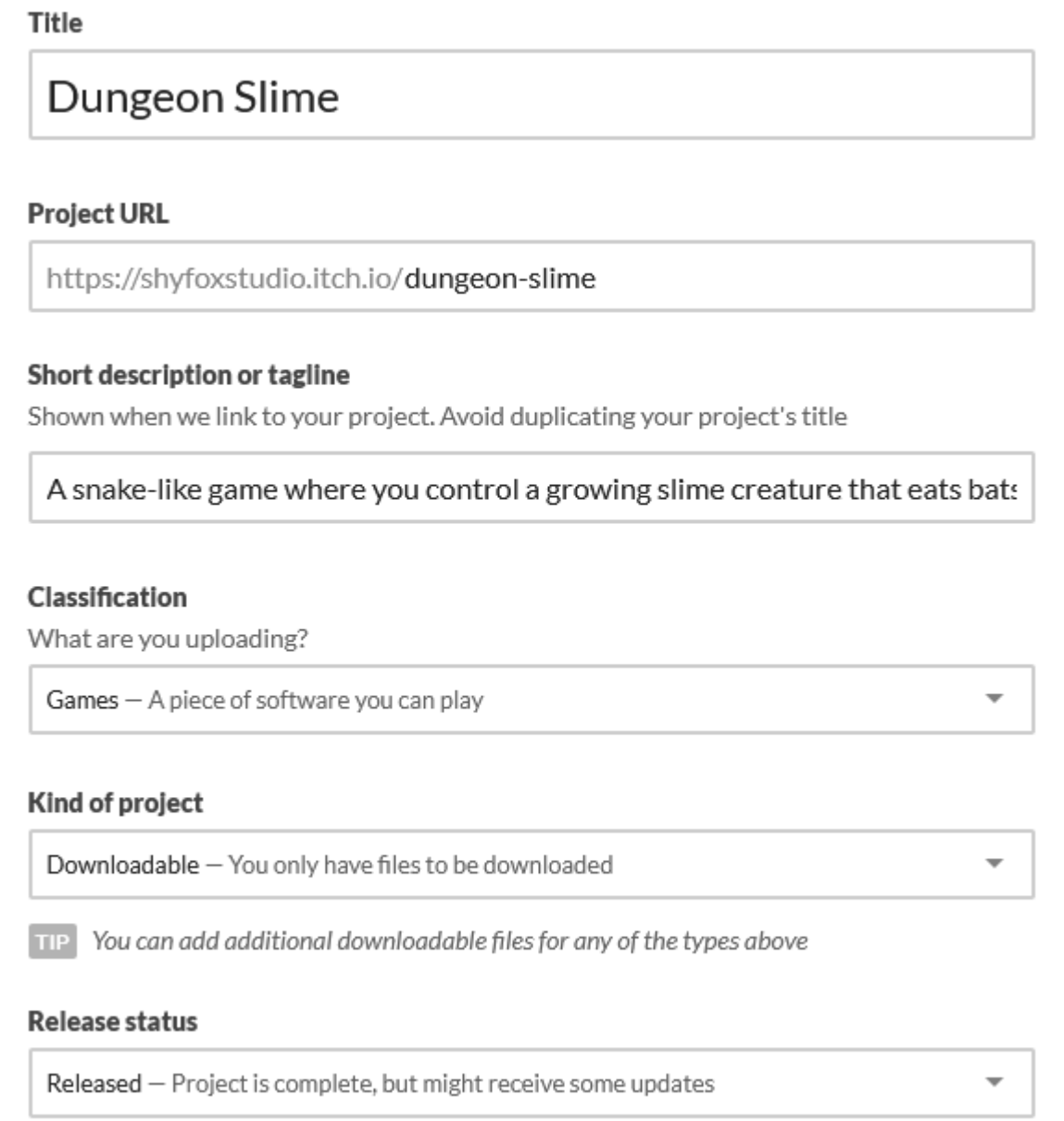


Figure 26-3: The basic project information for the Dungeon Slime game

Pricing

Following the basic information about the project, the next section of the form is for setting the price users would pay to download the game. Itch.io offers three pricing models for a project:

1. **$0 or donate**: This model makes your game freely available while encouraging optional financial support. When users download your game, they are presented with a donation prompt showing your suggested amount. While they can contribute any amount they choose, they also have the option to download without payment by selecting "No thanks, take me to the download.".
2. **Paid**: This model requires users to purchase your game before downloading. You set a minimum price that all users must pay, though they have the option to contribute more if they wish.
3. **No Payment**: This model provides your game completely free with no payment prompts. When users click to download, they are taken directly to the download page without any suggestion to donate.

> **TIP**
>
> Consider your goals when selecting a pricing model. The **"$0 or donate"** option often provides a good balance for indie developers, removing financial barriers to trying your game while still providing a path for supportive players to contribute.

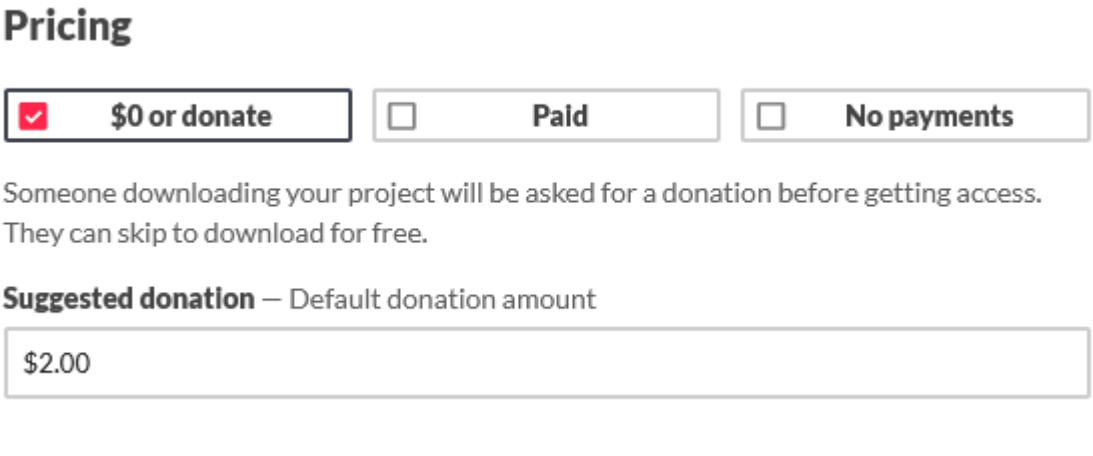


Figure 26-4: The pricing options for a new project on itch.io

Uploads

After choosing the pricing options for the project, the next part of the form is for uploading files that users can download for the project. This is where you will add the platform-specific builds you created using the steps in Chapter 25. Itch.io supports multiple file uploads, allowing you to provide the appropriate version for each platform you support.

> **NOTE**
>
> Before uploading, ensure the builds are properly packaged into either ZIP (Windows) or tar.gz (macOS/Linux) archives as discussed in Chapter 25.

To upload a file for the project:

1. Click the "Upload files" button.
2. In the file select dialog, navigate to and select the file you want to upload.

> **NOTE**
>
> For a MonoGame game project, you would upload the archived builds that you created through the steps in Chapter 25.

Once the upload has completed, you will be presented with options to set the following:

1. **Download Type**: Here you can set the type of file that the user will be downloading. The available options are:

    - **Executable**: The main game application that users will run to play your game. For MonoGame projects, your packaged builds fall into this category.
    - **Source code**: The underlying code of your game, useful for open-source projects or educational examples.
    - **Soundtrack**: Music files from your game, typically provided in MP3 or other audio formats.
    - **Book or Document**: Text content such as manuals, guides, or other written materials.
    - **Video**: Video content such as trailers, gameplay footage, or tutorials.
    - **Mod**: Add-ons or modifications that enhance or alter your base game.
    - **Graphical assets**: Art, textures, models, or other visual elements used in your game.
    - **Audio assets**: Sound effects, voice recordings, or other audio components.
    - **Documentation or Instructions**: Help files, control references, or other guidance for players.
    - **Other**: Any file type that does not fit into the above categories.

    > **TIP**
    > Even if the project you are creating is a game, you can include additional downloads for users such as the soundtrack and an instruction manual. Providing these supplementary materials can enhance the perceived value of your project and demonstrate professionalism and attention to detail.

2. If the download type is "Executable", you will be presented with checkboxes to choose which operating system it is for. The available options are:

    - **Windows**: For Windows-compatible executable files (.exe).
    - **Linux**: For Linux-compatible executable files.
    - **macOS**: For macOS-compatible application bundles (.app).
    - **Android**: For Android-compatible application packages (.apk).

For each file that is uploaded, you are also given the following options:

- **Set a different price for this file**: This allows you to adjust pricing for specific downloads, useful for offering premium content or special editions.
- **Hide this file and prevent it from being downloaded**: This can be used to temporarily remove access to a file without deleting it, which is helpful when updating content or preparing future releases.

NOTE

Itch.io also provides a command-line tool called **butler** that can automate the upload process for your game builds. This is particularly useful for developers who frequently update their games or need to manage multiple platforms. For more information, see the butler GitHub page.

MonoGame community member Jean-David Moisan (Apos) also has a guide written on automating releases using butler with GitHub Actions. For more information, see the Automate Release document.

Uploads

More... Delete file

DungeonSlime-win-x64.zip

31mb · Change display name Move

0 Downloads, Today at 1:22 PM

Executable for

☐ Set a different price for this file

☐ Hide this file and prevent it from being downloaded

More... Delete file

DungeonSlime-linux-x64.tar.gz

40mb · Change display name Move

0 Downloads, Today at 3:55 PM

Executable for

☐ Set a different price for this file

☐ Hide this file and prevent it from being downloaded

More... Delete file

DungeonSlime-osx-universal.tar.gz

60mb · Change display name Move

0 Downloads, Today at 10:59 PM

Executable for

☐ Set a different price for this file

☐ Hide this file and prevent it from being downloaded

Upload files or Choose from Dropbox Add External file ?

File size limit: 1 GB. Contact us if you need more space

TIP *Use **butler** to upload files: it only uploads what's changed, generates patches for the itch.io app, and you can automate it. Get started!*

Figure 26-5: Example of uploaded files for Windows, Linux, and macOS platforms

Description

The description section is where you can showcase your game's unique appeal and provide potential players with compelling reasons to download it. An effective game description serves multiple purposes: it explains what your game is about, highlights its distinctive features, and gives players an idea of what to expect.

When writing a description for your game, consider the following strategies to create a compelling presentation:

- **Start with a hook**: Begin with a concise, attention-grabbing summary that clearly communicates your game's concept and unique selling point.
- **Structure for readability**: Use short paragraphs, bullet points, and clear headings to make your description easy to scan.
- **Highlight key features**: List the most notable gameplay mechanics, visual elements, and player experiences your game offers.
- **Include technical information**: Mention the technologies used (such as MonoGame 😉), supported platforms, and any special hardware requirements or recommendations.
- **Describe the controls**: Briefly explain how players will interact with your game, especially if you support multiple input methods.
- **Add attribution**: Include proper credits for team members, external assets, and tools used in development that require attributions.
- **Consider your audience**: Write in a tone that resonates with your target players.

Here is an example structure for a game description:

```
[Game Name] is a [genre] game where players [main gameplay activity].  [One
additional sentence about a unique hook or setting].

Features:

- [Key Feature 1]
- [Key Feature 2]
- [Key Feature 3]
- [Key Feature 4]

Controls:
- [Control scheme for keyboard/mouse]
- [Control scheme for gamepad, if supported]

Developed with MonoGame
```

> **TIP**
>
> Consider reviewing descriptions of popular games in your genre for inspiration, but ensure your description authentically represents your game. False or exaggerated claims will lead to disappointed players and negative reviews.

Details

Description — This will make up the content of your game page.

Description

Dungeon Slime is a snake-like game where you control a growing slime creature navigating through a dungeon room environment. Eat bats to grow longer, but be careful not to collide with walls or your own body!

Features

- Classic snake-like gameplay with a dungeon theme.
- Animated pixel art visuals.
- Responsive controls with keyboard or gamepad support.
- Progressive difficulty as your slime grows longer.

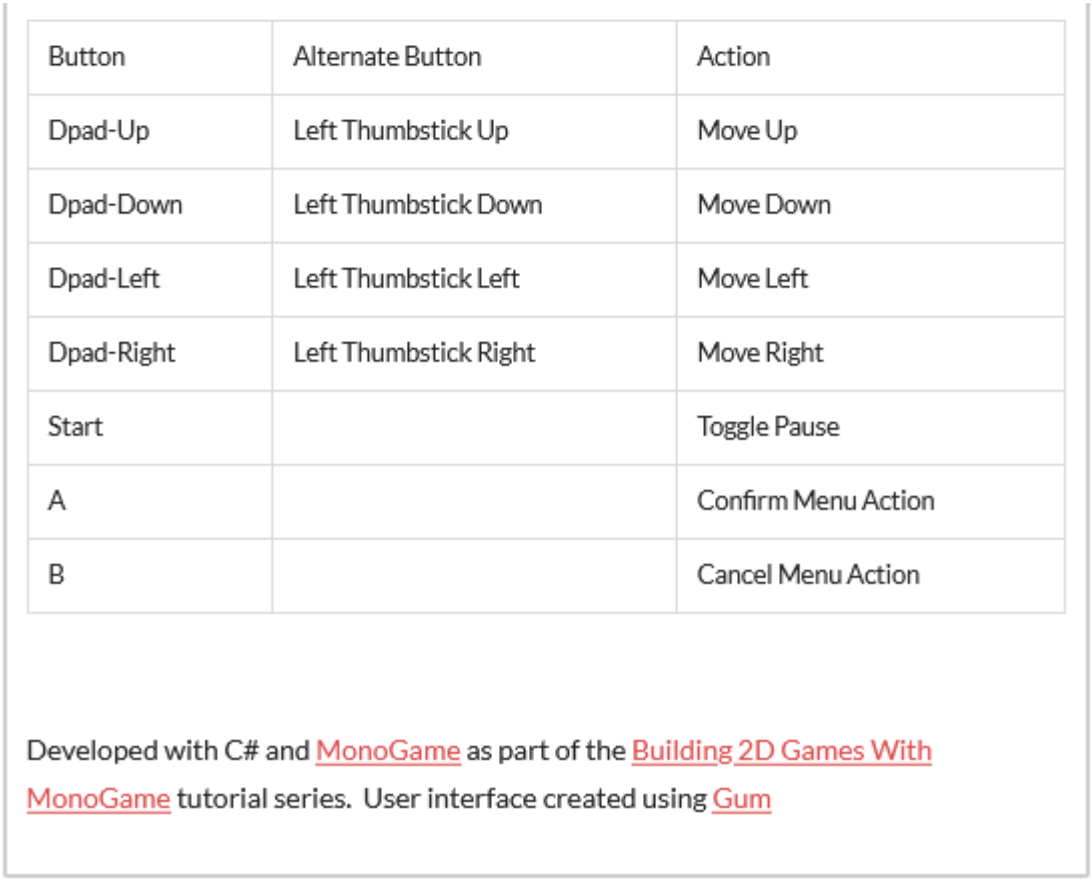
Controls

You can play the game with either Keyboard/Mouse or a Gamepad.

Keyboard Controls

Key	Alternate Key	Action
W	Up Arrow	Move Up
S	Down Arrow	Move Down
A	Left Arrow	Move Left
D	Right Arrow	Move Right
Escape		Toggle pause or cancel menu action
Enter		Confirm menu action

Gamepad Controls



Button	Alternate Button	Action
Dpad-Up	Left Thumbstick Up	Move Up
Dpad-Down	Left Thumbstick Down	Move Down
Dpad-Left	Left Thumbstick Left	Move Left
Dpad-Right	Left Thumbstick Right	Move Right
Start		Toggle Pause
A		Confirm Menu Action
B		Cancel Menu Action

Developed with C# and MonoGame as part of the Building 2D Games With MonoGame tutorial series. User interface created using Gum

Figure 26-6: Example of a game description outline

Metadata

Itch.io provides several metadata fields that help categorize your game and make it more discoverable when users search the platform. Properly configured metadata increases visibility and helps your game reach its intended audience. The following explores each of these fields:

1. **Genre**: Select the genre that most accurately represents your game's core gameplay experience. While many games span multiple genres, choose the one that best captures your game's primary appeal. Common genres include:

    - Action (for games focusing on reflexes and coordination)
    - Adventure (for exploration and narrative-driven experiences)
    - Puzzle (for games centered on problem-solving)
    - Role Playing (for character development and progression systems)
    - Simulation (for realistic system modeling)
    - Strategy (for tactical decision-making games)
2. **Tags**: Tags are keywords that help users find your game when searching or browsing. Effective tagging significantly impacts your game's discoverability. Consider including:

    - Platform identifiers (2d, 3d)
    - Visual style (pixel-art, low-poly, realistic)
    - Technical elements (monogame, procedural)
    - Player configuration (singleplayer, multiplayer)
    - Gameplay mechanics (shooter, platformer, roguelike)

> **TIP**
>
> Itch.io recommends using their suggested tags when possible. Review their tagging guidelines for best practices. Focus on accuracy rather than quantity—irrelevant tags may attract the wrong audience and lead to disappointed players.

3. **AI generation disclosure**: This field requires you to indicate whether your game contains AI-generated assets. This transparency helps users make informed decisions about the content they consume. Options include:

    - Yes: The project contains AI-generated assets.
    - No: The project does not contain any AI-generated assets.

> **NOTE**
>
> Itch.io takes this disclosure seriously. For detailed information about their AI disclosure policy, visit their generative AI guidelines.

4. **App store links**: If your game is available on other platforms like Steam, Apple App Store, or Google Play, you can provide links to these listings. This helps players find your game on their preferred platform and establishes your game's presence across multiple marketplaces.

5. **Custom Noun**: By default, itch.io refers to your project as a "game." If your project fits better with another descriptor (like "tool," "experience," or "interactive story"), you can specify a custom noun here. This affects how your project is described throughout the itch.io interface.

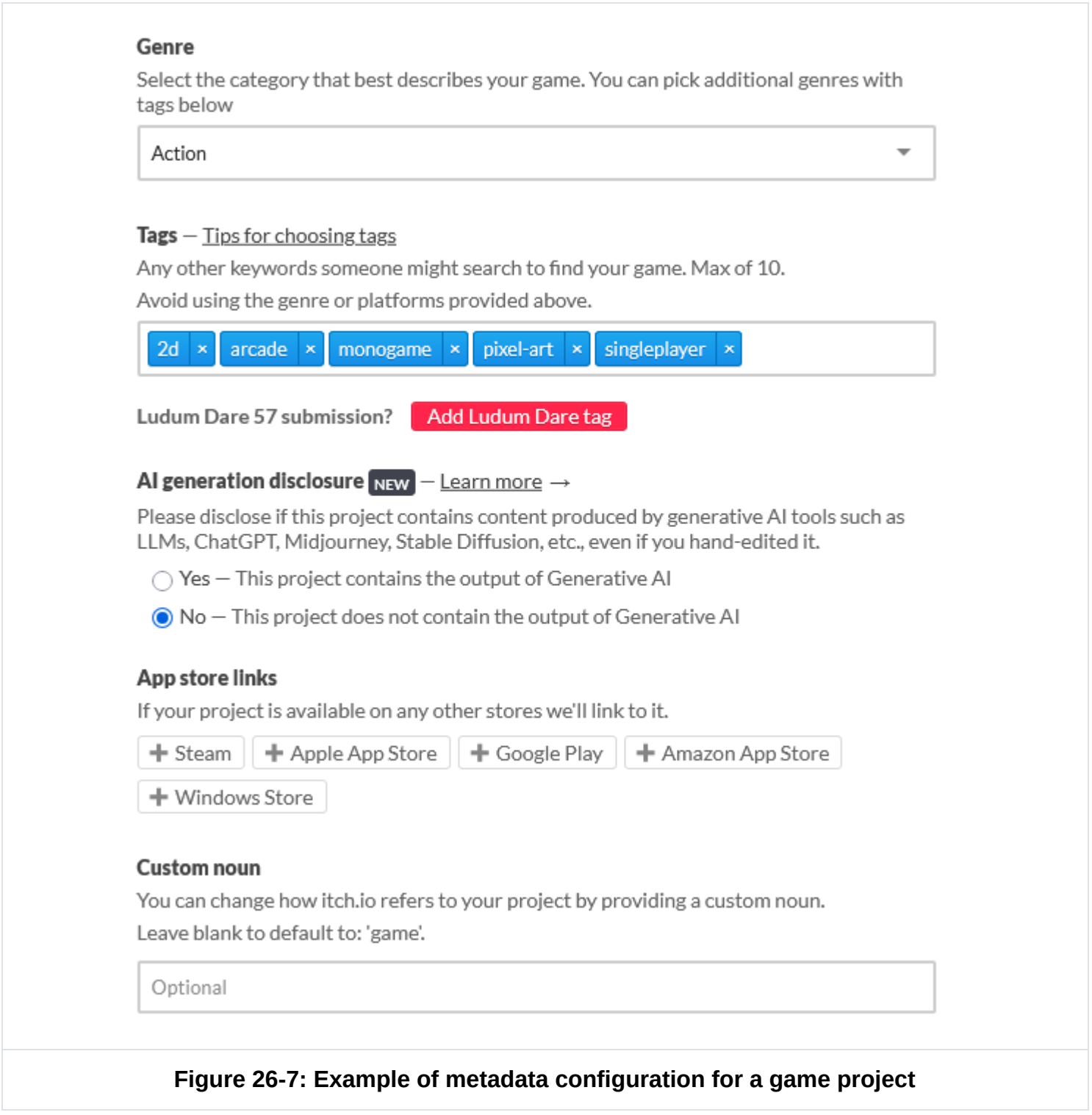


Figure 26-7: Example of metadata configuration for a game project

Download and Install Instructions

The download and installation instructions section is where you provide clear guidance to users on how to get your game up and running after they download it. Well-written instructions reduce confusion and ensure players can quickly start enjoying your game.

Since MonoGame projects are typically distributed as archived files (ZIP for Windows, tar.gz for macOS and Linux), it is essential to provide step-by-step instructions for extracting and launching your game on each supported platform.

- **Be platform-specific**: Separate instructions for each operating system you support.
- **Use numbered steps**: Sequential numbering makes instructions easier to follow.
- **Be explicit**: Specify exact filenames and actions to take.
- **Address common issues**: Mention any special permissions or requirements users might encounter.
- **Use simple language**: Avoid technical jargon when possible, as players may have varying technical expertise.

Here is an example template for installation instructions for a MonoGame project:

```
Once the download has completed:

For Windows:
    1. Extract the contents of the ZIP archive.
    2. Run the "[YourGameName].exe" executable from the extracted folder.

For Linux:
    1. Extract the contents of the tar.gz archive.
    2. Run the "[YourGameName]" executable from the extracted folder.
    3. If the game does not open or states it is not executable, you may need to
change the file permissions.  To do this:
        1. Open a terminal in the extracted directory.
        2. Make the game executable with the command: chmod +x ./[YourGameName]

For macOS:
    1. Extract the contents of the tar.gz archive.
    2. Run the [YourGameName].app executable from the extracted folder.
    3. If the game does not open or states it is not an executable, you may need to
change the file permissions.  To do this:
        1. Open a terminal in the extracted directory.
        2. Make the game executable with the command: chmod +x
./[YourGameName].app/Contents/MacOS/[YourGameName]
```

TIP

Consider testing your installation instructions on a clean system or with someone who has not seen your game before. This can help identify any missing steps or assumptions you might have made.

Download & install instructions

This text will be shown to someone when they are downloading your game through their browser. Include any information about how to install and run what they are downloading.

If your game requires any libraries or programs to be installed first you should also include that information

Once the download has completed:

Windows

1. Extract the contents of the ZIP archive.
2. Run the "DungeonSlime.exe" executable from the extracted folder.

LInux

1. Extract the contents of the tar.gz archive.
2. Run the "DungeonSlime" executable from the extracted folder.
3. If the game does not open
   or states it is not an executable, you may need to change the file permissions. To do this:
   a. Open a terminal in the extracted directory.
   b. Make the game executable with the command: chmod +x ./DungeonSlime

macOS

1. Extract the contents of the tar.gz archive.
2. Double-click "DungeonSlime.app" from the extracted contents. (It may appear as just DungeonSlime without the .app extension).
3. If the game does not
   open or states it is not an executable, you may need to change the file permissions. To do this:
   a. Open a terminal in the extracted directory.
   b. Make the game executable with the command: chmod +x ./DungeonSlime.app/Contents/MacOS/DungeonSlime

Figure 26-8: Example of download and installation instructions for a MonoGame project

Community

Building a community around your game can significantly enhance its lifespan and success. Itch.io offers several options for enabling player interaction directly on your project page. Choosing the right level of community engagement depends on your game's needs and your capacity to moderate interactions.

The platform provides three community feature options:

- **Disabled**: No community interaction features will be available on your project page. This option is appropriate when you prefer to direct community engagement elsewhere (such as Discord or social media), or when you do not have the resources to moderate community content. This is also suitable for projects in very early development stages where you are not yet ready for public feedback.
- **Comments**: This enables a simple comment section at the bottom of your project page where players can leave feedback, ask questions, or share their experiences. As the developer, you receive notifications when new comments are posted, allowing you to respond promptly. This option provides a good balance between community engagement and moderation requirements, making it ideal for most indie games.
- **Discussion board**: This creates a full-featured forum dedicated to your project, with support for multiple topics, threaded replies, and content organization. Discussion boards are beneficial for games with complex mechanics, active development cycles, or strong communities. However, they require more active moderation and engagement from developers to remain valuable.

> **TIP**
>
> Regardless of which option you choose, regularly engaging with your community demonstrates that you value player feedback and can build a loyal following. Consider setting aside specific times each week to respond to comments or discussions.

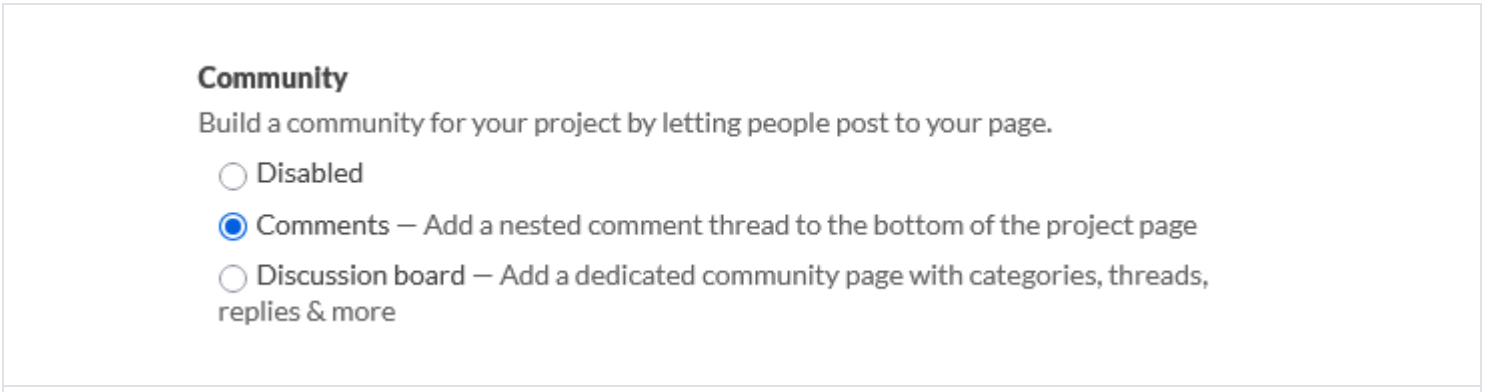


Figure 26-9: Community feature settings available for game projects

Visibility and Access

The visibility settings control who can access your project page. This feature allows you to prepare your page privately before making it public, or to create restricted access for testers or supporters. Understanding these options helps you manage your game's release strategy effectively.

Itch.io offers three visibility levels:

- **Draft**: This setting makes your project page visible only to you and any collaborators you have explicitly given edit access. Draft mode is ideal during the initial setup phase when you are still uploading files, writing descriptions, and configuring your page. It provides a safe environment to experiment with different layouts and content without public exposure.
- **Restricted**: This option limits access to specific individuals you designate. When you select this option, you can generate and share access keys with testers, reviewers, or early supporters. Restricted access is valuable for beta testing, gathering feedback from a controlled group, or offering early access to supporters before a wider release.
- **Public**: This makes your project accessible to everyone visiting itch.io. Once public, your game will appear in search results and category listings, making it discoverable by the platform's user base. This is the appropriate setting when your game is ready for distribution.

> **NOTE**
>
> Itch.io intentionally limits your initial options to either "Draft" or "Restricted" mode. This design encourages you to completely prepare your project page before making it public. Only after saving your initial project setup can you then return to change the visibility to "Public." This workflow helps prevent incomplete or unpolished project pages from being publicly visible.

> **TIP**
>
> Consider using a phased release approach: start in Draft mode while building your page, switch to Restricted for beta testing with a select audience, then finally move to Public when you are confident in your game's presentation and stability.

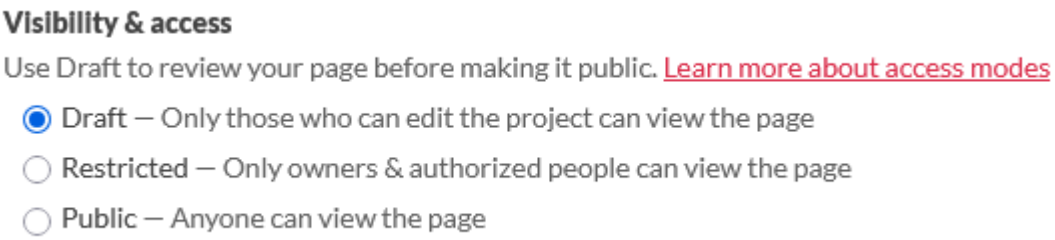


Figure 26-10: Visibility and access settings for publishing your game

Adding Visual Assets

Visual assets are often the first elements potential players notice about your game and play a crucial role in marketing. According to industry research, games with high-quality visual presentations typically receive more downloads and engagement than those with minimal or poor-quality visuals. Taking time to create compelling visual assets can significantly improve your game's appeal on the itch.io marketplace.

At minimum, you should prepare the following visual assets for your project:

1. **Cover Image**: This is your game's primary visual representation, displayed on the game's page, in search results, and in embedded links when shared on social media. Itch.io recommends a size of 630×500 pixels, with a minimum requirement of 315×250 pixels. Your cover image should:
    - Clearly represent your game's visual style
    - Include your game's title
    - Avoid cluttered compositions that become illegible when scaled down
2. **Screenshots**: Include 3-5 high-quality screenshots that showcase different aspects of your game. Effective screenshots should:
    - Demonstrate core gameplay mechanics
    - Highlight unique visual elements or features
    - Show different environments or levels if applicable
    - Include UI elements only when relevant to showcase
    - Maintain consistent aspect ratios across all screenshots
3. **Trailer or Gameplay Video** (optional but recommended): A short video (30-90 seconds) can increase player interest. Videos allow you to demonstrate gameplay flow, audio design, and interactive elements that static images cannot capture.

> **TIP**
>
> When capturing screenshots and video:
>
> - **Windows**: Use the built-in Game Bar (Windows+G), Snipping Tool, or press `Windows+Shift+S` for screenshots. OBS Studio works well for video capture.
> - **macOS**: Press `Command+Shift+4` for area screenshots or `Command+Shift+5` for screen recording. QuickTime Player also offers screen recording functionality.
> - **Linux**: Use tools like GNOME Screenshot, Flameshot, or SimpleScreenRecorder.

To add a cover image to your project:

1. Prepare your image at the recommended 630×500 pixel size.
2. At the top-right of the "Create a new project" page, click the "Upload Cover Image" button.
3. Select your cover image file in the file browser dialog.

The cover image is used whenever itch.io wants to link to your project from another part of the site. Required (Minimum: 315x250, Recommended: 630x500)

Figure 26-11: Example of uploading a cover image for a game project

To add screenshots to your project:

1. Prepare 3-5 screenshots showcasing different aspects of your game.
2. Click the "Add Screenshots" button located below the cover image section.
3. Select your screenshot files in the file browser dialog.
4. Arrange screenshots in order of importance by hovering over them and using the "Move Up" and "Move Down" actions after uploading.

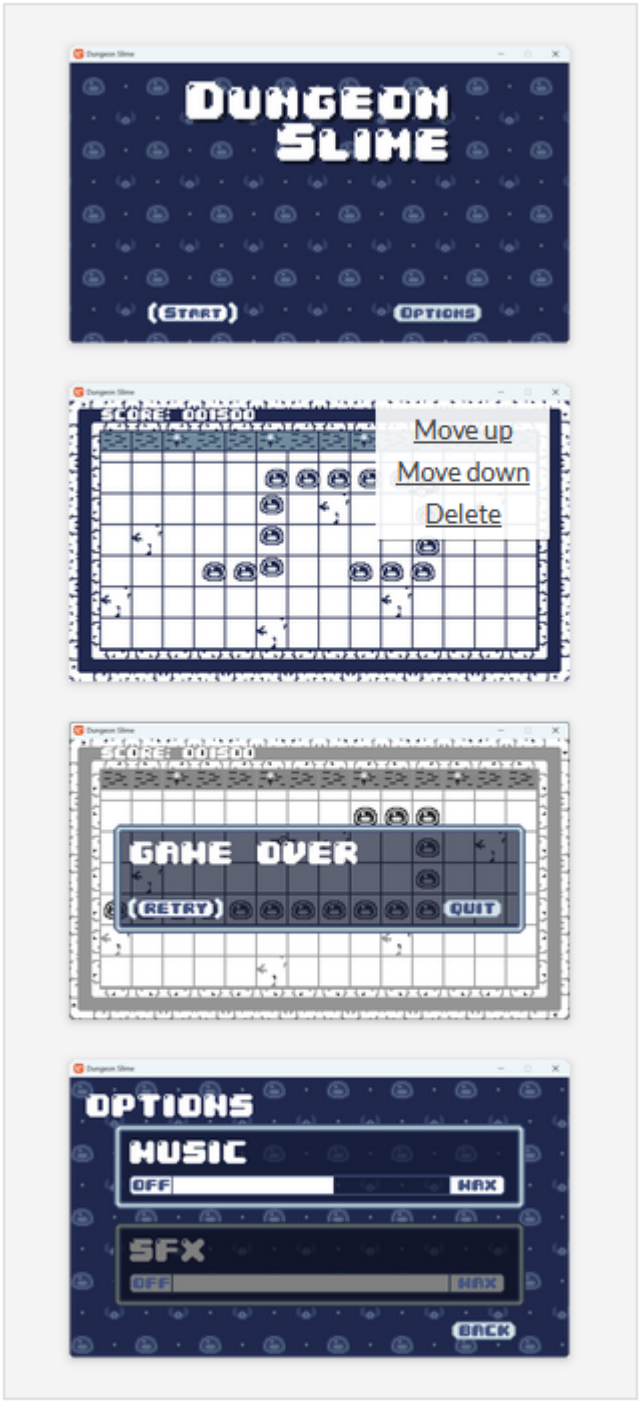


Figure 26-12: Example of multiple screenshots added to a game project

> **NOTE**
>
> While itch.io does not strictly require these visual assets, projects with complete visual presentations receive more attention than those without.

Saving and Publishing Your Project

After filling out all the necessary form fields, uploading your game files, and adding visual assets, it is time to save your project and prepare it for publishing. This process involves several steps designed to ensure your project is ready for your intended audience before publishing.

Initial Save and Preview

Once you have completed the project creation form:

1. Scroll to the bottom of the form and click the "Save & view page" button.
2. This action saves all your current information and generates a preview of your project page.
3. A popup will appear with important information about your project's status and next steps.

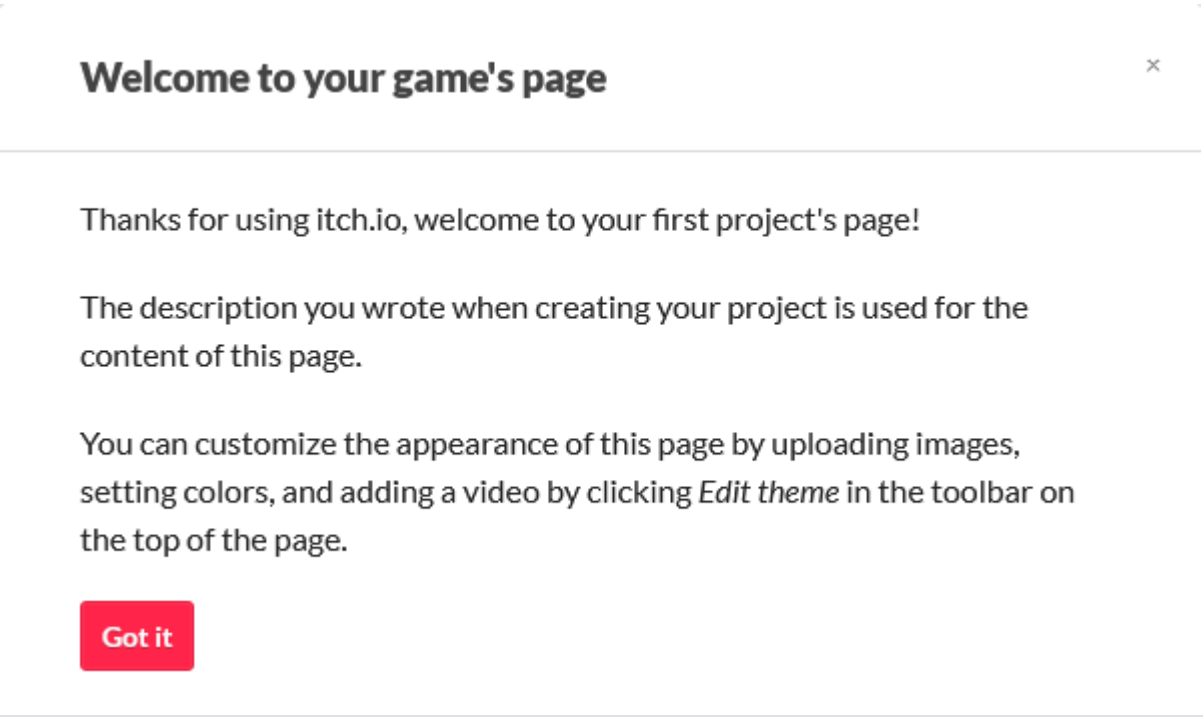


Figure 26-13: Initial information popup after saving a new project

4. Click the "Got it" button after reading to close the popup and view your project page preview.

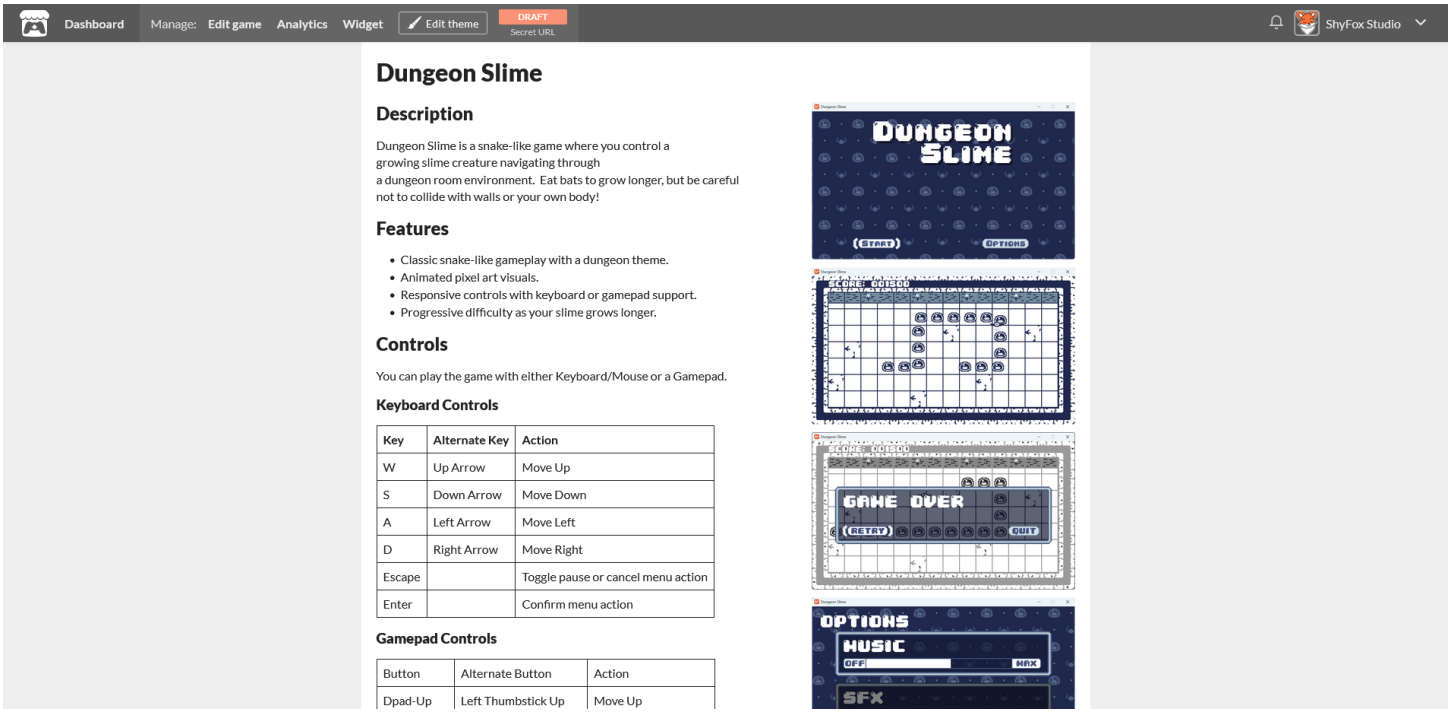


Figure 26-14: Example of a project page preview after initial save

Visual Customization

The preview page provides additional customization options beyond the basic project information you have already entered. As the project owner, you will notice an "Edit theme" button at the top of your preview page. This feature allows you to customize the visual presentation of your project page with options including:

- **Background color**: Change the page's background to match your game's aesthetic.
- **Font selection**: Choose typography that complements your game's style.
- **Layout options**: Modify how elements are arranged on the page.
- **Banner image**: Add a wider image at the top of your page for additional visual impact.
- **Background image**: Set a tiled or full-page background image.

These visual customizations can significantly enhance your project's presentation and help it stand out among other games on the platform. Experiment with different combinations to find a design that complements your game's identity while maintaining readability and usability.

Publishing Your Project

When you are satisfied with your project page and ready to make it publicly available:

1. Click the "Edit game" link in the top navigation bar to return to the project form.
2. Scroll to the bottom of the form to the **Visibility & access** section.
3. Change the setting from "Draft" to "Public."
4. Click the "Save" button to apply the change.

Your game is now publicly accessible to the itch.io community. It will appear in search results and category listings based on the metadata you provided.

> **TIP**
>
> Before widely promoting your game, consider a soft launch approach:
>
> 1. Share the link with a small group of trusted friends or fellow developers.
> 2. Ask them to test the download and installation process on different devices and operating systems.
> 3. Collect feedback on any unclear instructions or technical issues.
> 4. Make necessary adjustments before promoting your game to a wider audience.
>
> This approach helps identify potential issues you may have missed before presenting it to a larger audience.

Dungeon Slime on itch.io

To see a completed version of a project uploaded to itch.io, you can explore the Dungeon Slime example at https://monogame.itch.io/dungeon-slime.

This published version demonstrates the concepts we covered in this chapter, showing how they come together in a finished project.

Conclusion

In this chapter, you learned how to publish your MonoGame project on itch.io, one of the most accessible digital distribution platforms for indie game developers. You have gained a comprehensive understanding of the entire publishing process, from creating an account to making your game available to players worldwide.

> **NOTE**
>
> While you might think "why should I publish my demo or small game", we would recommend you do it anyway. The experience you gain from completing a project from start to finish and then publish will gain you more and more experience over time, so that when you finally come to roll out your "big game", you will already know everything involved and be ready for it.
>
> The old saying that "the last 20% takes 80% of the effort" is VERY true when it comes to game development and it pays to be prepared.

Through this chapter, you have learned how to:

- Create and configure an itch.io developer account with the appropriate security measures.
- Set up a project page with compelling descriptions and relevant metadata.
- Upload platform-specific builds for different operating systems.
- Configure pricing options that align with your goals.
- Write clear download and installation instructions to ensure a smooth player experience.
- Add visual assets that showcase your game effectively.
- Enable appropriate community features to engage with your audience.
- Customize your project page's visual presentation.

Publishing a game represents a significant milestone in your journey as a game developer. While itch.io is just one of many distribution platforms available, it provides an excellent starting point for indie developers due to its accessible nature, developer-friendly policies, and supportive community. The skills you have learned in this chapter extend beyond itch.io and establish a foundation for publishing on other platforms.

Chapter 27: Conclusion and Next Steps

Review key MonoGame concepts learned, discover community resources, and get practical advice for beginning your own game development projects.

As our journey together through this tutorial series comes to a close, it is worth taking a moment to reflect on how far you have come. What began as a simple window displaying the iconic "Cornflower Blue" has evolved into a fully functional game with animated sprites, collision detection, sound effects, and user interfaces. This transformation mirrors your own growth as a game developer.

What You Have Learned

Throughout these chapters you have built a complete 2D game from scratch, learning the fundamental concepts and techniques of game development along the way. This journey has equipped you with the knowledge and skills to start creating your own 2D games using MonoGame. Looking back at our journey, we have covered quite a lot of ground:

Fundamentals

Our journey began with essential building blocks of game development with MonoGame. We explored the framework's architecture and capabilities, comparing it to alternatives to understand its unique position in the game development ecosystem. These fundamental concepts provided the foundation upon which we built increasingly complex systems throughout the tutorial series.

These concepts included:

- Understanding what MonoGame is and how it compares to other game development frameworks.
- Setting up your development environment across different operating systems.
- Learning the basic structure of a MonoGame project and the game loop.
- Creating a reusable class library for game components.

Graphics and Rendering

With the foundations in place, we ventured into the visual aspects of game development that bring our virtual worlds to life. We explored MonoGame's asset management through the content pipeline, rendering capabilities, and learning techniques to efficiently display graphics while optimizing performance.

These included:

- Loading and rendering textures with SpriteBatch.
- Optimizing texture rendering with texture atlases.
- Creating sprites and animated sprites.
- Implementing tilemaps for creating game environments.
- Working with sprite fonts to render text.
- Using shaders to create visual effects.

Game Systems

Beyond graphics, we delved into the interactive systems that transform static visuals into a dynamic gameplay experience. These interconnected components form the backbone of our game, enabling player interaction, providing feedback, and creating a coherent experience across different game states. By implementing these systems, we learned how the various elements of game design work together to create an engaging and responsive player experience.

These systems included:

- Handling input from keyboard, mouse, and gamepad.
- Creating an input management system for improved responsiveness.
- Implementing collision detection between game objects.
- Managing audio with sound effects and background music.
- Developing a scene management system for different game states.
- Building a user interface.

Production and Distribution

After creating a fully functioning game, we turned our attention to sharing our creation with the world. The final stages of game development involve preparing your project for players across different platforms and making it available through distribution channels. These steps take your personal project and puts it into a publicly accessible space where others can download, play, and enjoy.

These stages were:

- Packaging your game for different platforms.
- Publishing your game on distribution platforms like itch.io.

Community and Continued Learning

The MonoGame community represents one of the most valuable resources available to you as you continue your development journey. Unlike closed commercial engines, MonoGame thrives through the collective efforts of developers who share knowledge, tools, and solutions.

The official MonoGame Discord server offers a space where developers of all skill levels exchange ideas and troubleshoot challenges together. Here, a question about optimization techniques might spark a conversation about architecture patterns, leading to insights you might never have discovered on your own.

For structured learning, several community members have created comprehensive educational resources:

- RB Whitaker's provides in-depth coverage of MonoGame fundamentals and advanced topics.
- Learn MonoGame by community member Jean-David Moisan (Apos) provides a collection of focused tutorials on specific MonoGame topics.
- Dark Genesis Blog by foundation member Simon Jackson offers a range of articles on MonoGame development.
- GameDevQuickie offers video tutorials focused on practical game development techniques with MonoGame.
- Darkside of MonoGame video series provides comprehensive tutorials on getting started with MonoGame across different platforms and scenarios.

The MonoGame Samples repository offers practical examples of features implemented in working code. Similarly, the archived XNA Game Studio educational resources maintained by Simon Jackson provide a wealth of examples that remain relevant despite XNA's official discontinuation.

> **TIP**
>
> **REMEMBER**, if you see content written for XNA, then it is more than likely it will still work for MonoGame due to MonoGame's commitment for backwards compatibility. There may be a few bumps, minor changes (especially with earlier versions of XNA, like XNA 2) or differences with shaders (the biggest pain). But remember the community is here to help, so just ASK!.

Your First Original Project

As you contemplate your next project, consider starting with a focused concept that builds on what you have learned while introducing one or two new challenges. This balanced approach allows you to reinforce existing skills while gradually expanding your capabilities.

Perhaps you might create a puzzle game where the core mechanics revolve around tile manipulation, or a side-scrolling adventure that expands on the collision and movement systems you have learned. Whatever you choose, the process will affirm and deepen your understanding in ways that tutorials alone cannot do. In other words, get your hands dirty.

Remember that your first project does not need to be commercially successful or technically groundbreaking. Its primary value lies in the creative freedom it offers and the practical experience it provides. Each bug you fix and feature you implement strengthens your developer intuition; the hard-to-define sense of how games work beneath their surface.

When you inevitably encounter obstacles, approach them with patience and curiosity. The solutions you discover will not just resolve immediate problems; they will become part of your growing toolkit for future projects. Every challenging moment represents an opportunity to become a more resourceful and knowledgeable developer.

The Art of Finishing

Perhaps the most valuable skill you have practiced throughout this tutorial series is the art of finishing. Game development history is littered with ambitious projects that grew beyond their creators' capacity to complete them. By following this series to its conclusion, you have demonstrated the discipline and persistence in bringing games to life.

When working on longer projects, break development into milestones with clear, achievable goals. Celebrate these incremental victories to maintain momentum through the inevitable challenges of extended development. Remember that a simple, finished game will always provide more value, both to players and to your growth as a developer, than an ambitious project that remains perpetually incomplete.

Final Note From Author

The knowledge you have gained throughout this tutorial series represents not an endpoint but a beginning. You now possess the fundamental skills to bring your game ideas to life, the architectural understanding to build systems that grow with your ambitions, and the practical experience of completing a functional game.

The MonoGame framework offers a rare combination of freedom and structure; providing the essential tools without limiting your creative expression. As you continue exploring its capabilities, you join a lineage of independent developers who have used these very same tools to create experiences that resonate with players around the world.

Your journey from here will be uniquely yours, shaped by the games you want to make and by the challenges you choose to overcome. Whatever path you follow, approach it with the same curiosity, persistence, and analytical thinking that brought you through this tutorial series. The road may not always be easy, but it leads to one of the most rewarding creative disciplines available today - the art and science of making games.

Instead of ending this like all other chapters with a section to "test your knowledge", I will just leave you with one simple question:

What will you create next?

- Christopher Whitley (Aristurtle)

Console Access

How do I get access to console versions of MonoGame?

MonoGame is free to use on all platforms from the public repository, but the code for supporting console platforms is only accessible to authorized console developers.

These platforms are provided as private code repositories that add integrations with the console vendor's APIs and platform-specific documentation.

Application process

The process for access to the private console repositories is as follows:

1. Apply to the vendor developer program (required for publishing).
2. Through the program, request access to the MonoGame console repositories.

> **IMPORTANT**
>
> The MonoGame Foundation cannot directly give anyone access to the private console repositories without prior approval from the vendor due to NDA requirements set out by each vendor.

The instructions below will help you gain access to each platform.

Nintendo Switch

Your first step is to register as a Nintendo Developer for Nintendo Switch.

Once you are in the program, you can go to the middleware page and fill out the MonoGame - Nintendo Developer Authorization form.

We will then contact you with further instructions.

PlayStation 4, PlayStation 5

To access PlayStation platforms you must register as a PlayStation Partner.

Once you are registered, you can submit a request in the PlayStation 4 forums and the PlayStation 5 forums.

We will process these requests to give you access and further instructions.

Xbox One, Xbox Series X

To publish on Xbox, you must first be a member of the ID@Xbox Program.

Once you are active in the program you can email `idsetup@xbox.com` and copy the MonoGame `console@monogame.net` account requesting Xbox Developer verification for access to MonoGame for Xbox.

Once Microsoft approves your development status we will respond with further instructions.

Questions

If you have any further questions feel free to contact us at `console@monogame.net`.

Help and Support

Where to get help and support when using MonoGame.

There is a wealth of community-created content, blogs and tutorials available.

You can also chat with other MonoGame users via Discord.

If you want to find an answer to a more specific problem, you can ask it on our GitHub Discussions page.

Bugs and feature requests

If you find a bug or have a feature request, please open a new issue. Before opening an issue, please check for existing ones.

Please note that the issue tracker is not for requesting help.

Namespace Microsoft.Xna.Framework

Classes

BoundingFrustum

Defines a viewing frustum for intersection operations.

Curve

Contains a collection of CurveKey points in 2D space and provides methods for evaluating features of the curve they define.

CurveKey

Key point on the Curve.

CurveKeyCollection

The collection of the CurveKey elements and a part of the Curve class.

DrawableGameComponent

A drawable object that, when added to the Game.Components collection of a Game instance, will have it's Draw(GameTime) method called when Game.Draw(GameTime) is called.

ExitingEventArgs

Event arguments for Exiting.

FrameworkDispatcher

Helper class for processing internal framework events.

Game

This class is the entry point for most games. Handles setting up a window and graphics and runs a game loop that calls Update(GameTime) and Draw(GameTime).

GameComponent

An object that can be attached to a Game and have its Update(GameTime) method called when Update(GameTime) is called.

GameComponentCollection

A collection of IGameComponent instances.

GameComponentCollectionEventArgs

Arguments for the ComponentAdded and ComponentRemoved events.

GameServiceContainer

A container for services for a Game.

GameTime
Holds the time state of a Game.

GameWindow
The system window used by a Game.

GraphicsDeviceInformation
The settings used in creation of the graphics device. See PreparingDeviceSettings.

GraphicsDeviceManager
Used to initialize and control the presentation of the graphics device.

LaunchParameters
The parameters for launching a Game.

MathHelper
Contains commonly used precalculated values and mathematical operations.

MathHelper.Random
Random Number Generator based on Permutation-Congruential-Generator (PCG), which is a fancy wording to describe a family of RNG which are simple, fast, statistically excellent, and hardly predictable.

More interestingly, PCG allows to generate multiple sequences with the same seed, which is very handy in game development to have a unique seed per game session while using different streams for each RNG which requires an isolated context (e.g. generating a procedural level, but we don't want the loot generation to affect subsequent level generations).

https://www.pcg-random.org/

This code is derived from the minimal C implementation. https://github.com/imneme/pcg-c-basic

PreparingDeviceSettingsEventArgs
The arguments to the PreparingDeviceSettings event.

TitleContainer
Provides functionality for opening a stream in the title storage area.

Structs

BoundingBox
Represents an axis-aligned bounding box (AABB) in 3D space.

BoundingSphere

Describes a sphere in 3D-space for bounding operations.

Color

Describes a 32-bit packed color.

FileDropEventArgs

This class is used in the FileDrop event as EventArgs.

InputKeyEventArgs

Provides data for the KeyUp and KeyDown events.

Matrix

Represents the right-handed 4x4 floating point matrix, which can store translation, scale and rotation information.

Plane

A plane in 3d space, represented by its normal away from the origin and its distance from the origin, D.

Point

Describes a 2D-point.

Quaternion

An efficient mathematical representation for three dimensional rotations.

Ray

Represents a ray with an origin and a direction in 3D space.

Rectangle

Describes a 2D-rectangle.

TextInputEventArgs

This class is used in GameWindow's TextInput event as EventArgs.

Vector2

Describes a 2D-vector.

Vector3

Describes a 3D-vector.

Vector4

Describes a 4D-vector.

Interfaces

ICurveEvaluator<T>

Interface for curve evaluation. Implemented by Curve

IDrawable

Interface for drawable entities.

IGameComponent

An interface for GameComponent.

IGraphicsDeviceManager

Used by the platform code to control the graphics device.

IPlatformBackButton

Allows for platform specific handling of the Back button.

IUpdateable

Interface for updateable entities.

Enums

ContainmentType

Defines how the bounding volumes intersects or contain one another.

CurveContinuity

Defines the continuity of keys on a Curve.

CurveLoopType

Defines how the Curve value is determined for position before first point or after the end point on the Curve.

CurveTangent

Defines the different tangent types to be calculated for CurveKey points in a Curve.

DisplayOrientation

Defines the orientation of the display.

GameRunBehavior

Defines how the Game should be run.

PlaneIntersectionType

Defines the intersection between a Plane and a bounding volume.

PlayerIndex

Defines the index of player for various MonoGame components.

Struct BoundingBox

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Represents an axis-aligned bounding box (AABB) in 3D space.

```
[DataContract]
public struct BoundingBox : IEquatable<BoundingBox>
```

Implements

IEquatable<BoundingBox>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

BoundingBox(Vector3, Vector3)

Create a BoundingBox.

```
public BoundingBox(Vector3 min, Vector3 max)
```

Parameters

`min` Vector3

The minimum extent of the BoundingBox.

`max` Vector3

The maximum extent of the BoundingBox.

Fields

CornerCount

The number of corners in a BoundingBox. This is equal to 8.

```
public const int CornerCount = 8
```

Field Value

int

Max

The maximum extent of this BoundingBox.

```
[DataMember]
public Vector3 Max
```

Field Value

Vector3

Min

The minimum extent of this BoundingBox.

```
[DataMember]
public Vector3 Min
```

Field Value

Vector3

Methods

Contains(BoundingBox)

Check if this BoundingBox contains another BoundingBox.

```
public ContainmentType Contains(BoundingBox box)
```

Parameters

`box` BoundingBox

The BoundingBox to test for overlap.

Returns

ContainmentType

A value indicating if this BoundingBox contains, intersects with or is disjoint with `box`.

Contains(ref BoundingBox, out ContainmentType)

Check if this BoundingBox contains another BoundingBox.

```
public void Contains(ref BoundingBox box, out ContainmentType result)
```

Parameters

`box` BoundingBox

The BoundingBox to test for overlap.

`result` ContainmentType

A value indicating if this BoundingBox contains, intersects with or is disjoint with `box`.

Contains(BoundingFrustum)

Determines if this BoundingBox contains or intersects with a specified BoundingFrustum. NOTE: This method may return false positives (indicating an intersection or containment when there is none) to improve performance. Use with caution if precision is critical.

```
public ContainmentType Contains(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The BoundingFrustum to test for overlap.

Returns

ContainmentType

A ContainmentType value indicating whether this BoundingBox contains or intersects the `frustum`.

Contains(BoundingSphere)

Check if this BoundingBox contains a BoundingSphere.

```
public ContainmentType Contains(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to test for overlap.

Returns

ContainmentType

A value indicating if this BoundingBox contains, intersects with or is disjoint with `sphere`.

Contains(ref BoundingSphere, out ContainmentType)

Check if this BoundingBox contains a BoundingSphere.

```
public void Contains(ref BoundingSphere sphere, out ContainmentType result)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to test for overlap.

`result` ContainmentType

A value indicating if this BoundingBox contains, intersects with or is disjoint with `sphere`.

Contains(Vector3)

Check if this BoundingBox contains a point.

```
public ContainmentType Contains(Vector3 point)
```

Parameters

`point` Vector3

The Vector3 to test.

Returns

ContainmentType

Contains if this BoundingBox contains `point` or Disjoint if it does not.

Contains(ref Vector3, out ContainmentType)

Check if this BoundingBox contains a point.

```
public void Contains(ref Vector3 point, out ContainmentType result)
```

Parameters

`point` Vector3

The Vector3 to test.

`result` ContainmentType

Contains if this BoundingBox contains `point` or Disjoint if it does not.

ContainsPrecise(BoundingFrustum)

Determines if this BoundingBox contains or intersects with a specified BoundingFrustum. This method is precise and will significantly affect performance.

```
public ContainmentType ContainsPrecise(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The BoundingFrustum to test for overlap.

Returns

ContainmentType

A ContainmentType value indicating whether this BoundingBox contains or intersects the `frustum`.

CreateFromPoints(Vector3[], int, int)

Create a bounding box from the given list of points.

```
public static BoundingBox CreateFromPoints(Vector3[] points, int index = 0, int
count = -1)
```

Parameters

`points` Vector3[]

The array of Vector3 instances defining the point cloud to bound

`index` int

The base index to start iterating from

`count` int

The number of points to iterate

Returns

BoundingBox

A bounding box that encapsulates the given point cloud.

Exceptions

ArgumentException

Thrown if the given array is null or has no points.

CreateFromPoints(IEnumerable<Vector3>)

Create the enclosing BoundingBox from the given list of points.

```
public static BoundingBox CreateFromPoints(IEnumerable<Vector3> points)
```

Parameters

`points` IEnumerable<Vector3>

The list of Vector3 instances defining the point cloud to bound.

Returns

BoundingBox

A BoundingBox that encloses the given point cloud.

Exceptions

ArgumentException

Thrown if the given list has no points.

CreateFromPoints(List<Vector3>, int, int)

Create a bounding box from the given list of points.

```
public static BoundingBox CreateFromPoints(List<Vector3> points, int index = 0, int
count = -1)
```

Parameters

`points` List<Vector3>

The list of Vector3 instances defining the point cloud to bound

`index` int

The base index to start iterating from

`count` int

The number of points to iterate

Returns

BoundingBox

A bounding box that encapsulates the given point cloud.

Exceptions

ArgumentException

Thrown if the given list is null or has no points.

CreateFromSphere(BoundingSphere)

Create the enclosing BoundingBox of a BoundingSphere.

```
public static BoundingBox CreateFromSphere(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to enclose.

Returns

BoundingBox

A BoundingBox enclosing `sphere`.

CreateFromSphere(ref BoundingSphere, out BoundingBox)

Create the enclosing BoundingBox of a BoundingSphere.

```
public static void CreateFromSphere(ref BoundingSphere sphere, out
BoundingBox result)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to enclose.

`result` BoundingBox

A BoundingBox enclosing `sphere`.

CreateMerged(BoundingBox, BoundingBox)

Create the BoundingBox enclosing two other BoundingBox instances.

```
public static BoundingBox CreateMerged(BoundingBox original, BoundingBox additional)
```

Parameters

`original` BoundingBox

A BoundingBox to enclose.

`additional` BoundingBox

A BoundingBox to enclose.

Returns

BoundingBox

The BoundingBox enclosing `original` and `additional`.

CreateMerged(ref BoundingBox, ref BoundingBox, out BoundingBox)

Create the BoundingBox enclosing two other BoundingBox instances.

```
public static void CreateMerged(ref BoundingBox original, ref BoundingBox
additional, out BoundingBox result)
```

Parameters

`original` BoundingBox

A BoundingBox to enclose.

`additional` BoundingBox

A BoundingBox to enclose.

`result` BoundingBox

The BoundingBox enclosing `original` and `additional`.

Deconstruct(out Vector3, out Vector3)

Deconstruction method for BoundingBox.

```
public void Deconstruct(out Vector3 min, out Vector3 max)
```

Parameters

`min` Vector3

`max` Vector3

Equals(BoundingBox)

Check if two BoundingBox instances are equal.

```
public bool Equals(BoundingBox other)
```

Parameters

`other` BoundingBox

The BoundingBox to compare with this BoundingBox.

Returns

bool

`true`

if `other` is equal to this BoundingBox,

`false`

if it is not.

Equals(object)

Check if two BoundingBox instances are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with this BoundingBox.

Returns

bool

`true`

if `obj` is equal to this BoundingBox,

`false`

if it is not.

GetCorners()

Get an array of Vector3 containing the corners of this BoundingBox.

```
public Vector3[] GetCorners()
```

Returns

Vector3[]

An array of Vector3 containing the corners of this BoundingBox.

GetCorners(Vector3[])

Fill the first 8 places of an array of Vector3 with the corners of this BoundingBox.

```
public void GetCorners(Vector3[] corners)
```

Parameters

`corners` Vector3[]

The array to fill.

Exceptions

ArgumentNullException

If `corners` is

`null`

.

ArgumentOutOfRangeException

If `corners` has a length of less than 8.

GetHashCode()

Get the hash code for this BoundingBox.

```
public override int GetHashCode()
```

Returns

int

A hash code for this BoundingBox.

Intersects(BoundingBox)

Check if this BoundingBox intersects another BoundingBox.

```
public bool Intersects(BoundingBox box)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

Returns

bool

`true`

if this BoundingBox intersects `box`,

`false`

if it does not.

Intersects(ref BoundingBox, out bool)

Check if this BoundingBox intersects another BoundingBox.

```
public void Intersects(ref BoundingBox box, out bool result)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

`result` bool

`true`

if this BoundingBox intersects `box`,

`false`

if it does not.

Intersects(BoundingFrustum)

Check if this BoundingBox intersects a BoundingFrustum.

```
public bool Intersects(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The BoundingFrustum to test for intersection.

Returns

bool

`true`

if this BoundingBox intersects `frustum`,

`false`

if it does not.

Intersects(BoundingSphere)

Check if this BoundingBox intersects a BoundingFrustum.

```
public bool Intersects(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The BoundingFrustum to test for intersection.

Returns

bool

`true`

if this BoundingBox intersects `sphere`,

`false`

if it does not.

Intersects(ref BoundingSphere, out bool)

Check if this BoundingBox intersects a BoundingFrustum.

```
public void Intersects(ref BoundingSphere sphere, out bool result)
```

Parameters

`sphere` BoundingSphere

The BoundingFrustum to test for intersection.

`result` bool

`true`

if this BoundingBox intersects `sphere`,

`false`

if it does not.

Intersects(Plane)

Check if this BoundingBox intersects a Plane.

```
public PlaneIntersectionType Intersects(Plane plane)
```

Parameters

`plane` Plane

The Plane to test for intersection.

Returns

PlaneIntersectionType

`true`

if this BoundingBox intersects `plane`,

`false`

if it does not.

Intersects(ref Plane, out PlaneIntersectionType)

Check if this BoundingBox intersects a Plane.

```
public void Intersects(ref Plane plane, out PlaneIntersectionType result)
```

Parameters

`plane` Plane

The Plane to test for intersection.

`result` PlaneIntersectionType

`true`

if this BoundingBox intersects `plane`,

`false`

if it does not.

Intersects(Ray)

Check if this BoundingBox intersects a Ray.

```
public float? Intersects(Ray ray)
```

Parameters

`ray` Ray

The Ray to test for intersection.

Returns

float?

The distance along the Ray to the intersection point or

`null`

if the Ray does not intesect this BoundingBox.

Intersects(ref Ray, out float?)

Check if this BoundingBox intersects a Ray.

```
public void Intersects(ref Ray ray, out float? result)
```

Parameters

`ray` Ray

The Ray to test for intersection.

`result` float?

The distance along the Ray to the intersection point or

`null`

if the Ray does not intesect this BoundingBox.

ToString()

Get a string representation of this BoundingBox.

```
public override string ToString()
```

Returns

string

A string representation of this BoundingBox.

Operators

operator ==(BoundingBox, BoundingBox)

Check if two BoundingBox instances are equal.

```
public static bool operator ==(BoundingBox a, BoundingBox b)
```

Parameters

a BoundingBox

A BoundingBox to compare the other.

b BoundingBox

A BoundingBox to compare the other.

Returns

bool

`true`

if `a` is equal to this `b`,

`false`

if it is not.

operator !=(BoundingBox, BoundingBox)

Check if two BoundingBox instances are not equal.

```
public static bool operator !=(BoundingBox a, BoundingBox b)
```

Parameters

a BoundingBox

A BoundingBox to compare the other.

b BoundingBox

A BoundingBox to compare the other.

Returns

bool

`true`

if `a` is not equal to this `b`,

`false`

if it is.

Class BoundingFrustum

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines a viewing frustum for intersection operations.

```
public class BoundingFrustum : IEquatable<BoundingFrustum>
```

Inheritance

object ← BoundingFrustum

Implements

IEquatable<BoundingFrustum>

Inherited Members

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

BoundingFrustum(Matrix)

Constructs the frustum by extracting the view planes from a matrix.

```
public BoundingFrustum(Matrix value)
```

Parameters

`value` Matrix

Combined matrix which usually is (View * Projection).

Fields

CornerCount

The number of corner points in the frustum.

```
public const int CornerCount = 8
```

Field Value

int

PlaneCount

The number of planes in the frustum.

```
public const int PlaneCount = 6
```

Field Value

int

Properties

Bottom

Gets the bottom plane of the frustum.

```
public Plane Bottom { get; }
```

Property Value

Plane

Far

Gets the far plane of the frustum.

```
public Plane Far { get; }
```

Property Value

Plane

Left

Gets the left plane of the frustum.

```
public Plane Left { get; }
```

Property Value

Plane

Matrix

Gets or sets the Matrix of the frustum.

```
public Matrix Matrix { get; set; }
```

Property Value

Matrix

Near

Gets the near plane of the frustum.

```
public Plane Near { get; }
```

Property Value

Plane

Right

Gets the right plane of the frustum.

```
public Plane Right { get; }
```

Property Value

Plane

Top

Gets the top plane of the frustum.

```
public Plane Top { get; }
```

Property Value

Plane

Methods

Contains(BoundingBox)

Containment test between this BoundingFrustum and specified BoundingBox.

```
public ContainmentType Contains(BoundingBox box)
```

Parameters

`box` BoundingBox

A BoundingBox for testing.

Returns

ContainmentType

Result of testing for containment between this BoundingFrustum and specified BoundingBox.

Contains(ref BoundingBox, out ContainmentType)

Containment test between this BoundingFrustum and specified BoundingBox.

```
public void Contains(ref BoundingBox box, out ContainmentType result)
```

Parameters

`box` BoundingBox

A BoundingBox for testing.

`result` ContainmentType

Result of testing for containment between this BoundingFrustum and specified BoundingBox as an output parameter.

Contains(BoundingFrustum)

Containment test between this BoundingFrustum and specified BoundingFrustum.

```
public ContainmentType Contains(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

A BoundingFrustum for testing.

Returns

ContainmentType

Result of testing for containment between this BoundingFrustum and specified BoundingFrustum.

Contains(BoundingSphere)

Containment test between this BoundingFrustum and specified BoundingSphere.

```
public ContainmentType Contains(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

A BoundingSphere for testing.

Returns

ContainmentType

Result of testing for containment between this BoundingFrustum and specified BoundingSphere.

Contains(ref BoundingSphere, out ContainmentType)

Containment test between this BoundingFrustum and specified BoundingSphere.

```
public void Contains(ref BoundingSphere sphere, out ContainmentType result)
```

Parameters

`sphere` BoundingSphere

A BoundingSphere for testing.

`result` ContainmentType

Result of testing for containment between this BoundingFrustum and specified BoundingSphere as an output parameter.

Contains(Vector3)

Containment test between this BoundingFrustum and specified Vector3.

```
public ContainmentType Contains(Vector3 point)
```

Parameters

`point` Vector3

A Vector3 for testing.

Returns

ContainmentType

Result of testing for containment between this BoundingFrustum and specified Vector3.

Contains(ref Vector3, out ContainmentType)

Containment test between this BoundingFrustum and specified Vector3.

```
public void Contains(ref Vector3 point, out ContainmentType result)
```

Parameters

`point` Vector3

A Vector3 for testing.

`result` ContainmentType

Result of testing for containment between this BoundingFrustum and specified Vector3 as an output parameter.

Equals(BoundingFrustum)

Compares whether current instance is equal to specified BoundingFrustum.

```
public bool Equals(BoundingFrustum other)
```

Parameters

`other` BoundingFrustum

The BoundingFrustum to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified BoundingFrustum.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetCorners()

Returns a copy of internal corners array.

```
public Vector3[] GetCorners()
```

Returns

Vector3[]

The array of corners.

GetCorners(Vector3[])

Returns a copy of internal corners array.

```
public void GetCorners(Vector3[] corners)
```

Parameters

`corners` Vector3[]

The array which values will be replaced to corner values of this instance. It must have size of Corner Count.

GetHashCode()

Gets the hash code of this BoundingFrustum.

```
public override int GetHashCode()
```

Returns

int

Hash code of this BoundingFrustum.

Intersects(BoundingBox)

Gets whether or not a specified BoundingBox intersects with this BoundingFrustum.

```
public bool Intersects(BoundingBox box)
```

Parameters

`box` BoundingBox

A BoundingBox for intersection test.

Returns

bool

`true` if specified BoundingBox intersects with this BoundingFrustum; `false` otherwise.

Intersects(ref BoundingBox, out bool)

Gets whether or not a specified BoundingBox intersects with this BoundingFrustum.

```
public void Intersects(ref BoundingBox box, out bool result)
```

Parameters

`box` BoundingBox

A BoundingBox for intersection test.

`result` bool

`true` if specified BoundingBox intersects with this BoundingFrustum; `false` otherwise as an output parameter.

Intersects(BoundingFrustum)

Gets whether or not a specified BoundingFrustum intersects with this BoundingFrustum.

```
public bool Intersects(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

An other BoundingFrustum for intersection test.

Returns

bool

`true` if other BoundingFrustum intersects with this BoundingFrustum; `false` otherwise.

Intersects(BoundingSphere)

Gets whether or not a specified BoundingSphere intersects with this BoundingFrustum.

```
public bool Intersects(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

A BoundingSphere for intersection test.

Returns

bool

`true` if specified BoundingSphere intersects with this BoundingFrustum; `false` otherwise.

Intersects(ref BoundingSphere, out bool)

Gets whether or not a specified BoundingSphere intersects with this BoundingFrustum.

```
public void Intersects(ref BoundingSphere sphere, out bool result)
```

Parameters

`sphere` BoundingSphere

A BoundingSphere for intersection test.

`result` bool

`true` if specified BoundingSphere intersects with this BoundingFrustum; `false` otherwise as an output parameter.

Intersects(Plane)

Gets type of intersection between specified Plane and this BoundingFrustum.

```
public PlaneIntersectionType Intersects(Plane plane)
```

Parameters

`plane` Plane

A Plane for intersection test.

Returns

PlaneIntersectionType

A plane intersection type.

Intersects(ref Plane, out PlaneIntersectionType)

Gets type of intersection between specified Plane and this BoundingFrustum.

```
public void Intersects(ref Plane plane, out PlaneIntersectionType result)
```

Parameters

`plane` Plane

A Plane for intersection test.

`result` PlaneIntersectionType

A plane intersection type as an output parameter.

Intersects(Ray)

Gets the distance of intersection of Ray and this BoundingFrustum or null if no intersection happens.

```
public float? Intersects(Ray ray)
```

Parameters

`ray` Ray

A Ray for intersection test.

Returns

float?

Distance at which ray intersects with this BoundingFrustum or null if no intersection happens.

Intersects(ref Ray, out float?)

Gets the distance of intersection of Ray and this BoundingFrustum or null if no intersection happens.

```
public void Intersects(ref Ray ray, out float? result)
```

Parameters

`ray` Ray

A Ray for intersection test.

`result` float?

Distance at which ray intersects with this BoundingFrustum or null if no intersection happens as an output parameter.

ToString()

Returns a string representation of this BoundingFrustum in the format: {Near:[nearPlane] Far:[farPlane] Left:[leftPlane] Right:[rightPlane] Top:[topPlane] Bottom:[bottomPlane]}

```
public override string ToString()
```

Returns

string

string representation of this BoundingFrustum.

Operators

operator ==(BoundingFrustum, BoundingFrustum)

Compares whether two BoundingFrustum instances are equal.

```
public static bool operator ==(BoundingFrustum a, BoundingFrustum b)
```

Parameters

a BoundingFrustum

BoundingFrustum instance on the left of the equal sign.

b BoundingFrustum

BoundingFrustum instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

operator !=(BoundingFrustum, BoundingFrustum)

Compares whether two BoundingFrustum instances are not equal.

```
public static bool operator !=(BoundingFrustum a, BoundingFrustum b)
```

Parameters

a BoundingFrustum

BoundingFrustum instance on the left of the not equal sign.

b BoundingFrustum

BoundingFrustum instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

Struct BoundingSphere

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a sphere in 3D-space for bounding operations.

```
[DataContract]
public struct BoundingSphere : IEquatable<BoundingSphere>
```

Implements

IEquatable<BoundingSphere>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

BoundingSphere(Vector3, float)

Constructs a bounding sphere with the specified center and radius.

```
public BoundingSphere(Vector3 center, float radius)
```

Parameters

`center` Vector3

The sphere center.

`radius` float

The sphere radius.

Fields

Center

The sphere center.

```
[DataMember]
public Vector3 Center
```

Field Value

Vector3

Radius

The sphere radius.

```
[DataMember]
public float Radius
```

Field Value

float

Methods

Contains(BoundingBox)

Test if a bounding box is fully inside, outside, or just intersecting the sphere.

```
public ContainmentType Contains(BoundingBox box)
```

Parameters

`box` BoundingBox

The box for testing.

Returns

ContainmentType

The containment type.

Contains(ref BoundingBox, out ContainmentType)

Test if a bounding box is fully inside, outside, or just intersecting the sphere.

```
public void Contains(ref BoundingBox box, out ContainmentType result)
```

Parameters

`box` BoundingBox

The box for testing.

`result` ContainmentType

The containment type as an output parameter.

Contains(BoundingFrustum)

Test if a frustum is fully inside, outside, or just intersecting the sphere.

```
public ContainmentType Contains(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The frustum for testing.

Returns

ContainmentType

The containment type.

Contains(ref BoundingFrustum, out ContainmentType)

Test if a frustum is fully inside, outside, or just intersecting the sphere.

```
public void Contains(ref BoundingFrustum frustum, out ContainmentType result)
```

Parameters

`frustum` BoundingFrustum

The frustum for testing.

`result` ContainmentType

The containment type as an output parameter.

Contains(BoundingSphere)

Test if a sphere is fully inside, outside, or just intersecting the sphere.

```
public ContainmentType Contains(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The other sphere for testing.

Returns

ContainmentType

The containment type.

Contains(ref BoundingSphere, out ContainmentType)

Test if a sphere is fully inside, outside, or just intersecting the sphere.

```
public void Contains(ref BoundingSphere sphere, out ContainmentType result)
```

Parameters

`sphere` BoundingSphere

The other sphere for testing.

`result` ContainmentType

The containment type as an output parameter.

Contains(Vector3)

Test if a point is fully inside, outside, or just intersecting the sphere.

```
public ContainmentType Contains(Vector3 point)
```

Parameters

`point` Vector3

The vector in 3D-space for testing.

Returns

ContainmentType

The containment type.

Contains(ref Vector3, out ContainmentType)

Test if a point is fully inside, outside, or just intersecting the sphere.

```
public void Contains(ref Vector3 point, out ContainmentType result)
```

Parameters

`point` Vector3

The vector in 3D-space for testing.

`result` ContainmentType

The containment type as an output parameter.

CreateFromBoundingBox(BoundingBox)

Creates the smallest BoundingSphere that can contain a specified BoundingBox.

```
public static BoundingSphere CreateFromBoundingBox(BoundingBox box)
```

Parameters

`box` BoundingBox

The box to create the sphere from.

Returns

BoundingSphere

The new BoundingSphere.

CreateFromBoundingBox(ref BoundingBox, out BoundingSphere)

Creates the smallest BoundingSphere that can contain a specified BoundingBox.

```
public static void CreateFromBoundingBox(ref BoundingBox box, out
BoundingSphere result)
```

Parameters

`box` BoundingBox

The box to create the sphere from.

`result` BoundingSphere

The new BoundingSphere as an output parameter.

CreateFromFrustum(BoundingFrustum)

Creates the smallest BoundingSphere that can contain a specified BoundingFrustum.

```
public static BoundingSphere CreateFromFrustum(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The frustum to create the sphere from.

Returns

BoundingSphere

The new BoundingSphere.

CreateFromPoints(IEnumerable<Vector3>)

Creates the smallest BoundingSphere that can contain a specified list of points in 3D-space.

```
public static BoundingSphere CreateFromPoints(IEnumerable<Vector3> points)
```

Parameters

`points` IEnumerable<Vector3>

List of point to create the sphere from.

Returns

BoundingSphere

The new BoundingSphere.

CreateMerged(BoundingSphere, BoundingSphere)

Creates the smallest BoundingSphere that can contain two spheres.

```
public static BoundingSphere CreateMerged(BoundingSphere original,
BoundingSphere additional)
```

Parameters

`original` BoundingSphere

First sphere.

`additional` BoundingSphere

Second sphere.

Returns

BoundingSphere

The new BoundingSphere.

CreateMerged(ref BoundingSphere, ref BoundingSphere, out BoundingSphere)

Creates the smallest BoundingSphere that can contain two spheres.

```
public static void CreateMerged(ref BoundingSphere original, ref BoundingSphere
additional, out BoundingSphere result)
```

Parameters

`original` BoundingSphere

First sphere.

`additional` BoundingSphere

Second sphere.

`result` BoundingSphere

The new BoundingSphere as an output parameter.

Deconstruct(out Vector3, out float)

Deconstruction method for BoundingSphere.

```
public void Deconstruct(out Vector3 center, out float radius)
```

Parameters

`center` Vector3

`radius` float

Equals(BoundingSphere)

Compares whether current instance is equal to specified BoundingSphere.

```
public bool Equals(BoundingSphere other)
```

Parameters

`other` BoundingSphere

The BoundingSphere to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetHashCode()

Gets the hash code of this BoundingSphere.

```
public override int GetHashCode()
```

Returns

int

Hash code of this BoundingSphere.

Intersects(BoundingBox)

Gets whether or not a specified BoundingBox intersects with this sphere.

```
public bool Intersects(BoundingBox box)
```

Parameters

`box` BoundingBox

The box for testing.

Returns

bool

`true` if BoundingBox intersects with this sphere; `false` otherwise.

Intersects(ref BoundingBox, out bool)

Gets whether or not a specified BoundingBox intersects with this sphere.

```
public void Intersects(ref BoundingBox box, out bool result)
```

Parameters

`box` BoundingBox

The box for testing.

`result` bool

`true` if BoundingBox intersects with this sphere; `false` otherwise. As an output parameter.

Intersects(BoundingSphere)

Gets whether or not the other BoundingSphere intersects with this sphere.

```
public bool Intersects(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The other sphere for testing.

Returns

bool

`true` if other BoundingSphere intersects with this sphere; `false` otherwise.

Intersects(ref BoundingSphere, out bool)

Gets whether or not the other BoundingSphere intersects with this sphere.

```
public void Intersects(ref BoundingSphere sphere, out bool result)
```

Parameters

`sphere` BoundingSphere

The other sphere for testing.

`result` bool

`true` if other BoundingSphere intersects with this sphere; `false` otherwise. As an output parameter.

Intersects(Plane)

Gets whether or not a specified Plane intersects with this sphere.

```
public PlaneIntersectionType Intersects(Plane plane)
```

Parameters

`plane` Plane

The plane for testing.

Returns

PlaneIntersectionType

Type of intersection.

Intersects(ref Plane, out PlaneIntersectionType)

Gets whether or not a specified Plane intersects with this sphere.

```
public void Intersects(ref Plane plane, out PlaneIntersectionType result)
```

Parameters

`plane` Plane

The plane for testing.

`result` PlaneIntersectionType

Type of intersection as an output parameter.

Intersects(Ray)

Gets whether or not a specified Ray intersects with this sphere.

```
public float? Intersects(Ray ray)
```

Parameters

`ray` Ray

The ray for testing.

Returns

float?

Distance of ray intersection or `null` if there is no intersection.

Intersects(ref Ray, out float?)

Gets whether or not a specified Ray intersects with this sphere.

```
public void Intersects(ref Ray ray, out float? result)
```

Parameters

`ray` Ray

The ray for testing.

`result` float?

Distance of ray intersection or `null` if there is no intersection as an output parameter.

ToString()

Returns a string representation of this BoundingSphere in the format: {Center:[Center] Radius:[Radius]}

```
public override string ToString()
```

Returns

string

A string representation of this BoundingSphere.

Transform(Matrix)

Creates a new BoundingSphere that contains a transformation of translation and scale from this sphere by the specified Matrix.

```
public BoundingSphere Transform(Matrix matrix)
```

Parameters

`matrix` Matrix

The transformation Matrix.

Returns

BoundingSphere

Transformed BoundingSphere.

Transform(ref Matrix, out BoundingSphere)

Creates a new BoundingSphere that contains a transformation of translation and scale from this sphere by the specified Matrix.

```
public void Transform(ref Matrix matrix, out BoundingSphere result)
```

Parameters

`matrix` Matrix

The transformation Matrix.

`result` BoundingSphere

Transformed BoundingSphere as an output parameter.

Operators

operator ==(BoundingSphere, BoundingSphere)

Compares whether two BoundingSphere instances are equal.

```
public static bool operator ==(BoundingSphere a, BoundingSphere b)
```

Parameters

a BoundingSphere

BoundingSphere instance on the left of the equal sign.

b BoundingSphere

BoundingSphere instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

operator !=(BoundingSphere, BoundingSphere)

Compares whether two BoundingSphere instances are not equal.

```
public static bool operator !=(BoundingSphere a, BoundingSphere b)
```

Parameters

a BoundingSphere

BoundingSphere instance on the left of the not equal sign.

b BoundingSphere

BoundingSphere instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

Struct Color

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 32-bit packed color.

```
[DataContract]
public struct Color : IEquatable<Color>
```

Implements

IEquatable<Color>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Color(Color, int)

Constructs an RGBA color from a Color and an alpha value.

```
public Color(Color color, int alpha)
```

Parameters

`color` Color

A Color for RGB values of new Color instance.

`alpha` int

The alpha component value from 0 to 255.

Color(Color, float)

Constructs an RGBA color from color and alpha value.

```
public Color(Color color, float alpha)
```

Parameters

`color` Color

A Color for RGB values of new Color instance.

`alpha` float

Alpha component value from 0.0f to 1.0f.

Color(Vector3)

Constructs an RGBA color from the XYZ unit length components of a vector. Alpha value will be opaque.

```
public Color(Vector3 color)
```

Parameters

`color` Vector3

A Vector3 representing color.

Color(Vector4)

Constructs an RGBA color from the XYZW unit length components of a vector.

```
public Color(Vector4 color)
```

Parameters

`color` Vector4

A Vector4 representing color.

Color(byte, byte, byte, byte)

Constructs an RGBA color from scalars representing red, green, blue and alpha values.

```
public Color(byte r, byte g, byte b, byte alpha)
```

Parameters

`r` byte

`g` byte

`b` byte

`alpha` byte

Remarks

This overload sets the values directly without clamping, and may therefore be faster than the other overloads.

Color(int, int, int)

Constructs an RGBA color from scalars representing red, green and blue values. Alpha value will be opaque.

```
public Color(int r, int g, int b)
```

Parameters

`r` int

Red component value from 0 to 255.

`g` int

Green component value from 0 to 255.

`b` int

Blue component value from 0 to 255.

Color(int, int, int, int)

Constructs an RGBA color from scalars representing red, green, blue and alpha values.

```
public Color(int r, int g, int b, int alpha)
```

Parameters

`r` int

Red component value from 0 to 255.

`g` int

Green component value from 0 to 255.

`b` int

Blue component value from 0 to 255.

`alpha` int

Alpha component value from 0 to 255.

Color(float, float, float)

Constructs an RGBA color from scalars representing red, green and blue values. Alpha value will be opaque.

```
public Color(float r, float g, float b)
```

Parameters

`r` float

Red component value from 0.0f to 1.0f.

`g` float

Green component value from 0.0f to 1.0f.

`b` float

Blue component value from 0.0f to 1.0f.

Color(float, float, float, float)

Constructs an RGBA color from scalars representing red, green, blue and alpha values.

```
public Color(float r, float g, float b, float alpha)
```

Parameters

`r` float

Red component value from 0.0f to 1.0f.

`g` float

Green component value from 0.0f to 1.0f.

`b` float

Blue component value from 0.0f to 1.0f.

`alpha` float

Alpha component value from 0.0f to 1.0f.

Color(uint)

Constructs an RGBA color from a packed value. The value is a 32-bit unsigned integer, with R in the least significant octet.

```
public Color(uint packedValue)
```

Parameters

`packedValue` uint

The packed value.

Properties

A

Gets or sets the alpha component.

```
[DataMember]
public byte A { get; set; }
```

Property Value

byte

AliceBlue

AliceBlue color (R:240,G:248,B:255,A:255).

```
public static Color AliceBlue { get; }
```

Property Value

Color

AntiqueWhite

AntiqueWhite color (R:250,G:235,B:215,A:255).

```
public static Color AntiqueWhite { get; }
```

Property Value

Color

Aqua

Aqua color (R:0,G:255,B:255,A:255).

```
public static Color Aqua { get; }
```

Property Value

Color

Aquamarine

Aquamarine color (R:127,G:255,B:212,A:255).

```
public static Color Aquamarine { get; }
```

Property Value

Color

Azure

Azure color (R:240,G:255,B:255,A:255).

```
public static Color Azure { get; }
```

Property Value

Color

B

Gets or sets the blue component.

```
[DataMember]
public byte B { get; set; }
```

Property Value

byte

Beige

Beige color (R:245,G:245,B:220,A:255).

```
public static Color Beige { get; }
```

Property Value

Color

Bisque

Bisque color (R:255,G:228,B:196,A:255).

```
public static Color Bisque { get; }
```

Property Value

Color

Black

Black color (R:0,G:0,B:0,A:255).

```
public static Color Black { get; }
```

Property Value

Color

BlanchedAlmond

BlanchedAlmond color (R:255,G:235,B:205,A:255).

```
public static Color BlanchedAlmond { get; }
```

Property Value

Color

Blue

Blue color (R:0,G:0,B:255,A:255).

```
public static Color Blue { get; }
```

Property Value

Color

BlueViolet

BlueViolet color (R:138,G:43,B:226,A:255).

```
public static Color BlueViolet { get; }
```

Property Value

Color

Brown

Brown color (R:165,G:42,B:42,A:255).

```
public static Color Brown { get; }
```

Property Value

Color

BurlyWood

BurlyWood color (R:222,G:184,B:135,A:255).

```
public static Color BurlyWood { get; }
```

Property Value

Color

CadetBlue

CadetBlue color (R:95,G:158,B:160,A:255).

```
public static Color CadetBlue { get; }
```

Property Value

Color

Chartreuse

Chartreuse color (R:127,G:255,B:0,A:255).

```
public static Color Chartreuse { get; }
```

Property Value

Color

Chocolate

Chocolate color (R:210,G:105,B:30,A:255).

```
public static Color Chocolate { get; }
```

Property Value

Color

Coral

Coral color (R:255,G:127,B:80,A:255).

```
public static Color Coral { get; }
```

Property Value

Color

CornflowerBlue

CornflowerBlue color (R:100,G:149,B:237,A:255).

```
public static Color CornflowerBlue { get; }
```

Property Value

Color

Cornsilk

Cornsilk color (R:255,G:248,B:220,A:255).

```
public static Color Cornsilk { get; }
```

Property Value

Color

Crimson

Crimson color (R:220,G:20,B:60,A:255).

```
public static Color Crimson { get; }
```

Property Value

Color

Cyan

Cyan color (R:0,G:255,B:255,A:255).

```
public static Color Cyan { get; }
```

Property Value

Color

DarkBlue

DarkBlue color (R:0,G:0,B:139,A:255).

```
public static Color DarkBlue { get; }
```

Property Value

Color

DarkCyan

DarkCyan color (R:0,G:139,B:139,A:255).

```
public static Color DarkCyan { get; }
```

Property Value

Color

DarkGoldenrod

DarkGoldenrod color (R:184,G:134,B:11,A:255).

```
public static Color DarkGoldenrod { get; }
```

Property Value

Color

DarkGray

DarkGray color (R:169,G:169,B:169,A:255).

```
public static Color DarkGray { get; }
```

Property Value

Color

DarkGreen

DarkGreen color (R:0,G:100,B:0,A:255).

```
public static Color DarkGreen { get; }
```

Property Value

Color

DarkKhaki

DarkKhaki color (R:189,G:183,B:107,A:255).

```
public static Color DarkKhaki { get; }
```

Property Value

Color

DarkMagenta

DarkMagenta color (R:139,G:0,B:139,A:255).

```
public static Color DarkMagenta { get; }
```

Property Value

Color

DarkOliveGreen

DarkOliveGreen color (R:85,G:107,B:47,A:255).

```
public static Color DarkOliveGreen { get; }
```

Property Value

Color

DarkOrange

DarkOrange color (R:255,G:140,B:0,A:255).

```
public static Color DarkOrange { get; }
```

Property Value

Color

DarkOrchid

DarkOrchid color (R:153,G:50,B:204,A:255).

```
public static Color DarkOrchid { get; }
```

Property Value

Color

DarkRed

DarkRed color (R:139,G:0,B:0,A:255).

```
public static Color DarkRed { get; }
```

Property Value

Color

DarkSalmon

DarkSalmon color (R:233,G:150,B:122,A:255).

```
public static Color DarkSalmon { get; }
```

Property Value

Color

DarkSeaGreen

DarkSeaGreen color (R:143,G:188,B:139,A:255).

```
public static Color DarkSeaGreen { get; }
```

Property Value

Color

DarkSlateBlue

DarkSlateBlue color (R:72,G:61,B:139,A:255).

```
public static Color DarkSlateBlue { get; }
```

Property Value

Color

DarkSlateGray

DarkSlateGray color (R:47,G:79,B:79,A:255).

```
public static Color DarkSlateGray { get; }
```

Property Value

Color

DarkTurquoise

DarkTurquoise color (R:0,G:206,B:209,A:255).

```
public static Color DarkTurquoise { get; }
```

Property Value

Color

DarkViolet

DarkViolet color (R:148,G:0,B:211,A:255).

```
public static Color DarkViolet { get; }
```

Property Value

Color

DeepPink

DeepPink color (R:255,G:20,B:147,A:255).

```
public static Color DeepPink { get; }
```

Property Value

Color

DeepSkyBlue

DeepSkyBlue color (R:0,G:191,B:255,A:255).

```
public static Color DeepSkyBlue { get; }
```

Property Value

Color

DimGray

DimGray color (R:105,G:105,B:105,A:255).

```
public static Color DimGray { get; }
```

Property Value

Color

DodgerBlue

DodgerBlue color (R:30,G:144,B:255,A:255).

```
public static Color DodgerBlue { get; }
```

Property Value

Color

Firebrick

Firebrick color (R:178,G:34,B:34,A:255).

```
public static Color Firebrick { get; }
```

Property Value

Color

FloralWhite

FloralWhite color (R:255,G:250,B:240,A:255).

```
public static Color FloralWhite { get; }
```

Property Value

Color

ForestGreen

ForestGreen color (R:34,G:139,B:34,A:255).

```
public static Color ForestGreen { get; }
```

Property Value

Color

Fuchsia

Fuchsia color (R:255,G:0,B:255,A:255).

```
public static Color Fuchsia { get; }
```

Property Value

Color

G

Gets or sets the green component.

```
[DataMember]
public byte G { get; set; }
```

Property Value

byte

Gainsboro

Gainsboro color (R:220,G:220,B:220,A:255).

```
public static Color Gainsboro { get; }
```

Property Value

Color

GhostWhite

GhostWhite color (R:248,G:248,B:255,A:255).

```
public static Color GhostWhite { get; }
```

Property Value

Color

Gold

Gold color (R:255,G:215,B:0,A:255).

```
public static Color Gold { get; }
```

Property Value

Color

Goldenrod

Goldenrod color (R:218,G:165,B:32,A:255).

```
public static Color Goldenrod { get; }
```

Property Value

Color

Gray

Gray color (R:128,G:128,B:128,A:255).

```
public static Color Gray { get; }
```

Property Value

Color

Green

Green color (R:0,G:128,B:0,A:255).

```
public static Color Green { get; }
```

Property Value

Color

GreenYellow

GreenYellow color (R:173,G:255,B:47,A:255).

```
public static Color GreenYellow { get; }
```

Property Value

Color

Honeydew

Honeydew color (R:240,G:255,B:240,A:255).

```
public static Color Honeydew { get; }
```

Property Value

Color

HotPink

HotPink color (R:255,G:105,B:180,A:255).

```
public static Color HotPink { get; }
```

Property Value

Color

IndianRed

IndianRed color (R:205,G:92,B:92,A:255).

```
public static Color IndianRed { get; }
```

Property Value

Color

Indigo

Indigo color (R:75,G:0,B:130,A:255).

```
public static Color Indigo { get; }
```

Property Value

Color

Ivory

Ivory color (R:255,G:255,B:240,A:255).

```
public static Color Ivory { get; }
```

Property Value

Color

Khaki

Khaki color (R:240,G:230,B:140,A:255).

```
public static Color Khaki { get; }
```

Property Value

Color

Lavender

Lavender color (R:230,G:230,B:250,A:255).

```
public static Color Lavender { get; }
```

Property Value

Color

LavenderBlush

LavenderBlush color (R:255,G:240,B:245,A:255).

```
public static Color LavenderBlush { get; }
```

Property Value

Color

LawnGreen

LawnGreen color (R:124,G:252,B:0,A:255).

```
public static Color LawnGreen { get; }
```

Property Value

Color

LemonChiffon

LemonChiffon color (R:255,G:250,B:205,A:255).

```
public static Color LemonChiffon { get; }
```

Property Value

Color

LightBlue

LightBlue color (R:173,G:216,B:230,A:255).

```
public static Color LightBlue { get; }
```

Property Value

Color

LightCoral

LightCoral color (R:240,G:128,B:128,A:255).

```
public static Color LightCoral { get; }
```

Property Value

Color

LightCyan

LightCyan color (R:224,G:255,B:255,A:255).

```
public static Color LightCyan { get; }
```

Property Value

Color

LightGoldenrodYellow

LightGoldenrodYellow color (R:250,G:250,B:210,A:255).

```
public static Color LightGoldenrodYellow { get; }
```

Property Value

Color

LightGray

LightGray color (R:211,G:211,B:211,A:255).

```
public static Color LightGray { get; }
```

Property Value

Color

LightGreen

LightGreen color (R:144,G:238,B:144,A:255).

```
public static Color LightGreen { get; }
```

Property Value

Color

LightPink

LightPink color (R:255,G:182,B:193,A:255).

```
public static Color LightPink { get; }
```

Property Value

Color

LightSalmon

LightSalmon color (R:255,G:160,B:122,A:255).

```
public static Color LightSalmon { get; }
```

Property Value

Color

LightSeaGreen

LightSeaGreen color (R:32,G:178,B:170,A:255).

```
public static Color LightSeaGreen { get; }
```

Property Value

Color

LightSkyBlue

LightSkyBlue color (R:135,G:206,B:250,A:255).

```
public static Color LightSkyBlue { get; }
```

Property Value

Color

LightSlateGray

LightSlateGray color (R:119,G:136,B:153,A:255).

```
public static Color LightSlateGray { get; }
```

Property Value

Color

LightSteelBlue

LightSteelBlue color (R:176,G:196,B:222,A:255).

```
public static Color LightSteelBlue { get; }
```

Property Value

Color

LightYellow

LightYellow color (R:255,G:255,B:224,A:255).

```
public static Color LightYellow { get; }
```

Property Value

Color

Lime

Lime color (R:0,G:255,B:0,A:255).

```
public static Color Lime { get; }
```

Property Value

Color

LimeGreen

LimeGreen color (R:50,G:205,B:50,A:255).

```
public static Color LimeGreen { get; }
```

Property Value

Color

Linen

Linen color (R:250,G:240,B:230,A:255).

```
public static Color Linen { get; }
```

Property Value

Color

Magenta

Magenta color (R:255,G:0,B:255,A:255).

```
public static Color Magenta { get; }
```

Property Value

Color

Maroon

Maroon color (R:128,G:0,B:0,A:255).

```
public static Color Maroon { get; }
```

Property Value

Color

MediumAquamarine

MediumAquamarine color (R:102,G:205,B:170,A:255).

```
public static Color MediumAquamarine { get; }
```

Property Value

Color

MediumBlue

MediumBlue color (R:0,G:0,B:205,A:255).

```
public static Color MediumBlue { get; }
```

Property Value

Color

MediumOrchid

MediumOrchid color (R:186,G:85,B:211,A:255).

```
public static Color MediumOrchid { get; }
```

Property Value

Color

MediumPurple

MediumPurple color (R:147,G:112,B:219,A:255).

```
public static Color MediumPurple { get; }
```

Property Value

Color

MediumSeaGreen

MediumSeaGreen color (R:60,G:179,B:113,A:255).

```
public static Color MediumSeaGreen { get; }
```

Property Value

Color

MediumSlateBlue

MediumSlateBlue color (R:123,G:104,B:238,A:255).

```
public static Color MediumSlateBlue { get; }
```

Property Value

Color

MediumSpringGreen

MediumSpringGreen color (R:0,G:250,B:154,A:255).

```
public static Color MediumSpringGreen { get; }
```

Property Value

Color

MediumTurquoise

MediumTurquoise color (R:72,G:209,B:204,A:255).

```
public static Color MediumTurquoise { get; }
```

Property Value

Color

MediumVioletRed

MediumVioletRed color (R:199,G:21,B:133,A:255).

```
public static Color MediumVioletRed { get; }
```

Property Value

Color

MidnightBlue

MidnightBlue color (R:25,G:25,B:112,A:255).

```
public static Color MidnightBlue { get; }
```

Property Value

Color

MintCream

MintCream color (R:245,G:255,B:250,A:255).

```
public static Color MintCream { get; }
```

Property Value

Color

MistyRose

MistyRose color (R:255,G:228,B:225,A:255).

```
public static Color MistyRose { get; }
```

Property Value

Color

Moccasin

Moccasin color (R:255,G:228,B:181,A:255).

```
public static Color Moccasin { get; }
```

Property Value

Color

MonoGameOrange

MonoGame orange theme color (R:231,G:60,B:0,A:255).

```
public static Color MonoGameOrange { get; }
```

Property Value

Color

NavajoWhite

NavajoWhite color (R:255,G:222,B:173,A:255).

```
public static Color NavajoWhite { get; }
```

Property Value

Color

Navy

Navy color (R:0,G:0,B:128,A:255).

```
public static Color Navy { get; }
```

Property Value

Color

OldLace

OldLace color (R:253,G:245,B:230,A:255).

```
public static Color OldLace { get; }
```

Property Value

Color

Olive

Olive color (R:128,G:128,B:0,A:255).

```
public static Color Olive { get; }
```

Property Value

Color

OliveDrab

OliveDrab color (R:107,G:142,B:35,A:255).

```
public static Color OliveDrab { get; }
```

Property Value

Color

Orange

Orange color (R:255,G:165,B:0,A:255).

```
public static Color Orange { get; }
```

Property Value

Color

OrangeRed

OrangeRed color (R:255,G:69,B:0,A:255).

```
public static Color OrangeRed { get; }
```

Property Value

Color

Orchid

Orchid color (R:218,G:112,B:214,A:255).

```
public static Color Orchid { get; }
```

Property Value

Color

PackedValue

Gets or sets packed value of this Color.

```
public uint PackedValue { get; set; }
```

Property Value

uint

PaleGoldenrod

PaleGoldenrod color (R:238,G:232,B:170,A:255).

```
public static Color PaleGoldenrod { get; }
```

Property Value

Color

PaleGreen

PaleGreen color (R:152,G:251,B:152,A:255).

```
public static Color PaleGreen { get; }
```

Property Value

Color

PaleTurquoise

PaleTurquoise color (R:175,G:238,B:238,A:255).

```
public static Color PaleTurquoise { get; }
```

Property Value

Color

PaleVioletRed

PaleVioletRed color (R:219,G:112,B:147,A:255).

```
public static Color PaleVioletRed { get; }
```

Property Value

Color

PapayaWhip

PapayaWhip color (R:255,G:239,B:213,A:255).

```
public static Color PapayaWhip { get; }
```

Property Value

Color

PeachPuff

PeachPuff color (R:255,G:218,B:185,A:255).

```
public static Color PeachPuff { get; }
```

Property Value

Color

Peru

Peru color (R:205,G:133,B:63,A:255).

```
public static Color Peru { get; }
```

Property Value

Color

Pink

Pink color (R:255,G:192,B:203,A:255).

```
public static Color Pink { get; }
```

Property Value

Color

Plum

Plum color (R:221,G:160,B:221,A:255).

```
public static Color Plum { get; }
```

Property Value

Color

PowderBlue

PowderBlue color (R:176,G:224,B:230,A:255).

```
public static Color PowderBlue { get; }
```

Property Value

Color

Purple

Purple color (R:128,G:0,B:128,A:255).

```
public static Color Purple { get; }
```

Property Value

Color

R

Gets or sets the red component.

```
[DataMember]
public byte R { get; set; }
```

Property Value

byte

Red

Red color (R:255,G:0,B:0,A:255).

```
public static Color Red { get; }
```

Property Value

Color

RosyBrown

RosyBrown color (R:188,G:143,B:143,A:255).

```
public static Color RosyBrown { get; }
```

Property Value

Color

RoyalBlue

RoyalBlue color (R:65,G:105,B:225,A:255).

```
public static Color RoyalBlue { get; }
```

Property Value

Color

SaddleBrown

SaddleBrown color (R:139,G:69,B:19,A:255).

```
public static Color SaddleBrown { get; }
```

Property Value

Color

Salmon

Salmon color (R:250,G:128,B:114,A:255).

```
public static Color Salmon { get; }
```

Property Value

Color

SandyBrown

SandyBrown color (R:244,G:164,B:96,A:255).

```
public static Color SandyBrown { get; }
```

Property Value

Color

SeaGreen

SeaGreen color (R:46,G:139,B:87,A:255).

```
public static Color SeaGreen { get; }
```

Property Value

Color

SeaShell

SeaShell color (R:255,G:245,B:238,A:255).

```
public static Color SeaShell { get; }
```

Property Value

Color

Sienna

Sienna color (R:160,G:82,B:45,A:255).

```
public static Color Sienna { get; }
```

Property Value

Color

Silver

Silver color (R:192,G:192,B:192,A:255).

```
public static Color Silver { get; }
```

Property Value

Color

SkyBlue

SkyBlue color (R:135,G:206,B:235,A:255).

```
public static Color SkyBlue { get; }
```

Property Value

Color

SlateBlue

SlateBlue color (R:106,G:90,B:205,A:255).

```
public static Color SlateBlue { get; }
```

Property Value

Color

SlateGray

SlateGray color (R:112,G:128,B:144,A:255).

```
public static Color SlateGray { get; }
```

Property Value

Color

Snow

Snow color (R:255,G:250,B:250,A:255).

```
public static Color Snow { get; }
```

Property Value

Color

SpringGreen

SpringGreen color (R:0,G:255,B:127,A:255).

```
public static Color SpringGreen { get; }
```

Property Value

Color

SteelBlue

SteelBlue color (R:70,G:130,B:180,A:255).

```
public static Color SteelBlue { get; }
```

Property Value

Color

Tan

Tan color (R:210,G:180,B:140,A:255).

```
public static Color Tan { get; }
```

Property Value

Color

Teal

Teal color (R:0,G:128,B:128,A:255).

```
public static Color Teal { get; }
```

Property Value

Color

Thistle

Thistle color (R:216,G:191,B:216,A:255).

```
public static Color Thistle { get; }
```

Property Value

Color

Tomato

Tomato color (R:255,G:99,B:71,A:255).

```
public static Color Tomato { get; }
```

Property Value

Color

Transparent

Transparent color (R:0,G:0,B:0,A:0).

```
public static Color Transparent { get; }
```

Property Value

Color

Turquoise

Turquoise color (R:64,G:224,B:208,A:255).

```
public static Color Turquoise { get; }
```

Property Value

Color

Violet

Violet color (R:238,G:130,B:238,A:255).

```
public static Color Violet { get; }
```

Property Value

Color

Wheat

Wheat color (R:245,G:222,B:179,A:255).

```
public static Color Wheat { get; }
```

Property Value

Color

White

White color (R:255,G:255,B:255,A:255).

```
public static Color White { get; }
```

Property Value

Color

WhiteSmoke

WhiteSmoke color (R:245,G:245,B:245,A:255).

```
public static Color WhiteSmoke { get; }
```

Property Value

Color

Yellow

Yellow color (R:255,G:255,B:0,A:255).

```
public static Color Yellow { get; }
```

Property Value

Color

YellowGreen

YellowGreen color (R:154,G:205,B:50,A:255).

```
public static Color YellowGreen { get; }
```

Property Value

Color

Methods

Deconstruct(out byte, out byte, out byte)

Deconstruction method for Color.

```
public void Deconstruct(out byte r, out byte g, out byte b)
```

Parameters

r byte

Red component value from 0 to 255.

g byte

Green component value from 0 to 255.

b byte

Blue component value from 0 to 255.

Deconstruct(out byte, out byte, out byte, out byte)

Deconstruction method for Color with Alpha.

```
public void Deconstruct(out byte r, out byte g, out byte b, out byte a)
```

Parameters

r byte

Red component value from 0 to 255.

g byte

Green component value from 0 to 255.

b byte

Blue component value from 0 to 255.

a byte

Alpha component value from 0 to 255.

Deconstruct(out float, out float, out float)

Deconstruction method for Color.

```
public void Deconstruct(out float r, out float g, out float b)
```

Parameters

r float

Red component value from 0.0f to 1.0f.

g float

Green component value from 0.0f to 1.0f.

b float

Blue component value from 0.0f to 1.0f.

Deconstruct(out float, out float, out float, out float)

Deconstruction method for Color with Alpha.

```
public void Deconstruct(out float r, out float g, out float b, out float a)
```

Parameters

r float

Red component value from 0.0f to 1.0f.

g float

Green component value from 0.0f to 1.0f.

b float

Blue component value from 0.0f to 1.0f.

a float

Alpha component value from 0.0f to 1.0f.

Equals(Color)

Compares whether current instance is equal to specified Color.

```
public bool Equals(Color other)
```

Parameters

other Color

The Color to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

obj object

The Color to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

FromHSL(float, float, float)

Creates a Color from HSL values

```
public static Color FromHSL(float h, float s, float l)
```

Parameters

`h` float

Hue component value, from 0.0f to 360.0f

`s` float

Saturation component value, from 0.0f to 100.0f

`l` float

Luminosity (brightness) component value, from 0.0f to 100.0f

Returns

Color

Color with the HSL values

FromHSV(float, float, float)

Creates a Color from HSV values.

```
public static Color FromHSV(float h, float s, float v)
```

Parameters

h float

Hue component value, ranging from 0.0f to 360.0f

s float

Saturation component value, ranging from 0.0f to 1.0f

v float

Value component value, ranging from 0.0f to 1.0f

Returns

Color

Color with the HSV values

FromNonPremultiplied(Vector4)

Translate a non-premultipled alpha Color to a Color that contains premultiplied alpha.

```
public static Color FromNonPremultiplied(Vector4 vector)
```

Parameters

vector Vector4

A Vector4 representing color.

Returns

Color

A Color which contains premultiplied alpha data.

FromNonPremultiplied(int, int, int, int)

Translate a non-premultipled alpha Color to a Color that contains premultiplied alpha.

```
public static Color FromNonPremultiplied(int r, int g, int b, int a)
```

Parameters

r int

Red component value.

g int

Green component value.

b int

Blue component value.

a int

Alpha component value.

Returns

Color

A Color which contains premultiplied alpha data.

GetHashCode()

Gets the hash code of this Color.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Color.

Lerp(Color, Color, float)

Performs linear interpolation of Color.

```
public static Color Lerp(Color value1, Color value2, float amount)
```

Parameters

`value1` Color

Source Color.

`value2` Color

Destination Color.

`amount` float

Interpolation factor.

Returns

Color

Interpolated Color.

LerpPrecise(Color, Color, float)

Lerp(Color, Color, float) should be used instead of this function.

```
[Obsolete("Color.Lerp should be used instead of this function.")]
public static Color LerpPrecise(Color value1, Color value2, float amount)
```

Parameters

`value1` Color

`value2` Color

`amount` float

Returns

Color

Interpolated Color.

Multiply(Color, float)

Multiplies the RGBA component values of a color by the specified value.

```
public static Color Multiply(Color value, float scale)
```

Parameters

`value` Color

The source color value to multiply.

`scale` float

The value to multiply the RGBA component values by.

Returns

Color

The new color value created as a result of the multiplication.

MultiplyAlpha(Color, float)

Multiplies the Alpha component value of a color by the specified value.

```
public static Color MultiplyAlpha(Color value, float scale)
```

Parameters

`value` Color

The source color value to multiply.

`scale` float

The value to multiply the Alpha component value by.

Returns

Color

The new color value created as a result of the multiplication.

ToHSL(out float, out float, out float)

Converts Color into HSL components.

```
public void ToHSL(out float h, out float s, out float l)
```

Parameters

`h` float

Hue component from 0.0f to 360.0f.

`s` float

Saturation component from 0.0f to 100.0f.

`l` float

Luminosity (or brightness) component from 0.0f to 100.0f.

ToHSV(out float, out float, out float)

Converts Color into HSV components

```
public void ToHSV(out float h, out float s, out float v)
```

Parameters

`h` float

Hue component value from 0.0f to 360.0f

s float

Saturation component value from 0.0f to 100.0f

v float

Value component value from 0.0f to 100.0f

ToString()

Returns a string representation of this Color in the format: {R:[red] G:[green] B:[blue] A:[alpha]}

```
public override string ToString()
```

Returns

string

string representation of this Color.

ToVector3()

Gets a Vector3 representation for this object.

```
public Vector3 ToVector3()
```

Returns

Vector3

A Vector3 representation for this object.

ToVector4()

Gets a Vector4 representation for this object.

```
public Vector4 ToVector4()
```

Returns

Vector4

A Vector4 representation for this object.

Operators

operator ==(Color, Color)

Compares whether two Color instances are equal.

```
public static bool operator ==(Color a, Color b)
```

Parameters

a Color

Color instance on the left of the equal sign.

b Color

Color instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Color(Vector4)

Converts a Vector4 to a Vector4 by mapping XYZW -> RGBA.

```
public static implicit operator Color(Vector4 value)
```

Parameters

`value` Vector4

The converted value.

Returns

Color

operator !=(Color, Color)

Compares whether two Color instances are not equal.

```
public static bool operator !=(Color a, Color b)
```

Parameters

a Color

Color instance on the left of the not equal sign.

b Color

Color instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Color, Color)

Multiplies two colors compenent-wise.

```
public static Color operator *(Color color1, Color color2)
```

Parameters

`color1` Color

The first color to be multiplied.

`color2` Color

The second color to be multiplied.

Returns

Color

The new color value created as a result of the multiplication.

operator *(Color, float)

Multiplies the RGBA component values of a color by the specified value.

```
public static Color operator *(Color value, float scale)
```

Parameters

`value` Color

The source color value to multiply.

`scale` float

The value to multiply the RGBA component values by.

Returns

Color

The new color value created as a result of the multiplication.

operator *(float, Color)

Multiplies the RGBA component values of a color by the specified value.

```
public static Color operator *(float scale, Color value)
```

Parameters

`scale` float

The value to multiply the RGBA component values by.

`value` Color

The source color value to multiply.

Returns

Color

The new color value created as a result of the multiplication.

Enum ContainmentType

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines how the bounding volumes intersects or contain one another.

```
public enum ContainmentType
```

Fields

`Contains = 1`

Indicates that one bounding volume completely contains another volume.

`Disjoint = 0`

Indicates that there is no overlap between two bounding volumes.

`Intersects = 2`

Indicates that bounding volumes partially overlap one another.

Class Curve

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Contains a collection of CurveKey points in 2D space and provides methods for evaluating features of the curve they define.

```
[DataContract]
public class Curve : ICurveEvaluator<float>
```

Inheritance

object ← Curve

Implements

ICurveEvaluator<float>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Curve()

Constructs a curve.

```
public Curve()
```

Properties

IsConstant

Returns `true` if this curve is constant (has zero or one points); `false` otherwise.

```
[DataMember]
public bool IsConstant { get; }
```

Property Value

bool

Keys

The collection of curve keys.

```
[DataMember]
public CurveKeyCollection Keys { get; }
```

Property Value

CurveKeyCollection

PostLoop

Defines how to handle weighting values that are greater than the last control point in the curve.

```
[DataMember]
public CurveLoopType PostLoop { get; set; }
```

Property Value

CurveLoopType

PreLoop

Defines how to handle weighting values that are less than the first control point in the curve.

```
[DataMember]
public CurveLoopType PreLoop { get; set; }
```

Property Value

CurveLoopType

Methods

Clone()

Creates a copy of this curve.

```
public Curve Clone()
```

Returns

Curve

A copy of this curve.

ComputeTangent(int, CurveTangent)

Computes tangent for the specific key in the collection.

```
public void ComputeTangent(int keyIndex, CurveTangent tangentType)
```

Parameters

`keyIndex` int

The index of a key in the collection.

`tangentType` CurveTangent

The tangent type for both in and out.

ComputeTangent(int, CurveTangent, CurveTangent)

Computes tangent for the specific key in the collection.

```
public void ComputeTangent(int keyIndex, CurveTangent tangentInType,
CurveTangent tangentOutType)
```

Parameters

`keyIndex` int

The index of key in the collection.

`tangentInType` CurveTangent

The tangent in-type. TangentIn for more details.

`tangentOutType` CurveTangent

The tangent out-type. TangentOut for more details.

ComputeTangents(CurveTangent)

Computes tangents for all keys in the collection.

```
public void ComputeTangents(CurveTangent tangentType)
```

Parameters

`tangentType` CurveTangent

The tangent type for both in and out.

ComputeTangents(CurveTangent, CurveTangent)

Computes tangents for all keys in the collection.

```
public void ComputeTangents(CurveTangent tangentInType, CurveTangent tangentOutType)
```

Parameters

`tangentInType` CurveTangent

The tangent in-type. TangentIn for more details.

`tangentOutType` CurveTangent

The tangent out-type. TangentOut for more details.

Evaluate(float)

Evaluate the value at a position of this Curve.

```
public float Evaluate(float position)
```

Parameters

`position` float

The position on this Curve.

Returns

float

Value at the position on this Curve.

Enum CurveContinuity

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines the continuity of keys on a Curve.

```
public enum CurveContinuity
```

Fields

`Smooth = 0`

Interpolation can be used between this key and the next.

`Step = 1`

Interpolation cannot be used. A position between the two points returns this point.

Class CurveKey

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Key point on the Curve.

```
[DataContract]
public class CurveKey : IEquatable<CurveKey>, IComparable<CurveKey>
```

Inheritance

object ← CurveKey

Implements

IEquatable<CurveKey>, IComparable<CurveKey>

Inherited Members

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CurveKey()

Creates a new instance of CurveKey class with position: 0 and value: 0.

```
public CurveKey()
```

CurveKey(float, float)

Creates a new instance of CurveKey class.

```
public CurveKey(float position, float value)
```

Parameters

`position` float

Position on the curve.

`value` float

Value of the control point.

CurveKey(float, float, float, float)

Creates a new instance of CurveKey class.

```
public CurveKey(float position, float value, float tangentIn, float tangentOut)
```

Parameters

`position` float

Position on the curve.

`value` float

Value of the control point.

`tangentIn` float

Tangent approaching point from the previous point on the curve.

`tangentOut` float

Tangent leaving point toward next point on the curve.

CurveKey(float, float, float, float, CurveContinuity)

Creates a new instance of CurveKey class.

```
public CurveKey(float position, float value, float tangentIn, float tangentOut,
CurveContinuity continuity)
```

Parameters

`position` float

Position on the curve.

`value` float

Value of the control point.

`tangentIn` float

Tangent approaching point from the previous point on the curve.

`tangentOut` float

Tangent leaving point toward next point on the curve.

`continuity` CurveContinuity

Indicates whether the curve is discrete or continuous.

Properties

Continuity

Gets or sets the indicator whether the segment between this point and the next point on the curve is discrete or continuous.

```
[DataMember]
public CurveContinuity Continuity { get; set; }
```

Property Value

CurveContinuity

Position

Gets a position of the key on the curve.

```
[DataMember]
public float Position { get; }
```

Property Value

float

TangentIn

Gets or sets a tangent when approaching this point from the previous point on the curve.

```
[DataMember]
public float TangentIn { get; set; }
```

Property Value

float

TangentOut

Gets or sets a tangent when leaving this point to the next point on the curve.

```
[DataMember]
public float TangentOut { get; set; }
```

Property Value

float

Value

Gets a value of this point.

```
[DataMember]
public float Value { get; set; }
```

Property Value

float

Methods

Clone()

Creates a copy of this key.

```
public CurveKey Clone()
```

Returns

CurveKey

A copy of this key.

CompareTo(CurveKey)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

```
public int CompareTo(CurveKey other)
```

Parameters

`other` CurveKey

An object to compare with this instance.

Returns

int

A value that indicates the relative order of the objects being compared. The return value has these meanings:

Value	Meaning
Less than zero	This instance precedes `other` in the sort order.
Zero	This instance occurs in the same position in the sort order as `other`.
Greater than zero	This instance follows `other` in the sort order.

Equals(CurveKey)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(CurveKey other)
```

Parameters

`other` CurveKey

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to the current object; otherwise, false.

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

int

A hash code for the current object.

Operators

operator ==(CurveKey, CurveKey)

Compares whether two CurveKey instances are equal.

```
public static bool operator ==(CurveKey value1, CurveKey value2)
```

Parameters

`value1` CurveKey

CurveKey instance on the left of the equal sign.

`value2` CurveKey

CurveKey instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

operator !=(CurveKey, CurveKey)

Compares whether two CurveKey instances are not equal.

```
public static bool operator !=(CurveKey value1, CurveKey value2)
```

Parameters

`value1` CurveKey

CurveKey instance on the left of the not equal sign.

`value2` CurveKey

CurveKey instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

Class CurveKeyCollection

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

The collection of the CurveKey elements and a part of the Curve class.

```
[DataContract]
public class CurveKeyCollection : ICollection<CurveKey>, IEnumerable<CurveKey>,
IEnumerable
```

Inheritance

object ← CurveKeyCollection

Implements

ICollection<CurveKey>, IEnumerable<CurveKey>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CurveKeyCollection()

Creates a new instance of CurveKeyCollection class.

```
public CurveKeyCollection()
```

Properties

Count

Returns the count of keys in this collection.

```
[DataMember]
public int Count { get; }
```

Property Value

int

IsReadOnly

Returns false because it is not a read-only collection.

```
[DataMember]
public bool IsReadOnly { get; }
```

Property Value

bool

this[int]

Indexer.

```
[DataMember(Name = "Items")]
public CurveKey this[int index] { get; set; }
```

Parameters

`index` int

The index of key in this collection.

Property Value

CurveKey

CurveKey at `index` position.

Methods

Add(CurveKey)

Adds a key to this collection.

```
public void Add(CurveKey item)
```

Parameters

`item` CurveKey

New key for the collection.

Remarks

The new key would be added respectively to a position of that key and the position of other keys.

Exceptions

ArgumentNullException

Throws if `item` is null.

Clear()

Removes all keys from this collection.

```
public void Clear()
```

Clone()

Creates a copy of this collection.

```
public CurveKeyCollection Clone()
```

Returns

CurveKeyCollection

A copy of this collection.

Contains(CurveKey)

Determines whether this collection contains a specific key.

```
public bool Contains(CurveKey item)
```

Parameters

`item` CurveKey

The key to locate in this collection.

Returns

bool

`true` if the key is found; `false` otherwise.

CopyTo(CurveKey[], int)

Copies the keys of this collection to an array, starting at the array index provided.

```
public void CopyTo(CurveKey[] array, int arrayIndex)
```

Parameters

`array` CurveKey[]

Destination array where elements will be copied.

`arrayIndex` int

The zero-based index in the array to start copying from.

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<CurveKey> GetEnumerator()
```

Returns

IEnumerator<CurveKey>

An enumerator for the CurveKeyCollection.

IndexAtPosition(float)

Searches for the key with the lowest position greater than or equal to the specified position.

```
public int IndexAtPosition(float position)
```

Parameters

`position` float

Position to search for.

Returns

int

The zero-based index of the first matching position if there is a match; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than `position` or, if there is no larger element, the bitwise complement of `Count`.

IndexOf(CurveKey)

Finds element in the collection and returns its index.

```
public int IndexOf(CurveKey item)
```

Parameters

`item` CurveKey

Element for the search.

Returns

int

Index of the element; or -1 if item is not found.

Remove(CurveKey)

Removes specific element.

```
public bool Remove(CurveKey item)
```

Parameters

`item` CurveKey

The element

Returns

bool

`true` if item is successfully removed; `false` otherwise. This method also returns `false` if item was not found.

RemoveAt(int)

Removes element at the specified index.

```
public void RemoveAt(int index)
```

Parameters

`index` int

The index which element will be removed.

Enum CurveLoopType

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines how the Curve value is determined for position before first point or after the end point on the Curve.

```
public enum CurveLoopType
```

Fields

`Constant = 0`

The value of Curve will be evaluated as first point for positions before the beginning and end point for positions after the end.

`Cycle = 1`

The positions will wrap around from the end to beginning of the Curve for determined the value.

`CycleOffset = 2`

The positions will wrap around from the end to beginning of the Curve. The value will be offset by the difference between the values of first and end CurveKey multiplied by the wrap amount. If the position is before the beginning of the Curve the difference will be subtracted from its value; otherwise the difference will be added.

`Linear = 4`

The linear interpolation will be performed for determined the value.

`Oscillate = 3`

The value at the end of the Curve act as an offset from the same side of the Curve toward the opposite side.

Enum CurveTangent

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines the different tangent types to be calculated for CurveKey points in a Curve.

```
public enum CurveTangent
```

Fields

`Flat = 0`

The tangent which always has a value equal to zero.

`Linear = 1`

The tangent which contains a difference between current tangent value and the tangent value from the previous CurveKey.

`Smooth = 2`

The smoouth tangent which contains the inflection between TangentIn and TangentOut by taking into account the values of both neighbors of the CurveKey.

Enum DisplayOrientation

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines the orientation of the display.

```
[Flags]
public enum DisplayOrientation
```

Fields

`Default = 0`

The default orientation.

`LandscapeLeft = 1`

The display is rotated counterclockwise into a landscape orientation. Width is greater than height.

`LandscapeRight = 2`

The display is rotated clockwise into a landscape orientation. Width is greater than height.

`Portrait = 4`

The display is rotated as portrait, where height is greater than width.

`PortraitDown = 8`

The display is rotated as inverted portrait, where height is greater than width.

`Unknown = 16`

Unknown display orientation.

Class DrawableGameComponent

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

A drawable object that, when added to the Game.Components collection of a Game instance, will have it's Draw(GameTime) method called when Game.Draw(GameTime) is called.

```
public class DrawableGameComponent : GameComponent, IGameComponent, IUpdateable,
IDisposable, IDrawable
```

Inheritance

object ← GameComponent ← DrawableGameComponent

Implements

IGameComponent, IUpdateable, IDisposable, IDrawable

Inherited Members

GameComponent.Game , GameComponent.Enabled , GameComponent.UpdateOrder , GameComponent.EnabledChanged , GameComponent.UpdateOrderChanged , GameComponent.Update(GameTime) , GameComponent.OnUpdateOrderChanged(object, EventArgs) , GameComponent.OnEnabledChanged(object, EventArgs) , GameComponent.Dispose() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This inherits from GameComponent so it's Update(GameTime) method will also be called when Game.Update(GameTime) is called.

Constructors

DrawableGameComponent(Game)

Create a DrawableGameComponent.

```
public DrawableGameComponent(Game game)
```

Parameters

`game` Game

The game that this component will belong to.

Properties

DrawOrder

Gets the order in which this component should be drawn, relative to other components that are in the same GameComponentCollection.

```
public int DrawOrder { get; set; }
```

Property Value

int

Remarks

This value can be any integer. Components in the GameComponentCollection are drawn in ascending order based on their **DrawOrder**.

GraphicsDevice

Get the GraphicsDevice that this DrawableGameComponent uses for drawing.

```
public GraphicsDevice GraphicsDevice { get; }
```

Property Value

GraphicsDevice

Visible

Gets or Sets a value that indicates whether the Draw(GameTime) method of this component should be called.

```
public bool Visible { get; set; }
```

Property Value

bool

Methods

Dispose(bool)

Shuts down the component.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

Draw(GameTime)

Draw this component.

```
public virtual void Draw(GameTime gameTime)
```

Parameters

`gameTime` GameTime

The time elapsed since the last call to Draw(GameTime).

Initialize()

Called when the GameComponent needs to be initialized.

```
public override void Initialize()
```

LoadContent()

Load graphical resources needed by this component.

```
protected virtual void LoadContent()
```

OnDrawOrderChanged(object, EventArgs)

Called when DrawOrder changed.

```
protected virtual void OnDrawOrderChanged(object sender, EventArgs args)
```

Parameters

`sender` object

This DrawableGameComponent.

`args` EventArgs

Arguments to the DrawOrderChanged event.

OnVisibleChanged(object, EventArgs)

Called when Visible changed.

```
protected virtual void OnVisibleChanged(object sender, EventArgs args)
```

Parameters

`sender` object

This DrawableGameComponent.

`args` EventArgs

Arguments to the VisibleChanged event.

UnloadContent()

Unload graphical resources needed by this component.

```
protected virtual void UnloadContent()
```

Events

DrawOrderChanged

Raised when DrawOrder changed.

```
public event EventHandler<EventArgs> DrawOrderChanged
```

Event Type

EventHandler<EventArgs>

VisibleChanged

Raised when Visible changed.

```
public event EventHandler<EventArgs> VisibleChanged
```

Event Type

EventHandler<EventArgs>

Class ExitingEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Event arguments for Exiting.

```
public class ExitingEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← ExitingEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Cancel

Set to `true` to cancel closing the game.

```
public bool Cancel { get; set; }
```

Property Value

bool

Struct FileDropEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

This class is used in the FileDrop event as EventArgs.

```
public struct FileDropEventArgs
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

FileDropEventArgs(string[])

Creates an instance of FileDropEventArgs.

```
public FileDropEventArgs(string[] files)
```

Parameters

`files` string[]

Array of paths to dropped files.

Properties

Files

The paths of dropped files

```
public readonly string[] Files { get; }
```

Property Value

string[]

Class FrameworkDispatcher

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Helper class for processing internal framework events.

```
public static class FrameworkDispatcher
```

Inheritance

object ← FrameworkDispatcher

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

If you use Game class, Update() is called automatically. Otherwise you must call it as part of your game loop.

Methods

Update()

Processes internal framework updates and raises events. If you don't use the Game class you should call this from your update loop. This method is not thread safe.

```
public static void Update()
```

Class Game

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

This class is the entry point for most games. Handles setting up a window and graphics and runs a game loop that calls Update(GameTime) and Draw(GameTime).

```
public class Game : IDisposable
```

Inheritance

object ← Game

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Game()

Create a Game.

```
public Game()
```

Properties

Components

A collection of game components attached to this Game.

```
public GameComponentCollection Components { get; }
```

Property Value

GameComponentCollection

Content

The ContentManager of this Game.

```
public ContentManager Content { get; set; }
```

Property Value

ContentManager

Exceptions

ArgumentNullException

If Content is set to

```
null
```

.

GraphicsDevice

Gets the GraphicsDevice used for rendering by this Game.

```
public GraphicsDevice GraphicsDevice { get; }
```

Property Value

GraphicsDevice

Exceptions

InvalidOperationException

There is no GraphicsDevice attached to this Game.

InactiveSleepTime

Gets or sets time to sleep between frames when the game is not active

```
public TimeSpan InactiveSleepTime { get; set; }
```

Property Value

TimeSpan

IsActive

Indicates if the game is the focused application.

```
public bool IsActive { get; }
```

Property Value

bool

IsFixedTimeStep

Indicates if this game is running with a fixed time between frames.

When set to

```
true
```

the target time between frames is given by TargetElapsedTime.

```
public bool IsFixedTimeStep { get; set; }
```

Property Value

bool

IsMouseVisible

Indicates if the mouse cursor is visible on the game screen.

```
public bool IsMouseVisible { get; set; }
```

Property Value

bool

LaunchParameters

The start up parameters for this Game.

```
public LaunchParameters LaunchParameters { get; }
```

Property Value

LaunchParameters

MaxElapsedTime

The maximum amount of time we will frameskip over and only perform Update calls with no Draw calls. MonoGame extension.

```
public TimeSpan MaxElapsedTime { get; set; }
```

Property Value

TimeSpan

Services

Get a container holding service providers attached to this Game.

```
public GameServiceContainer Services { get; }
```

Property Value

GameServiceContainer

TargetElapsedTime

The time between frames when running with a fixed time step. IsFixedTimeStep

```
public TimeSpan TargetElapsedTime { get; set; }
```

Property Value

TimeSpan

Exceptions

ArgumentOutOfRangeException

> Target elapsed time must be strictly larger than zero.

Window

The system window that this game is displayed on.

```
public GameWindow Window { get; }
```

Property Value

GameWindow

Methods

BeginDraw()

Called right before Draw(GameTime) is normally called. Can return

```
false
```

to let the game loop not call Draw(GameTime).

```
protected virtual bool BeginDraw()
```

Returns

bool

```
true
```

if Draw(GameTime) should be called,

```
false
```

if it should not.

BeginRun()

Called after Initialize(), but before the first call to Update(GameTime).

```
protected virtual void BeginRun()
```

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

Draw(GameTime)

Called when the game should draw a frame.

Draws the DrawableGameComponent instances attached to this game. Override this to render your game.

```
protected virtual void Draw(GameTime gameTime)
```

Parameters

`gameTime` GameTime

A GameTime instance containing the elapsed time since the last call to Draw(GameTime) and the total time elapsed since the game started.

EndDraw()

Called right after Draw(GameTime). Presents the rendered frame in the GameWindow.

```
protected virtual void EndDraw()
```

EndRun()

Called when the game loop has been terminated before exiting.

```
protected virtual void EndRun()
```

Exit()

Exit the game at the end of this tick.

```
public void Exit()
```

~Game()

```
protected ~Game()
```

Initialize()

Override this to initialize the game and load any needed non-graphical resources.

Initializes attached GameComponent instances and calls LoadContent().

```
protected virtual void Initialize()
```

LoadContent()

Override this to load graphical resources required by the game.

```
protected virtual void LoadContent()
```

OnActivated(object, EventArgs)

Called when the game gains focus. Raises the Activated event.

```
protected virtual void OnActivated(object sender, EventArgs args)
```

Parameters

`sender` object

This Game.

`args` EventArgs

The arguments to the Activated event.

OnDeactivated(object, EventArgs)

Called when the game loses focus. Raises the Deactivated event.

```
protected virtual void OnDeactivated(object sender, EventArgs args)
```

Parameters

`sender` object

This Game.

`args` EventArgs

The arguments to the Deactivated event.

OnExiting(object, ExitingEventArgs)

Called when the game is exiting. Raises the Exiting event.

```
protected virtual void OnExiting(object sender, ExitingEventArgs args)
```

Parameters

`sender` object

This Game.

`args` ExitingEventArgs

The arguments to the Exiting event.

ResetElapsedTime()

Reset the elapsed game time to Zero.

```
public void ResetElapsedTime()
```

Run()

Run the game using the default GameRunBehavior for the current platform.

```
public void Run()
```

Run(GameRunBehavior)

Run the game.

```
public void Run(GameRunBehavior runBehavior)
```

Parameters

`runBehavior` GameRunBehavior

Indicate if the game should be run synchronously or asynchronously.

RunOneFrame()

Run the game for one frame, then exit.

```
public void RunOneFrame()
```

SuppressDraw()

Supress calling Draw(GameTime) in the game loop.

```
public void SuppressDraw()
```

Tick()

Run one iteration of the game loop.

Makes at least one call to Update(GameTime) and exactly one call to Draw(GameTime) if drawing is not supressed. When IsFixedTimeStep is set to

```
false
```

this will make exactly one call to Update(GameTime).

```
public void Tick()
```

UnloadContent()

Override this to unload graphical resources loaded by the game.

```
protected virtual void UnloadContent()
```

Update(GameTime)

Called when the game should update.

Updates the GameComponent instances attached to this game. Override this to update your game.

```
protected virtual void Update(GameTime gameTime)
```

Parameters

`gameTime` GameTime

The elapsed time since the last call to Update(GameTime).

Events

Activated

Raised when the game gains focus.

```
public event EventHandler<EventArgs> Activated
```

Event Type

EventHandler<EventArgs>

Deactivated

Raised when the game loses focus.

```
public event EventHandler<EventArgs> Deactivated
```

Event Type

EventHandler<EventArgs>

Disposed

Raised when this game is being disposed.

```
public event EventHandler<EventArgs> Disposed
```

Event Type

EventHandler<EventArgs>

Exiting

Raised when this game is exiting.

```
public event EventHandler<ExitingEventArgs> Exiting
```

Event Type

EventHandler<ExitingEventArgs>

Class GameComponent

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

An object that can be attached to a Game and have its Update(GameTime) method called when Update(GameTime) is called.

```
public class GameComponent : IGameComponent, IUpdateable, IDisposable
```

Inheritance

object ← GameComponent

Implements

IGameComponent, IUpdateable, IDisposable

Derived

DrawableGameComponent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GameComponent(Game)

Create a GameComponent.

```
public GameComponent(Game game)
```

Parameters

`game` Game

The game that this component will belong to.

Properties

Enabled

Indicates whether GameComponent.Update(GameTime) should be called when Game.Update(GameTime) is called.

```
public bool Enabled { get; set; }
```

Property Value

bool

Game

The Game that owns this GameComponent.

```
public Game Game { get; }
```

Property Value

Game

UpdateOrder

Indicates the order in which the GameComponent should be updated relative to other GameComponent instances. Lower values are updated first.

```
public int UpdateOrder { get; set; }
```

Property Value

int

Methods

Dispose()

Shuts down the component.

```
public void Dispose()
```

Dispose(bool)

Shuts down the component.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

~GameComponent()

```
protected ~GameComponent()
```

Initialize()

Called when the GameComponent needs to be initialized.

```
public virtual void Initialize()
```

OnEnabledChanged(object, EventArgs)

Called when Enabled changed. Raises the EnabledChanged event.

```
protected virtual void OnEnabledChanged(object sender, EventArgs args)
```

Parameters

`sender` object

This GameComponent.

`args` EventArgs

Arguments to the EnabledChanged event.

OnUpdateOrderChanged(object, EventArgs)

Called when UpdateOrder changed. Raises the UpdateOrderChanged event.

```
protected virtual void OnUpdateOrderChanged(object sender, EventArgs args)
```

Parameters

`sender` object

This GameComponent.

`args` EventArgs

Arguments to the UpdateOrderChanged event.

Update(GameTime)

Update the component.

```
public virtual void Update(GameTime gameTime)
```

Parameters

`gameTime` GameTime

GameTime of the Game.

Events

EnabledChanged

Raised when Enabled changed.

```
public event EventHandler<EventArgs> EnabledChanged
```

Event Type

EventHandler<EventArgs>

UpdateOrderChanged

Raised when UpdateOrder changed.

```
public event EventHandler<EventArgs> UpdateOrderChanged
```

Event Type

EventHandler<EventArgs>

Class GameComponentCollection

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

A collection of IGameComponent instances.

```
public sealed class GameComponentCollection : Collection<IGameComponent>,
IList<IGameComponent>, ICollection<IGameComponent>, IReadOnlyList<IGameComponent>,
IReadOnlyCollection<IGameComponent>, IEnumerable<IGameComponent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<IGameComponent> ← GameComponentCollection

Implements

IList<IGameComponent>, ICollection<IGameComponent>, IReadOnlyList<IGameComponent>, IReadOnlyCollection<IGameComponent>, IEnumerable<IGameComponent>, IList, ICollection, IEnumerable

Inherited Members

Collection<IGameComponent>.Add(IGameComponent) , Collection<IGameComponent>.Clear() , Collection<IGameComponent>.Contains(IGameComponent) , Collection<IGameComponent>.CopyTo(IGameComponent[], int) , Collection<IGameComponent>.GetEnumerator() , Collection<IGameComponent>.IndexOf(IGameComponent) , Collection<IGameComponent>.Insert(int, IGameComponent) , Collection<IGameComponent>.Remove(IGameComponent) , Collection<IGameComponent>.RemoveAt(int) , Collection<IGameComponent>.Count , Collection<IGameComponent>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ClearItems()

Removes every GameComponent from this GameComponentCollection. Triggers ComponentRemoved once for each GameComponent removed.

```
protected override void ClearItems()
```

InsertItem(int, IGameComponent)

Inserts an element into the collection at the specified index. Triggers ComponentAdded.

```
protected override void InsertItem(int index, IGameComponent item)
```

Parameters

`index` int

The zero-based index at which item should be inserted.

`item` IGameComponent

The object to insert.

Exceptions

ArgumentException

Specified IGameComponent is already present in the collection.

RemoveItem(int)

Removes the element at the specified index of the GameComponentCollection. Triggers Component Removed.

```
protected override void RemoveItem(int index)
```

Parameters

`index` int

The zero-based index of the element to remove.

SetItem(int, IGameComponent)

Replaces the element at the specified index.

```
protected override void SetItem(int index, IGameComponent item)
```

Parameters

`index` int

The zero-based index of the element to replace.

`item` IGameComponent

The new value for the element at the specified index.

Remarks

This method is not supported and will always throw NotSupportedException

Exceptions

NotSupportedException

Thrown always

Events

ComponentAdded

Event that is triggered when a GameComponent is added to this GameComponentCollection.

```
public event EventHandler<GameComponentCollectionEventArgs> ComponentAdded
```

Event Type

EventHandler<GameComponentCollectionEventArgs>

ComponentRemoved

Event that is triggered when a GameComponent is removed from this GameComponentCollection.

```
public event EventHandler<GameComponentCollectionEventArgs> ComponentRemoved
```

Event Type

EventHandler<GameComponentCollectionEventArgs>

Class GameComponentCollectionEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Arguments for the ComponentAdded and ComponentRemoved events.

```
public class GameComponentCollectionEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← GameComponentCollectionEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GameComponentCollectionEventArgs(IGameComponent)

Create a GameComponentCollectionEventArgs instance.

```
public GameComponentCollectionEventArgs(IGameComponent gameComponent)
```

Parameters

`gameComponent` IGameComponent

The IGameComponent that the event notifies about.

Properties

GameComponent

The IGameComponent that the event notifies about.

```
public IGameComponent GameComponent { get; }
```

Property Value

IGameComponent

Enum GameRunBehavior

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines how the Game should be run.

```
public enum GameRunBehavior
```

Fields

`Asynchronous = 0`

The game loop will be run asynchronously.

`Synchronous = 1`

The game loop will be run synchronously.

Class GameServiceContainer

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

A container for services for a Game.

```
public class GameServiceContainer : IServiceProvider
```

Inheritance

object ← GameServiceContainer

Implements

IServiceProvider

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GameServiceContainer()

Create an empty GameServiceContainer.

```
public GameServiceContainer()
```

Methods

AddService(Type, object)

Add a service provider to this container.

```
public void AddService(Type type, object provider)
```

Parameters

`type` Type

The type of the service.

`provider` object

The provider of the service.

Exceptions

ArgumentNullException

If `type` or `provider` is

```
null
```

.

ArgumentException

If `provider` cannot be assigned to `type`.

AddService<T>(T)

Add a service provider to this container.

```
public void AddService<T>(T provider)
```

Parameters

`provider` T

The provider of the service.

Type Parameters

`T`

The type of the service.

Exceptions

ArgumentNullException

If `provider` is

`null`

.

GetService(Type)

Get a service provider for the service of the specified type.

```
public object GetService(Type type)
```

Parameters

`type` Type

The type of the service.

Returns

object

A service provider for the service of the specified type or

`null`

if no suitable service provider is registered in this container.

Exceptions

ArgumentNullException

If the specified type is

`null`

.

GetService<T>()

Get a service provider of the specified type.

```
public T GetService<T>() where T : class
```

Returns

T

A service provider of the specified type or

`null`

if no suitable service provider is registered in this container.

Type Parameters

`T`

The type of the service provider.

RemoveService(Type)

Remove the service with the specified type. Does nothing no service of the specified type is registered.

```
public void RemoveService(Type type)
```

Parameters

`type` Type

The type of the service to remove.

Exceptions

ArgumentNullException

If the specified type is

`null`

.

Class GameTime

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Holds the time state of a Game.

```
public class GameTime
```

Inheritance

object ← GameTime

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GameTime()

Create a GameTime instance with a TotalGameTime and ElapsedGameTime of

```
0
```

.

```
public GameTime()
```

GameTime(TimeSpan, TimeSpan)

Create a GameTime with the specified TotalGameTime and ElapsedGameTime.

```
public GameTime(TimeSpan totalGameTime, TimeSpan elapsedGameTime)
```

Parameters

`totalGameTime` TimeSpan

The total game time elapsed since the start of the Game.

`elapsedGameTime` TimeSpan

The time elapsed since the last call to Update(GameTime).

GameTime(TimeSpan, TimeSpan, bool)

Create a GameTime with the specified TotalGameTime and ElapsedGameTime.

```
public GameTime(TimeSpan totalRealTime, TimeSpan elapsedRealTime,
bool isRunningSlowly)
```

Parameters

`totalRealTime` TimeSpan

The total game time elapsed since the start of the Game.

`elapsedRealTime` TimeSpan

The time elapsed since the last call to Update(GameTime).

`isRunningSlowly` bool

A value indicating if the Game is running slowly.

Properties

ElapsedGameTime

Time since the last call to Update(GameTime).

```
public TimeSpan ElapsedGameTime { get; set; }
```

Property Value

TimeSpan

IsRunningSlowly

Indicates whether the Game is running slowly.

This flag is set to `true` when IsFixedTimeStep is set to `true` and a tick of the game loop takes longer than TargetElapsedTime for a few frames in a row.

```
public bool IsRunningSlowly { get; set; }
```

Property Value

bool

TotalGameTime

Time since the start of the Game.

```
public TimeSpan TotalGameTime { get; set; }
```

Property Value

TimeSpan

Class GameWindow

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

The system window used by a Game.

```
public abstract class GameWindow
```

Inheritance

object ← GameWindow

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GameWindow()

Create a GameWindow.

```
protected GameWindow()
```

Properties

AllowAltF4

Gets or sets a bool that enables usage of Alt+F4 for window closing on desktop platforms. Value is true by default.

```
public virtual bool AllowAltF4 { get; set; }
```

Property Value

bool

AllowUserResizing

Indicates if users can resize this GameWindow.

```
public abstract bool AllowUserResizing { get; set; }
```

Property Value

bool

ClientBounds

The client rectangle of the GameWindow.

```
public abstract Rectangle ClientBounds { get; }
```

Property Value

Rectangle

CurrentOrientation

The display orientation on a mobile device.

```
public abstract DisplayOrientation CurrentOrientation { get; }
```

Property Value

DisplayOrientation

Handle

The handle to the window used by the backend windowing service.

For WindowsDX this is the Win32 window handle (HWND). For DesktopGL this is the SDL window handle.

```
public abstract nint Handle { get; }
```

Property Value

nint

IsBorderless

Determines whether the border of the window is visible. Currently only supported on the WindowsDX and DesktopGL platforms.

```
public virtual bool IsBorderless { get; set; }
```

Property Value

bool

Exceptions

NotImplementedException

> Thrown when trying to use this property on a platform other than WinowsDX or DesktopGL.

Position

The location of this window on the desktop, eg: global coordinate space which stretches across all screens.

May be zero on platforms where it is not supported.

```
public abstract Point Position { get; set; }
```

Property Value

Point

ScreenDeviceName

The name of the screen the window is currently on.

```
public abstract string ScreenDeviceName { get; }
```

Property Value

string

Title

Gets or sets the title of the game window.

```
public string Title { get; set; }
```

Property Value

string

Methods

BeginScreenDeviceChange(bool)

Called before a game switches from windowed to full screen mode or vice versa.

```
public abstract void BeginScreenDeviceChange(bool willBeFullScreen)
```

Parameters

`willBeFullScreen` bool

Indicates what mode the game will switch to.

EndScreenDeviceChange(string)

Called when a transition from windowed to full screen or vice versa ends, or when the GraphicsDevice is reset.

```
public void EndScreenDeviceChange(string screenDeviceName)
```

Parameters

`screenDeviceName` string

Name of the screen to move the window to.

EndScreenDeviceChange(string, int, int)

Called when a transition from windowed to full screen or vice versa ends, or when the GraphicsDevice is reset.

```
public abstract void EndScreenDeviceChange(string screenDeviceName, int clientWidth,
int clientHeight)
```

Parameters

`screenDeviceName` string

Name of the screen to move the window to.

`clientWidth` int

The new width of the client rectangle.

`clientHeight` int

The new height of the client rectangle.

OnActivated()

Called when the window gains focus.

```
protected void OnActivated()
```

OnDeactivated()

Called when the window loses focus.

```
protected void OnDeactivated()
```

OnOrientationChanged()

Called when CurrentOrientation changed. Raises the OnOrientationChanged() event.

```
protected void OnOrientationChanged()
```

OnPaint()

Called when the window needs to be painted.

```
protected void OnPaint()
```

OnScreenDeviceNameChanged()

Called when ScreenDeviceName changed. Raises the ScreenDeviceNameChanged event.

```
protected void OnScreenDeviceNameChanged()
```

SetSupportedOrientations(DisplayOrientation)

Sets the supported display orientations.

```
protected abstract void SetSupportedOrientations(DisplayOrientation orientations)
```

Parameters

`orientations` DisplayOrientation

Supported display orientations

SetTitle(string)

Set the title of this window to the given string.

```
protected abstract void SetTitle(string title)
```

Parameters

`title` string

The new title of the window.

Events

ClientSizeChanged

Raised when the user resized the window or the window switches from fullscreen mode to windowed mode or vice versa.

```
public event EventHandler<EventArgs> ClientSizeChanged
```

Event Type

EventHandler<EventArgs>

FileDrop

This event is raised when user drops a file into the game window

```
public event EventHandler<FileDropEventArgs> FileDrop
```

Event Type

EventHandler<FileDropEventArgs>

Remarks

This event is only supported on desktop platforms.

KeyDown

Buffered keyboard KeyDown event.

```
public event EventHandler<InputKeyEventArgs> KeyDown
```

Event Type

EventHandler<InputKeyEventArgs>

KeyUp

Buffered keyboard KeyUp event.

```
public event EventHandler<InputKeyEventArgs> KeyUp
```

Event Type

EventHandler<InputKeyEventArgs>

OrientationChanged

Raised when CurrentOrientation changed.

```
public event EventHandler<EventArgs> OrientationChanged
```

Event Type

EventHandler<EventArgs>

ScreenDeviceNameChanged

Raised when ScreenDeviceName changed.

```
public event EventHandler<EventArgs> ScreenDeviceNameChanged
```

Event Type

EventHandler<EventArgs>

TextInput

Use this event to user text input.

This event is not raised by noncharacter keys except control characters such as backspace, tab, carriage return and escape. This event also supports key repeat.

```
public event EventHandler<TextInputEventArgs> TextInput
```

Event Type

EventHandler<TextInputEventArgs>

Remarks

This event is only supported on desktop platforms.

Class GraphicsDeviceInformation

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

The settings used in creation of the graphics device. See PreparingDeviceSettings.

```
public class GraphicsDeviceInformation
```

Inheritance

object ← GraphicsDeviceInformation

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Adapter

The graphics adapter on which the graphics device will be created.

```
public GraphicsAdapter Adapter { get; set; }
```

Property Value

GraphicsAdapter

Remarks

This is only valid on desktop systems where multiple graphics adapters are possible. Defaults to Default Adapter.

GraphicsProfile

The requested graphics device feature set.

```
public GraphicsProfile GraphicsProfile { get; set; }
```

Property Value

GraphicsProfile

PresentationParameters

The settings that define how graphics will be presented to the display.

```
public PresentationParameters PresentationParameters { get; set; }
```

Property Value

PresentationParameters

Class GraphicsDeviceManager

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Used to initialize and control the presentation of the graphics device.

```
public class GraphicsDeviceManager : IGraphicsDeviceService,
IDisposable, IGraphicsDeviceManager
```

Inheritance

object ← GraphicsDeviceManager

Implements

IGraphicsDeviceService, IDisposable, IGraphicsDeviceManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GraphicsDeviceManager(Game)

Associates this graphics device manager to a game instances.

```
public GraphicsDeviceManager(Game game)
```

Parameters

`game` Game

The game instance to attach.

Fields

DefaultBackBufferHeight

The default back buffer height.

```
public static readonly int DefaultBackBufferHeight
```

Field Value

int

DefaultBackBufferWidth

The default back buffer width.

```
public static readonly int DefaultBackBufferWidth
```

Field Value

int

Properties

GraphicsDevice

Returns the graphics device for this manager.

```
public GraphicsDevice GraphicsDevice { get; }
```

Property Value

GraphicsDevice

GraphicsProfile

The profile which determines the graphics feature level.

```
public GraphicsProfile GraphicsProfile { get; set; }
```

Property Value

GraphicsProfile

HardwareModeSwitch

Gets or sets the boolean which defines how window switches from windowed to fullscreen state. "Hard" mode(true) is slow to switch, but more effecient for performance, while "soft" mode(false) is vice versa. The default value is `true`.

```
public bool HardwareModeSwitch { get; set; }
```

Property Value

bool

IsFullScreen

Indicates the desire to switch into fullscreen mode.

```
public bool IsFullScreen { get; set; }
```

Property Value

bool

Remarks

When called at startup this will automatically set fullscreen mode during initialization. If set after startup you must call ApplyChanges() for the fullscreen mode to be changed. Note that for some platforms that do not support windowed modes this property has no affect.

PreferHalfPixelOffset

Indicates if DX9 style pixel addressing or current standard pixel addressing should be used. This flag is set to `false` by default. It should be set to `true` for XNA compatibility. It is recommended to leave this flag set to `false` for projects that are not ported from XNA. This value is passed to UseHalfPixelOffset.

```
public bool PreferHalfPixelOffset { get; set; }
```

Property Value

bool

Remarks

XNA uses DirectX9 for its graphics. DirectX9 interprets UV coordinates differently from other graphics API's. This is typically referred to as the half-pixel offset. MonoGame replicates XNA behavior if this flag is set to `true`.

PreferMultiSampling

Indicates the desire for a multisampled back buffer.

```
public bool PreferMultiSampling { get; set; }
```

Property Value

bool

Remarks

When called at startup this will automatically set the MSAA mode during initialization. If set after startup you must call ApplyChanges() for the MSAA mode to be changed.

PreferredBackBufferFormat

Indicates the desired back buffer color format.

```
public SurfaceFormat PreferredBackBufferFormat { get; set; }
```

Property Value

SurfaceFormat

Remarks

When called at startup this will automatically set the format during initialization. If set after startup you must call ApplyChanges() for the format to be changed.

PreferredBackBufferHeight

Indicates the desired back buffer height in pixels.

```
public int PreferredBackBufferHeight { get; set; }
```

Property Value

int

Remarks

When called at startup this will automatically set the height during initialization. If set after startup you must call ApplyChanges() for the height to be changed.

PreferredBackBufferWidth

Indicates the desired back buffer width in pixels.

```
public int PreferredBackBufferWidth { get; set; }
```

Property Value

int

Remarks

When called at startup this will automatically set the width during initialization. If set after startup you must call ApplyChanges() for the width to be changed.

PreferredDepthStencilFormat

Indicates the desired depth-stencil buffer format.

```
public DepthFormat PreferredDepthStencilFormat { get; set; }
```

Property Value

DepthFormat

Remarks

The depth-stencil buffer format defines the scene depth precision and stencil bits available for effects during rendering. When called at startup this will automatically set the format during initialization. If set after startup you must call ApplyChanges() for the format to be changed.

SupportedOrientations

Indicates the desired allowable display orientations when the device is rotated.

```
public DisplayOrientation SupportedOrientations { get; set; }
```

Property Value

DisplayOrientation

Remarks

This property only applies to mobile platforms with automatic display rotation. When called at startup this will automatically apply the supported orientations during initialization. If set after startup you must call ApplyChanges() for the supported orientations to be changed.

SynchronizeWithVerticalRetrace

Indicates the desire for vsync when presenting the back buffer.

```
public bool SynchronizeWithVerticalRetrace { get; set; }
```

Property Value

bool

Remarks

Vsync limits the frame rate of the game to the monitor referesh rate to prevent screen tearing. When called at startup this will automatically set the vsync mode during initialization. If set after startup you must call ApplyChanges() for the vsync mode to be changed. Depending on a user's video card settings, vsync settings can be ignored and cause unexpected behaviour in frame rate.

Methods

ApplyChanges()

Applies any pending property changes to the graphics device.

```
public void ApplyChanges()
```

BeginDraw()

Begins the drawing process.

```
public bool BeginDraw()
```

Returns

bool

true if the drawing process begins successfully; false otherwise.

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

EndDraw()

Ends the drawing process and calls Present() for the current graphics device.

```
public void EndDraw()
```

~GraphicsDeviceManager()

```
protected ~GraphicsDeviceManager()
```

OnDeviceCreated(EventArgs)

Called when a GraphicsDevice is created. Raises the DeviceCreated event.

```
protected void OnDeviceCreated(EventArgs e)
```

Parameters

e EventArgs

OnDeviceDisposing(EventArgs)

Called when a GraphicsDevice is disposed. Raises the DeviceDisposing event.

```
protected void OnDeviceDisposing(EventArgs e)
```

Parameters

e EventArgs

OnDeviceReset(EventArgs)

Called after a GraphicsDevice is reset. Raises the DeviceReset event.

```
protected void OnDeviceReset(EventArgs e)
```

Parameters

e EventArgs

OnDeviceResetting(EventArgs)

Called before a GraphicsDevice is reset. Raises the DeviceResetting event.

```
protected void OnDeviceResetting(EventArgs e)
```

Parameters

e EventArgs

ToggleFullScreen()

Toggles between windowed and fullscreen modes.

```
public void ToggleFullScreen()
```

Remarks

Note that on platforms that do not support windowed modes this has no affect.

Events

DeviceCreated

Raised when a new GraphicsDevice has been created.

```
public event EventHandler<EventArgs> DeviceCreated
```

Event Type

EventHandler<EventArgs>

DeviceDisposing

Raised when the GraphicsDevice is disposed.

```
public event EventHandler<EventArgs> DeviceDisposing
```

Event Type

EventHandler<EventArgs>

DeviceReset

Raised when the GraphicsDevice has reset.

```
public event EventHandler<EventArgs> DeviceReset
```

Event Type

EventHandler<EventArgs>

See Also
Reset()

DeviceResetting

Raised before the GraphicsDevice is resetting.

```
public event EventHandler<EventArgs> DeviceResetting
```

Event Type

EventHandler<EventArgs>

Disposed

Raised when this GraphicsDeviceManager is disposed.

```
public event EventHandler<EventArgs> Disposed
```

Event Type

EventHandler<EventArgs>

PreparingDeviceSettings

Raised by ApplyChanges(). Allows users to override the PresentationParameters to pass to the Graphics Device.

```
public event EventHandler<PreparingDeviceSettingsEventArgs> PreparingDeviceSettings
```

Event Type

EventHandler<PreparingDeviceSettingsEventArgs>

Interface ICurveEvaluator<T>

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Interface for curve evaluation. Implemented by Curve

```
public interface ICurveEvaluator<T>
```

Type Parameters

T

Interface for curve evaluation. Implemented by

Methods

Evaluate(float)

Evaluate the value at a position of this ICurveEvaluator<T>.

```
T Evaluate(float position)
```

Parameters

position float

The position on this ICurveEvaluator<T>.

Returns

T

Value at the position on this ICurveEvaluator<T>.

Interface IDrawable

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Interface for drawable entities.

```
public interface IDrawable
```

Properties

DrawOrder

The draw order of this IDrawable relative to other IDrawable instances.

```
int DrawOrder { get; }
```

Property Value

int

Visible

Indicates if Draw(GameTime) will be called.

```
bool Visible { get; }
```

Property Value

bool

Methods

Draw(GameTime)

Called when this IDrawable should draw itself.

```
void Draw(GameTime gameTime)
```

Parameters

`gameTime` GameTime

The elapsed time since the last call to Draw(GameTime).

Events

DrawOrderChanged

Raised when DrawOrder changed.

```
event EventHandler<EventArgs> DrawOrderChanged
```

Event Type

EventHandler<EventArgs>

VisibleChanged

Raised when Visible changed.

```
event EventHandler<EventArgs> VisibleChanged
```

Event Type

EventHandler<EventArgs>

Interface IGameComponent

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

An interface for GameComponent.

```
public interface IGameComponent
```

Methods

Initialize()

Initializes the component. Used to load non-graphical resources.

```
void Initialize()
```

Interface IGraphicsDeviceManager

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Used by the platform code to control the graphics device.

```
public interface IGraphicsDeviceManager
```

Methods

BeginDraw()

Called at the start of rendering a frame.

```
bool BeginDraw()
```

Returns

bool

Returns true if the frame should be rendered.

CreateDevice()

Called to create the graphics device.

```
void CreateDevice()
```

Remarks

Does nothing if the graphics device is already created.

EndDraw()

Called after rendering to present the frame to the screen.

```
void EndDraw()
```

Interface IPlatformBackButton

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Allows for platform specific handling of the Back button.

```
public interface IPlatformBackButton
```

Methods

Handled()

Return true if your game has handled the back button event return false if you want the operating system to handle it.

```
bool Handled()
```

Returns

bool

Interface IUpdateable

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Interface for updateable entities.

```
public interface IUpdateable
```

Properties

Enabled

Indicates if Update(GameTime) will be called.

```
bool Enabled { get; }
```

Property Value

bool

UpdateOrder

The update order of this GameComponent relative to other GameComponent instances.

```
int UpdateOrder { get; }
```

Property Value

int

Methods

Update(GameTime)

Called when this IUpdateable should update itself.

```
void Update(GameTime gameTime)
```

Parameters

`gameTime` GameTime

The elapsed time since the last call to Update(GameTime).

Events

EnabledChanged

Raised when Enabled changed.

```
event EventHandler<EventArgs> EnabledChanged
```

Event Type

EventHandler<EventArgs>

UpdateOrderChanged

Raised when UpdateOrder changed.

```
event EventHandler<EventArgs> UpdateOrderChanged
```

Event Type

EventHandler<EventArgs>

Struct InputKeyEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Provides data for the KeyUp and KeyDown events.

```
public struct InputKeyEventArgs
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

InputKeyEventArgs(Keys)

Create a new keyboard input event

```
public InputKeyEventArgs(Keys key = Keys.None)
```

Parameters

`key` Keys

The key involved in this event

Fields

Key

The key that was either pressed or released.

```
public readonly Keys Key
```

Field Value

Keys

Class LaunchParameters

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

The parameters for launching a Game.

```
public class LaunchParameters : Dictionary<string, string>, IDictionary<string,
string>, ICollection<KeyValuePair<string, string>>, IReadOnlyDictionary<string,
string>, IReadOnlyCollection<KeyValuePair<string, string>>,
IEnumerable<KeyValuePair<string, string>>, IDictionary, ICollection, IEnumerable,
IDeserializationCallback, ISerializable
```

Inheritance

object ← Dictionary<string, string> ← LaunchParameters

Implements

IDictionary<string, string>, ICollection<KeyValuePair<string, string>>, IReadOnlyDictionary<string, string>, IReadOnlyCollection<KeyValuePair<string, string>>, IEnumerable<KeyValuePair<string, string>>, IDictionary, ICollection, IEnumerable, IDeserializationCallback, ISerializable

Inherited Members

Dictionary<string, string>.Add(string, string) , Dictionary<string, string>.Clear() , Dictionary<string, string>.ContainsKey(string) , Dictionary<string, string>.ContainsValue(string) , Dictionary<string, string>.EnsureCapacity(int) , Dictionary<string, string>.GetEnumerator() , Dictionary<string, string>.OnDeserialization(object) , Dictionary<string, string>.Remove(string) , Dictionary<string, string>.Remove(string, out string) , Dictionary<string, string>.TrimExcess() , Dictionary<string, string>.TrimExcess(int) , Dictionary<string, string>.TryAdd(string, string) , Dictionary<string, string>.TryGetValue(string, out string) , Dictionary<string, string>.Comparer , Dictionary<string, string>.Count , Dictionary<string, string>.this[string] , Dictionary<string, string>.Keys , Dictionary<string, string>.Values , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class MathHelper

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Contains commonly used precalculated values and mathematical operations.

```
public static class MathHelper
```

Inheritance

object ← MathHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

E

Represents the mathematical constant e(2.71828175).

```
public const float E = 2.7182817
```

Field Value

float

Log10E

Represents the log base ten of e(0.4342945).

```
public const float Log10E = 0.4342945
```

Field Value

float

Log2E

Represents the log base two of e(1.442695).

```
public const float Log2E = 1.442695
```

Field Value

float

Pi

Represents the value of pi(3.14159274).

```
public const float Pi = 3.1415927
```

Field Value

float

PiOver2

Represents the value of pi divided by two(1.57079637).

```
public const float PiOver2 = 1.5707964
```

Field Value

float

PiOver4

Represents the value of pi divided by four(0.7853982).

```
public const float PiOver4 = 0.7853982
```

Field Value

float

Tau

Represents the value of pi times two(6.28318548). This is an alias of TwoPi.

```
public const float Tau = 6.2831855
```

Field Value

float

TwoPi

Represents the value of pi times two(6.28318548).

```
public const float TwoPi = 6.2831855
```

Field Value

float

Methods

Barycentric(float, float, float, float, float)

Returns the Cartesian coordinate for one axis of a point that is defined by a given triangle and two normalized barycentric (areal) coordinates.

```
public static float Barycentric(float value1, float value2, float value3, float
amount1, float amount2)
```

Parameters

`value1` float

The coordinate on one axis of vertex 1 of the defining triangle.

`value2` float

The coordinate on the same axis of vertex 2 of the defining triangle.

`value3` float

The coordinate on the same axis of vertex 3 of the defining triangle.

`amount1` float

The normalized barycentric (areal) coordinate b2, equal to the weighting factor for vertex 2, the coordinate of which is specified in value2.

`amount2` float

The normalized barycentric (areal) coordinate b3, equal to the weighting factor for vertex 3, the coordinate of which is specified in value3.

Returns

float

Cartesian coordinate of the specified point with respect to the axis being used.

CatmullRom(float, float, float, float, float)

Performs a Catmull-Rom interpolation using the specified positions.

```
public static float CatmullRom(float value1, float value2, float value3, float
value4, float amount)
```

Parameters

`value1` float

The first position in the interpolation.

`value2` float

The second position in the interpolation.

`value3` float

The third position in the interpolation.

`value4` float

The fourth position in the interpolation.

`amount` float

Weighting factor.

Returns

float

A position that is the result of the Catmull-Rom interpolation.

Clamp(int, int, int)

Restricts a value to be within a specified range.

```
public static int Clamp(int value, int min, int max)
```

Parameters

`value` int

The value to clamp.

`min` int

The minimum value. If `value` is less than `min`, `min` will be returned.

`max` int

The maximum value. If `value` is greater than `max`, `max` will be returned.

Returns

int

The clamped value.

Clamp(float, float, float)

Restricts a value to be within a specified range.

```
public static float Clamp(float value, float min, float max)
```

Parameters

`value` float

The value to clamp.

`min` float

The minimum value. If `value` is less than `min`, `min` will be returned.

`max` float

The maximum value. If `value` is greater than `max`, `max` will be returned.

Returns

float

The clamped value.

Distance(float, float)

Calculates the absolute value of the difference of two values.

```
public static float Distance(float value1, float value2)
```

Parameters

`value1` float

Source value.

`value2` float

Source value.

Returns

float

Distance between the two values.

Hermite(float, float, float, float, float)

Performs a Hermite spline interpolation.

```
public static float Hermite(float value1, float tangent1, float value2, float
tangent2, float amount)
```

Parameters

`value1` float

Source position.

`tangent1` float

Source tangent.

`value2` float

Source position.

`tangent2` float

Source tangent.

`amount` float

Weighting factor.

Returns

float

The result of the Hermite spline interpolation.

IsPowerOfTwo(int)

Determines if value is powered by two.

```
public static bool IsPowerOfTwo(int value)
```

Parameters

`value` int

A value.

Returns

bool

`true` if `value` is powered by two; otherwise `false`.

Lerp(float, float, float)

Linearly interpolates between two values.

```
public static float Lerp(float value1, float value2, float amount)
```

Parameters

`value1` float

Source value.

`value2` float

Destination value.

`amount` float

Value between 0 and 1 indicating the weight of value2.

Returns

float

Interpolated value.

Remarks

This method performs the linear interpolation based on the following formula:

```
value1 + (value2 - value1) * amount
```

. Passing amount a value of 0 will cause value1 to be returned, a value of 1 will cause value2 to be returned. See LerpPrecise(float, float, float) for a less efficient version with more precision around edge cases.

LerpPrecise(float, float, float)

Linearly interpolates between two values. This method is a less efficient, more precise version of Lerp(float, float, float). See remarks for more info.

```
public static float LerpPrecise(float value1, float value2, float amount)
```

Parameters

`value1` float

Source value.

`value2` float

Destination value.

`amount` float

Value between 0 and 1 indicating the weight of value2.

Returns

float

Interpolated value.

Remarks

This method performs the linear interpolation based on the following formula:

```
((1 - amount) * value1) + (value2 * amount)
```

. Passing amount a value of 0 will cause value1 to be returned, a value of 1 will cause value2 to be returned. This method does not have the floating point precision issue that Lerp(float, float, float) has. i.e. If there is a big gap between value1 and value2 in magnitude (e.g. value1=10000000000000000, value2=1), right at the edge of the interpolation range (amount=1), Lerp(float, float, float) will return 0 (whereas it should return 1). This also holds for value1=10^17, value2=10; value1=10^18,value2=10^2... so on. For an in depth explanation of the issue, see below references: Relevant Wikipedia Article: https://en.wikipedia.org/wiki/Linear_interpolation#Programming_language_support Relevant StackOverflow Answer: http://stackoverflow.com/questions/4353525/floating-point-linear-interpolation#answer-23716956

Max(int, int)

Returns the greater of two values.

```
public static int Max(int value1, int value2)
```

Parameters

`value1` int

Source value.

`value2` int

Source value.

Returns

int

The greater value.

Max(float, float)

Returns the greater of two values.

```
public static float Max(float value1, float value2)
```

Parameters

`value1` float

Source value.

`value2` float

Source value.

Returns

float

The greater value.

Min(int, int)

Returns the lesser of two values.

```
public static int Min(int value1, int value2)
```

Parameters

`value1` int

Source value.

`value2` int

Source value.

Returns

int

The lesser value.

Min(float, float)

Returns the lesser of two values.

```
public static float Min(float value1, float value2)
```

Parameters

`value1` float

Source value.

`value2` float

Source value.

Returns

float

The lesser value.

SmoothStep(float, float, float)

Interpolates between two values using a cubic equation.

```
public static float SmoothStep(float value1, float value2, float amount)
```

Parameters

`value1` float

Source value.

`value2` float

Source value.

`amount` float

Weighting value.

Returns

float

Interpolated value.

ToDegrees(float)

Converts radians to degrees.

```
public static float ToDegrees(float radians)
```

Parameters

`radians` float

The angle in radians.

Returns

float

The angle in degrees.

Remarks

This method uses double precision internally, though it returns single float Factor = 180 / pi

ToRadians(float)

Converts degrees to radians.

```
public static float ToRadians(float degrees)
```

Parameters

`degrees` float

The angle in degrees.

Returns

float

The angle in radians.

Remarks

This method uses double precision internally, though it returns single float Factor = pi / 180

WrapAngle(float)

Reduces a given angle to a value between π and -π.

```
public static float WrapAngle(float angle)
```

Parameters

`angle` float

The angle to reduce, in radians.

Returns

float

The new angle, in radians.

Class MathHelper.Random

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Random Number Generator based on Permutation-Congruential-Generator (PCG), which is a fancy wording to describe a family of RNG which are simple, fast, statistically excellent, and hardly predictable.

More interestingly, PCG allows to generate multiple sequences with the same seed, which is very handy in game development to have a unique seed per game session while using different streams for each RNG which requires an isolated context (e.g. generating a procedural level, but we don't want the loot generation to affect subsequent level generations).

https://www.pcg-random.org/

This code is derived from the minimal C implementation. https://github.com/imneme/pcg-c-basic

```
public class MathHelper.Random
```

Inheritance

object ← MathHelper.Random

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Random()

Initialize a random number generator.

```
public Random()
```

Random(ulong, ulong)

Initialize a random number generator. Specified in two parts, state initializer (a.k.a. seed) and a sequence selection constant (a.k.a. stream id).

```
public Random(ulong state, ulong streamID = 0)
```

Parameters

`state` ulong

State initializer (a.k.a. seed).

`streamID` ulong

Sequence selection constant (a.k.a. stream id). Defaults to 0.

Properties

State

State of the random number generator.

```
public ulong State { get; set; }
```

Property Value

ulong

Methods

Next()

Generate a random positive number.

```
public int Next()
```

Returns

int

A random positive number.

Next(int)

Generate a random number with an exclusive `maxValue` upper bound.

```
public int Next(int maxValue)
```

Parameters

`maxValue` int

Exclusive upper bound.

Returns

int

A random number with an exclusive `maxValue` upper bound.

Next(int, int)

Generate a random number ranging from `minValue` to `maxValue`.

```
public int Next(int minValue, int maxValue)
```

Parameters

`minValue` int

Lower bound.

`maxValue` int

Upper bound.

Returns

int

A random number ranging from `minValue` to `maxValue`.

NextBool()

Generate a random bool.

```
public bool NextBool()
```

Returns

bool

A random bool.

NextFloat()

Generate a random float ranging from 0.0f to 1.0f.

```
public float NextFloat()
```

Returns

float

A random float ranging from 0.0f to 1.0f.

NextFloat(float, float)

Generate a random float ranging from `minValue` to `maxValue`.

```
public float NextFloat(float minValue, float maxValue)
```

Parameters

`minValue` float

Lower bound. MUST be greater than zero.

`maxValue` float

Upper bound. MUST be greater than zero.

Returns

float

A random float ranging from `minValue` to `maxValue`.

Struct Matrix

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Represents the right-handed 4x4 floating point matrix, which can store translation, scale and rotation information.

```
[DataContract]
public struct Matrix : IEquatable<Matrix>
```

Implements

IEquatable<Matrix>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Matrix(Vector4, Vector4, Vector4, Vector4)

Constructs a matrix.

```
public Matrix(Vector4 row1, Vector4 row2, Vector4 row3, Vector4 row4)
```

Parameters

`row1` Vector4

A first row of the created matrix.

`row2` Vector4

A second row of the created matrix.

`row3` Vector4

A third row of the created matrix.

`row4` Vector4

A fourth row of the created matrix.

Matrix(float, float, float, float, float, float, float, float, float, float, float, float, float, float, float, float)

Constructs a matrix.

```
public Matrix(float m11, float m12, float m13, float m14, float m21, float m22,
float m23, float m24, float m31, float m32, float m33, float m34, float m41, float
m42, float m43, float m44)
```

Parameters

`m11` float

A first row and first column value.

`m12` float

A first row and second column value.

`m13` float

A first row and third column value.

`m14` float

A first row and fourth column value.

`m21` float

A second row and first column value.

`m22` float

A second row and second column value.

`m23` float

A second row and third column value.

`m24` float

A second row and fourth column value.

`m31` float

A third row and first column value.

`m32` float

A third row and second column value.

`m33` float

A third row and third column value.

`m34` float

A third row and fourth column value.

`m41` float

A fourth row and first column value.

`m42` float

A fourth row and second column value.

`m43` float

A fourth row and third column value.

`m44` float

A fourth row and fourth column value.

Fields

M11

A first row and first column value.

```
[DataMember]
public float M11
```

Field Value

float

M12

A first row and second column value.

```
[DataMember]
public float M12
```

Field Value

float

M13

A first row and third column value.

```
[DataMember]
public float M13
```

Field Value

float

M14

A first row and fourth column value.

```
[DataMember]
public float M14
```

Field Value

float

M21

A second row and first column value.

```
[DataMember]
public float M21
```

Field Value

float

M22

A second row and second column value.

```
[DataMember]
public float M22
```

Field Value

float

M23

A second row and third column value.

```
[DataMember]
public float M23
```

Field Value

float

M24

A second row and fourth column value.

```
[DataMember]
public float M24
```

Field Value

float

M31

A third row and first column value.

```
[DataMember]
public float M31
```

Field Value

float

M32

A third row and second column value.

```
[DataMember]
public float M32
```

Field Value

float

M33

A third row and third column value.

```
[DataMember]
public float M33
```

Field Value

float

M34

A third row and fourth column value.

```
[DataMember]
public float M34
```

Field Value

float

M41

A fourth row and first column value.

```
[DataMember]
public float M41
```

Field Value

float

M42

A fourth row and second column value.

```
[DataMember]
public float M42
```

Field Value

float

M43

A fourth row and third column value.

```
[DataMember]
public float M43
```

Field Value

float

M44

A fourth row and fourth column value.

```
[DataMember]
public float M44
```

Field Value

float

Properties

Backward

The backward vector formed from the third row M31, M32, M33 elements.

```
public Vector3 Backward { get; set; }
```

Property Value

Vector3

Down

The down vector formed from the second row -M21, -M22, -M23 elements.

```
public Vector3 Down { get; set; }
```

Property Value

Vector3

Forward

The forward vector formed from the third row -M31, -M32, -M33 elements.

```
public Vector3 Forward { get; set; }
```

Property Value

Vector3

Identity

Returns the identity matrix.

```
public static Matrix Identity { get; }
```

Property Value

Matrix

this[int]

Get or set the matrix element at the given index, indexed in row major order.

```
public float this[int index] { get; set; }
```

Parameters

`index` int

The linearized, zero-based index of the matrix element.

Property Value

float

Exceptions

ArgumentOutOfRangeException

If the index is less than

`0`

or larger than

`15`

.

this[int, int]

Get or set the value at the specified row and column (indices are zero-based).

```
public float this[int row, int column] { get; set; }
```

Parameters

`row` int

The row of the element.

`column` int

The column of the element.

Property Value

float

Exceptions

ArgumentOutOfRangeException

If the row or column is less than

```
0
```

or larger than

```
3
```

.

Left

The left vector formed from the first row -M11, -M12, -M13 elements.

```
public Vector3 Left { get; set; }
```

Property Value

Vector3

Right

The right vector formed from the first row M11, M12, M13 elements.

```
public Vector3 Right { get; set; }
```

Property Value

Vector3

Translation

Position stored in this matrix.

```
public Vector3 Translation { get; set; }
```

Property Value

Vector3

Up

The upper vector formed from the second row M21, M22, M23 elements.

```
public Vector3 Up { get; set; }
```

Property Value

Vector3

Methods

Add(Matrix, Matrix)

Creates a new Matrix which contains sum of two matrixes.

```
public static Matrix Add(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

The first matrix to add.

`matrix2` Matrix

The second matrix to add.

Returns

Matrix

The result of the matrix addition.

Add(ref Matrix, ref Matrix, out Matrix)

Creates a new Matrix which contains sum of two matrixes.

```
public static void Add(ref Matrix matrix1, ref Matrix matrix2, out Matrix result)
```

Parameters

`matrix1` Matrix

The first matrix to add.

`matrix2` Matrix

The second matrix to add.

`result` Matrix

The result of the matrix addition as an output parameter.

CreateBillboard(Vector3, Vector3, Vector3, Vector3?)

Creates a new Matrix for spherical billboarding that rotates around specified object position.

```
public static Matrix CreateBillboard(Vector3 objectPosition, Vector3 cameraPosition,
Vector3 cameraUpVector, Vector3? cameraForwardVector)
```

Parameters

`objectPosition` Vector3

Position of billboard object. It will rotate around that vector.

`cameraPosition` Vector3

The camera position.

`cameraUpVector` Vector3

The camera up vector.

`cameraForwardVector` Vector3?

Optional camera forward vector.

Returns

Matrix

The Matrix for spherical billboarding.

CreateBillboard(ref Vector3, ref Vector3, ref Vector3, Vector3?, out Matrix)

Creates a new Matrix for spherical billboarding that rotates around specified object position.

```
public static void CreateBillboard(ref Vector3 objectPosition, ref Vector3
cameraPosition, ref Vector3 cameraUpVector, Vector3? cameraForwardVector, out
Matrix result)
```

Parameters

`objectPosition` Vector3

Position of billboard object. It will rotate around that vector.

`cameraPosition` Vector3

The camera position.

`cameraUpVector` Vector3

The camera up vector.

`cameraForwardVector` Vector3?

Optional camera forward vector.

`result` Matrix

The Matrix for spherical billboarding as an output parameter.

CreateConstrainedBillboard(Vector3, Vector3, Vector3, Vector3?, Vector3?)

Creates a new Matrix for cylindrical billboarding that rotates around specified axis.

```
public static Matrix CreateConstrainedBillboard(Vector3 objectPosition,
Vector3 cameraPosition, Vector3 rotateAxis, Vector3? cameraForwardVector,
Vector3? objectForwardVector)
```

Parameters

`objectPosition` Vector3

Object position the billboard will rotate around.

`cameraPosition` Vector3

Camera position.

`rotateAxis` Vector3

Axis of billboard for rotation.

`cameraForwardVector` Vector3?

Optional camera forward vector.

`objectForwardVector` Vector3?

Optional object forward vector.

Returns

Matrix

The Matrix for cylindrical billboarding.

CreateConstrainedBillboard(ref Vector3, ref Vector3, ref Vector3, Vector3?, Vector3?, out Matrix)

Creates a new Matrix for cylindrical billboarding that rotates around specified axis.

```
public static void CreateConstrainedBillboard(ref Vector3 objectPosition, ref
Vector3 cameraPosition, ref Vector3 rotateAxis, Vector3? cameraForwardVector,
Vector3? objectForwardVector, out Matrix result)
```

Parameters

`objectPosition` Vector3

Object position the billboard will rotate around.

`cameraPosition` Vector3

Camera position.

`rotateAxis` Vector3

Axis of billboard for rotation.

`cameraForwardVector` Vector3?

Optional camera forward vector.

`objectForwardVector` Vector3?

Optional object forward vector.

`result` Matrix

The Matrix for cylindrical billboarding as an output parameter.

CreateFromAxisAngle(Vector3, float)

Creates a new Matrix which contains the rotation moment around specified axis.

```
public static Matrix CreateFromAxisAngle(Vector3 axis, float angle)
```

Parameters

`axis` Vector3

The axis of rotation.

`angle` float

The angle of rotation in radians.

Returns

Matrix

The rotation Matrix.

CreateFromAxisAngle(ref Vector3, float, out Matrix)

Creates a new Matrix which contains the rotation moment around specified axis.

```
public static void CreateFromAxisAngle(ref Vector3 axis, float angle, out
Matrix result)
```

Parameters

`axis` Vector3

The axis of rotation.

`angle` float

The angle of rotation in radians.

`result` Matrix

The rotation Matrix as an output parameter.

CreateFromQuaternion(Quaternion)

Creates a new rotation Matrix from a Quaternion.

```
public static Matrix CreateFromQuaternion(Quaternion quaternion)
```

Parameters

`quaternion` Quaternion

Quaternion of rotation moment.

Returns

Matrix

The rotation Matrix.

CreateFromQuaternion(ref Quaternion, out Matrix)

Creates a new rotation Matrix from a Quaternion.

```
public static void CreateFromQuaternion(ref Quaternion quaternion, out
Matrix result)
```

Parameters

`quaternion` Quaternion

Quaternion of rotation moment.

`result` Matrix

The rotation Matrix as an output parameter.

CreateFromYawPitchRoll(float, float, float)

Creates a new rotation Matrix from the specified yaw, pitch and roll values.

```
public static Matrix CreateFromYawPitchRoll(float yaw, float pitch, float roll)
```

Parameters

`yaw` float

The yaw rotation value in radians.

`pitch` float

The pitch rotation value in radians.

`roll` float

The roll rotation value in radians.

Returns

Matrix

The rotation Matrix.

Remarks

For more information about yaw, pitch and roll visit http://en.wikipedia.org/wiki/Euler_angles.

CreateFromYawPitchRoll(float, float, float, out Matrix)

Creates a new rotation Matrix from the specified yaw, pitch and roll values.

```
public static void CreateFromYawPitchRoll(float yaw, float pitch, float roll, out
Matrix result)
```

Parameters

`yaw` float

The yaw rotation value in radians.

`pitch` float

The pitch rotation value in radians.

`roll` float

The roll rotation value in radians.

`result` Matrix

The rotation Matrix as an output parameter.

Remarks

For more information about yaw, pitch and roll visit http://en.wikipedia.org/wiki/Euler_angles.

CreateLookAt(Vector3, Vector3, Vector3)

Creates a new viewing Matrix.

```
public static Matrix CreateLookAt(Vector3 cameraPosition, Vector3 cameraTarget,
Vector3 cameraUpVector)
```

Parameters

`cameraPosition` Vector3

Position of the camera.

`cameraTarget` Vector3

Lookup vector of the camera.

`cameraUpVector` Vector3

The direction of the upper edge of the camera.

Returns

Matrix

The viewing Matrix.

CreateLookAt(ref Vector3, ref Vector3, ref Vector3, out Matrix)

Creates a new viewing Matrix.

```
public static void CreateLookAt(ref Vector3 cameraPosition, ref Vector3
cameraTarget, ref Vector3 cameraUpVector, out Matrix result)
```

Parameters

`cameraPosition` Vector3

Position of the camera.

`cameraTarget` Vector3

Lookup vector of the camera.

`cameraUpVector` Vector3

The direction of the upper edge of the camera.

`result` Matrix

The viewing Matrix as an output parameter.

CreateOrthographic(float, float, float, float)

Creates a new projection Matrix for orthographic view.

```
public static Matrix CreateOrthographic(float width, float height, float zNearPlane,
float zFarPlane)
```

Parameters

`width` float

Width of the viewing volume.

`height` float

Height of the viewing volume.

`zNearPlane` float

Depth of the near plane.

`zFarPlane` float

Depth of the far plane.

Returns

Matrix

The new projection Matrix for orthographic view.

CreateOrthographic(float, float, float, float, out Matrix)

Creates a new projection Matrix for orthographic view.

```
public static void CreateOrthographic(float width, float height, float zNearPlane,
float zFarPlane, out Matrix result)
```

Parameters

`width` float

Width of the viewing volume.

`height` float

Height of the viewing volume.

`zNearPlane` float

Depth of the near plane.

`zFarPlane` float

Depth of the far plane.

`result` Matrix

The new projection Matrix for orthographic view as an output parameter.

CreateOrthographicOffCenter(Rectangle, float, float)

Creates a new projection Matrix for customized orthographic view.

```
public static Matrix CreateOrthographicOffCenter(Rectangle viewingVolume, float
zNearPlane, float zFarPlane)
```

Parameters

`viewingVolume` Rectangle

The viewing volume.

`zNearPlane` float

Depth of the near plane.

`zFarPlane` float

Depth of the far plane.

Returns

Matrix

The new projection Matrix for customized orthographic view.

CreateOrthographicOffCenter(float, float, float, float, float, float)

Creates a new projection Matrix for customized orthographic view.

```
public static Matrix CreateOrthographicOffCenter(float left, float right, float
bottom, float top, float zNearPlane, float zFarPlane)
```

Parameters

`left` float

Lower x-value at the near plane.

`right` float

Upper x-value at the near plane.

`bottom` float

Lower y-coordinate at the near plane.

`top` float

Upper y-value at the near plane.

`zNearPlane` float

Depth of the near plane.

`zFarPlane` float

Depth of the far plane.

Returns

Matrix

The new projection Matrix for customized orthographic view.

CreateOrthographicOffCenter(float, float, float, float, float, float, out Matrix)

Creates a new projection Matrix for customized orthographic view.

```
public static void CreateOrthographicOffCenter(float left, float right, float
bottom, float top, float zNearPlane, float zFarPlane, out Matrix result)
```

Parameters

`left` float

Lower x-value at the near plane.

`right` float

Upper x-value at the near plane.

`bottom` float

Lower y-coordinate at the near plane.

`top` float

Upper y-value at the near plane.

`zNearPlane` float

Depth of the near plane.

`zFarPlane` float

Depth of the far plane.

`result` Matrix

The new projection Matrix for customized orthographic view as an output parameter.

CreatePerspective(float, float, float, float)

Creates a new projection Matrix for perspective view.

```
public static Matrix CreatePerspective(float width, float height, float
nearPlaneDistance, float farPlaneDistance)
```

Parameters

`width` float

Width of the viewing volume.

`height` float

Height of the viewing volume.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane.

Returns

Matrix

The new projection Matrix for perspective view.

CreatePerspective(float, float, float, float, out Matrix)

Creates a new projection Matrix for perspective view.

```
public static void CreatePerspective(float width, float height, float
nearPlaneDistance, float farPlaneDistance, out Matrix result)
```

Parameters

`width` float

Width of the viewing volume.

`height` float

Height of the viewing volume.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane, or PositiveInfinity.

`result` Matrix

The new projection Matrix for perspective view as an output parameter.

CreatePerspectiveFieldOfView(float, float, float, float)

Creates a new projection Matrix for perspective view with field of view.

```
public static Matrix CreatePerspectiveFieldOfView(float fieldOfView, float
aspectRatio, float nearPlaneDistance, float farPlaneDistance)
```

Parameters

`fieldOfView` float

Field of view in the y direction in radians.

`aspectRatio` float

Width divided by height of the viewing volume.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane, or PositiveInfinity.

Returns

Matrix

The new projection Matrix for perspective view with FOV.

CreatePerspectiveFieldOfView(float, float, float, float, out Matrix)

Creates a new projection Matrix for perspective view with field of view.

```
public static void CreatePerspectiveFieldOfView(float fieldOfView, float
aspectRatio, float nearPlaneDistance, float farPlaneDistance, out Matrix result)
```

Parameters

`fieldOfView` float

Field of view in the y direction in radians.

`aspectRatio` float

Width divided by height of the viewing volume.

`nearPlaneDistance` float

Distance of the near plane.

`farPlaneDistance` float

Distance of the far plane, or PositiveInfinity.

`result` Matrix

The new projection Matrix for perspective view with FOV as an output parameter.

CreatePerspectiveOffCenter(Rectangle, float, float)

Creates a new projection Matrix for customized perspective view.

```
public static Matrix CreatePerspectiveOffCenter(Rectangle viewingVolume, float
nearPlaneDistance, float farPlaneDistance)
```

Parameters

`viewingVolume` Rectangle

The viewing volume.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane.

Returns

Matrix

The new Matrix for customized perspective view.

CreatePerspectiveOffCenter(float, float, float, float, float, float)

Creates a new projection Matrix for customized perspective view.

```
public static Matrix CreatePerspectiveOffCenter(float left, float right, float
bottom, float top, float nearPlaneDistance, float farPlaneDistance)
```

Parameters

`left` float

Lower x-value at the near plane.

`right` float

Upper x-value at the near plane.

`bottom` float

Lower y-coordinate at the near plane.

`top` float

Upper y-value at the near plane.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane.

Returns

Matrix

The new Matrix for customized perspective view.

CreatePerspectiveOffCenter(float, float, float, float, float, float, out Matrix)

Creates a new projection Matrix for customized perspective view.

```
public static void CreatePerspectiveOffCenter(float left, float right, float bottom,
float top, float nearPlaneDistance, float farPlaneDistance, out Matrix result)
```

Parameters

`left` float

Lower x-value at the near plane.

`right` float

Upper x-value at the near plane.

`bottom` float

Lower y-coordinate at the near plane.

`top` float

Upper y-value at the near plane.

`nearPlaneDistance` float

Distance to the near plane.

`farPlaneDistance` float

Distance to the far plane.

`result` Matrix

The new Matrix for customized perspective view as an output parameter.

CreateReflection(Plane)

Creates a new reflection Matrix.

```
public static Matrix CreateReflection(Plane value)
```

Parameters

`value` Plane

The plane that used for reflection calculation.

Returns

Matrix

The reflection Matrix.

CreateReflection(ref Plane, out Matrix)

Creates a new reflection Matrix.

```
public static void CreateReflection(ref Plane value, out Matrix result)
```

Parameters

`value` Plane

The plane that used for reflection calculation.

`result` Matrix

The reflection Matrix as an output parameter.

CreateRotationX(float)

Creates a new rotation Matrix around X axis.

```
public static Matrix CreateRotationX(float radians)
```

Parameters

`radians` float

Angle in radians.

Returns

Matrix

The rotation Matrix around X axis.

CreateRotationX(float, out Matrix)

Creates a new rotation Matrix around X axis.

```
public static void CreateRotationX(float radians, out Matrix result)
```

Parameters

`radians` float

Angle in radians.

`result` Matrix

The rotation Matrix around X axis as an output parameter.

CreateRotationY(float)

Creates a new rotation Matrix around Y axis.

```
public static Matrix CreateRotationY(float radians)
```

Parameters

`radians` float

Angle in radians.

Returns

Matrix

The rotation Matrix around Y axis.

CreateRotationY(float, out Matrix)

Creates a new rotation Matrix around Y axis.

```
public static void CreateRotationY(float radians, out Matrix result)
```

Parameters

`radians` float

Angle in radians.

`result` Matrix

The rotation Matrix around Y axis as an output parameter.

CreateRotationZ(float)

Creates a new rotation Matrix around Z axis.

```
public static Matrix CreateRotationZ(float radians)
```

Parameters

`radians` float

Angle in radians.

Returns

Matrix

The rotation Matrix around Z axis.

CreateRotationZ(float, out Matrix)

Creates a new rotation Matrix around Z axis.

```
public static void CreateRotationZ(float radians, out Matrix result)
```

Parameters

`radians` float

Angle in radians.

`result` Matrix

The rotation Matrix around Z axis as an output parameter.

CreateScale(Vector3)

Creates a new scaling Matrix.

```
public static Matrix CreateScale(Vector3 scales)
```

Parameters

`scales` Vector3

Vector3 representing x,y and z scale values.

Returns

Matrix

The scaling Matrix.

CreateScale(ref Vector3, out Matrix)

Creates a new scaling Matrix.

```
public static void CreateScale(ref Vector3 scales, out Matrix result)
```

Parameters

`scales` Vector3

Vector3 representing x,y and z scale values.

`result` Matrix

The scaling Matrix as an output parameter.

CreateScale(float)

Creates a new scaling Matrix.

```
public static Matrix CreateScale(float scale)
```

Parameters

`scale` float

Scale value for all three axises.

Returns

Matrix

The scaling Matrix.

CreateScale(float, out Matrix)

Creates a new scaling Matrix.

```
public static void CreateScale(float scale, out Matrix result)
```

Parameters

`scale` float

Scale value for all three axises.

`result` Matrix

The scaling Matrix as an output parameter.

CreateScale(float, float, float)

Creates a new scaling Matrix.

```
public static Matrix CreateScale(float xScale, float yScale, float zScale)
```

Parameters

`xScale` float

Scale value for X axis.

`yScale` float

Scale value for Y axis.

`zScale` float

Scale value for Z axis.

Returns

Matrix

The scaling Matrix.

CreateScale(float, float, float, out Matrix)

Creates a new scaling Matrix.

```
public static void CreateScale(float xScale, float yScale, float zScale, out
Matrix result)
```

Parameters

`xScale` float

Scale value for X axis.

`yScale` float

Scale value for Y axis.

`zScale` float

Scale value for Z axis.

`result` Matrix

The scaling Matrix as an output parameter.

CreateShadow(Vector3, Plane)

Creates a new Matrix that flattens geometry into a specified Plane as if casting a shadow from a specified light source.

```
public static Matrix CreateShadow(Vector3 lightDirection, Plane plane)
```

Parameters

`lightDirection` Vector3

A vector specifying the direction from which the light that will cast the shadow is coming.

`plane` Plane

The plane onto which the new matrix should flatten geometry so as to cast a shadow.

Returns

Matrix

A Matrix that can be used to flatten geometry onto the specified plane from the specified direction.

CreateShadow(ref Vector3, ref Plane, out Matrix)

Creates a new Matrix that flattens geometry into a specified Plane as if casting a shadow from a specified light source.

```
public static void CreateShadow(ref Vector3 lightDirection, ref Plane plane, out
Matrix result)
```

Parameters

`lightDirection` Vector3

A vector specifying the direction from which the light that will cast the shadow is coming.

`plane` Plane

The plane onto which the new matrix should flatten geometry so as to cast a shadow.

`result` Matrix

A Matrix that can be used to flatten geometry onto the specified plane from the specified direction as an output parameter.

CreateTranslation(Vector3)

Creates a new translation Matrix.

```
public static Matrix CreateTranslation(Vector3 position)
```

Parameters

`position` Vector3

X,Y and Z coordinates of translation.

Returns

Matrix

The translation Matrix.

CreateTranslation(ref Vector3, out Matrix)

Creates a new translation Matrix.

```
public static void CreateTranslation(ref Vector3 position, out Matrix result)
```

Parameters

`position` Vector3

X,Y and Z coordinates of translation.

`result` Matrix

The translation Matrix as an output parameter.

CreateTranslation(float, float, float)

Creates a new translation Matrix.

```
public static Matrix CreateTranslation(float xPosition, float yPosition,
float zPosition)
```

Parameters

`xPosition` float

X coordinate of translation.

`yPosition` float

Y coordinate of translation.

`zPosition` float

Z coordinate of translation.

Returns

Matrix

The translation Matrix.

CreateTranslation(float, float, float, out Matrix)

Creates a new translation Matrix.

```
public static void CreateTranslation(float xPosition, float yPosition, float
zPosition, out Matrix result)
```

Parameters

`xPosition` float

X coordinate of translation.

`yPosition` float

Y coordinate of translation.

`zPosition` float

Z coordinate of translation.

`result` Matrix

The translation Matrix as an output parameter.

CreateWorld(Vector3, Vector3, Vector3)

Creates a new world Matrix.

```
public static Matrix CreateWorld(Vector3 position, Vector3 forward, Vector3 up)
```

Parameters

`position` Vector3

The position vector.

`forward` Vector3

The forward direction vector.

`up` Vector3

The upward direction vector. Usually Up.

Returns

Matrix

The world Matrix.

CreateWorld(ref Vector3, ref Vector3, ref Vector3, out Matrix)

Creates a new world Matrix.

```
public static void CreateWorld(ref Vector3 position, ref Vector3 forward, ref
Vector3 up, out Matrix result)
```

Parameters

`position` Vector3

The position vector.

`forward` Vector3

The forward direction vector.

`up` Vector3

The upward direction vector. Usually Up.

`result` Matrix

The world Matrix as an output parameter.

Decompose(out Vector3, out Quaternion, out Vector3)

Decomposes this matrix to translation, rotation and scale elements. Returns `true` if matrix can be decomposed; `false` otherwise.

```
public bool Decompose(out Vector3 scale, out Quaternion rotation, out
Vector3 translation)
```

Parameters

`scale` Vector3

Scale vector as an output parameter.

`rotation` Quaternion

Rotation quaternion as an output parameter.

`translation` Vector3

Translation vector as an output parameter.

Returns

bool

`true` if matrix can be decomposed; `false` otherwise.

Determinant()

Returns a determinant of this Matrix.

```
public float Determinant()
```

Returns

float

Determinant of this Matrix

Remarks

See more about determinant here - http://en.wikipedia.org/wiki/Determinant.

Divide(Matrix, Matrix)

Divides the elements of a Matrix by the elements of another matrix.

```
public static Matrix Divide(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix.

`matrix2` Matrix

Divisor Matrix.

Returns

Matrix

The result of dividing the matrix.

Divide(Matrix, float)

Divides the elements of a Matrix by a scalar.

```
public static Matrix Divide(Matrix matrix1, float divider)
```

Parameters

`matrix1` Matrix

Source Matrix.

`divider` float

Divisor scalar.

Returns

Matrix

The result of dividing a matrix by a scalar.

Divide(ref Matrix, ref Matrix, out Matrix)

Divides the elements of a Matrix by the elements of another matrix.

```
public static void Divide(ref Matrix matrix1, ref Matrix matrix2, out Matrix result)
```

Parameters

`matrix1` Matrix

Source Matrix.

`matrix2` Matrix

Divisor Matrix.

`result` Matrix

The result of dividing the matrix as an output parameter.

Divide(ref Matrix, float, out Matrix)

Divides the elements of a Matrix by a scalar.

```
public static void Divide(ref Matrix matrix1, float divider, out Matrix result)
```

Parameters

`matrix1` Matrix

Source Matrix.

`divider` float

Divisor scalar.

`result` Matrix

The result of dividing a matrix by a scalar as an output parameter.

Equals(Matrix)

Compares whether current instance is equal to specified Matrix without any tolerance.

```
public bool Equals(Matrix other)
```

Parameters

`other` Matrix

The Matrix to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object without any tolerance.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetHashCode()

Gets the hash code of this Matrix.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Matrix.

Invert(Matrix)

Creates a new Matrix which contains inversion of the specified matrix.

```
public static Matrix Invert(Matrix matrix)
```

Parameters

`matrix` Matrix

Source Matrix.

Returns

Matrix

The inverted matrix.

Invert(ref Matrix, out Matrix)

Creates a new Matrix which contains inversion of the specified matrix.

```
public static void Invert(ref Matrix matrix, out Matrix result)
```

Parameters

`matrix` Matrix

Source Matrix.

`result` Matrix

The inverted matrix as output parameter.

Lerp(Matrix, Matrix, float)

Creates a new Matrix that contains linear interpolation of the values in specified matrixes.

```
public static Matrix Lerp(Matrix matrix1, Matrix matrix2, float amount)
```

Parameters

`matrix1` Matrix

The first Matrix.

`matrix2` Matrix

The second Vector2.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Matrix

> The result of linear interpolation of the specified matrixes.

Lerp(ref Matrix, ref Matrix, float, out Matrix)

Creates a new Matrix that contains linear interpolation of the values in specified matrixes.

```
public static void Lerp(ref Matrix matrix1, ref Matrix matrix2, float amount, out
Matrix result)
```

Parameters

`matrix1` Matrix

The first Matrix.

`matrix2` Matrix

The second Vector2.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Matrix

The result of linear interpolation of the specified matrixes as an output parameter.

Multiply(Matrix, Matrix)

Creates a new Matrix that contains a multiplication of two matrix.

```
public static Matrix Multiply(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix.

`matrix2` Matrix

Source Matrix.

Returns

Matrix

Result of the matrix multiplication.

Multiply(Matrix, float)

Creates a new Matrix that contains a multiplication of Matrix and a scalar.

```
public static Matrix Multiply(Matrix matrix1, float scaleFactor)
```

Parameters

`matrix1` Matrix

Source Matrix.

`scaleFactor` float

Scalar value.

Returns

Matrix

Result of the matrix multiplication with a scalar.

Multiply(ref Matrix, ref Matrix, out Matrix)

Creates a new Matrix that contains a multiplication of two matrix.

```
public static void Multiply(ref Matrix matrix1, ref Matrix matrix2, out
Matrix result)
```

Parameters

`matrix1` Matrix

Source Matrix.

`matrix2` Matrix

Source Matrix.

`result` Matrix

Result of the matrix multiplication as an output parameter.

Multiply(ref Matrix, float, out Matrix)

Creates a new Matrix that contains a multiplication of Matrix and a scalar.

```
public static void Multiply(ref Matrix matrix1, float scaleFactor, out
Matrix result)
```

Parameters

`matrix1` Matrix

Source Matrix.

`scaleFactor` float

Scalar value.

`result` Matrix

Result of the matrix multiplication with a scalar as an output parameter.

Negate(Matrix)

Returns a matrix with the all values negated.

```
public static Matrix Negate(Matrix matrix)
```

Parameters

`matrix` Matrix

Source Matrix.

Returns

Matrix

Result of the matrix negation.

Negate(ref Matrix, out Matrix)

Returns a matrix with the all values negated.

```
public static void Negate(ref Matrix matrix, out Matrix result)
```

Parameters

`matrix` Matrix

Source Matrix.

`result` Matrix

Result of the matrix negation as an output parameter.

Subtract(Matrix, Matrix)

Creates a new Matrix that contains subtraction of one matrix from another.

```
public static Matrix Subtract(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

The first Matrix.

`matrix2` Matrix

The second Matrix.

Returns

Matrix

The result of the matrix subtraction.

Subtract(ref Matrix, ref Matrix, out Matrix)

Creates a new Matrix that contains subtraction of one matrix from another.

```
public static void Subtract(ref Matrix matrix1, ref Matrix matrix2, out
Matrix result)
```

Parameters

`matrix1` Matrix

The first Matrix.

`matrix2` Matrix

The second Matrix.

`result` Matrix

The result of the matrix subtraction as an output parameter.

ToFloatArray(Matrix)

Copy the values of specified Matrix to the float array.

```
public static float[] ToFloatArray(Matrix matrix)
```

Parameters

`matrix` Matrix

The source Matrix.

Returns

float[]

The array which matrix values will be stored.

Remarks

Required for OpenGL 2.0 projection matrix stuff.

ToNumerics()

Returns a Matrix4x4.

```
public Matrix4x4 ToNumerics()
```

Returns

Matrix4x4

ToString()

Returns a string representation of this Matrix in the format: {M11:[M11] M12:[M12] M13:[M13] M14:[M14]} {M21:[M21] M12:[M22] M13:[M23] M14:[M24]} {M31:[M31] M32:[M32] M33:[M33] M34:[M34]} {M41:[M41] M42:[M42] M43:[M43] M44:[M44]}

```
public override string ToString()
```

Returns

string

A string representation of this Matrix.

Transpose(Matrix)

Swap the matrix rows and columns.

```
public static Matrix Transpose(Matrix matrix)
```

Parameters

`matrix` Matrix

The matrix for transposing operation.

Returns

Matrix

The new Matrix which contains the transposing result.

Transpose(ref Matrix, out Matrix)

Swap the matrix rows and columns.

```
public static void Transpose(ref Matrix matrix, out Matrix result)
```

Parameters

`matrix` Matrix

The matrix for transposing operation.

`result` Matrix

The new Matrix which contains the transposing result as an output parameter.

Operators

operator +(Matrix, Matrix)

Adds two matrixes.

```
public static Matrix operator +(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the add sign.

`matrix2` Matrix

Source Matrix on the right of the add sign.

Returns

Matrix

Sum of the matrixes.

operator /(Matrix, Matrix)

Divides the elements of a Matrix by the elements of another Matrix.

```
public static Matrix operator /(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the div sign.

`matrix2` Matrix

Divisor Matrix on the right of the div sign.

Returns

Matrix

The result of dividing the matrixes.

operator /(Matrix, float)

Divides the elements of a Matrix by a scalar.

```
public static Matrix operator /(Matrix matrix, float divider)
```

Parameters

`matrix` Matrix

Source Matrix on the left of the div sign.

`divider` float

Divisor scalar on the right of the div sign.

Returns

Matrix

The result of dividing a matrix by a scalar.

operator ==(Matrix, Matrix)

Compares whether two Matrix instances are equal without any tolerance.

```
public static bool operator ==(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the equal sign.

`matrix2` Matrix

Source Matrix on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Matrix(Matrix4x4)

Converts a Matrix4x4 to a Matrix.

```
public static implicit operator Matrix(Matrix4x4 value)
```

Parameters

`value` Matrix4x4

The converted value.

Returns

Matrix

operator !=(Matrix, Matrix)

Compares whether two Matrix instances are not equal without any tolerance.

```
public static bool operator !=(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the not equal sign.

`matrix2` Matrix

Source Matrix on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Matrix, Matrix)

Multiplies two matrixes.

```
public static Matrix operator *(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the mul sign.

`matrix2` Matrix

Source Matrix on the right of the mul sign.

Returns

Matrix

Result of the matrix multiplication.

Remarks

Using matrix multiplication algorithm - see http://en.wikipedia.org/wiki/Matrix_multiplication.

operator *(Matrix, float)

Multiplies the elements of matrix by a scalar.

```
public static Matrix operator *(Matrix matrix, float scaleFactor)
```

Parameters

`matrix` Matrix

Source Matrix on the left of the mul sign.

`scaleFactor` float

Scalar value on the right of the mul sign.

Returns

Matrix

Result of the matrix multiplication with a scalar.

operator -(Matrix, Matrix)

Subtracts the values of one Matrix from another Matrix.

```
public static Matrix operator -(Matrix matrix1, Matrix matrix2)
```

Parameters

`matrix1` Matrix

Source Matrix on the left of the sub sign.

`matrix2` Matrix

Source Matrix on the right of the sub sign.

Returns

Matrix

Result of the matrix subtraction.

operator -(Matrix)

Inverts values in the specified Matrix.

```
public static Matrix operator -(Matrix matrix)
```

Parameters

`matrix` Matrix

Source Matrix on the right of the sub sign.

Returns

Matrix

Result of the inversion.

Struct Plane

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

A plane in 3d space, represented by its normal away from the origin and its distance from the origin, D.

```
[DataContract]
public struct Plane : IEquatable<Plane>
```

Implements

IEquatable<Plane>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Plane(Vector3, Vector3)

Create a Plane that contains the specified point and has the specified Normal vector.

```
public Plane(Vector3 pointOnPlane, Vector3 normal)
```

Parameters

`pointOnPlane` Vector3

A point the created Plane should contain.

`normal` Vector3

The normal of the plane.

Plane(Vector3, Vector3, Vector3)

Create the Plane that contains the three specified points.

```
public Plane(Vector3 a, Vector3 b, Vector3 c)
```

Parameters

a Vector3

A point the created Plane should contain.

b Vector3

A point the created Plane should contain.

c Vector3

A point the created Plane should contain.

Plane(Vector3, float)

Create a Plane with the specified normal and distance to the origin.

```
public Plane(Vector3 normal, float d)
```

Parameters

`normal` Vector3

The normal of the plane.

d float

The distance to the origin.

Plane(Vector4)

Create a Plane with the first three components of the specified Vector4 as the normal and the last component as the distance to the origin.

```
public Plane(Vector4 value)
```

Parameters

`value` Vector4

A vector holding the normal and distance to origin.

Plane(float, float, float, float)

Create a Plane with the first three values as the X, Y and Z components of the normal and the last value as the distance to the origin.

```
public Plane(float a, float b, float c, float d)
```

Parameters

a float

The X component of the normal.

b float

The Y component of the normal.

c float

The Z component of the normal.

d float

The distance to the origin.

Fields

D

The distance of the Plane to the origin.

```
[DataMember]
public float D
```

Field Value

float

Normal

The normal of the Plane.

```
[DataMember]
public Vector3 Normal
```

Field Value

Vector3

Methods

Deconstruct(out Vector3, out float)

Deconstruction method for Plane.

```
public void Deconstruct(out Vector3 normal, out float d)
```

Parameters

`normal` Vector3

`d` float

Dot(Vector4)

Get the dot product of a Vector4 with this Plane.

```
public float Dot(Vector4 value)
```

Parameters

`value` Vector4

The Vector4 to calculate the dot product with.

Returns

float

The dot product of the specified Vector4 and this Plane.

Dot(ref Vector4, out float)

Get the dot product of a Vector4 with this Plane.

```
public void Dot(ref Vector4 value, out float result)
```

Parameters

`value` Vector4

The Vector4 to calculate the dot product with.

`result` float

The dot product of the specified Vector4 and this Plane.

DotCoordinate(Vector3)

Get the dot product of a Vector3 with the Normal vector of this Plane plus the D value of this Plane.

```
public float DotCoordinate(Vector3 value)
```

Parameters

`value` Vector3

The Vector3 to calculate the dot product with.

Returns

float

The dot product of the specified Vector3 and the normal of this Plane plus the D value of this Plane.

DotCoordinate(ref Vector3, out float)

Get the dot product of a Vector3 with the Normal vector of this Plane plus the D value of this Plane.

```
public void DotCoordinate(ref Vector3 value, out float result)
```

Parameters

`value` Vector3

The Vector3 to calculate the dot product with.

`result` float

The dot product of the specified Vector3 and the normal of this Plane plus the D value of this Plane.

DotNormal(Vector3)

Get the dot product of a Vector3 with the Normal vector of this Plane.

```
public float DotNormal(Vector3 value)
```

Parameters

`value` Vector3

The Vector3 to calculate the dot product with.

Returns

float

The dot product of the specified Vector3 and the normal of this Plane.

DotNormal(ref Vector3, out float)

Get the dot product of a Vector3 with the Normal vector of this Plane.

```
public void DotNormal(ref Vector3 value, out float result)
```

Parameters

`value` Vector3

The Vector3 to calculate the dot product with.

`result` float

The dot product of the specified Vector3 and the normal of this Plane.

Equals(Plane)

Check if this Plane is equal to another Plane.

```
public bool Equals(Plane other)
```

Parameters

`other` Plane

A Plane to check for equality with this Plane.

Returns

bool

`true`

if the specified Plane is equal to this one,

`false`

if it is not.

Equals(object)

Check if this Plane is equal to another Plane.

```
public override bool Equals(object other)
```

Parameters

`other` object

An object to check for equality with this Plane.

Returns

bool

`true`

if the specified object is equal to this Plane,

`false`

if it is not.

GetHashCode()

Get a hash code for this Plane.

```
public override int GetHashCode()
```

Returns

int

A hash code for this Plane.

Intersects(BoundingBox)

Check if this Plane intersects a BoundingBox.

```
public PlaneIntersectionType Intersects(BoundingBox box)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

Returns

PlaneIntersectionType

The type of intersection of this Plane with the specified BoundingBox.

Intersects(ref BoundingBox, out PlaneIntersectionType)

Check if this Plane intersects a BoundingBox.

```
public void Intersects(ref BoundingBox box, out PlaneIntersectionType result)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

`result` PlaneIntersectionType

The type of intersection of this Plane with the specified BoundingBox.

Intersects(BoundingFrustum)

Check if this Plane intersects a BoundingFrustum.

```
public PlaneIntersectionType Intersects(BoundingFrustum frustum)
```

Parameters

`frustum` BoundingFrustum

The BoundingFrustum to test for intersection.

Returns

PlaneIntersectionType

The type of intersection of this Plane with the specified BoundingFrustum.

Intersects(BoundingSphere)

Check if this Plane intersects a BoundingSphere.

```
public PlaneIntersectionType Intersects(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to test for intersection.

Returns

PlaneIntersectionType

The type of intersection of this Plane with the specified BoundingSphere.

Intersects(ref BoundingSphere, out PlaneIntersectionType)

Check if this Plane intersects a BoundingSphere.

```
public void Intersects(ref BoundingSphere sphere, out PlaneIntersectionType result)
```

Parameters

`sphere` BoundingSphere

The BoundingSphere to test for intersection.

`result` PlaneIntersectionType

The type of intersection of this Plane with the specified BoundingSphere.

Normalize()

Normalize the normal vector of this plane.

```
public void Normalize()
```

Normalize(Plane)

Get a normalized version of the specified plane.

```
public static Plane Normalize(Plane value)
```

Parameters

`value` Plane

The Plane to normalize.

Returns

Plane

A normalized version of the specified Plane.

Normalize(ref Plane, out Plane)

Get a normalized version of the specified plane.

```
public static void Normalize(ref Plane value, out Plane result)
```

Parameters

`value` Plane

The Plane to normalize.

`result` Plane

A normalized version of the specified Plane.

ToNumerics()

Returns a Plane.

```
public Plane ToNumerics()
```

Returns

Plane

ToString()

Get a string representation of this Plane.

```
public override string ToString()
```

Returns

string

A string representation of this Plane.

Transform(Plane, Matrix)

Transforms a normalized plane by a matrix.

```
public static Plane Transform(Plane plane, Matrix matrix)
```

Parameters

`plane` Plane

The normalized plane to transform.

`matrix` Matrix

The transformation matrix.

Returns

Plane

The transformed plane.

Transform(Plane, Quaternion)

Transforms a normalized plane by a quaternion rotation.

```
public static Plane Transform(Plane plane, Quaternion rotation)
```

Parameters

`plane` Plane

The normalized plane to transform.

`rotation` Quaternion

The quaternion rotation.

Returns

Plane

The transformed plane.

Transform(ref Plane, ref Matrix, out Plane)

Transforms a normalized plane by a matrix.

```
public static void Transform(ref Plane plane, ref Matrix matrix, out Plane result)
```

Parameters

`plane` Plane

The normalized plane to transform.

`matrix` Matrix

The transformation matrix.

`result` Plane

The transformed plane.

Transform(ref Plane, ref Quaternion, out Plane)

Transforms a normalized plane by a quaternion rotation.

```
public static void Transform(ref Plane plane, ref Quaternion rotation, out
Plane result)
```

Parameters

`plane` Plane

The normalized plane to transform.

`rotation` Quaternion

The quaternion rotation.

`result` Plane

The transformed plane.

Operators

operator ==(Plane, Plane)

Check if two planes are equal.

```
public static bool operator ==(Plane plane1, Plane plane2)
```

Parameters

`plane1` Plane

A Plane to check for equality.

`plane2` Plane

A Plane to check for equality.

Returns

bool

`true`

if the two planes are equal,

`false`

if they are not.

implicit operator Plane(Plane)

Converts a Plane to a Plane.

```
public static implicit operator Plane(Plane value)
```

Parameters

`value` Plane

The converted value.

Returns

Plane

operator !=(Plane, Plane)

Check if two planes are not equal.

```
public static bool operator !=(Plane plane1, Plane plane2)
```

Parameters

`plane1` Plane

A Plane to check for inequality.

`plane2` Plane

A Plane to check for inequality.

Returns

bool

`true`

if the two planes are not equal,

`false`

if they are.

Enum PlaneIntersectionType

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines the intersection between a Plane and a bounding volume.

```
public enum PlaneIntersectionType
```

Fields

`Back = 1`

There is no intersection, the bounding volume is in the positive half space of the plane.

`Front = 0`

There is no intersection, the bounding volume is in the negative half space of the plane.

`Intersecting = 2`

The plane is intersected.

Enum PlayerIndex

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Defines the index of player for various MonoGame components.

```
public enum PlayerIndex
```

Fields

`Eight = 7`

The eighth player index.

`Five = 4`

The fifth player index.

`Four = 3`

The fourth player index.

`One = 0`

The first player index.

`Seven = 6`

The seventh player index.

`Six = 5`

The sixth player index.

`Three = 2`

The third player index.

`Two = 1`

The second player index.

Remarks

Use MaximumGamePadCount to determine the number of supported gamepads on the current platform to ensure that a valid PlayerIndex is used when accessing gamepad input. Not all platforms support all eight player indices.

Struct Point

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 2D-point.

```
[DataContract]
public struct Point : IEquatable<Point>
```

Implements

IEquatable<Point>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Point(int)

Constructs a point with X and Y set to the same value.

```
public Point(int value)
```

Parameters

`value` int

The x and y coordinates in 2d-space.

Point(int, int)

Constructs a point with X and Y from two values.

```
public Point(int x, int y)
```

Parameters

x int

The x coordinate in 2d-space.

y int

The y coordinate in 2d-space.

Fields

X

The x coordinate of this Point.

```
[DataMember]
public int X
```

Field Value

int

Y

The y coordinate of this Point.

```
[DataMember]
public int Y
```

Field Value

int

Properties

Zero

Returns a Point with coordinates 0, 0.

```
public static Point Zero { get; }
```

Property Value

Point

Methods

Deconstruct(out int, out int)

Deconstruction method for Point.

```
public void Deconstruct(out int x, out int y)
```

Parameters

`x` int

`y` int

Equals(Point)

Compares whether current instance is equal to specified Point.

```
public bool Equals(Point other)
```

Parameters

`other` Point

The Point to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetHashCode()

Gets the hash code of this Point.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Point.

ToString()

Returns a string representation of this Point in the format: {X:[X] Y:[Y]}

```
public override string ToString()
```

Returns

string

string representation of this Point.

ToVector2()

Gets a Vector2 representation for this object.

```
public Vector2 ToVector2()
```

Returns

Vector2

A Vector2 representation for this object.

Operators

operator +(Point, Point)

Adds two points.

```
public static Point operator +(Point value1, Point value2)
```

Parameters

`value1` Point

Source Point on the left of the add sign.

`value2` Point

Source Point on the right of the add sign.

Returns

Point

Sum of the points.

operator /(Point, Point)

Divides the components of a Point by the components of another Point.

```
public static Point operator /(Point source, Point divisor)
```

Parameters

`source` Point

Source Point on the left of the div sign.

`divisor` Point

Divisor Point on the right of the div sign.

Returns

Point

The result of dividing the points.

operator ==(Point, Point)

Compares whether two Point instances are equal.

```
public static bool operator ==(Point a, Point b)
```

Parameters

a Point

Point instance on the left of the equal sign.

b Point

Point instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

operator !=(Point, Point)

Compares whether two Point instances are not equal.

```
public static bool operator !=(Point a, Point b)
```

Parameters

a Point

Point instance on the left of the not equal sign.

b Point

Point instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Point, Point)

Multiplies the components of two points by each other.

```
public static Point operator *(Point value1, Point value2)
```

Parameters

`value1` Point

Source Point on the left of the mul sign.

`value2` Point

Source Point on the right of the mul sign.

Returns

Point

Result of the multiplication.

operator -(Point, Point)

Subtracts a Point from a Point.

```
public static Point operator -(Point value1, Point value2)
```

Parameters

`value1` Point

Source Point on the left of the sub sign.

`value2` Point

Source Point on the right of the sub sign.

Returns

Point

Result of the subtraction.

Class PreparingDeviceSettingsEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

The arguments to the PreparingDeviceSettings event.

```
public class PreparingDeviceSettingsEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← PreparingDeviceSettingsEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PreparingDeviceSettingsEventArgs(GraphicsDeviceInformation)

Create a new instance of the event.

```
public PreparingDeviceSettingsEventArgs(GraphicsDeviceInformation
graphicsDeviceInformation)
```

Parameters

`graphicsDeviceInformation` GraphicsDeviceInformation

The default settings to be used in device creation.

Properties

GraphicsDeviceInformation

The default settings that will be used in device creation.

```
public GraphicsDeviceInformation GraphicsDeviceInformation { get; }
```

Property Value

GraphicsDeviceInformation

Struct Quaternion

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

An efficient mathematical representation for three dimensional rotations.

```
[DataContract]
public struct Quaternion : IEquatable<Quaternion>
```

Implements

IEquatable<Quaternion>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Quaternion(Vector3, float)

Constructs a quaternion with X, Y, Z from Vector3 and rotation component from a scalar.

```
public Quaternion(Vector3 value, float w)
```

Parameters

`value` Vector3

The x, y, z coordinates in 3d-space.

`w` float

The rotation component.

Quaternion(Vector4)

Constructs a quaternion from Vector4.

```
public Quaternion(Vector4 value)
```

Parameters

`value` Vector4

The x, y, z coordinates in 3d-space and the rotation component.

Quaternion(float, float, float, float)

Constructs a quaternion with X, Y, Z and W from four values.

```
public Quaternion(float x, float y, float z, float w)
```

Parameters

`x` float

The x coordinate in 3d-space.

`y` float

The y coordinate in 3d-space.

`z` float

The z coordinate in 3d-space.

`w` float

The rotation component.

Fields

W

The rotation component of this Quaternion.

```
[DataMember]
public float W
```

Field Value

float

X

The x coordinate of this Quaternion.

```
[DataMember]
public float X
```

Field Value

float

Y

The y coordinate of this Quaternion.

```
[DataMember]
public float Y
```

Field Value

float

Z

The z coordinate of this Quaternion.

```
[DataMember]
public float Z
```

Field Value

float

Properties

Identity

Returns a quaternion representing no rotation.

```
public static Quaternion Identity { get; }
```

Property Value

Quaternion

Methods

Add(Quaternion, Quaternion)

Creates a new Quaternion that contains the sum of two quaternions.

```
public static Quaternion Add(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

Returns

Quaternion

The result of the quaternion addition.

Add(ref Quaternion, ref Quaternion, out Quaternion)

Creates a new Quaternion that contains the sum of two quaternions.

```
public static void Add(ref Quaternion quaternion1, ref Quaternion quaternion2, out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`result` Quaternion

The result of the quaternion addition as an output parameter.

Concatenate(Quaternion, Quaternion)

Creates a new Quaternion that contains concatenation between two quaternion.

```
public static Quaternion Concatenate(Quaternion value1, Quaternion value2)
```

Parameters

`value1` Quaternion

The first Quaternion to concatenate.

`value2` Quaternion

The second Quaternion to concatenate.

Returns

Quaternion

The result of rotation of `value1` followed by `value2` rotation.

Concatenate(ref Quaternion, ref Quaternion, out Quaternion)

Creates a new Quaternion that contains concatenation between two quaternion.

```
public static void Concatenate(ref Quaternion value1, ref Quaternion value2, out
Quaternion result)
```

Parameters

`value1` Quaternion

The first Quaternion to concatenate.

`value2` Quaternion

The second Quaternion to concatenate.

`result` Quaternion

The result of rotation of `value1` followed by `value2` rotation as an output parameter.

Conjugate()

Transforms this quaternion into its conjugated version.

```
public void Conjugate()
```

Conjugate(Quaternion)

Creates a new Quaternion that contains conjugated version of the specified quaternion.

```
public static Quaternion Conjugate(Quaternion value)
```

Parameters

`value` Quaternion

The quaternion which values will be used to create the conjugated version.

Returns

Quaternion

The conjugate version of the specified quaternion.

Conjugate(ref Quaternion, out Quaternion)

Creates a new Quaternion that contains conjugated version of the specified quaternion.

```
public static void Conjugate(ref Quaternion value, out Quaternion result)
```

Parameters

`value` Quaternion

The quaternion which values will be used to create the conjugated version.

`result` Quaternion

The conjugated version of the specified quaternion as an output parameter.

CreateFromAxisAngle(Vector3, float)

Creates a new Quaternion from the specified axis and angle.

```
public static Quaternion CreateFromAxisAngle(Vector3 axis, float angle)
```

Parameters

`axis` Vector3

The axis of rotation.

`angle` float

The angle in radians.

Returns

Quaternion

The new quaternion builded from axis and angle.

CreateFromAxisAngle(ref Vector3, float, out Quaternion)

Creates a new Quaternion from the specified axis and angle.

```
public static void CreateFromAxisAngle(ref Vector3 axis, float angle, out
Quaternion result)
```

Parameters

`axis` Vector3

The axis of rotation.

`angle` float

The angle in radians.

`result` Quaternion

The new quaternion builded from axis and angle as an output parameter.

CreateFromRotationMatrix(Matrix)

Creates a new Quaternion from the specified Matrix.

```
public static Quaternion CreateFromRotationMatrix(Matrix matrix)
```

Parameters

`matrix` Matrix

The rotation matrix.

Returns

Quaternion

A quaternion composed from the rotation part of the matrix.

CreateFromRotationMatrix(ref Matrix, out Quaternion)

Creates a new Quaternion from the specified Matrix.

```
public static void CreateFromRotationMatrix(ref Matrix matrix, out
Quaternion result)
```

Parameters

`matrix` Matrix

The rotation matrix.

`result` Quaternion

A quaternion composed from the rotation part of the matrix as an output parameter.

CreateFromYawPitchRoll(float, float, float)

Creates a new Quaternion from the specified yaw, pitch and roll angles.

```
public static Quaternion CreateFromYawPitchRoll(float yaw, float pitch, float roll)
```

Parameters

`yaw` float

Yaw around the y axis in radians.

`pitch` float

Pitch around the x axis in radians.

`roll` float

Roll around the z axis in radians.

Returns

Quaternion

A new quaternion from the concatenated yaw, pitch, and roll angles.

CreateFromYawPitchRoll(float, float, float, out Quaternion)

Creates a new Quaternion from the specified yaw, pitch and roll angles.

```
public static void CreateFromYawPitchRoll(float yaw, float pitch, float roll, out
Quaternion result)
```

Parameters

`yaw` float

Yaw around the y axis in radians.

`pitch` float

Pitch around the x axis in radians.

`roll` float

Roll around the z axis in radians.

`result` Quaternion

A new quaternion from the concatenated yaw, pitch, and roll angles as an output parameter.

Deconstruct(out float, out float, out float, out float)

Deconstruction method for Quaternion.

```
public void Deconstruct(out float x, out float y, out float z, out float w)
```

Parameters

`x` float

The x coordinate in 3d-space.

`y` float

The y coordinate in 3d-space.

`z` float

The z coordinate in 3d-space.

`w` float

The rotation component.

Divide(Quaternion, Quaternion)

Divides a Quaternion by the other Quaternion.

```
public static Quaternion Divide(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Divisor Quaternion.

Returns

Quaternion

The result of dividing the quaternions.

Divide(ref Quaternion, ref Quaternion, out Quaternion)

Divides a Quaternion by the other Quaternion.

```
public static void Divide(ref Quaternion quaternion1, ref Quaternion quaternion2,
out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Divisor Quaternion.

`result` Quaternion

The result of dividing the quaternions as an output parameter.

Dot(Quaternion, Quaternion)

Returns a dot product of two quaternions.

```
public static float Dot(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

The first quaternion.

`quaternion2` Quaternion

The second quaternion.

Returns

float

The dot product of two quaternions.

Dot(ref Quaternion, ref Quaternion, out float)

Returns a dot product of two quaternions.

```
public static void Dot(ref Quaternion quaternion1, ref Quaternion quaternion2, out float result)
```

Parameters

`quaternion1` Quaternion

The first quaternion.

`quaternion2` Quaternion

The second quaternion.

`result` float

The dot product of two quaternions as an output parameter.

Equals(Quaternion)

Compares whether current instance is equal to specified Quaternion.

```
public bool Equals(Quaternion other)
```

Parameters

`other` Quaternion

The Quaternion to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetHashCode()

Gets the hash code of this Quaternion.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Quaternion.

Inverse(Quaternion)

Returns the inverse quaternion which represents the opposite rotation.

```
public static Quaternion Inverse(Quaternion quaternion)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

Returns

Quaternion

The inverse quaternion.

Inverse(ref Quaternion, out Quaternion)

Returns the inverse quaternion which represents the opposite rotation.

```
public static void Inverse(ref Quaternion quaternion, out Quaternion result)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

`result` Quaternion

The inverse quaternion as an output parameter.

Length()

Returns the magnitude of the quaternion components.

```
public float Length()
```

Returns

float

The magnitude of the quaternion components.

LengthSquared()

Returns the squared magnitude of the quaternion components.

```
public float LengthSquared()
```

Returns

float

The squared magnitude of the quaternion components.

Lerp(Quaternion, Quaternion, float)

Performs a linear blend between two quaternions.

```
public static Quaternion Lerp(Quaternion quaternion1, Quaternion quaternion2,
float amount)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`amount` float

The blend amount where 0 returns `quaternion1` and 1 `quaternion2`.

Returns

Quaternion

The result of linear blending between two quaternions.

Lerp(ref Quaternion, ref Quaternion, float, out Quaternion)

Performs a linear blend between two quaternions.

```
public static void Lerp(ref Quaternion quaternion1, ref Quaternion quaternion2,
float amount, out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`amount` float

The blend amount where 0 returns `quaternion1` and 1 `quaternion2`.

`result` Quaternion

The result of linear blending between two quaternions as an output parameter.

Multiply(Quaternion, Quaternion)

Creates a new Quaternion that contains a multiplication of two quaternions.

```
public static Quaternion Multiply(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

Returns

Quaternion

The result of the quaternion multiplication.

Multiply(Quaternion, float)

Creates a new Quaternion that contains a multiplication of Quaternion and a scalar.

```
public static Quaternion Multiply(Quaternion quaternion1, float scaleFactor)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`scaleFactor` float

Scalar value.

Returns

Quaternion

The result of the quaternion multiplication with a scalar.

Multiply(ref Quaternion, ref Quaternion, out Quaternion)

Creates a new Quaternion that contains a multiplication of two quaternions.

```
public static void Multiply(ref Quaternion quaternion1, ref Quaternion quaternion2,
out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`result` Quaternion

The result of the quaternion multiplication as an output parameter.

Multiply(ref Quaternion, float, out Quaternion)

Creates a new Quaternion that contains a multiplication of Quaternion and a scalar.

```
public static void Multiply(ref Quaternion quaternion1, float scaleFactor, out
Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`scaleFactor` float

Scalar value.

`result` Quaternion

The result of the quaternion multiplication with a scalar as an output parameter.

Negate(Quaternion)

Flips the sign of the all the quaternion components.

```
public static Quaternion Negate(Quaternion quaternion)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

Returns

Quaternion

The result of the quaternion negation.

Negate(ref Quaternion, out Quaternion)

Flips the sign of the all the quaternion components.

```
public static void Negate(ref Quaternion quaternion, out Quaternion result)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

`result` Quaternion

The result of the quaternion negation as an output parameter.

Normalize()

Scales the quaternion magnitude to unit length.

```
public void Normalize()
```

Normalize(Quaternion)

Scales the quaternion magnitude to unit length.

```
public static Quaternion Normalize(Quaternion quaternion)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

Returns

Quaternion

The unit length quaternion.

Normalize(ref Quaternion, out Quaternion)

Scales the quaternion magnitude to unit length.

```
public static void Normalize(ref Quaternion quaternion, out Quaternion result)
```

Parameters

`quaternion` Quaternion

Source Quaternion.

`result` Quaternion

The unit length quaternion an output parameter.

Slerp(Quaternion, Quaternion, float)

Performs a spherical linear blend between two quaternions.

```
public static Quaternion Slerp(Quaternion quaternion1, Quaternion quaternion2,
float amount)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`amount` float

The blend amount where 0 returns `quaternion1` and 1 `quaternion2`.

Returns

Quaternion

The result of spherical linear blending between two quaternions.

Slerp(ref Quaternion, ref Quaternion, float, out Quaternion)

Performs a spherical linear blend between two quaternions.

```
public static void Slerp(ref Quaternion quaternion1, ref Quaternion quaternion2,
float amount, out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`amount` float

The blend amount where 0 returns `quaternion1` and 1 `quaternion2`.

`result` Quaternion

The result of spherical linear blending between two quaternions as an output parameter.

Subtract(Quaternion, Quaternion)

Creates a new Quaternion that contains subtraction of one Quaternion from another.

```
public static Quaternion Subtract(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

Returns

Quaternion

The result of the quaternion subtraction.

Subtract(ref Quaternion, ref Quaternion, out Quaternion)

Creates a new Quaternion that contains subtraction of one Quaternion from another.

```
public static void Subtract(ref Quaternion quaternion1, ref Quaternion quaternion2,
out Quaternion result)
```

Parameters

`quaternion1` Quaternion

Source Quaternion.

`quaternion2` Quaternion

Source Quaternion.

`result` Quaternion

The result of the quaternion subtraction as an output parameter.

ToNumerics()

Returns a Quaternion.

```
public Quaternion ToNumerics()
```

Returns

Quaternion

ToString()

Returns a string representation of this Quaternion in the format: {X:[X] Y:[Y] Z:[Z] W:[W]}

```
public override string ToString()
```

Returns

string

A string representation of this Quaternion.

ToVector4()

Gets a Vector4 representation for this object.

```
public Vector4 ToVector4()
```

Returns

Vector4

A Vector4 representation for this object.

Operators

operator +(Quaternion, Quaternion)

Adds two quaternions.

```
public static Quaternion operator +(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion on the left of the add sign.

`quaternion2` Quaternion

Source Quaternion on the right of the add sign.

Returns

Quaternion

Sum of the vectors.

operator /(Quaternion, Quaternion)

Divides a Quaternion by the other Quaternion.

```
public static Quaternion operator /(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion on the left of the div sign.

`quaternion2` Quaternion

Divisor Quaternion on the right of the div sign.

Returns

Quaternion

The result of dividing the quaternions.

operator ==(Quaternion, Quaternion)

Compares whether two Quaternion instances are equal.

```
public static bool operator ==(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Quaternion instance on the left of the equal sign.

`quaternion2` Quaternion

Quaternion instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Quaternion(Quaternion)

Converts a Quaternion to a Quaternion.

```
public static implicit operator Quaternion(Quaternion value)
```

Parameters

`value` Quaternion

The converted value.

Returns

Quaternion

operator !=(Quaternion, Quaternion)

Compares whether two Quaternion instances are not equal.

```
public static bool operator !=(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Quaternion instance on the left of the not equal sign.

`quaternion2` Quaternion

Quaternion instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Quaternion, Quaternion)

Multiplies two quaternions.

```
public static Quaternion operator *(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Quaternion on the left of the mul sign.

`quaternion2` Quaternion

Source Quaternion on the right of the mul sign.

Returns

Quaternion

Result of the quaternions multiplication.

operator *(Quaternion, float)

Multiplies the components of quaternion by a scalar.

```
public static Quaternion operator *(Quaternion quaternion1, float scaleFactor)
```

Parameters

`quaternion1` Quaternion

Source Vector3 on the left of the mul sign.

`scaleFactor` float

Scalar value on the right of the mul sign.

Returns

Quaternion

Result of the quaternion multiplication with a scalar.

operator -(Quaternion, Quaternion)

Subtracts a Quaternion from a Quaternion.

```
public static Quaternion operator -(Quaternion quaternion1, Quaternion quaternion2)
```

Parameters

`quaternion1` Quaternion

Source Vector3 on the left of the sub sign.

`quaternion2` Quaternion

Source Vector3 on the right of the sub sign.

Returns

Quaternion

Result of the quaternion subtraction.

operator -(Quaternion)

Flips the sign of the all the quaternion components.

```
public static Quaternion operator -(Quaternion quaternion)
```

Parameters

`quaternion` Quaternion

Source Quaternion on the right of the sub sign.

Returns

Quaternion

The result of the quaternion negation.

Struct Ray

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Represents a ray with an origin and a direction in 3D space.

```
[DataContract]
public struct Ray : IEquatable<Ray>
```

Implements

IEquatable<Ray>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Ray(Vector3, Vector3)

Create a Ray.

```
public Ray(Vector3 position, Vector3 direction)
```

Parameters

`position` Vector3

The origin of the Ray.

`direction` Vector3

The direction of the Ray.

Fields

Direction

The direction of this Ray.

```
[DataMember]
public Vector3 Direction
```

Field Value

Vector3

Position

The origin of this Ray.

```
[DataMember]
public Vector3 Position
```

Field Value

Vector3

Methods

Deconstruct(out Vector3, out Vector3)

Deconstruction method for Ray.

```
public void Deconstruct(out Vector3 position, out Vector3 direction)
```

Parameters

`position` Vector3

Receives the start position of the ray.

`direction` Vector3

Receives the direction of the ray.

Equals(Ray)

Check if the specified Ray is equal to this Ray.

```
public bool Equals(Ray other)
```

Parameters

`other` Ray

The Ray to test for equality with this Ray.

Returns

bool

`true`

if the specified Ray is equal to this Ray,

`false`

if it is not.

Equals(object)

Check if the specified object is equal to this Ray.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to test for equality with this Ray.

Returns

bool

`true`

if the specified object is equal to this Ray,

`false`

if it is not.

GetHashCode()

Get a hash code for this Ray.

```
public override int GetHashCode()
```

Returns

int

A hash code for this Ray.

Intersects(BoundingBox)

Check if this Ray intersects the specified BoundingBox.

```
public float? Intersects(BoundingBox box)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

Returns

float?

The distance along the ray of the intersection or

`null`

if this Ray does not intersect the BoundingBox.

Intersects(ref BoundingBox, out float?)

Check if this Ray intersects the specified BoundingBox.

```
public void Intersects(ref BoundingBox box, out float? result)
```

Parameters

`box` BoundingBox

The BoundingBox to test for intersection.

`result` float?

The distance along the ray of the intersection or

`null`

if this Ray does not intersect the BoundingBox.

Intersects(BoundingSphere)

Check if this Ray intersects the specified BoundingSphere.

```
public float? Intersects(BoundingSphere sphere)
```

Parameters

`sphere` BoundingSphere

The BoundingBox to test for intersection.

Returns

float?

The distance along the ray of the intersection or

`null`

if this Ray does not intersect the BoundingSphere.

Intersects(ref BoundingSphere, out float?)

Check if this Ray intersects the specified BoundingSphere.

```
public void Intersects(ref BoundingSphere sphere, out float? result)
```

Parameters

`sphere` BoundingSphere

The BoundingBox to test for intersection.

`result` float?

The distance along the ray of the intersection or

`null`

if this Ray does not intersect the BoundingSphere.

Intersects(Plane)

Check if this Ray intersects the specified Plane.

```
public float? Intersects(Plane plane)
```

Parameters

`plane` Plane

The Plane to test for intersection.

Returns

float?

The distance along the ray of the intersection or

```
null
```

if this Ray does not intersect the Plane.

Intersects(ref Plane, out float?)

Check if this Ray intersects the specified Plane.

```
public void Intersects(ref Plane plane, out float? result)
```

Parameters

`plane` Plane

The Plane to test for intersection.

`result` float?

The distance along the ray of the intersection or

```
null
```

if this Ray does not intersect the Plane.

ToString()

Get a string representation of this Ray.

```
public override string ToString()
```

Returns

string

A string representation of this Ray.

Operators

operator ==(Ray, Ray)

Check if two rays are equal.

```
public static bool operator ==(Ray a, Ray b)
```

Parameters

a Ray

A ray to check for equality.

b Ray

A ray to check for equality.

Returns

bool

`true`

if the two rays are equals,

`false`

if they are not.

operator !=(Ray, Ray)

Check if two rays are not equal.

```
public static bool operator !=(Ray a, Ray b)
```

Parameters

a Ray

A ray to check for inequality.

b Ray

A ray to check for inequality.

Returns

bool

`true`

if the two rays are not equal,

`false`

if they are.

Struct Rectangle

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 2D-rectangle.

```
[DataContract]
public struct Rectangle : IEquatable<Rectangle>
```

Implements

IEquatable<Rectangle>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Rectangle(Point, Point)

Creates a new instance of Rectangle struct, with the specified location and size.

```
public Rectangle(Point location, Point size)
```

Parameters

`location` Point

The x and y coordinates of the top-left corner of the created Rectangle.

`size` Point

The width and height of the created Rectangle.

Rectangle(int, int, int, int)

Creates a new instance of Rectangle struct, with the specified position, width, and height.

```
public Rectangle(int x, int y, int width, int height)
```

Parameters

`x` int

The x coordinate of the top-left corner of the created Rectangle.

`y` int

The y coordinate of the top-left corner of the created Rectangle.

`width` int

The width of the created Rectangle.

`height` int

The height of the created Rectangle.

Fields

Height

The height of this Rectangle.

```
[DataMember]
public int Height
```

Field Value

int

Width

The width of this Rectangle.

```
[DataMember]
public int Width
```

Field Value

int

X

The x coordinate of the top-left corner of this Rectangle.

```
[DataMember]
public int X
```

Field Value

int

Y

The y coordinate of the top-left corner of this Rectangle.

```
[DataMember]
public int Y
```

Field Value

int

Properties

Bottom

Returns the y coordinate of the bottom edge of this Rectangle.

```
public int Bottom { get; }
```

Property Value

int

Center

A Point located in the center of this Rectangle.

```
public Point Center { get; }
```

Property Value

Point

Remarks

If Width or Height is an odd number, the center point will be rounded down.

Empty

Returns a Rectangle with X=0, Y=0, Width=0, Height=0.

```
public static Rectangle Empty { get; }
```

Property Value

Rectangle

IsEmpty

Whether or not this Rectangle has a Width and Height of 0, and a Location of (0, 0).

```
public bool IsEmpty { get; }
```

Property Value

bool

Left

Returns the x coordinate of the left edge of this Rectangle.

```
public int Left { get; }
```

Property Value

int

Location

The top-left coordinates of this Rectangle.

```
public Point Location { get; set; }
```

Property Value

Point

Right

Returns the x coordinate of the right edge of this Rectangle.

```
public int Right { get; }
```

Property Value

int

Size

The width-height coordinates of this Rectangle.

```
public Point Size { get; set; }
```

Property Value

Point

Top

Returns the y coordinate of the top edge of this Rectangle.

```
public int Top { get; }
```

Property Value

int

Methods

Contains(Point)

Gets whether or not the provided Point lies within the bounds of this Rectangle.

```
public bool Contains(Point value)
```

Parameters

`value` Point

The coordinates to check for inclusion in this Rectangle.

Returns

bool

`true` if the provided Point lies inside this Rectangle; `false` otherwise.

Contains(ref Point, out bool)

Gets whether or not the provided Point lies within the bounds of this Rectangle.

```
public void Contains(ref Point value, out bool result)
```

Parameters

`value` Point

The coordinates to check for inclusion in this Rectangle.

`result` bool

`true` if the provided Point lies inside this Rectangle; `false` otherwise. As an output parameter.

Contains(Rectangle)

Gets whether or not the provided Rectangle lies within the bounds of this Rectangle.

```
public bool Contains(Rectangle value)
```

Parameters

`value` Rectangle

The Rectangle to check for inclusion in this Rectangle.

Returns

bool

`true` if the provided Rectangle's bounds lie entirely inside this Rectangle; `false` otherwise.

Contains(ref Rectangle, out bool)

Gets whether or not the provided Rectangle lies within the bounds of this Rectangle.

```
public void Contains(ref Rectangle value, out bool result)
```

Parameters

`value` Rectangle

The Rectangle to check for inclusion in this Rectangle.

`result` bool

`true` if the provided Rectangle's bounds lie entirely inside this Rectangle; `false` otherwise. As an output parameter.

Contains(Vector2)

Gets whether or not the provided Vector2 lies within the bounds of this Rectangle.

```
public bool Contains(Vector2 value)
```

Parameters

`value` Vector2

The coordinates to check for inclusion in this Rectangle.

Returns

bool

`true` if the provided Vector2 lies inside this Rectangle; `false` otherwise.

Contains(ref Vector2, out bool)

Gets whether or not the provided Vector2 lies within the bounds of this Rectangle.

```
public void Contains(ref Vector2 value, out bool result)
```

Parameters

`value` Vector2

The coordinates to check for inclusion in this Rectangle.

`result` bool

`true` if the provided Vector2 lies inside this Rectangle; `false` otherwise. As an output parameter.

Contains(int, int)

Gets whether or not the provided coordinates lie within the bounds of this Rectangle.

```
public bool Contains(int x, int y)
```

Parameters

`x` int

The x coordinate of the point to check for containment.

`y` int

The y coordinate of the point to check for containment.

Returns

bool

`true` if the provided coordinates lie inside this Rectangle; `false` otherwise.

Contains(float, float)

Gets whether or not the provided coordinates lie within the bounds of this Rectangle.

```
public bool Contains(float x, float y)
```

Parameters

`x` float

The x coordinate of the point to check for containment.

`y` float

The y coordinate of the point to check for containment.

Returns

bool

`true` if the provided coordinates lie inside this Rectangle; `false` otherwise.

Deconstruct(out int, out int, out int, out int)

Deconstruction method for Rectangle.

```
public void Deconstruct(out int x, out int y, out int width, out int height)
```

Parameters

`x` int

`y` int

`width` int

`height` int

Equals(Rectangle)

Compares whether current instance is equal to specified Rectangle.

```
public bool Equals(Rectangle other)
```

Parameters

`other` Rectangle

The Rectangle to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

GetHashCode()

Gets the hash code of this Rectangle.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Rectangle.

Inflate(int, int)

Adjusts the edges of this Rectangle by specified horizontal and vertical amounts.

```
public void Inflate(int horizontalAmount, int verticalAmount)
```

Parameters

`horizontalAmount` int

Value to adjust the left and right edges.

`verticalAmount` int

Value to adjust the top and bottom edges.

Inflate(float, float)

Adjusts the edges of this Rectangle by specified horizontal and vertical amounts.

```
public void Inflate(float horizontalAmount, float verticalAmount)
```

Parameters

`horizontalAmount` float

Value to adjust the left and right edges.

`verticalAmount` float

Value to adjust the top and bottom edges.

Intersect(Rectangle, Rectangle)

Creates a new Rectangle that contains overlapping region of two other rectangles.

```
public static Rectangle Intersect(Rectangle value1, Rectangle value2)
```

Parameters

`value1` Rectangle

The first Rectangle.

`value2` Rectangle

The second Rectangle.

Returns

Rectangle

Overlapping region of the two rectangles.

Intersect(ref Rectangle, ref Rectangle, out Rectangle)

Creates a new Rectangle that contains overlapping region of two other rectangles.

```
public static void Intersect(ref Rectangle value1, ref Rectangle value2, out
Rectangle result)
```

Parameters

`value1` Rectangle

The first Rectangle.

`value2` Rectangle

The second Rectangle.

`result` Rectangle

Overlapping region of the two rectangles as an output parameter.

Intersects(Rectangle)

Gets whether or not the other Rectangle intersects with this rectangle.

```
public bool Intersects(Rectangle value)
```

Parameters

`value` Rectangle

The other rectangle for testing.

Returns

bool

`true` if other Rectangle intersects with this rectangle; `false` otherwise.

Intersects(ref Rectangle, out bool)

Gets whether or not the other Rectangle intersects with this rectangle.

```
public void Intersects(ref Rectangle value, out bool result)
```

Parameters

`value` Rectangle

The other rectangle for testing.

`result` bool

`true` if other Rectangle intersects with this rectangle; `false` otherwise. As an output parameter.

Offset(Point)

Changes the Location of this Rectangle.

```
public void Offset(Point amount)
```

Parameters

`amount` Point

The x and y components to add to this Rectangle.

Offset(Vector2)

Changes the Location of this Rectangle.

```
public void Offset(Vector2 amount)
```

Parameters

`amount` Vector2

The x and y components to add to this Rectangle.

Offset(int, int)

Changes the Location of this Rectangle.

```
public void Offset(int offsetX, int offsetY)
```

Parameters

`offsetX` int

The x coordinate to add to this Rectangle.

`offsetY` int

The y coordinate to add to this Rectangle.

Offset(float, float)

Changes the Location of this Rectangle.

```
public void Offset(float offsetX, float offsetY)
```

Parameters

`offsetX` float

The x coordinate to add to this Rectangle.

`offsetY` float

The y coordinate to add to this Rectangle.

ToString()

Returns a string representation of this Rectangle in the format: {X:[X] Y:[Y] Width:[Width] Height:[Height]}

```
public override string ToString()
```

Returns

string

string representation of this Rectangle.

Union(Rectangle, Rectangle)

Creates a new Rectangle that completely contains two other rectangles.

```
public static Rectangle Union(Rectangle value1, Rectangle value2)
```

Parameters

`value1` Rectangle

The first Rectangle.

`value2` Rectangle

The second Rectangle.

Returns

Rectangle

The union of the two rectangles.

Union(ref Rectangle, ref Rectangle, out Rectangle)

Creates a new Rectangle that completely contains two other rectangles.

```
public static void Union(ref Rectangle value1, ref Rectangle value2, out
Rectangle result)
```

Parameters

`value1` Rectangle

The first Rectangle.

`value2` Rectangle

The second Rectangle.

`result` Rectangle

The union of the two rectangles as an output parameter.

Operators

operator ==(Rectangle, Rectangle)

Compares whether two Rectangle instances are equal.

```
public static bool operator ==(Rectangle a, Rectangle b)
```

Parameters

a Rectangle

Rectangle instance on the left of the equal sign.

b Rectangle

Rectangle instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

operator !=(Rectangle, Rectangle)

Compares whether two Rectangle instances are not equal.

```
public static bool operator !=(Rectangle a, Rectangle b)
```

Parameters

a Rectangle

Rectangle instance on the left of the not equal sign.

b Rectangle

Rectangle instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

Struct TextInputEventArgs

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

This class is used in GameWindow's TextInput event as EventArgs.

```
public struct TextInputEventArgs
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

TextInputEventArgs(char, Keys)

Creates an instance of TextInputEventArgs.

```
public TextInputEventArgs(char character, Keys key = Keys.None)
```

Parameters

`character` char

Character for the key that was pressed.

`key` Keys

The pressed key.

Fields

Character

The character for the key that was pressed.

```
public readonly char Character
```

Field Value

char

Key

The pressed key.

```
public readonly Keys Key
```

Field Value

Keys

Class TitleContainer

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Provides functionality for opening a stream in the title storage area.

```
public static class TitleContainer
```

Inheritance

object ← TitleContainer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ContentProvider

Set this to a valid IContentProvider for the TitleContainer to check any content requests by it.

```
public static IContentProvider ContentProvider { get; set; }
```

Property Value

IContentProvider

Methods

OpenStream(string)

Returns an open stream to an existing file in the title storage area.

```
public static Stream OpenStream(string name)
```

Parameters

`name` string

The filepath relative to the title storage area.

Returns

Stream

An open stream if a file is found.

Exceptions

ArgumentNullException

If the name is null or invalid.

Struct Vector2

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 2D-vector.

```
[TypeConverter(typeof(Vector2TypeConverter))]
[DataContract]
public struct Vector2 : IEquatable<Vector2>
```

Implements

IEquatable<Vector2>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Vector2(float)

Constructs a 2d vector with X and Y set to the same value.

```
public Vector2(float value)
```

Parameters

`value` float

The x and y coordinates in 2d-space.

Vector2(float, float)

Constructs a 2d vector with X and Y from two values.

```
public Vector2(float x, float y)
```

Parameters

x float

The x coordinate in 2d-space.

y float

The y coordinate in 2d-space.

Fields

X

The x coordinate of this Vector2.

```
[DataMember]
public float X
```

Field Value

float

Y

The y coordinate of this Vector2.

```
[DataMember]
public float Y
```

Field Value

float

Properties

One

Returns a Vector2 with components 1, 1.

```
public static Vector2 One { get; }
```

Property Value

Vector2

UnitX

Returns a Vector2 with components 1, 0.

```
public static Vector2 UnitX { get; }
```

Property Value

Vector2

UnitY

Returns a Vector2 with components 0, 1.

```
public static Vector2 UnitY { get; }
```

Property Value

Vector2

Zero

Returns a Vector2 with components 0, 0.

```
public static Vector2 Zero { get; }
```

Property Value

Vector2

Methods

Add(Vector2, Vector2)

Performs vector addition on `value1` and `value2`.

```
public static Vector2 Add(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

The first vector to add.

`value2` Vector2

The second vector to add.

Returns

Vector2

The result of the vector addition.

Add(ref Vector2, ref Vector2, out Vector2)

Performs vector addition on `value1` and `value2`, storing the result of the addition in `result`.

```
public static void Add(ref Vector2 value1, ref Vector2 value2, out Vector2 result)
```

Parameters

`value1` Vector2

The first vector to add.

`value2` Vector2

The second vector to add.

`result` Vector2

The result of the vector addition.

Barycentric(Vector2, Vector2, Vector2, float, float)

Creates a new Vector2 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 2d-triangle.

```
public static Vector2 Barycentric(Vector2 value1, Vector2 value2, Vector2 value3,
float amount1, float amount2)
```

Parameters

`value1` Vector2

The first vector of 2d-triangle.

`value2` Vector2

The second vector of 2d-triangle.

`value3` Vector2

The third vector of 2d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 2d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 2d-triangle.

Returns

Vector2

The cartesian translation of barycentric coordinates.

Barycentric(ref Vector2, ref Vector2, ref Vector2, float, float, out Vector2)

Creates a new Vector2 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 2d-triangle.

```
public static void Barycentric(ref Vector2 value1, ref Vector2 value2, ref Vector2
value3, float amount1, float amount2, out Vector2 result)
```

Parameters

`value1` Vector2

The first vector of 2d-triangle.

`value2` Vector2

The second vector of 2d-triangle.

`value3` Vector2

The third vector of 2d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 2d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 2d-triangle.

`result` Vector2

The cartesian translation of barycentric coordinates as an output parameter.

CatmullRom(Vector2, Vector2, Vector2, Vector2, float)

Creates a new Vector2 that contains CatmullRom interpolation of the specified vectors.

```
public static Vector2 CatmullRom(Vector2 value1, Vector2 value2, Vector2 value3,
Vector2 value4, float amount)
```

Parameters

`value1` Vector2

The first vector in interpolation.

`value2` Vector2

The second vector in interpolation.

`value3` Vector2

The third vector in interpolation.

`value4` Vector2

The fourth vector in interpolation.

`amount` float

Weighting factor.

Returns

Vector2

The result of CatmullRom interpolation.

CatmullRom(ref Vector2, ref Vector2, ref Vector2, ref Vector2, float, out Vector2)

Creates a new Vector2 that contains CatmullRom interpolation of the specified vectors.

```
public static void CatmullRom(ref Vector2 value1, ref Vector2 value2, ref Vector2
value3, ref Vector2 value4, float amount, out Vector2 result)
```

Parameters

`value1` Vector2

The first vector in interpolation.

`value2` Vector2

The second vector in interpolation.

`value3` Vector2

The third vector in interpolation.

`value4` Vector2

The fourth vector in interpolation.

`amount` float

Weighting factor.

`result` Vector2

The result of CatmullRom interpolation as an output parameter.

Ceiling()

Round the members of this Vector2 towards positive infinity.

```
public void Ceiling()
```

Ceiling(Vector2)

Creates a new Vector2 that contains members from another vector rounded towards positive infinity.

```
public static Vector2 Ceiling(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2.

Returns

Vector2

The rounded Vector2.

Ceiling(ref Vector2, out Vector2)

Creates a new Vector2 that contains members from another vector rounded towards positive infinity.

```
public static void Ceiling(ref Vector2 value, out Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`result` Vector2

The rounded Vector2.

Clamp(Vector2, Vector2, Vector2)

Clamps the specified value within a range.

```
public static Vector2 Clamp(Vector2 value1, Vector2 min, Vector2 max)
```

Parameters

`value1` Vector2

The value to clamp.

`min` Vector2

The min value.

`max` Vector2

The max value.

Returns

Vector2

The clamped value.

Clamp(ref Vector2, ref Vector2, ref Vector2, out Vector2)

Clamps the specified value within a range.

```
public static void Clamp(ref Vector2 value1, ref Vector2 min, ref Vector2 max, out
Vector2 result)
```

Parameters

`value1` Vector2

The value to clamp.

`min` Vector2

The min value.

`max` Vector2

The max value.

`result` Vector2

The clamped value as an output parameter.

Deconstruct(out float, out float)

Deconstruction method for Vector2.

```
public void Deconstruct(out float x, out float y)
```

Parameters

x float

y float

Distance(Vector2, Vector2)

Returns the distance between two vectors.

```
public static float Distance(Vector2 value1, Vector2 value2)
```

Parameters

value1 Vector2

The first vector.

value2 Vector2

The second vector.

Returns

float

The distance between two vectors.

Distance(ref Vector2, ref Vector2, out float)

Returns the distance between two vectors.

```
public static void Distance(ref Vector2 value1, ref Vector2 value2, out
float result)
```

Parameters

value1 Vector2

The first vector.

`value2` Vector2

The second vector.

`result` float

The distance between two vectors as an output parameter.

DistanceSquared(Vector2, Vector2)

Returns the squared distance between two vectors.

```
public static float DistanceSquared(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

Returns

float

The squared distance between two vectors.

DistanceSquared(ref Vector2, ref Vector2, out float)

Returns the squared distance between two vectors.

```
public static void DistanceSquared(ref Vector2 value1, ref Vector2 value2, out
float result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`result` float

The squared distance between two vectors as an output parameter.

Divide(Vector2, Vector2)

Divides the components of a Vector2 by the components of another Vector2.

```
public static Vector2 Divide(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Divisor Vector2.

Returns

Vector2

The result of dividing the vectors.

Divide(Vector2, float)

Divides the components of a Vector2 by a scalar.

```
public static Vector2 Divide(Vector2 value1, float divider)
```

Parameters

`value1` Vector2

Source Vector2.

`divider` float

Divisor scalar.

Returns

Vector2

The result of dividing a vector by a scalar.

Divide(ref Vector2, ref Vector2, out Vector2)

Divides the components of a Vector2 by the components of another Vector2.

```
public static void Divide(ref Vector2 value1, ref Vector2 value2, out
Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Divisor Vector2.

`result` Vector2

The result of dividing the vectors as an output parameter.

Divide(ref Vector2, float, out Vector2)

Divides the components of a Vector2 by a scalar.

```
public static void Divide(ref Vector2 value1, float divider, out Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`divider` float

Divisor scalar.

`result` Vector2

The result of dividing a vector by a scalar as an output parameter.

Dot(Vector2, Vector2)

Returns a dot product of two vectors.

```
public static float Dot(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

Returns

float

The dot product of two vectors.

Dot(ref Vector2, ref Vector2, out float)

Returns a dot product of two vectors.

```
public static void Dot(ref Vector2 value1, ref Vector2 value2, out float result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`result` float

The dot product of two vectors as an output parameter.

Equals(Vector2)

Compares whether current instance is equal to specified Vector2.

```
public bool Equals(Vector2 other)
```

Parameters

`other` Vector2

The Vector2 to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Floor()

Round the members of this Vector2 towards negative infinity.

```
public void Floor()
```

Floor(Vector2)

Creates a new Vector2 that contains members from another vector rounded towards negative infinity.

```
public static Vector2 Floor(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2.

Returns

Vector2

The rounded Vector2.

Floor(ref Vector2, out Vector2)

Creates a new Vector2 that contains members from another vector rounded towards negative infinity.

```
public static void Floor(ref Vector2 value, out Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`result` Vector2

The rounded Vector2.

GetHashCode()

Gets the hash code of this Vector2.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Vector2.

Hermite(Vector2, Vector2, Vector2, Vector2, float)

Creates a new Vector2 that contains hermite spline interpolation.

```
public static Vector2 Hermite(Vector2 value1, Vector2 tangent1, Vector2 value2,
Vector2 tangent2, float amount)
```

Parameters

`value1` Vector2

The first position vector.

`tangent1` Vector2

The first tangent vector.

`value2` Vector2

The second position vector.

`tangent2` Vector2

The second tangent vector.

`amount` float

Weighting factor.

Returns

Vector2

The hermite spline interpolation vector.

Hermite(ref Vector2, ref Vector2, ref Vector2, ref Vector2, float, out Vector2)

Creates a new Vector2 that contains hermite spline interpolation.

```
public static void Hermite(ref Vector2 value1, ref Vector2 tangent1, ref Vector2
value2, ref Vector2 tangent2, float amount, out Vector2 result)
```

Parameters

`value1` Vector2

The first position vector.

`tangent1` Vector2

The first tangent vector.

`value2` Vector2

The second position vector.

`tangent2` Vector2

The second tangent vector.

`amount` float

Weighting factor.

`result` Vector2

The hermite spline interpolation vector as an output parameter.

Length()

Returns the length of this Vector2.

```
public float Length()
```

Returns

float

The length of this Vector2.

LengthSquared()

Returns the squared length of this Vector2.

```
public float LengthSquared()
```

Returns

float

The squared length of this Vector2.

Lerp(Vector2, Vector2, float)

Creates a new Vector2 that contains linear interpolation of the specified vectors.

```
public static Vector2 Lerp(Vector2 value1, Vector2 value2, float amount)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector2

The result of linear interpolation of the specified vectors.

Lerp(ref Vector2, ref Vector2, float, out Vector2)

Creates a new Vector2 that contains linear interpolation of the specified vectors.

```
public static void Lerp(ref Vector2 value1, ref Vector2 value2, float amount, out
Vector2 result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector2

The result of linear interpolation of the specified vectors as an output parameter.

LerpPrecise(Vector2, Vector2, float)

Creates a new Vector2 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(Vector2, Vector2, float). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static Vector2 LerpPrecise(Vector2 value1, Vector2 value2, float amount)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector2

The result of linear interpolation of the specified vectors.

LerpPrecise(ref Vector2, ref Vector2, float, out Vector2)

Creates a new Vector2 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(ref Vector2, ref Vector2, float, out Vector2). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static void LerpPrecise(ref Vector2 value1, ref Vector2 value2, float amount,
out Vector2 result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector2

The result of linear interpolation of the specified vectors as an output parameter.

Max(Vector2, Vector2)

Creates a new Vector2 that contains a maximal values from the two vectors.

```
public static Vector2 Max(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

Returns

Vector2

The Vector2 with maximal values from the two vectors.

Max(ref Vector2, ref Vector2, out Vector2)

Creates a new Vector2 that contains a maximal values from the two vectors.

```
public static void Max(ref Vector2 value1, ref Vector2 value2, out Vector2 result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`result` Vector2

The Vector2 with maximal values from the two vectors as an output parameter.

Min(Vector2, Vector2)

Creates a new Vector2 that contains a minimal values from the two vectors.

```
public static Vector2 Min(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

Returns

Vector2

The Vector2 with minimal values from the two vectors.

Min(ref Vector2, ref Vector2, out Vector2)

Creates a new Vector2 that contains a minimal values from the two vectors.

```
public static void Min(ref Vector2 value1, ref Vector2 value2, out Vector2 result)
```

Parameters

`value1` Vector2

The first vector.

`value2` Vector2

The second vector.

`result` Vector2

The Vector2 with minimal values from the two vectors as an output parameter.

Multiply(Vector2, Vector2)

Creates a new Vector2 that contains a multiplication of two vectors.

```
public static Vector2 Multiply(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

Returns

Vector2

The result of the vector multiplication.

Multiply(Vector2, float)

Creates a new Vector2 that contains a multiplication of Vector2 and a scalar.

```
public static Vector2 Multiply(Vector2 value1, float scaleFactor)
```

Parameters

`value1` Vector2

Source Vector2.

`scaleFactor` float

Scalar value.

Returns

Vector2

The result of the vector multiplication with a scalar.

Multiply(ref Vector2, ref Vector2, out Vector2)

Creates a new Vector2 that contains a multiplication of two vectors.

```
public static void Multiply(ref Vector2 value1, ref Vector2 value2, out
Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

`result` Vector2

The result of the vector multiplication as an output parameter.

Multiply(ref Vector2, float, out Vector2)

Creates a new Vector2 that contains a multiplication of Vector2 and a scalar.

```
public static void Multiply(ref Vector2 value1, float scaleFactor, out
Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`scaleFactor` float

Scalar value.

`result` Vector2

The result of the multiplication with a scalar as an output parameter.

Negate(Vector2)

Creates a new Vector2 that contains the specified vector inversion.

```
public static Vector2 Negate(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2.

Returns

Vector2

The result of the vector inversion.

Negate(ref Vector2, out Vector2)

Creates a new Vector2 that contains the specified vector inversion.

```
public static void Negate(ref Vector2 value, out Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`result` Vector2

The result of the vector inversion as an output parameter.

Normalize()

Turns this Vector2 to a unit vector with the same direction.

```
public void Normalize()
```

Normalize(Vector2)

Creates a new Vector2 that contains a normalized values from another vector.

```
public static Vector2 Normalize(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2.

Returns

Vector2

Unit vector.

Normalize(ref Vector2, out Vector2)

Creates a new Vector2 that contains a normalized values from another vector.

```
public static void Normalize(ref Vector2 value, out Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`result` Vector2

Unit vector as an output parameter.

Reflect(Vector2, Vector2)

Creates a new Vector2 that contains reflect vector of the given vector and normal.

```
public static Vector2 Reflect(Vector2 vector, Vector2 normal)
```

Parameters

`vector` Vector2

Source Vector2.

`normal` Vector2

Reflection normal.

Returns

Vector2

Reflected vector.

Reflect(ref Vector2, ref Vector2, out Vector2)

Creates a new Vector2 that contains reflect vector of the given vector and normal.

```
public static void Reflect(ref Vector2 vector, ref Vector2 normal, out
Vector2 result)
```

Parameters

`vector` Vector2

Source Vector2.

`normal` Vector2

Reflection normal.

`result` Vector2

Reflected vector as an output parameter.

Rotate(Vector2, float)

Rotates a vector by the specified number of radians

```
public static Vector2 Rotate(Vector2 value, float radians)
```

Parameters

`value` Vector2

The vector to be rotated.

`radians` float

The amount to rotate the vector.

Returns

Vector2

A rotated copy of value.

Remarks

A positive angle and negative angle would rotate counterclockwise and clockwise, respectively

Rotate(float)

Rotates a Vector2 by the specified number of radians

```
public void Rotate(float radians)
```

Parameters

`radians` float

The amount to rotate this Vector2.

Remarks

A positive angle and negative angle would rotate counterclockwise and clockwise, respectively

RotateAround(Vector2, Vector2, float)

Rotates a Vector2 around another Vector2 representing a location

```
public static Vector2 RotateAround(Vector2 value, Vector2 origin, float radians)
```

Parameters

`value` Vector2

The Vector2 to be rotated

`origin` Vector2

The origin location to be rotated around

`radians` float

The amount to rotate by in radians

Returns

Vector2

The rotated Vector2

Remarks

A positive angle and negative angle would rotate counterclockwise and clockwise, respectively

RotateAround(Vector2, float)

Rotates a Vector2 around another Vector2 representing a location

```
public void RotateAround(Vector2 origin, float radians)
```

Parameters

`origin` Vector2

The origin location to be rotated around

`radians` float

The amount to rotate by in radians

Remarks

A positive angle and negative angle would rotate counterclockwise and clockwise, respectively

Round()

Round the members of this Vector2 to the nearest integer value.

```
public void Round()
```

Round(Vector2)

Creates a new Vector2 that contains members from another vector rounded to the nearest integer value.

```
public static Vector2 Round(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2.

Returns

Vector2

The rounded Vector2.

Round(ref Vector2, out Vector2)

Creates a new Vector2 that contains members from another vector rounded to the nearest integer value.

```
public static void Round(ref Vector2 value, out Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`result` Vector2

The rounded Vector2.

SmoothStep(Vector2, Vector2, float)

Creates a new Vector2 that contains cubic interpolation of the specified vectors.

```
public static Vector2 SmoothStep(Vector2 value1, Vector2 value2, float amount)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

`amount` float

Weighting value.

Returns

Vector2

Cubic interpolation of the specified vectors.

SmoothStep(ref Vector2, ref Vector2, float, out Vector2)

Creates a new Vector2 that contains cubic interpolation of the specified vectors.

```
public static void SmoothStep(ref Vector2 value1, ref Vector2 value2, float amount,
out Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

`amount` float

Weighting value.

`result` Vector2

Cubic interpolation of the specified vectors as an output parameter.

Subtract(Vector2, Vector2)

Creates a new Vector2 that contains subtraction of on Vector2 from a another.

```
public static Vector2 Subtract(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

Returns

Vector2

The result of the vector subtraction.

Subtract(ref Vector2, ref Vector2, out Vector2)

Creates a new Vector2 that contains subtraction of on Vector2 from a another.

```
public static void Subtract(ref Vector2 value1, ref Vector2 value2, out
Vector2 result)
```

Parameters

`value1` Vector2

Source Vector2.

`value2` Vector2

Source Vector2.

`result` Vector2

The result of the vector subtraction as an output parameter.

ToNumerics()

Returns a Vector2.

```
public Vector2 ToNumerics()
```

Returns

Vector2

ToPoint()

Gets a Point representation for this object.

```
public Point ToPoint()
```

Returns

Point

A Point representation for this object.

ToString()

Returns a string representation of this Vector2 in the format: {X:[X] Y:[Y]}

```
public override string ToString()
```

Returns

string

A string representation of this Vector2.

Transform(Vector2, Matrix)

Creates a new Vector2 that contains a transformation of 2d-vector by the specified Matrix.

```
public static Vector2 Transform(Vector2 position, Matrix matrix)
```

Parameters

`position` Vector2

Source Vector2.

`matrix` Matrix

The transformation Matrix.

Returns

Vector2

Transformed Vector2.

Transform(Vector2, Quaternion)

Creates a new Vector2 that contains a transformation of 2d-vector by the specified Quaternion, representing the rotation.

```
public static Vector2 Transform(Vector2 value, Quaternion rotation)
```

Parameters

`value` Vector2

Source Vector2.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

Returns

Vector2

Transformed Vector2.

Transform(ref Vector2, ref Matrix, out Vector2)

Creates a new Vector2 that contains a transformation of 2d-vector by the specified Matrix.

```
public static void Transform(ref Vector2 position, ref Matrix matrix, out
Vector2 result)
```

Parameters

`position` Vector2

Source Vector2.

`matrix` Matrix

The transformation Matrix.

`result` Vector2

Transformed Vector2 as an output parameter.

Transform(ref Vector2, ref Quaternion, out Vector2)

Creates a new Vector2 that contains a transformation of 2d-vector by the specified Quaternion, representing the rotation.

```
public static void Transform(ref Vector2 value, ref Quaternion rotation, out
Vector2 result)
```

Parameters

`value` Vector2

Source Vector2.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`result` Vector2

Transformed Vector2 as an output parameter.

Transform(Vector2[], ref Matrix, Vector2[])

Apply transformation on all vectors within array of Vector2 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector2[] sourceArray, ref Matrix matrix,
Vector2[] destinationArray)
```

Parameters

`sourceArray` Vector2[]

Source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector2[]

Destination array.

Transform(Vector2[], ref Quaternion, Vector2[])

Apply transformation on all vectors within array of Vector2 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector2[] sourceArray, ref Quaternion rotation,
Vector2[] destinationArray)
```

Parameters

`sourceArray` Vector2[]

Source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector2[]

Destination array.

Transform(Vector2[], int, ref Matrix, Vector2[], int, int)

Apply transformation on vectors within array of Vector2 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector2[] sourceArray, int sourceIndex, ref Matrix
matrix, Vector2[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector2[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector2[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector2 should be written.

`length` int

The number of vectors to be transformed.

Transform(Vector2[], int, ref Quaternion, Vector2[], int, int)

Apply transformation on vectors within array of Vector2 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector2[] sourceArray, int sourceIndex, ref Quaternion
rotation, Vector2[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector2[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector2[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector2 should be written.

`length` int

The number of vectors to be transformed.

TransformNormal(Vector2, Matrix)

Creates a new Vector2 that contains a transformation of the specified normal by the specified Matrix.

```
public static Vector2 TransformNormal(Vector2 normal, Matrix matrix)
```

Parameters

`normal` Vector2

Source Vector2 which represents a normal vector.

`matrix` Matrix

The transformation Matrix.

Returns

Vector2

Transformed normal.

TransformNormal(ref Vector2, ref Matrix, out Vector2)

Creates a new Vector2 that contains a transformation of the specified normal by the specified Matrix.

```
public static void TransformNormal(ref Vector2 normal, ref Matrix matrix, out
Vector2 result)
```

Parameters

`normal` Vector2

Source Vector2 which represents a normal vector.

`matrix` Matrix

The transformation Matrix.

`result` Vector2

Transformed normal as an output parameter.

TransformNormal(Vector2[], ref Matrix, Vector2[])

Apply transformation on all normals within array of Vector2 by the specified Matrix and places the results in an another array.

```
public static void TransformNormal(Vector2[] sourceArray, ref Matrix matrix,
Vector2[] destinationArray)
```

Parameters

`sourceArray` Vector2[]

Source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector2[]

Destination array.

TransformNormal(Vector2[], int, ref Matrix, Vector2[], int, int)

Apply transformation on normals within array of Vector2 by the specified Matrix and places the results in an another array.

```
public static void TransformNormal(Vector2[] sourceArray, int sourceIndex, ref
Matrix matrix, Vector2[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector2[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector2[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector2 should be written.

`length` int

The number of normals to be transformed.

Operators

operator +(Vector2, Vector2)

Adds two vectors.

```
public static Vector2 operator +(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2 on the left of the add sign.

`value2` Vector2

Source Vector2 on the right of the add sign.

Returns

Vector2

Sum of the vectors.

operator /(Vector2, Vector2)

Divides the components of a Vector2 by the components of another Vector2.

```
public static Vector2 operator /(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2 on the left of the div sign.

`value2` Vector2

Divisor Vector2 on the right of the div sign.

Returns

Vector2

The result of dividing the vectors.

operator /(Vector2, float)

Divides the components of a Vector2 by a scalar.

```
public static Vector2 operator /(Vector2 value1, float divider)
```

Parameters

`value1` Vector2

Source Vector2 on the left of the div sign.

`divider` float

Divisor scalar on the right of the div sign.

Returns

Vector2

The result of dividing a vector by a scalar.

operator ==(Vector2, Vector2)

Compares whether two Vector2 instances are equal.

```
public static bool operator ==(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Vector2 instance on the left of the equal sign.

`value2` Vector2

Vector2 instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Vector2(Vector2)

Converts a Vector2 to a Vector2.

```
public static implicit operator Vector2(Vector2 value)
```

Parameters

`value` Vector2

The converted value.

Returns

Vector2

operator !=(Vector2, Vector2)

Compares whether two Vector2 instances are not equal.

```
public static bool operator !=(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Vector2 instance on the left of the not equal sign.

`value2` Vector2

Vector2 instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Vector2, Vector2)

Multiplies the components of two vectors by each other.

```
public static Vector2 operator *(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2 on the left of the mul sign.

`value2` Vector2

Source Vector2 on the right of the mul sign.

Returns

Vector2

Result of the vector multiplication.

operator *(Vector2, float)

Multiplies the components of vector by a scalar.

```
public static Vector2 operator *(Vector2 value, float scaleFactor)
```

Parameters

`value` Vector2

Source Vector2 on the left of the mul sign.

`scaleFactor` float

Scalar value on the right of the mul sign.

Returns

Vector2

Result of the vector multiplication with a scalar.

operator *(float, Vector2)

Multiplies the components of vector by a scalar.

```
public static Vector2 operator *(float scaleFactor, Vector2 value)
```

Parameters

`scaleFactor` float

Scalar value on the left of the mul sign.

`value` Vector2

Source Vector2 on the right of the mul sign.

Returns

Vector2

Result of the vector multiplication with a scalar.

operator -(Vector2, Vector2)

Subtracts a Vector2 from a Vector2.

```
public static Vector2 operator -(Vector2 value1, Vector2 value2)
```

Parameters

`value1` Vector2

Source Vector2 on the left of the sub sign.

`value2` Vector2

Source Vector2 on the right of the sub sign.

Returns

Vector2

Result of the vector subtraction.

operator -(Vector2)

Inverts values in the specified Vector2.

```
public static Vector2 operator -(Vector2 value)
```

Parameters

`value` Vector2

Source Vector2 on the right of the sub sign.

Returns

Vector2

Result of the inversion.

Struct Vector3

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 3D-vector.

```
[TypeConverter(typeof(Vector3TypeConverter))]
[DataContract]
public struct Vector3 : IEquatable<Vector3>
```

Implements

IEquatable<Vector3>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Vector3(Vector2, float)

Constructs a 3d vector with X, Y from Vector2 and Z from a scalar.

```
public Vector3(Vector2 value, float z)
```

Parameters

`value` Vector2

The x and y coordinates in 3d-space.

`z` float

The z coordinate in 3d-space.

Vector3(float)

Constructs a 3d vector with X, Y and Z set to the same value.

```
public Vector3(float value)
```

Parameters

`value` float

The x, y and z coordinates in 3d-space.

Vector3(float, float, float)

Constructs a 3d vector with X, Y and Z from three values.

```
public Vector3(float x, float y, float z)
```

Parameters

`x` float

The x coordinate in 3d-space.

`y` float

The y coordinate in 3d-space.

`z` float

The z coordinate in 3d-space.

Fields

X

The x coordinate of this Vector3.

```
[DataMember]
public float X
```

Field Value

float

Y

The y coordinate of this Vector3.

```
[DataMember]
public float Y
```

Field Value

float

Z

The z coordinate of this Vector3.

```
[DataMember]
public float Z
```

Field Value

float

Properties

Backward

Returns a Vector3 with components 0, 0, 1.

```
public static Vector3 Backward { get; }
```

Property Value

Vector3

Down

Returns a Vector3 with components 0, -1, 0.

```
public static Vector3 Down { get; }
```

Property Value

Vector3

Forward

Returns a Vector3 with components 0, 0, -1.

```
public static Vector3 Forward { get; }
```

Property Value

Vector3

Left

Returns a Vector3 with components -1, 0, 0.

```
public static Vector3 Left { get; }
```

Property Value

Vector3

One

Returns a Vector3 with components 1, 1, 1.

```
public static Vector3 One { get; }
```

Property Value

Vector3

Right

Returns a Vector3 with components 1, 0, 0.

```
public static Vector3 Right { get; }
```

Property Value

Vector3

UnitX

Returns a Vector3 with components 1, 0, 0.

```
public static Vector3 UnitX { get; }
```

Property Value

Vector3

UnitY

Returns a Vector3 with components 0, 1, 0.

```
public static Vector3 UnitY { get; }
```

Property Value

Vector3

UnitZ

Returns a Vector3 with components 0, 0, 1.

```
public static Vector3 UnitZ { get; }
```

Property Value

Vector3

Up

Returns a Vector3 with components 0, 1, 0.

```
public static Vector3 Up { get; }
```

Property Value

Vector3

Zero

Returns a Vector3 with components 0, 0, 0.

```
public static Vector3 Zero { get; }
```

Property Value

Vector3

Methods

Add(Vector3, Vector3)

Performs vector addition on `value1` and `value2`.

```
public static Vector3 Add(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector to add.

`value2` Vector3

The second vector to add.

Returns

Vector3

The result of the vector addition.

Add(ref Vector3, ref Vector3, out Vector3)

Performs vector addition on `value1` and `value2`, storing the result of the addition in `result`.

```
public static void Add(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
```

Parameters

`value1` Vector3

The first vector to add.

`value2` Vector3

The second vector to add.

`result` Vector3

The result of the vector addition.

Barycentric(Vector3, Vector3, Vector3, float, float)

Creates a new Vector3 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 3d-triangle.

```
public static Vector3 Barycentric(Vector3 value1, Vector3 value2, Vector3 value3,
float amount1, float amount2)
```

Parameters

`value1` Vector3

The first vector of 3d-triangle.

`value2` Vector3

The second vector of 3d-triangle.

`value3` Vector3

The third vector of 3d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 3d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 3d-triangle.

Returns

Vector3

The cartesian translation of barycentric coordinates.

Barycentric(ref Vector3, ref Vector3, ref Vector3, float, float, out Vector3)

Creates a new Vector3 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 3d-triangle.

```
public static void Barycentric(ref Vector3 value1, ref Vector3 value2, ref Vector3
value3, float amount1, float amount2, out Vector3 result)
```

Parameters

`value1` Vector3

The first vector of 3d-triangle.

`value2` Vector3

The second vector of 3d-triangle.

`value3` Vector3

The third vector of 3d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 3d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 3d-triangle.

`result` Vector3

The cartesian translation of barycentric coordinates as an output parameter.

CatmullRom(Vector3, Vector3, Vector3, Vector3, float)

Creates a new Vector3 that contains CatmullRom interpolation of the specified vectors.

```
public static Vector3 CatmullRom(Vector3 value1, Vector3 value2, Vector3 value3,
Vector3 value4, float amount)
```

Parameters

`value1` Vector3

The first vector in interpolation.

`value2` Vector3

The second vector in interpolation.

`value3` Vector3

The third vector in interpolation.

`value4` Vector3

The fourth vector in interpolation.

`amount` float

Weighting factor.

Returns

Vector3

The result of CatmullRom interpolation.

CatmullRom(ref Vector3, ref Vector3, ref Vector3, ref Vector3, float, out Vector3)

Creates a new Vector3 that contains CatmullRom interpolation of the specified vectors.

```
public static void CatmullRom(ref Vector3 value1, ref Vector3 value2, ref Vector3
value3, ref Vector3 value4, float amount, out Vector3 result)
```

Parameters

`value1` Vector3

The first vector in interpolation.

`value2` Vector3

The second vector in interpolation.

`value3` Vector3

The third vector in interpolation.

`value4` Vector3

The fourth vector in interpolation.

`amount` float

Weighting factor.

`result` Vector3

The result of CatmullRom interpolation as an output parameter.

Ceiling()

Round the members of this Vector3 towards positive infinity.

```
public void Ceiling()
```

Ceiling(Vector3)

Creates a new Vector3 that contains members from another vector rounded towards positive infinity.

```
public static Vector3 Ceiling(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3.

Returns

Vector3

The rounded Vector3.

Ceiling(ref Vector3, out Vector3)

Creates a new Vector3 that contains members from another vector rounded towards positive infinity.

```
public static void Ceiling(ref Vector3 value, out Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`result` Vector3

The rounded Vector3.

Clamp(Vector3, Vector3, Vector3)

Clamps the specified value within a range.

```
public static Vector3 Clamp(Vector3 value1, Vector3 min, Vector3 max)
```

Parameters

`value1` Vector3

The value to clamp.

`min` Vector3

The min value.

`max` Vector3

The max value.

Returns

Vector3

The clamped value.

Clamp(ref Vector3, ref Vector3, ref Vector3, out Vector3)

Clamps the specified value within a range.

```
public static void Clamp(ref Vector3 value1, ref Vector3 min, ref Vector3 max, out
Vector3 result)
```

Parameters

`value1` Vector3

The value to clamp.

`min` Vector3

The min value.

`max` Vector3

The max value.

`result` Vector3

The clamped value as an output parameter.

Cross(Vector3, Vector3)

Computes the cross product of two vectors.

```
public static Vector3 Cross(Vector3 vector1, Vector3 vector2)
```

Parameters

`vector1` Vector3

The first vector.

`vector2` Vector3

The second vector.

Returns

Vector3

The cross product of two vectors.

Cross(ref Vector3, ref Vector3, out Vector3)

Computes the cross product of two vectors.

```
public static void Cross(ref Vector3 vector1, ref Vector3 vector2, out
Vector3 result)
```

Parameters

`vector1` Vector3

The first vector.

`vector2` Vector3

The second vector.

`result` Vector3

The cross product of two vectors as an output parameter.

Deconstruct(out float, out float, out float)

Deconstruction method for Vector3.

```
public void Deconstruct(out float x, out float y, out float z)
```

Parameters

`x` float

`y` float

`z` float

Distance(Vector3, Vector3)

Returns the distance between two vectors.

```
public static float Distance(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

Returns

float

The distance between two vectors.

Distance(ref Vector3, ref Vector3, out float)

Returns the distance between two vectors.

```
public static void Distance(ref Vector3 value1, ref Vector3 value2, out
float result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`result` float

The distance between two vectors as an output parameter.

DistanceSquared(Vector3, Vector3)

Returns the squared distance between two vectors.

```
public static float DistanceSquared(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

Returns

float

The squared distance between two vectors.

DistanceSquared(ref Vector3, ref Vector3, out float)

Returns the squared distance between two vectors.

```
public static void DistanceSquared(ref Vector3 value1, ref Vector3 value2, out
float result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`result` float

The squared distance between two vectors as an output parameter.

Divide(Vector3, Vector3)

Divides the components of a Vector3 by the components of another Vector3.

```
public static Vector3 Divide(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Divisor Vector3.

Returns

Vector3

The result of dividing the vectors.

Divide(Vector3, float)

Divides the components of a Vector3 by a scalar.

```
public static Vector3 Divide(Vector3 value1, float divider)
```

Parameters

`value1` Vector3

Source Vector3.

`divider` float

Divisor scalar.

Returns

Vector3

The result of dividing a vector by a scalar.

Divide(ref Vector3, ref Vector3, out Vector3)

Divides the components of a Vector3 by the components of another Vector3.

```
public static void Divide(ref Vector3 value1, ref Vector3 value2, out
Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Divisor Vector3.

`result` Vector3

The result of dividing the vectors as an output parameter.

Divide(ref Vector3, float, out Vector3)

Divides the components of a Vector3 by a scalar.

```
public static void Divide(ref Vector3 value1, float divider, out Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`divider` float

Divisor scalar.

`result` Vector3

The result of dividing a vector by a scalar as an output parameter.

Dot(Vector3, Vector3)

Returns a dot product of two vectors.

```
public static float Dot(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

Returns

float

The dot product of two vectors.

Dot(ref Vector3, ref Vector3, out float)

Returns a dot product of two vectors.

```
public static void Dot(ref Vector3 value1, ref Vector3 value2, out float result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`result` float

The dot product of two vectors as an output parameter.

Equals(Vector3)

Compares whether current instance is equal to specified Vector3.

```
public bool Equals(Vector3 other)
```

Parameters

`other` Vector3

The Vector3 to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Floor()

Round the members of this Vector3 towards negative infinity.

```
public void Floor()
```

Floor(Vector3)

Creates a new Vector3 that contains members from another vector rounded towards negative infinity.

```
public static Vector3 Floor(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3.

Returns

Vector3

The rounded Vector3.

Floor(ref Vector3, out Vector3)

Creates a new Vector3 that contains members from another vector rounded towards negative infinity.

```
public static void Floor(ref Vector3 value, out Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`result` Vector3

The rounded Vector3.

GetHashCode()

Gets the hash code of this Vector3.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Vector3.

Hermite(Vector3, Vector3, Vector3, Vector3, float)

Creates a new Vector3 that contains hermite spline interpolation.

```
public static Vector3 Hermite(Vector3 value1, Vector3 tangent1, Vector3 value2,
Vector3 tangent2, float amount)
```

Parameters

`value1` Vector3

The first position vector.

`tangent1` Vector3

The first tangent vector.

`value2` Vector3

The second position vector.

`tangent2` Vector3

The second tangent vector.

`amount` float

Weighting factor.

Returns

Vector3

The hermite spline interpolation vector.

Hermite(ref Vector3, ref Vector3, ref Vector3, ref Vector3, float, out Vector3)

Creates a new Vector3 that contains hermite spline interpolation.

```
public static void Hermite(ref Vector3 value1, ref Vector3 tangent1, ref Vector3
value2, ref Vector3 tangent2, float amount, out Vector3 result)
```

Parameters

`value1` Vector3

The first position vector.

`tangent1` Vector3

The first tangent vector.

`value2` Vector3

The second position vector.

`tangent2` Vector3

The second tangent vector.

`amount` float

Weighting factor.

`result` Vector3

The hermite spline interpolation vector as an output parameter.

Length()

Returns the length of this Vector3.

```
public float Length()
```

Returns

float

The length of this Vector3.

LengthSquared()

Returns the squared length of this Vector3.

```
public float LengthSquared()
```

Returns

float

The squared length of this Vector3.

Lerp(Vector3, Vector3, float)

Creates a new Vector3 that contains linear interpolation of the specified vectors.

```
public static Vector3 Lerp(Vector3 value1, Vector3 value2, float amount)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector3

The result of linear interpolation of the specified vectors.

Lerp(ref Vector3, ref Vector3, float, out Vector3)

Creates a new Vector3 that contains linear interpolation of the specified vectors.

```
public static void Lerp(ref Vector3 value1, ref Vector3 value2, float amount, out
Vector3 result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector3

The result of linear interpolation of the specified vectors as an output parameter.

LerpPrecise(Vector3, Vector3, float)

Creates a new Vector3 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(Vector3, Vector3, float). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static Vector3 LerpPrecise(Vector3 value1, Vector3 value2, float amount)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector3

The result of linear interpolation of the specified vectors.

LerpPrecise(ref Vector3, ref Vector3, float, out Vector3)

Creates a new Vector3 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(ref Vector3, ref Vector3, float, out Vector3). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static void LerpPrecise(ref Vector3 value1, ref Vector3 value2, float amount,
out Vector3 result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector3

The result of linear interpolation of the specified vectors as an output parameter.

Max(Vector3, Vector3)

Creates a new Vector3 that contains a maximal values from the two vectors.

```
public static Vector3 Max(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

Returns

Vector3

The Vector3 with maximal values from the two vectors.

Max(ref Vector3, ref Vector3, out Vector3)

Creates a new Vector3 that contains a maximal values from the two vectors.

```
public static void Max(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`result` Vector3

The Vector3 with maximal values from the two vectors as an output parameter.

Min(Vector3, Vector3)

Creates a new Vector3 that contains a minimal values from the two vectors.

```
public static Vector3 Min(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

Returns

Vector3

The Vector3 with minimal values from the two vectors.

Min(ref Vector3, ref Vector3, out Vector3)

Creates a new Vector3 that contains a minimal values from the two vectors.

```
public static void Min(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
```

Parameters

`value1` Vector3

The first vector.

`value2` Vector3

The second vector.

`result` Vector3

The Vector3 with minimal values from the two vectors as an output parameter.

Multiply(Vector3, Vector3)

Creates a new Vector3 that contains a multiplication of two vectors.

```
public static Vector3 Multiply(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

Returns

Vector3

The result of the vector multiplication.

Multiply(Vector3, float)

Creates a new Vector3 that contains a multiplication of Vector3 and a scalar.

```
public static Vector3 Multiply(Vector3 value1, float scaleFactor)
```

Parameters

`value1` Vector3

Source Vector3.

`scaleFactor` float

Scalar value.

Returns

Vector3

The result of the vector multiplication with a scalar.

Multiply(ref Vector3, ref Vector3, out Vector3)

Creates a new Vector3 that contains a multiplication of two vectors.

```
public static void Multiply(ref Vector3 value1, ref Vector3 value2, out
Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

`result` Vector3

The result of the vector multiplication as an output parameter.

Multiply(ref Vector3, float, out Vector3)

Creates a new Vector3 that contains a multiplication of Vector3 and a scalar.

```
public static void Multiply(ref Vector3 value1, float scaleFactor, out
Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`scaleFactor` float

Scalar value.

`result` Vector3

The result of the multiplication with a scalar as an output parameter.

Negate(Vector3)

Creates a new Vector3 that contains the specified vector inversion.

```
public static Vector3 Negate(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3.

Returns

Vector3

The result of the vector inversion.

Negate(ref Vector3, out Vector3)

Creates a new Vector3 that contains the specified vector inversion.

```
public static void Negate(ref Vector3 value, out Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`result` Vector3

The result of the vector inversion as an output parameter.

Normalize()

Turns this Vector3 to a unit vector with the same direction.

```
public void Normalize()
```

Normalize(Vector3)

Creates a new Vector3 that contains a normalized values from another vector.

```
public static Vector3 Normalize(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3.

Returns

Vector3

Unit vector.

Normalize(ref Vector3, out Vector3)

Creates a new Vector3 that contains a normalized values from another vector.

```
public static void Normalize(ref Vector3 value, out Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`result` Vector3

Unit vector as an output parameter.

Reflect(Vector3, Vector3)

Creates a new Vector3 that contains reflect vector of the given vector and normal.

```
public static Vector3 Reflect(Vector3 vector, Vector3 normal)
```

Parameters

`vector` Vector3

Source Vector3.

`normal` Vector3

Reflection normal.

Returns

Vector3

Reflected vector.

Reflect(ref Vector3, ref Vector3, out Vector3)

Creates a new Vector3 that contains reflect vector of the given vector and normal.

```
public static void Reflect(ref Vector3 vector, ref Vector3 normal, out
Vector3 result)
```

Parameters

`vector` Vector3

Source Vector3.

`normal` Vector3

Reflection normal.

`result` Vector3

Reflected vector as an output parameter.

Round()

Round the members of this Vector3 towards the nearest integer value.

```
public void Round()
```

Round(Vector3)

Creates a new Vector3 that contains members from another vector rounded to the nearest integer value.

```
public static Vector3 Round(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3.

Returns

Vector3

The rounded Vector3.

Round(ref Vector3, out Vector3)

Creates a new Vector3 that contains members from another vector rounded to the nearest integer value.

```
public static void Round(ref Vector3 value, out Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`result` Vector3

The rounded Vector3.

SmoothStep(Vector3, Vector3, float)

Creates a new Vector3 that contains cubic interpolation of the specified vectors.

```
public static Vector3 SmoothStep(Vector3 value1, Vector3 value2, float amount)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

`amount` float

Weighting value.

Returns

Vector3

Cubic interpolation of the specified vectors.

SmoothStep(ref Vector3, ref Vector3, float, out Vector3)

Creates a new Vector3 that contains cubic interpolation of the specified vectors.

```
public static void SmoothStep(ref Vector3 value1, ref Vector3 value2, float amount,
out Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

`amount` float

Weighting value.

`result` Vector3

Cubic interpolation of the specified vectors as an output parameter.

Subtract(Vector3, Vector3)

Creates a new Vector3 that contains subtraction of on Vector3 from a another.

```
public static Vector3 Subtract(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

Returns

Vector3

The result of the vector subtraction.

Subtract(ref Vector3, ref Vector3, out Vector3)

Creates a new Vector3 that contains subtraction of on Vector3 from a another.

```
public static void Subtract(ref Vector3 value1, ref Vector3 value2, out
Vector3 result)
```

Parameters

`value1` Vector3

Source Vector3.

`value2` Vector3

Source Vector3.

`result` Vector3

The result of the vector subtraction as an output parameter.

ToNumerics()

Returns a Vector3.

```
public Vector3 ToNumerics()
```

Returns

Vector3

ToString()

Returns a string representation of this Vector3 in the format: {X:[X] Y:[Y] Z:[Z]}

```
public override string ToString()
```

Returns

string

A string representation of this Vector3.

Transform(Vector3, Matrix)

Creates a new Vector3 that contains a transformation of 3d-vector by the specified Matrix.

```
public static Vector3 Transform(Vector3 position, Matrix matrix)
```

Parameters

`position` Vector3

Source Vector3.

`matrix` Matrix

The transformation Matrix.

Returns

Vector3

Transformed Vector3.

Transform(Vector3, Quaternion)

Creates a new Vector3 that contains a transformation of 3d-vector by the specified Quaternion, representing the rotation.

```
public static Vector3 Transform(Vector3 value, Quaternion rotation)
```

Parameters

`value` Vector3

Source Vector3.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

Returns

Vector3

Transformed Vector3.

Transform(ref Vector3, ref Matrix, out Vector3)

Creates a new Vector3 that contains a transformation of 3d-vector by the specified Matrix.

```
public static void Transform(ref Vector3 position, ref Matrix matrix, out
Vector3 result)
```

Parameters

`position` Vector3

Source Vector3.

`matrix` Matrix

The transformation Matrix.

`result` Vector3

Transformed Vector3 as an output parameter.

Transform(ref Vector3, ref Quaternion, out Vector3)

Creates a new Vector3 that contains a transformation of 3d-vector by the specified Quaternion, representing the rotation.

```
public static void Transform(ref Vector3 value, ref Quaternion rotation, out
Vector3 result)
```

Parameters

`value` Vector3

Source Vector3.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`result` Vector3

Transformed Vector3 as an output parameter.

Transform(Vector3[], ref Matrix, Vector3[])

Apply transformation on all vectors within array of Vector3 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector3[] sourceArray, ref Matrix matrix,
Vector3[] destinationArray)
```

Parameters

`sourceArray` Vector3[]

Source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector3[]

Destination array.

Transform(Vector3[], ref Quaternion, Vector3[])

Apply transformation on all vectors within array of Vector3 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector3[] sourceArray, ref Quaternion rotation,
Vector3[] destinationArray)
```

Parameters

`sourceArray` Vector3[]

Source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector3[]

Destination array.

Transform(Vector3[], int, ref Matrix, Vector3[], int, int)

Apply transformation on vectors within array of Vector3 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector3[] sourceArray, int sourceIndex, ref Matrix
matrix, Vector3[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector3[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector3[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector3 should be written.

`length` int

The number of vectors to be transformed.

Transform(Vector3[], int, ref Quaternion, Vector3[], int, int)

Apply transformation on vectors within array of Vector3 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector3[] sourceArray, int sourceIndex, ref Quaternion
rotation, Vector3[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector3[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector3[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector3 should be written.

`length` int

The number of vectors to be transformed.

TransformNormal(Vector3, Matrix)

Creates a new Vector3 that contains a transformation of the specified normal by the specified Matrix.

```
public static Vector3 TransformNormal(Vector3 normal, Matrix matrix)
```

Parameters

`normal` Vector3

Source Vector3 which represents a normal vector.

`matrix` Matrix

The transformation Matrix.

Returns

Vector3

Transformed normal.

TransformNormal(ref Vector3, ref Matrix, out Vector3)

Creates a new Vector3 that contains a transformation of the specified normal by the specified Matrix.

```
public static void TransformNormal(ref Vector3 normal, ref Matrix matrix, out
Vector3 result)
```

Parameters

`normal` Vector3

Source Vector3 which represents a normal vector.

`matrix` Matrix

The transformation Matrix.

`result` Vector3

Transformed normal as an output parameter.

TransformNormal(Vector3[], ref Matrix, Vector3[])

Apply transformation on all normals within array of Vector3 by the specified Matrix and places the results in an another array.

```
public static void TransformNormal(Vector3[] sourceArray, ref Matrix matrix,
Vector3[] destinationArray)
```

Parameters

`sourceArray` Vector3[]

Source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector3[]

Destination array.

TransformNormal(Vector3[], int, ref Matrix, Vector3[], int, int)

Apply transformation on normals within array of Vector3 by the specified Matrix and places the results in an another array.

```
public static void TransformNormal(Vector3[] sourceArray, int sourceIndex, ref
Matrix matrix, Vector3[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector3[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector3[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector3 should be written.

`length` int

The number of normals to be transformed.

Operators

operator +(Vector3, Vector3)

Adds two vectors.

```
public static Vector3 operator +(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3 on the left of the add sign.

`value2` Vector3

Source Vector3 on the right of the add sign.

Returns

Vector3

Sum of the vectors.

operator /(Vector3, Vector3)

Divides the components of a Vector3 by the components of another Vector3.

```
public static Vector3 operator /(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3 on the left of the div sign.

`value2` Vector3

Divisor Vector3 on the right of the div sign.

Returns

Vector3

The result of dividing the vectors.

operator /(Vector3, float)

Divides the components of a Vector3 by a scalar.

```
public static Vector3 operator /(Vector3 value1, float divider)
```

Parameters

`value1` Vector3

Source Vector3 on the left of the div sign.

`divider` float

Divisor scalar on the right of the div sign.

Returns

Vector3

The result of dividing a vector by a scalar.

operator ==(Vector3, Vector3)

Compares whether two Vector3 instances are equal.

```
public static bool operator ==(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Vector3 instance on the left of the equal sign.

`value2` Vector3

Vector3 instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Vector3(Vector3)

Converts a Vector3 to a Vector3.

```
public static implicit operator Vector3(Vector3 value)
```

Parameters

`value` Vector3

The converted value.

Returns

Vector3

operator !=(Vector3, Vector3)

Compares whether two Vector3 instances are not equal.

```
public static bool operator !=(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Vector3 instance on the left of the not equal sign.

`value2` Vector3

Vector3 instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Vector3, Vector3)

Multiplies the components of two vectors by each other.

```
public static Vector3 operator *(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3 on the left of the mul sign.

`value2` Vector3

Source Vector3 on the right of the mul sign.

Returns

Vector3

Result of the vector multiplication.

operator *(Vector3, float)

Multiplies the components of vector by a scalar.

```
public static Vector3 operator *(Vector3 value, float scaleFactor)
```

Parameters

`value` Vector3

Source Vector3 on the left of the mul sign.

`scaleFactor` float

Scalar value on the right of the mul sign.

Returns

Vector3

Result of the vector multiplication with a scalar.

operator *(float, Vector3)

Multiplies the components of vector by a scalar.

```
public static Vector3 operator *(float scaleFactor, Vector3 value)
```

Parameters

`scaleFactor` float

Scalar value on the left of the mul sign.

`value` Vector3

Source Vector3 on the right of the mul sign.

Returns

Vector3

Result of the vector multiplication with a scalar.

operator -(Vector3, Vector3)

Subtracts a Vector3 from a Vector3.

```
public static Vector3 operator -(Vector3 value1, Vector3 value2)
```

Parameters

`value1` Vector3

Source Vector3 on the left of the sub sign.

`value2` Vector3

Source Vector3 on the right of the sub sign.

Returns

Vector3

Result of the vector subtraction.

operator -(Vector3)

Inverts values in the specified Vector3.

```
public static Vector3 operator -(Vector3 value)
```

Parameters

`value` Vector3

Source Vector3 on the right of the sub sign.

Returns

Vector3

Result of the inversion.

Struct Vector4

Namespace: Microsoft.Xna.Framework

Assembly: MonoGame.Framework.dll

Describes a 4D-vector.

```
[TypeConverter(typeof(Vector4TypeConverter))]
[DataContract]
public struct Vector4 : IEquatable<Vector4>
```

Implements

IEquatable<Vector4>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Vector4(Vector2, float, float)

Constructs a 3d vector with X and Z from Vector2 and Z and W from the scalars.

```
public Vector4(Vector2 value, float z, float w)
```

Parameters

`value` Vector2

The x and y coordinates in 4d-space.

`z` float

The z coordinate in 4d-space.

`w` float

The w coordinate in 4d-space.

Vector4(Vector3, float)

Constructs a 3d vector with X, Y, Z from Vector3 and W from a scalar.

```
public Vector4(Vector3 value, float w)
```

Parameters

`value` Vector3

The x, y and z coordinates in 4d-space.

`w` float

The w coordinate in 4d-space.

Vector4(float)

Constructs a 4d vector with X, Y, Z and W set to the same value.

```
public Vector4(float value)
```

Parameters

`value` float

The x, y, z and w coordinates in 4d-space.

Vector4(float, float, float, float)

Constructs a 3d vector with X, Y, Z and W from four values.

```
public Vector4(float x, float y, float z, float w)
```

Parameters

`x` float

The x coordinate in 4d-space.

y float

The y coordinate in 4d-space.

z float

The z coordinate in 4d-space.

w float

The w coordinate in 4d-space.

Fields

W

The w coordinate of this Vector4.

```
[DataMember]
public float W
```

Field Value

float

X

The x coordinate of this Vector4.

```
[DataMember]
public float X
```

Field Value

float

Y

The y coordinate of this Vector4.

```
[DataMember]
public float Y
```

Field Value

float

Z

The z coordinate of this Vector4.

```
[DataMember]
public float Z
```

Field Value

float

Properties

One

Returns a Vector4 with components 1, 1, 1, 1.

```
public static Vector4 One { get; }
```

Property Value

Vector4

UnitW

Returns a Vector4 with components 0, 0, 0, 1.

```
public static Vector4 UnitW { get; }
```

Property Value

Vector4

UnitX

Returns a Vector4 with components 1, 0, 0, 0.

```
public static Vector4 UnitX { get; }
```

Property Value

Vector4

UnitY

Returns a Vector4 with components 0, 1, 0, 0.

```
public static Vector4 UnitY { get; }
```

Property Value

Vector4

UnitZ

Returns a Vector4 with components 0, 0, 1, 0.

```
public static Vector4 UnitZ { get; }
```

Property Value

Vector4

Zero

Returns a Vector4 with components 0, 0, 0, 0.

```
public static Vector4 Zero { get; }
```

Property Value

Vector4

Methods

Add(Vector4, Vector4)

Performs vector addition on `value1` and `value2`.

```
public static Vector4 Add(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector to add.

`value2` Vector4

The second vector to add.

Returns

Vector4

The result of the vector addition.

Add(ref Vector4, ref Vector4, out Vector4)

Performs vector addition on `value1` and `value2`, storing the result of the addition in `result`.

```
public static void Add(ref Vector4 value1, ref Vector4 value2, out Vector4 result)
```

Parameters

`value1` Vector4

The first vector to add.

`value2` Vector4

The second vector to add.

`result` Vector4

The result of the vector addition.

Barycentric(Vector4, Vector4, Vector4, float, float)

Creates a new Vector4 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 4d-triangle.

```
public static Vector4 Barycentric(Vector4 value1, Vector4 value2, Vector4 value3,
float amount1, float amount2)
```

Parameters

`value1` Vector4

The first vector of 4d-triangle.

`value2` Vector4

The second vector of 4d-triangle.

`value3` Vector4

The third vector of 4d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 4d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 4d-triangle.

Returns

Vector4

The cartesian translation of barycentric coordinates.

Barycentric(ref Vector4, ref Vector4, ref Vector4, float, float, out Vector4)

Creates a new Vector4 that contains the cartesian coordinates of a vector specified in barycentric coordinates and relative to 4d-triangle.

```
public static void Barycentric(ref Vector4 value1, ref Vector4 value2, ref Vector4
value3, float amount1, float amount2, out Vector4 result)
```

Parameters

`value1` Vector4

The first vector of 4d-triangle.

`value2` Vector4

The second vector of 4d-triangle.

`value3` Vector4

The third vector of 4d-triangle.

`amount1` float

Barycentric scalar `b2` which represents a weighting factor towards second vector of 4d-triangle.

`amount2` float

Barycentric scalar `b3` which represents a weighting factor towards third vector of 4d-triangle.

`result` Vector4

The cartesian translation of barycentric coordinates as an output parameter.

CatmullRom(Vector4, Vector4, Vector4, Vector4, float)

Creates a new Vector4 that contains CatmullRom interpolation of the specified vectors.

```
public static Vector4 CatmullRom(Vector4 value1, Vector4 value2, Vector4 value3,
Vector4 value4, float amount)
```

Parameters

`value1` Vector4

The first vector in interpolation.

`value2` Vector4

The second vector in interpolation.

`value3` Vector4

The third vector in interpolation.

`value4` Vector4

The fourth vector in interpolation.

`amount` float

Weighting factor.

Returns

Vector4

The result of CatmullRom interpolation.

CatmullRom(ref Vector4, ref Vector4, ref Vector4, ref Vector4, float, out Vector4)

Creates a new Vector4 that contains CatmullRom interpolation of the specified vectors.

```
public static void CatmullRom(ref Vector4 value1, ref Vector4 value2, ref Vector4
value3, ref Vector4 value4, float amount, out Vector4 result)
```

Parameters

`value1` Vector4

The first vector in interpolation.

`value2` Vector4

The second vector in interpolation.

`value3` Vector4

The third vector in interpolation.

`value4` Vector4

The fourth vector in interpolation.

`amount` float

Weighting factor.

`result` Vector4

The result of CatmullRom interpolation as an output parameter.

Ceiling()

Round the members of this Vector4 towards positive infinity.

```
public void Ceiling()
```

Ceiling(Vector4)

Creates a new Vector4 that contains members from another vector rounded towards positive infinity.

```
public static Vector4 Ceiling(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4.

Returns

Vector4

The rounded Vector4.

Ceiling(ref Vector4, out Vector4)

Creates a new Vector4 that contains members from another vector rounded towards positive infinity.

```
public static void Ceiling(ref Vector4 value, out Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`result` Vector4

The rounded Vector4.

Clamp(Vector4, Vector4, Vector4)

Clamps the specified value within a range.

```
public static Vector4 Clamp(Vector4 value1, Vector4 min, Vector4 max)
```

Parameters

`value1` Vector4

The value to clamp.

`min` Vector4

The min value.

`max` Vector4

The max value.

Returns

Vector4

The clamped value.

Clamp(ref Vector4, ref Vector4, ref Vector4, out Vector4)

Clamps the specified value within a range.

```
public static void Clamp(ref Vector4 value1, ref Vector4 min, ref Vector4 max, out
Vector4 result)
```

Parameters

`value1` Vector4

The value to clamp.

`min` Vector4

The min value.

`max` Vector4

The max value.

`result` Vector4

The clamped value as an output parameter.

Deconstruct(out float, out float, out float, out float)

Deconstruction method for Vector4.

```
public void Deconstruct(out float x, out float y, out float z, out float w)
```

Parameters

`x` float

`y` float

`z` float

`w` float

Distance(Vector4, Vector4)

Returns the distance between two vectors.

```
public static float Distance(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

Returns

float

The distance between two vectors.

Distance(ref Vector4, ref Vector4, out float)

Returns the distance between two vectors.

```
public static void Distance(ref Vector4 value1, ref Vector4 value2, out
float result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`result` float

The distance between two vectors as an output parameter.

DistanceSquared(Vector4, Vector4)

Returns the squared distance between two vectors.

```
public static float DistanceSquared(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

Returns

float

The squared distance between two vectors.

DistanceSquared(ref Vector4, ref Vector4, out float)

Returns the squared distance between two vectors.

```
public static void DistanceSquared(ref Vector4 value1, ref Vector4 value2, out
float result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`result` float

The squared distance between two vectors as an output parameter.

Divide(Vector4, Vector4)

Divides the components of a Vector4 by the components of another Vector4.

```
public static Vector4 Divide(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Divisor Vector4.

Returns

Vector4

The result of dividing the vectors.

Divide(Vector4, float)

Divides the components of a Vector4 by a scalar.

```
public static Vector4 Divide(Vector4 value1, float divider)
```

Parameters

`value1` Vector4

Source Vector4.

`divider` float

Divisor scalar.

Returns

Vector4

The result of dividing a vector by a scalar.

Divide(ref Vector4, ref Vector4, out Vector4)

Divides the components of a Vector4 by the components of another Vector4.

```
public static void Divide(ref Vector4 value1, ref Vector4 value2, out
Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Divisor Vector4.

`result` Vector4

The result of dividing the vectors as an output parameter.

Divide(ref Vector4, float, out Vector4)

Divides the components of a Vector4 by a scalar.

```
public static void Divide(ref Vector4 value1, float divider, out Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`divider` float

Divisor scalar.

`result` Vector4

The result of dividing a vector by a scalar as an output parameter.

Dot(Vector4, Vector4)

Returns a dot product of two vectors.

```
public static float Dot(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

Returns

float

The dot product of two vectors.

Dot(ref Vector4, ref Vector4, out float)

Returns a dot product of two vectors.

```
public static void Dot(ref Vector4 value1, ref Vector4 value2, out float result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`result` float

The dot product of two vectors as an output parameter.

Equals(Vector4)

Compares whether current instance is equal to specified Vector4.

```
public bool Equals(Vector4 other)
```

Parameters

`other` Vector4

The Vector4 to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

`true` if the instances are equal; `false` otherwise.

Floor()

Round the members of this Vector4 towards negative infinity.

```
public void Floor()
```

Floor(Vector4)

Creates a new Vector4 that contains members from another vector rounded towards negative infinity.

```
public static Vector4 Floor(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4.

Returns

Vector4

The rounded Vector4.

Floor(ref Vector4, out Vector4)

Creates a new Vector4 that contains members from another vector rounded towards negative infinity.

```
public static void Floor(ref Vector4 value, out Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`result` Vector4

The rounded Vector4.

GetHashCode()

Gets the hash code of this Vector4.

```
public override int GetHashCode()
```

Returns

int

Hash code of this Vector4.

Hermite(Vector4, Vector4, Vector4, Vector4, float)

Creates a new Vector4 that contains hermite spline interpolation.

```
public static Vector4 Hermite(Vector4 value1, Vector4 tangent1, Vector4 value2,
Vector4 tangent2, float amount)
```

Parameters

`value1` Vector4

The first position vector.

`tangent1` Vector4

The first tangent vector.

`value2` Vector4

The second position vector.

`tangent2` Vector4

The second tangent vector.

`amount` float

Weighting factor.

Returns

Vector4

The hermite spline interpolation vector.

Hermite(ref Vector4, ref Vector4, ref Vector4, ref Vector4, float, out Vector4)

Creates a new Vector4 that contains hermite spline interpolation.

```
public static void Hermite(ref Vector4 value1, ref Vector4 tangent1, ref Vector4
value2, ref Vector4 tangent2, float amount, out Vector4 result)
```

Parameters

`value1` Vector4

The first position vector.

`tangent1` Vector4

The first tangent vector.

`value2` Vector4

The second position vector.

`tangent2` Vector4

The second tangent vector.

`amount` float

Weighting factor.

`result` Vector4

The hermite spline interpolation vector as an output parameter.

Length()

Returns the length of this Vector4.

```
public float Length()
```

Returns

float

The length of this Vector4.

LengthSquared()

Returns the squared length of this Vector4.

```
public float LengthSquared()
```

Returns

float

The squared length of this Vector4.

Lerp(Vector4, Vector4, float)

Creates a new Vector4 that contains linear interpolation of the specified vectors.

```
public static Vector4 Lerp(Vector4 value1, Vector4 value2, float amount)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector4

The result of linear interpolation of the specified vectors.

Lerp(ref Vector4, ref Vector4, float, out Vector4)

Creates a new Vector4 that contains linear interpolation of the specified vectors.

```
public static void Lerp(ref Vector4 value1, ref Vector4 value2, float amount, out
Vector4 result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector4

The result of linear interpolation of the specified vectors as an output parameter.

LerpPrecise(Vector4, Vector4, float)

Creates a new Vector4 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(Vector4, Vector4, float). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static Vector4 LerpPrecise(Vector4 value1, Vector4 value2, float amount)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

Returns

Vector4

The result of linear interpolation of the specified vectors.

LerpPrecise(ref Vector4, ref Vector4, float, out Vector4)

Creates a new Vector4 that contains linear interpolation of the specified vectors. Uses LerpPrecise(float, float, float) on MathHelper for the interpolation. Less efficient but more precise compared to Lerp(ref Vector4, ref Vector4, float, out Vector4). See remarks section of LerpPrecise(float, float, float) on MathHelper for more info.

```
public static void LerpPrecise(ref Vector4 value1, ref Vector4 value2, float amount,
out Vector4 result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`amount` float

Weighting value(between 0.0 and 1.0).

`result` Vector4

The result of linear interpolation of the specified vectors as an output parameter.

Max(Vector4, Vector4)

Creates a new Vector4 that contains a maximal values from the two vectors.

```
public static Vector4 Max(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

Returns

Vector4

The Vector4 with maximal values from the two vectors.

Max(ref Vector4, ref Vector4, out Vector4)

Creates a new Vector4 that contains a maximal values from the two vectors.

```
public static void Max(ref Vector4 value1, ref Vector4 value2, out Vector4 result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`result` Vector4

The Vector4 with maximal values from the two vectors as an output parameter.

Min(Vector4, Vector4)

Creates a new Vector4 that contains a minimal values from the two vectors.

```
public static Vector4 Min(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

Returns

Vector4

The Vector4 with minimal values from the two vectors.

Min(ref Vector4, ref Vector4, out Vector4)

Creates a new Vector4 that contains a minimal values from the two vectors.

```
public static void Min(ref Vector4 value1, ref Vector4 value2, out Vector4 result)
```

Parameters

`value1` Vector4

The first vector.

`value2` Vector4

The second vector.

`result` Vector4

The Vector4 with minimal values from the two vectors as an output parameter.

Multiply(Vector4, Vector4)

Creates a new Vector4 that contains a multiplication of two vectors.

```
public static Vector4 Multiply(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

Returns

Vector4

The result of the vector multiplication.

Multiply(Vector4, float)

Creates a new Vector4 that contains a multiplication of Vector4 and a scalar.

```
public static Vector4 Multiply(Vector4 value1, float scaleFactor)
```

Parameters

`value1` Vector4

Source Vector4.

`scaleFactor` float

Scalar value.

Returns

Vector4

The result of the vector multiplication with a scalar.

Multiply(ref Vector4, ref Vector4, out Vector4)

Creates a new Vector4 that contains a multiplication of two vectors.

```
public static void Multiply(ref Vector4 value1, ref Vector4 value2, out
Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

`result` Vector4

The result of the vector multiplication as an output parameter.

Multiply(ref Vector4, float, out Vector4)

Creates a new Vector4 that contains a multiplication of Vector4 and a scalar.

```
public static void Multiply(ref Vector4 value1, float scaleFactor, out
Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`scaleFactor` float

Scalar value.

`result` Vector4

The result of the multiplication with a scalar as an output parameter.

Negate(Vector4)

Creates a new Vector4 that contains the specified vector inversion.

```
public static Vector4 Negate(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4.

Returns

Vector4

The result of the vector inversion.

Negate(ref Vector4, out Vector4)

Creates a new Vector4 that contains the specified vector inversion.

```
public static void Negate(ref Vector4 value, out Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`result` Vector4

The result of the vector inversion as an output parameter.

Normalize()

Turns this Vector4 to a unit vector with the same direction.

```
public void Normalize()
```

Normalize(Vector4)

Creates a new Vector4 that contains a normalized values from another vector.

```
public static Vector4 Normalize(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4.

Returns

Vector4

Unit vector.

Normalize(ref Vector4, out Vector4)

Creates a new Vector4 that contains a normalized values from another vector.

```
public static void Normalize(ref Vector4 value, out Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`result` Vector4

Unit vector as an output parameter.

Round()

Round the members of this Vector4 to the nearest integer value.

```
public void Round()
```

Round(Vector4)

Creates a new Vector4 that contains members from another vector rounded to the nearest integer value.

```
public static Vector4 Round(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4.

Returns

Vector4

The rounded Vector4.

Round(ref Vector4, out Vector4)

Creates a new Vector4 that contains members from another vector rounded to the nearest integer value.

```
public static void Round(ref Vector4 value, out Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`result` Vector4

The rounded Vector4.

SmoothStep(Vector4, Vector4, float)

Creates a new Vector4 that contains cubic interpolation of the specified vectors.

```
public static Vector4 SmoothStep(Vector4 value1, Vector4 value2, float amount)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

`amount` float

Weighting value.

Returns

Vector4

Cubic interpolation of the specified vectors.

SmoothStep(ref Vector4, ref Vector4, float, out Vector4)

Creates a new Vector4 that contains cubic interpolation of the specified vectors.

```
public static void SmoothStep(ref Vector4 value1, ref Vector4 value2, float amount,
out Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

`amount` float

Weighting value.

`result` Vector4

Cubic interpolation of the specified vectors as an output parameter.

Subtract(Vector4, Vector4)

Creates a new Vector4 that contains subtraction of on Vector4 from a another.

```
public static Vector4 Subtract(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

Returns

Vector4

The result of the vector subtraction.

Subtract(ref Vector4, ref Vector4, out Vector4)

Creates a new Vector4 that contains subtraction of on Vector4 from a another.

```
public static void Subtract(ref Vector4 value1, ref Vector4 value2, out
Vector4 result)
```

Parameters

`value1` Vector4

Source Vector4.

`value2` Vector4

Source Vector4.

`result` Vector4

The result of the vector subtraction as an output parameter.

ToNumerics()

Returns a Vector4.

```
public Vector4 ToNumerics()
```

Returns

Vector4

ToString()

Returns a string representation of this Vector4 in the format: {X:[X] Y:[Y] Z:[Z] W:[W]}

```
public override string ToString()
```

Returns

string

A string representation of this Vector4.

Transform(Vector2, Matrix)

Creates a new Vector4 that contains a transformation of 2d-vector by the specified Matrix.

```
public static Vector4 Transform(Vector2 value, Matrix matrix)
```

Parameters

`value` Vector2

Source Vector2.

`matrix` Matrix

The transformation Matrix.

Returns

Vector4

Transformed Vector4.

Transform(Vector2, Quaternion)

Creates a new Vector4 that contains a transformation of 2d-vector by the specified Quaternion.

```
public static Vector4 Transform(Vector2 value, Quaternion rotation)
```

Parameters

`value` Vector2

Source Vector2.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

Returns

Vector4

Transformed Vector4.

Transform(ref Vector2, ref Matrix, out Vector4)

Creates a new Vector4 that contains a transformation of 2d-vector by the specified Matrix.

```
public static void Transform(ref Vector2 value, ref Matrix matrix, out
Vector4 result)
```

Parameters

`value` Vector2

Source Vector2.

`matrix` Matrix

The transformation Matrix.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(ref Vector2, ref Quaternion, out Vector4)

Creates a new Vector4 that contains a transformation of 2d-vector by the specified Quaternion.

```
public static void Transform(ref Vector2 value, ref Quaternion rotation, out
Vector4 result)
```

Parameters

`value` Vector2

Source Vector2.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(Vector3, Matrix)

Creates a new Vector4 that contains a transformation of 3d-vector by the specified Matrix.

```
public static Vector4 Transform(Vector3 value, Matrix matrix)
```

Parameters

`value` Vector3

Source Vector3.

`matrix` Matrix

The transformation Matrix.

Returns

Vector4

Transformed Vector4.

Transform(Vector3, Quaternion)

Creates a new Vector4 that contains a transformation of 3d-vector by the specified Quaternion.

```
public static Vector4 Transform(Vector3 value, Quaternion rotation)
```

Parameters

`value` Vector3

Source Vector3.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

Returns

Vector4

Transformed Vector4.

Transform(ref Vector3, ref Matrix, out Vector4)

Creates a new Vector4 that contains a transformation of 3d-vector by the specified Matrix.

```
public static void Transform(ref Vector3 value, ref Matrix matrix, out
Vector4 result)
```

Parameters

`value` Vector3

Source Vector3.

`matrix` Matrix

The transformation Matrix.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(ref Vector3, ref Quaternion, out Vector4)

Creates a new Vector4 that contains a transformation of 3d-vector by the specified Quaternion.

```
public static void Transform(ref Vector3 value, ref Quaternion rotation, out
Vector4 result)
```

Parameters

`value` Vector3

Source Vector3.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(Vector4, Matrix)

Creates a new Vector4 that contains a transformation of 4d-vector by the specified Matrix.

```
public static Vector4 Transform(Vector4 value, Matrix matrix)
```

Parameters

`value` Vector4

Source Vector4.

`matrix` Matrix

The transformation Matrix.

Returns

Vector4

Transformed Vector4.

Transform(Vector4, Quaternion)

Creates a new Vector4 that contains a transformation of 4d-vector by the specified Quaternion.

```
public static Vector4 Transform(Vector4 value, Quaternion rotation)
```

Parameters

`value` Vector4

Source Vector4.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

Returns

Vector4

Transformed Vector4.

Transform(ref Vector4, ref Matrix, out Vector4)

Creates a new Vector4 that contains a transformation of 4d-vector by the specified Matrix.

```
public static void Transform(ref Vector4 value, ref Matrix matrix, out
Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`matrix` Matrix

The transformation Matrix.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(ref Vector4, ref Quaternion, out Vector4)

Creates a new Vector4 that contains a transformation of 4d-vector by the specified Quaternion.

```
public static void Transform(ref Vector4 value, ref Quaternion rotation, out
Vector4 result)
```

Parameters

`value` Vector4

Source Vector4.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`result` Vector4

Transformed Vector4 as an output parameter.

Transform(Vector4[], ref Matrix, Vector4[])

Apply transformation on all vectors within array of Vector4 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector4[] sourceArray, ref Matrix matrix,
Vector4[] destinationArray)
```

Parameters

`sourceArray` Vector4[]

Source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector4[]

Destination array.

Transform(Vector4[], ref Quaternion, Vector4[])

Apply transformation on all vectors within array of Vector4 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector4[] sourceArray, ref Quaternion rotation,
Vector4[] destinationArray)
```

Parameters

`sourceArray` Vector4[]

Source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector4[]

Destination array.

Transform(Vector4[], int, ref Matrix, Vector4[], int, int)

Apply transformation on vectors within array of Vector4 by the specified Matrix and places the results in an another array.

```
public static void Transform(Vector4[] sourceArray, int sourceIndex, ref Matrix
matrix, Vector4[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector4[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`matrix` Matrix

The transformation Matrix.

`destinationArray` Vector4[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector4 should be written.

`length` int

The number of vectors to be transformed.

Transform(Vector4[], int, ref Quaternion, Vector4[], int, int)

Apply transformation on vectors within array of Vector4 by the specified Quaternion and places the results in an another array.

```
public static void Transform(Vector4[] sourceArray, int sourceIndex, ref Quaternion
rotation, Vector4[] destinationArray, int destinationIndex, int length)
```

Parameters

`sourceArray` Vector4[]

Source array.

`sourceIndex` int

The starting index of transformation in the source array.

`rotation` Quaternion

The Quaternion which contains rotation transformation.

`destinationArray` Vector4[]

Destination array.

`destinationIndex` int

The starting index in the destination array, where the first Vector4 should be written.

`length` int

The number of vectors to be transformed.

Operators

operator +(Vector4, Vector4)

Adds two vectors.

```
public static Vector4 operator +(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4 on the left of the add sign.

`value2` Vector4

Source Vector4 on the right of the add sign.

Returns

Vector4

Sum of the vectors.

operator /(Vector4, Vector4)

Divides the components of a Vector4 by the components of another Vector4.

```
public static Vector4 operator /(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4 on the left of the div sign.

`value2` Vector4

Divisor Vector4 on the right of the div sign.

Returns

Vector4

The result of dividing the vectors.

operator /(Vector4, float)

Divides the components of a Vector4 by a scalar.

```
public static Vector4 operator /(Vector4 value1, float divider)
```

Parameters

`value1` Vector4

Source Vector4 on the left of the div sign.

`divider` float

Divisor scalar on the right of the div sign.

Returns

Vector4

The result of dividing a vector by a scalar.

operator ==(Vector4, Vector4)

Compares whether two Vector4 instances are equal.

```
public static bool operator ==(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Vector4 instance on the left of the equal sign.

`value2` Vector4

Vector4 instance on the right of the equal sign.

Returns

bool

`true` if the instances are equal; `false` otherwise.

implicit operator Vector4(Vector4)

Converts a Vector4 to a Vector4.

```
public static implicit operator Vector4(Vector4 value)
```

Parameters

`value` Vector4

The converted value.

Returns

Vector4

operator !=(Vector4, Vector4)

Compares whether two Vector4 instances are not equal.

```
public static bool operator !=(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Vector4 instance on the left of the not equal sign.

`value2` Vector4

Vector4 instance on the right of the not equal sign.

Returns

bool

`true` if the instances are not equal; `false` otherwise.

operator *(Vector4, Vector4)

Multiplies the components of two vectors by each other.

```
public static Vector4 operator *(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4 on the left of the mul sign.

`value2` Vector4

Source Vector4 on the right of the mul sign.

Returns

Vector4

Result of the vector multiplication.

operator *(Vector4, float)

Multiplies the components of vector by a scalar.

```
public static Vector4 operator *(Vector4 value, float scaleFactor)
```

Parameters

`value` Vector4

Source Vector4 on the left of the mul sign.

`scaleFactor` float

Scalar value on the right of the mul sign.

Returns

Vector4

Result of the vector multiplication with a scalar.

operator *(float, Vector4)

Multiplies the components of vector by a scalar.

```
public static Vector4 operator *(float scaleFactor, Vector4 value)
```

Parameters

`scaleFactor` float

Scalar value on the left of the mul sign.

`value` Vector4

Source Vector4 on the right of the mul sign.

Returns

Vector4

Result of the vector multiplication with a scalar.

operator -(Vector4, Vector4)

Subtracts a Vector4 from a Vector4.

```
public static Vector4 operator -(Vector4 value1, Vector4 value2)
```

Parameters

`value1` Vector4

Source Vector4 on the left of the sub sign.

`value2` Vector4

Source Vector4 on the right of the sub sign.

Returns

Vector4

Result of the vector subtraction.

operator -(Vector4)

Inverts values in the specified Vector4.

```
public static Vector4 operator -(Vector4 value)
```

Parameters

`value` Vector4

Source Vector4 on the right of the sub sign.

Returns

Vector4

Result of the inversion.

Namespace Microsoft.Xna.Framework.Audio

Classes

AudioEmitter

Represents a 3D audio emitter. Used to simulate 3D audio effects.

AudioEngine

Class used to create and manipulate code audio objects.

AudioListener

Represents a 3D audio listener. Used when simulating 3D Audio.

Cue

Manages the playback of a sound or set of sounds.

DynamicSoundEffectInstance

A SoundEffectInstance for which the audio buffer is provided by the game at run time.

InstancePlayLimitException

The exception thrown when the system attempts to play more SoundEffectInstances than allotted.

Microphone

Provides microphones capture features.

NoAudioHardwareException

The exception thrown when no audio hardware is present, or driver issues are detected.

NoMicrophoneConnectedException

The exception thrown when no audio hardware is present, or driver issues are detected.

SoundBank

Represents a collection of Cues.

SoundEffect

Represents a loaded sound resource.

SoundEffectInstance

Represents a single instance of a playing, paused, or stopped sound.

WaveBank

Represents a collection of wave files.

Structs

AudioCategory

Provides functionality for manipulating multiple sounds at a time.

Enums

AudioChannels

Represents how many channels are used in the audio data.

AudioStopOptions

Controls how Cue objects should cease playback when told to stop.

MicrophoneState

Microphone state.

SoundState

Described the playback state of a SoundEffectInstance.

Struct AudioCategory

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Provides functionality for manipulating multiple sounds at a time.

```
public struct AudioCategory : IEquatable<AudioCategory>
```

Implements

IEquatable<AudioCategory>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Name

Gets the category's friendly name.

```
public string Name { get; }
```

Property Value

string

Methods

Equals(AudioCategory)

Determines whether two AudioCategory instances are equal.

```
public bool Equals(AudioCategory other)
```

Parameters

`other` AudioCategory

AudioCategory to compare with this instance.

Returns

bool

true if the objects are equal or false if they aren't

Equals(object)

Determines whether two AudioCategory instances are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

Object to compare with this instance.

Returns

bool

true if the objects are equal or false if they aren't.

GetHashCode()

Gets the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

Hash code for this object.

Pause()

Pauses all associated sounds.

```
public void Pause()
```

Resume()

Resumes all associated paused sounds.

```
public void Resume()
```

SetVolume(float)

Set the volume for this AudioCategory.

```
public void SetVolume(float volume)
```

Parameters

`volume` float

The new volume of the category.

Exceptions

ArgumentException

If the volume is less than zero.

Stop(AudioStopOptions)

Stops all associated sounds.

```
public void Stop(AudioStopOptions options)
```

Parameters

`options` AudioStopOptions

ToString()

Returns the name of this AudioCategory

```
public override string ToString()
```

Returns

string

Friendly name of the AudioCategory

Operators

operator ==(AudioCategory, AudioCategory)

Determines whether two AudioCategory instances are equal.

```
public static bool operator ==(AudioCategory first, AudioCategory second)
```

Parameters

`first` AudioCategory

First AudioCategory instance to compare.

`second` AudioCategory

Second AudioCategory instance to compare.

Returns

bool

true if the objects are equal or false if they aren't.

operator !=(AudioCategory, AudioCategory)

Determines whether two AudioCategory instances are not equal.

```
public static bool operator !=(AudioCategory first, AudioCategory second)
```

Parameters

`first` AudioCategory

First AudioCategory instance to compare.

`second` AudioCategory

Second AudioCategory instance to compare.

Returns

bool

true if the objects are not equal or false if they are.

Enum AudioChannels

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents how many channels are used in the audio data.

```
public enum AudioChannels
```

Fields

`Mono = 1`

Single channel.

`Stereo = 2`

Two channels.

Class AudioEmitter

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a 3D audio emitter. Used to simulate 3D audio effects.

```
public class AudioEmitter
```

Inheritance

object ← AudioEmitter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AudioEmitter()

Initializes a new AudioEmitter instance.

```
public AudioEmitter()
```

Properties

DopplerScale

Gets or sets a scale applied to the Doppler effect between the AudioEmitter and an AudioListener.

```
public float DopplerScale { get; set; }
```

Property Value

float

Remarks

Defaults to 1.0

A value of 1.0 leaves the Doppler effect unmodified.

Forward

Gets or sets the emitter's forward vector.

```
public Vector3 Forward { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Forward. (new Vector3(0, 0, -1))

Used with AudioListener.Velocity to calculate Doppler values.

The Forward and Up values must be orthonormal.

Position

Gets or sets the position of this emitter.

```
public Vector3 Position { get; set; }
```

Property Value

Vector3

Up

Gets or sets the emitter's Up vector.

```
public Vector3 Up { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Up. (new Vector3(0, -1, 1)).

The Up and Forward vectors must be orthonormal.

Velocity

Gets or sets the emitter's velocity vector.

```
public Vector3 Velocity { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Zero.

This value is only used when calculating Doppler values.

Class AudioEngine

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Class used to create and manipulate code audio objects.

```
public class AudioEngine : IDisposable
```

Inheritance

object ← AudioEngine

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AudioEngine(string)

Initializes a new instance of the AudioEngine class by reading out the specified XACT `settingsFile`

```
public AudioEngine(string settingsFile)
```

Parameters

`settingsFile` string

Path to a XACT settings file.

Exceptions

ArgumentException

Invoked if `settingsFile` is IsNullOrEmpty(string)

AudioEngine(string, TimeSpan, string)

Initializes a new instance of the AudioEngine class by reading out the specified XACT `settingsFile`

```
[Obsolete("Use AudioEngine(string settingsFile) instead. The lookAheadTime and
rendererId parameters are not used.")]
public AudioEngine(string settingsFile, TimeSpan lookAheadTime, string rendererId)
```

Parameters

`settingsFile` string

Path to a XACT settings file.

`lookAheadTime` TimeSpan

`Not in use:` use the AudioEngine(string) constructor instead!

`rendererId` string

`Not in use:` use the AudioEngine(string) constructor instead!

Fields

ContentVersion

The current content version.

```
public const int ContentVersion = 39
```

Field Value

int

Properties

IsDisposed

Is true if the AudioEngine has been disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Methods

Dispose()

Disposes the AudioEngine.

```
public void Dispose()
```

~AudioEngine()

```
protected ~AudioEngine()
```

GetCategory(string)

Returns an audio category by name.

```
public AudioCategory GetCategory(string name)
```

Parameters

`name` string

Friendly name of the category to get.

Returns

AudioCategory

The AudioCategory with a matching name. Throws an exception if not found.

GetGlobalVariable(string)

Gets the value of a global variable.

```
public float GetGlobalVariable(string name)
```

Parameters

`name` string

Friendly name of the variable.

Returns

float

float value of the queried variable.

Remarks

A global variable has global scope. It can be accessed by all code within a project.

SetGlobalVariable(string, float)

Sets the value of a global variable.

```
public void SetGlobalVariable(string name, float value)
```

Parameters

`name` string

Friendly name of the variable.

`value` float

Value of the global variable.

Update()

Performs periodic work required by the audio engine.

```
public void Update()
```

Remarks

Must be called at least once per frame.

Events

Disposing

This event is triggered when the AudioEngine is disposed.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

Class AudioListener

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a 3D audio listener. Used when simulating 3D Audio.

```
public class AudioListener
```

Inheritance

object ← AudioListener

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AudioListener()

Create an AudioListener instance.

```
public AudioListener()
```

Properties

Forward

Gets or sets the listener's forward vector.

```
public Vector3 Forward { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Forward. (new Vector3(0, 0, -1))

Used with AudioListener.Velocity and AudioEmitter.Velocity to calculate Doppler values.

The Forward and Up vectors must be orthonormal.

Position

Gets or sets the listener's position.

```
public Vector3 Position { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Zero.

Up

Gets or sets the listener's up vector..

```
public Vector3 Up { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Up (New Vector3(0, -1, 0)).

Used with AudioListener.Velocity and AudioEmitter.Velocity to calculate Doppler values.

The values of the Forward and Up vectors must be orthonormal.

Velocity

Gets or sets the listener's velocity vector.

```
public Vector3 Velocity { get; set; }
```

Property Value

Vector3

Remarks

Defaults to Vector3.Zero.

Scaled by DopplerScale to calculate the Doppler effect value applied to a Cue.

This value is only used to calculate Doppler values.

Enum AudioStopOptions

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Controls how Cue objects should cease playback when told to stop.

```
public enum AudioStopOptions
```

Fields

`AsAuthored = 0`

Stop normally, playing any pending release phases or transitions.

`Immediate = 1`

Immediately stops the cue, ignoring any pending release phases or transitions.

Class Cue

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Manages the playback of a sound or set of sounds.

```
public class Cue : IDisposable
```

Inheritance

object ← Cue

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Cues are comprised of one or more sounds.

Cues also define specific properties such as pitch or volume.

Cues are referenced through SoundBank objects.

Properties

IsCreated

Returns whether the cue has been created.

```
public bool IsCreated { get; }
```

Property Value

bool

Remarks

Current implementation will always return false.

IsDisposed

Is true if the Cue has been disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsPaused

Indicates whether or not the cue is currently paused.

```
public bool IsPaused { get; }
```

Property Value

bool

Remarks

IsPlaying and IsPaused both return true if a cue is paused while playing.

IsPlaying

Indicates whether or not the cue is currently playing.

```
public bool IsPlaying { get; }
```

Property Value

bool

Remarks

IsPlaying and IsPaused both return true if a cue is paused while playing.

IsPrepared

Returns whether the cue is prepared to play.

```
public bool IsPrepared { get; }
```

Property Value

bool

Remarks

This property returns true only if the cue is prepared but has not yet been played. Cues that are played return true for IsPlaying if they are currently playing or true for IsStopped if they are stopped.

IsPreparing

Returns whether the cue is preparing to play.

```
public bool IsPreparing { get; }
```

Property Value

bool

Remarks

Current implementation will always return false.

IsStopped

Indicates whether or not the cue is currently stopped.

```
public bool IsStopped { get; }
```

Property Value

bool

IsStopping

Returns whether the cue is stopping playback.

```
public bool IsStopping { get; }
```

Property Value

bool

Remarks

Current implementation will always return false.

Name

Gets the friendly name of the cue.

```
public string Name { get; }
```

Property Value

string

Remarks

The friendly name is a value set from the designer.

Methods

Apply3D(AudioListener, AudioEmitter)

Updates the simulated 3D Audio settings calculated between an AudioEmitter and AudioListener.

```
public void Apply3D(AudioListener listener, AudioEmitter emitter)
```

Parameters

`listener` AudioListener

The listener to calculate.

`emitter` AudioEmitter

The emitter to calculate.

Remarks

This must be called before Play().

Calling this method automatically converts the sound to monoaural and sets the speaker mix for any sound played by this cue to a value calculated with the listener's and emitter's positions. Any stereo information in the sound will be discarded.

Dispose()

Disposes the Cue.

```
public void Dispose()
```

GetVariable(string)

Gets a cue-instance variable value based on its friendly name.

```
public float GetVariable(string name)
```

Parameters

`name` string

Friendly name of the variable.

Returns

float

Value of the variable.

Remarks

Cue-instance variables are useful when multiple instantiations of a single cue (and its associated sounds) are required (for example, a "car" cue where there may be more than one car at any given time). While a global variable allows multiple audio elements to be controlled in unison, a cue instance variable grants discrete control of each instance of a cue, even for each copy of the same cue.

The friendly name is a value set from the designer.

Pause()

Pauses playback.

```
public void Pause()
```

Play()

Requests playback of a prepared or preparing Cue.

```
public void Play()
```

Remarks

Calling Play when the Cue already is playing can result in an InvalidOperationException.

Resume()

Resumes playback of a paused Cue.

```
public void Resume()
```

SetVariable(string, float)

Sets the value of a cue-instance variable based on its friendly name.

```
public void SetVariable(string name, float value)
```

Parameters

`name` string

Friendly name of the variable to set.

`value` float

Value to assign to the variable.

Remarks

The friendly name is a value set from the designer.

Stop(AudioStopOptions)

Stops playback of a Cue.

```
public void Stop(AudioStopOptions options)
```

Parameters

`options` AudioStopOptions

Specifies if the sound should play any pending release phases or transitions before stopping.

Events

Disposing

This event is triggered when the Cue is disposed.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

Class DynamicSoundEffectInstance

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

A SoundEffectInstance for which the audio buffer is provided by the game at run time.

```
public sealed class DynamicSoundEffectInstance : SoundEffectInstance, IDisposable
```

Inheritance

object ← SoundEffectInstance ← DynamicSoundEffectInstance

Implements

IDisposable

Inherited Members

SoundEffectInstance.Pan , SoundEffectInstance.Pitch , SoundEffectInstance.Volume , SoundEffectInstance.IsDisposed , SoundEffectInstance.Apply3D(AudioListener, AudioEmitter) , SoundEffectInstance.Apply3D(AudioListener[], AudioEmitter) , SoundEffectInstance.Dispose() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DynamicSoundEffectInstance(int, AudioChannels)

```
public DynamicSoundEffectInstance(int sampleRate, AudioChannels channels)
```

Parameters

`sampleRate` int

Sample rate, in Hertz (Hz).

`channels` AudioChannels

Number of channels (mono or stereo).

Properties

IsLooped

This value has no effect on DynamicSoundEffectInstance. It may not be set.

```
public override bool IsLooped { get; set; }
```

Property Value

bool

PendingBufferCount

Returns the number of audio buffers queued for playback.

```
public int PendingBufferCount { get; }
```

Property Value

int

State

Gets the SoundEffectInstance's current playback state.

```
public override SoundState State { get; }
```

Property Value

SoundState

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GetSampleDuration(int)

Returns the duration of an audio buffer of the specified size, based on the settings of this instance.

```
public TimeSpan GetSampleDuration(int sizeInBytes)
```

Parameters

`sizeInBytes` int

Size of the buffer, in bytes.

Returns

TimeSpan

The playback length of the buffer.

GetSampleSizeInBytes(TimeSpan)

Returns the size, in bytes, of a buffer of the specified duration, based on the settings of this instance.

```
public int GetSampleSizeInBytes(TimeSpan duration)
```

Parameters

`duration` TimeSpan

The playback length of the buffer.

Returns

int

The data size of the buffer, in bytes.

Pause()

Pauses playback of the DynamicSoundEffectInstance.

```
public override void Pause()
```

Play()

Plays or resumes the DynamicSoundEffectInstance.

```
public override void Play()
```

Resume()

Resumes playback of the DynamicSoundEffectInstance.

```
public override void Resume()
```

Stop()

Immediately stops playing the DynamicSoundEffectInstance.

```
public override void Stop()
```

Remarks

Calling this also releases all queued buffers.

Stop(bool)

Stops playing the DynamicSoundEffectInstance. If the `immediate` parameter is false, this call has no effect.

```
public override void Stop(bool immediate)
```

Parameters

`immediate` bool

When set to false, this call has no effect.

Remarks

Calling this also releases all queued buffers.

SubmitBuffer(byte[])

Queues an audio buffer for playback.

```
public void SubmitBuffer(byte[] buffer)
```

Parameters

`buffer` byte[]

The buffer containing PCM audio data.

Remarks

The buffer length must conform to alignment requirements for the audio format.

SubmitBuffer(byte[], int, int)

Queues an audio buffer for playback.

```
public void SubmitBuffer(byte[] buffer, int offset, int count)
```

Parameters

`buffer` byte[]

The buffer containing PCM audio data.

`offset` int

The starting position of audio data.

`count` int

The amount of bytes to use.

Remarks

The buffer length must conform to alignment requirements for the audio format.

Events

BufferNeeded

The event that occurs when the number of queued audio buffers is less than or equal to 2.

```
public event EventHandler<EventArgs> BufferNeeded
```

Event Type

EventHandler<EventArgs>

Remarks

This event may occur when Play() is called or during playback when a buffer is completed.

Class InstancePlayLimitException

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

The exception thrown when the system attempts to play more SoundEffectInstances than allotted.

```
[DataContract]
public sealed class InstancePlayLimitException : ExternalException, ISerializable
```

Inheritance

object ← Exception ← SystemException ← ExternalException ← InstancePlayLimitException

Implements

ISerializable

Inherited Members

ExternalException.ToString() , ExternalException.ErrorCode , Exception.GetBaseException() , Exception.GetType() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Remarks

Most platforms have a hard limit on how many sounds can be played simultaneously. This exception is thrown when that limit is exceeded.

Class Microphone

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Provides microphones capture features.

```
public sealed class Microphone
```

Inheritance

object ← Microphone

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Fields

Name

Returns the friendly name of the microphone.

```
public readonly string Name
```

Field Value

string

Properties

All

Returns all compatible microphones.

```
public static ReadOnlyCollection<Microphone> All { get; }
```

Property Value

ReadOnlyCollection<Microphone>

BufferDuration

Gets or sets the capture buffer duration. This value must be greater than 100 milliseconds, lower than 1000 milliseconds, and must be 10 milliseconds aligned (BufferDuration % 10 == 10).

```
public TimeSpan BufferDuration { get; set; }
```

Property Value

TimeSpan

Default

Returns the default microphone.

```
public static Microphone Default { get; }
```

Property Value

Microphone

IsHeadset

Determines if the microphone is a wired headset. Note: XNA could know if a headset microphone was plugged in an Xbox 360 controller but MonoGame can't. Hence, this is always true on mobile platforms, and always false otherwise.

```
public bool IsHeadset { get; }
```

Property Value

bool

SampleRate

Returns the sample rate of the captured audio. Note: default value is 44100hz

```
public int SampleRate { get; }
```

Property Value

int

State

Returns the state of the Microphone.

```
public MicrophoneState State { get; }
```

Property Value

MicrophoneState

Methods

GetData(byte[])

Gets the latest available data from the microphone.

```
public int GetData(byte[] buffer)
```

Parameters

`buffer` byte[]

Buffer, in bytes, of the captured data (16-bit PCM).

Returns

int

The buffer size, in bytes, of the captured data.

GetData(byte[], int, int)

Gets the latest available data from the microphone.

```
public int GetData(byte[] buffer, int offset, int count)
```

Parameters

`buffer` byte[]

Buffer, in bytes, of the captured data (16-bit PCM).

`offset` int

Byte offset.

`count` int

Amount, in bytes.

Returns

int

The buffer size, in bytes, of the captured data.

GetSampleDuration(int)

Returns the duration based on the size of the buffer (assuming 16-bit PCM data).

```
public TimeSpan GetSampleDuration(int sizeInBytes)
```

Parameters

`sizeInBytes` int

Size, in bytes

Returns

TimeSpan

TimeSpan of the duration.

GetSampleSizeInBytes(TimeSpan)

Returns the size, in bytes, of the array required to hold the specified duration of 16-bit PCM data.

```
public int GetSampleSizeInBytes(TimeSpan duration)
```

Parameters

`duration` TimeSpan

TimeSpan of the duration of the sample.

Returns

int

Size, in bytes, of the buffer.

Start()

Starts microphone capture.

```
public void Start()
```

Stop()

Stops microphone capture.

```
public void Stop()
```

Events

BufferReady

Event fired when the audio data are available.

```
public event EventHandler<EventArgs> BufferReady
```

Event Type

EventHandler<EventArgs>

Enum MicrophoneState

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Microphone state.

```
public enum MicrophoneState
```

Fields

`Started = 0`

The Microphone audio capture has started.

`Stopped = 1`

The Microphone audio capture has stopped.

Class NoAudioHardwareException

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

The exception thrown when no audio hardware is present, or driver issues are detected.

```
[DataContract]
public sealed class NoAudioHardwareException : ExternalException, ISerializable
```

Inheritance

object ← Exception ← SystemException ← ExternalException ← NoAudioHardwareException

Implements

ISerializable

Inherited Members

ExternalException.ToString() , ExternalException.ErrorCode , Exception.GetBaseException() , Exception.GetType() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

NoAudioHardwareException(string)

```
public NoAudioHardwareException(string msg)
```

Parameters

`msg` string

A message describing the error.

NoAudioHardwareException(string, Exception)

```
public NoAudioHardwareException(string msg, Exception innerException)
```

Parameters

`msg` string

A message describing the error.

`innerException` Exception

The exception that is the underlying cause of the current exception. If not null, the current exception is raised in a try/catch block that handled the innerException.

Class NoMicrophoneConnectedException

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

The exception thrown when no audio hardware is present, or driver issues are detected.

```
[DataContract]
public sealed class NoMicrophoneConnectedException : Exception, ISerializable
```

Inheritance

object ← Exception ← NoMicrophoneConnectedException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

NoMicrophoneConnectedException(string)

```
public NoMicrophoneConnectedException(string msg)
```

Parameters

`msg` string

A message describing the error.

NoMicrophoneConnectedException(string, Exception)

```
public NoMicrophoneConnectedException(string msg, Exception innerException)
```

Parameters

`msg` string

A message describing the error.

`innerException` Exception

The exception that is the underlying cause of the current exception. If not null, the current exception is raised in a try/catch block that handled the innerException.

Class SoundBank

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a collection of Cues.

```
public class SoundBank : IDisposable
```

Inheritance

object ← SoundBank

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SoundBank(AudioEngine, string)

```
public SoundBank(AudioEngine audioEngine, string fileName)
```

Parameters

`audioEngine` AudioEngine

AudioEngine that will be associated with this sound bank.

`fileName` string

Path to a .xsb SoundBank file.

Properties

IsDisposed

Is true if the SoundBank has been disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsInUse

Is true if the SoundBank has any live Cues in use.

```
public bool IsInUse { get; }
```

Property Value

bool

Methods

Dispose()

Disposes the SoundBank.

```
public void Dispose()
```

~SoundBank()

```
protected ~SoundBank()
```

GetCue(string)

Returns a pooled Cue object.

```
public Cue GetCue(string name)
```

Parameters

`name` string

Friendly name of the cue to get.

Returns

Cue

a unique Cue object from a pool.

Remarks

Cue instances are unique, even when sharing the same name. This allows multiple instances to simultaneously play.

PlayCue(string)

Plays a cue.

```
public void PlayCue(string name)
```

Parameters

`name` string

Name of the cue to play.

PlayCue(string, AudioListener, AudioEmitter)

Plays a cue with static 3D positional information.

```
public void PlayCue(string name, AudioListener listener, AudioEmitter emitter)
```

Parameters

`name` string

The name of the cue to play.

`listener` AudioListener

The listener state.

`emitter` AudioEmitter

The cue emitter state.

Remarks

Commonly used for short lived effects. To dynamically change the 3D positional information on a cue over time use GetCue(string) and Apply3D(AudioListener, AudioEmitter).

Events

Disposing

This event is triggered when the SoundBank is disposed.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

Class SoundEffect

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a loaded sound resource.

```
public sealed class SoundEffect : IDisposable
```

Inheritance

object ← SoundEffect

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

A SoundEffect represents the buffer used to hold audio data and metadata. SoundEffectInstances are used to play from SoundEffects. Multiple SoundEffectInstance objects can be created and played from the same SoundEffect object.

The only limit on the number of loaded SoundEffects is restricted by available memory. When a SoundEffect is disposed, all SoundEffectInstances created from it will become invalid.

SoundEffect.Play() can be used for 'fire and forget' sounds. If advanced playback controls like volume or pitch is required, use SoundEffect.CreateInstance().

Constructors

SoundEffect(byte[], int, AudioChannels)

Create a sound effect.

```
public SoundEffect(byte[] buffer, int sampleRate, AudioChannels channels)
```

Parameters

`buffer` byte[]

The buffer with the sound data.

`sampleRate` int

The sound data sample rate in hertz.

`channels` AudioChannels

The number of channels in the sound data.

Remarks

This only supports uncompressed 16bit PCM wav data.

SoundEffect(byte[], int, int, int, AudioChannels, int, int)

Create a sound effect.

```
public SoundEffect(byte[] buffer, int offset, int count, int sampleRate,
AudioChannels channels, int loopStart, int loopLength)
```

Parameters

`buffer` byte[]

The buffer with the sound data.

`offset` int

The offset to the start of the sound data in bytes.

`count` int

The length of the sound data in bytes.

`sampleRate` int

The sound data sample rate in hertz.

`channels` AudioChannels

The number of channels in the sound data.

`loopStart` int

The position where the sound should begin looping in samples.

`loopLength` int

The duration of the sound data loop in samples.

Remarks

This only supports uncompressed 16bit PCM wav data.

Properties

DistanceScale

Gets or sets the scale of distance calculations.

```
public static float DistanceScale { get; set; }
```

Property Value

float

Remarks

DistanceScale defaults to 1.0 and must be greater than 0.0.

Higher values reduce the rate of falloff between the sound and listener.

DopplerScale

Gets or sets the scale of Doppler calculations applied to sounds.

```
public static float DopplerScale { get; set; }
```

Property Value

float

Remarks

DopplerScale defaults to 1.0 and must be greater or equal to 0.0

Affects the relative velocity of emitters and listeners.

Higher values more dramatically shift the pitch for the given relative velocity of the emitter and listener.

Duration

Gets the duration of the SoundEffect.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

IsDisposed

Indicates whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

MasterVolume

Gets or sets the master volume scale applied to all SoundEffectInstances.

```
public static float MasterVolume { get; set; }
```

Property Value

float

Remarks

Each SoundEffectInstance has its own Volume property that is independent to SoundEffect.MasterVolume. During playback SoundEffectInstance.Volume is multiplied by SoundEffect.MasterVolume.

This property is used to adjust the volume on all current and newly created SoundEffectInstances. The volume of an individual SoundEffectInstance can be adjusted on its own.

Name

Gets or sets the asset name of the SoundEffect.

```
public string Name { get; set; }
```

Property Value

string

SpeedOfSound

Returns the speed of sound used when calculating the Doppler effect..

```
public static float SpeedOfSound { get; set; }
```

Property Value

float

Remarks

Defaults to 343.5. Value is measured in meters per second.

Has no effect on distance attenuation.

Methods

CreateInstance()

Creates a new SoundEffectInstance for this SoundEffect.

```
public SoundEffectInstance CreateInstance()
```

Returns

SoundEffectInstance

A new SoundEffectInstance for this SoundEffect.

Remarks

Creating a SoundEffectInstance before calling SoundEffectInstance.Play() allows you to access advanced playback features, such as volume, pitch, and 3D positioning.

Dispose()

Releases the resources held by this SoundEffect.

```
public void Dispose()
```

~SoundEffect()

Releases unmanaged resources and performs other cleanup operations before the SoundEffect is reclaimed by garbage collection.

```
protected ~SoundEffect()
```

FromFile(string)

Creates a new SoundEffect object based on the specified data stream. This internally calls From Stream(Stream).

```
public static SoundEffect FromFile(string path)
```

Parameters

`path` string

The path to the audio file.

Returns

SoundEffect

The SoundEffect loaded from the given file.

Remarks

The stream must point to the head of a valid wave file in the RIFF bitstream format. The formats supported are:

- 8-bit unsigned PCM16-bit signed PCM24-bit signed PCM32-bit IEEE float PCMMS-ADPCM 4-bit compressedIMA/ADPCM (IMA4) 4-bit compressed

FromStream(Stream)

Creates a new SoundEffect object based on the specified data stream.

```
public static SoundEffect FromStream(Stream stream)
```

Parameters

`stream` Stream

A stream containing the wave data.

Returns

SoundEffect

A new SoundEffect object.

Remarks

The stream must point to the head of a valid wave file in the RIFF bitstream format. The formats supported are:

- 8-bit unsigned PCM16-bit signed PCM24-bit signed PCM32-bit IEEE float PCMMS-ADPCM 4-bit compressedIMA/ADPCM (IMA4) 4-bit compressed

GetSampleDuration(int, int, AudioChannels)

Returns the duration for 16-bit PCM audio.

```
public static TimeSpan GetSampleDuration(int sizeInBytes, int sampleRate,
AudioChannels channels)
```

Parameters

`sizeInBytes` int

The length of the audio data in bytes.

`sampleRate` int

Sample rate, in Hertz (Hz). Must be between 8000 Hz and 48000 Hz

`channels` AudioChannels

Number of channels in the audio data.

Returns

TimeSpan

The duration of the audio data.

GetSampleSizeInBytes(TimeSpan, int, AudioChannels)

Returns the data size in bytes for 16bit PCM audio.

```
public static int GetSampleSizeInBytes(TimeSpan duration, int sampleRate,
AudioChannels channels)
```

Parameters

`duration` TimeSpan

The total duration of the audio data.

`sampleRate` int

Sample rate, in Hertz (Hz), of audio data. Must be between 8,000 and 48,000 Hz.

`channels` AudioChannels

Number of channels in the audio data.

Returns

int

The size in bytes of a single sample of audio data.

Initialize()

Initializes the sound system for SoundEffect support. This method is automatically called when a SoundEffect is loaded, a DynamicSoundEffectInstance is created, or Microphone.All is queried. You can however call this method manually (preferably in, or before the Game constructor) to catch any Exception that may occur during the sound system initialization (and act accordingly).

```
public static void Initialize()
```

Play()

Gets an internal SoundEffectInstance and plays it.

```
public bool Play()
```

Returns

bool

True if a SoundEffectInstance was successfully played, false if not.

Remarks

Play returns false if more SoundEffectInstances are currently playing then the platform allows.

To loop a sound or apply 3D effects, call SoundEffect.CreateInstance() and SoundEffectInstance.Play() instead.

SoundEffectInstances used by SoundEffect.Play() are pooled internally.

Play(float, float, float)

Gets an internal SoundEffectInstance and plays it with the specified volume, pitch, and panning.

```
public bool Play(float volume, float pitch, float pan)
```

Parameters

`volume` float

Volume, ranging from 0.0 (silence) to 1.0 (full volume). Volume during playback is scaled by SoundEffect.MasterVolume.

`pitch` float

Pitch adjustment, ranging from -1.0 (down an octave) to 0.0 (no change) to 1.0 (up an octave).

`pan` float

Panning, ranging from -1.0 (left speaker) to 0.0 (centered), 1.0 (right speaker).

Returns

bool

True if a SoundEffectInstance was successfully created and played, false if not.

Remarks

Play returns false if more SoundEffectInstances are currently playing then the platform allows.

To apply looping or simulate 3D audio, call SoundEffect.CreateInstance() and SoundEffectInstance.Play() instead.

SoundEffectInstances used by SoundEffect.Play() are pooled internally.

Class SoundEffectInstance

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a single instance of a playing, paused, or stopped sound.

```
public class SoundEffectInstance : IDisposable
```

Inheritance

object ← SoundEffectInstance

Implements

IDisposable

Derived

DynamicSoundEffectInstance

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

SoundEffectInstances are created through SoundEffect.CreateInstance() and used internally by SoundEffect.Play()

Properties

IsDisposed

Indicates whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsLooped

Enables or Disables whether the SoundEffectInstance should repeat after playback.

```
public virtual bool IsLooped { get; set; }
```

Property Value

bool

Remarks

This value has no effect on an already playing sound.

Pan

Gets or sets the pan, or speaker balance..

```
public float Pan { get; set; }
```

Property Value

float

Pan value ranging from -1.0 (left speaker) to 0.0 (centered), 1.0 (right speaker). Values outside of this range will throw an exception.

Pitch

Gets or sets the pitch adjustment.

```
public float Pitch { get; set; }
```

Property Value

float

Pitch adjustment, where -1.0 is down an octave, 0.0 is no change, and 1.0 is up an octave.

Remarks

Android and iOS will be clamped this to [-1.0, 1.0]. DesktopGL and WindowsDX will clamp this to [-10.0, 10.0].

State

Gets the SoundEffectInstance's current playback state.

```
public virtual SoundState State { get; }
```

Property Value

SoundState

Volume

Gets or sets the volume of the SoundEffectInstance.

```
public float Volume { get; set; }
```

Property Value

float

Volume, ranging from 0.0 (silence) to 1.0 (full volume). Volume during playback is scaled by SoundEffect.MasterVolume.

Remarks

This is the volume relative to SoundEffect.MasterVolume. Before playback, this Volume property is multiplied by SoundEffect.MasterVolume when determining the final mix volume.

Methods

Apply3D(AudioListener, AudioEmitter)

Applies 3D positioning to the SoundEffectInstance using a single listener.

```
public void Apply3D(AudioListener listener, AudioEmitter emitter)
```

Parameters

`listener` AudioListener

Data about the listener.

`emitter` AudioEmitter

Data about the source of emission.

Apply3D(AudioListener[], AudioEmitter)

Applies 3D positioning to the SoundEffectInstance using multiple listeners.

```
public void Apply3D(AudioListener[] listeners, AudioEmitter emitter)
```

Parameters

`listeners` AudioListener[]

Data about each listener.

`emitter` AudioEmitter

Data about the source of emission.

Dispose()

Releases the resources held by this SoundEffectInstance.

```
public void Dispose()
```

Dispose(bool)

Releases the resources held by this SoundEffectInstance.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

If set to `true`, Dispose was called explicitly.

Remarks

If the disposing parameter is true, the Dispose method was called explicitly. This means that managed objects referenced by this instance should be disposed or released as required. If the disposing parameter is false, Dispose was called by the finalizer and no managed objects should be touched because we do not know if they are still valid or not at that time. Unmanaged resources should always be released.

~SoundEffectInstance()

Releases unmanaged resources and performs other cleanup operations before the SoundEffectInstance is reclaimed by garbage collection.

```
protected ~SoundEffectInstance()
```

Pause()

Pauses playback of a SoundEffectInstance.

```
public virtual void Pause()
```

Remarks

Paused instances can be resumed with SoundEffectInstance.Play() or SoundEffectInstance.Resume().

Play()

Plays or resumes a SoundEffectInstance.

```
public virtual void Play()
```

Remarks

Throws an exception if more sounds are playing than the platform allows.

Resume()

Resumes playback for a SoundEffectInstance.

```
public virtual void Resume()
```

Remarks

Only has effect on a SoundEffectInstance in a paused state.

Stop()

Immediately stops playing a SoundEffectInstance.

```
public virtual void Stop()
```

Stop(bool)

Stops playing a SoundEffectInstance, either immediately or as authored.

```
public virtual void Stop(bool immediate)
```

Parameters

`immediate` bool

Determined whether the sound stops immediately, or after playing its release phase and/or transitions.

Remarks

Stopping a sound with the immediate argument set to false will allow it to play any release phases, such as fade, before coming to a stop.

Enum SoundState

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Described the playback state of a SoundEffectInstance.

```
public enum SoundState
```

Fields

`Paused = 1`

The SoundEffectInstance is currently paused.

`Playing = 0`

The SoundEffectInstance is currently playing.

`Stopped = 2`

The SoundEffectInstance is currently stopped.

Class WaveBank

Namespace: Microsoft.Xna.Framework.Audio

Assembly: MonoGame.Framework.dll

Represents a collection of wave files.

```
public class WaveBank : IDisposable
```

Inheritance

object ← WaveBank

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

WaveBank(AudioEngine, string)

```
public WaveBank(AudioEngine audioEngine, string nonStreamingWaveBankFilename)
```

Parameters

`audioEngine` AudioEngine

Instance of the AudioEngine to associate this wave bank with.

`nonStreamingWaveBankFilename` string

Path to the .xwb file to load.

Remarks

This constructor immediately loads all wave data into memory at once.

WaveBank(AudioEngine, string, int, short)

```
public WaveBank(AudioEngine audioEngine, string streamingWaveBankFilename, int
offset, short packetsize)
```

Parameters

`audioEngine` AudioEngine

Instance of the AudioEngine to associate this wave bank with.

`streamingWaveBankFilename` string

Path to the .xwb to stream from.

`offset` int

DVD sector-aligned offset within the wave bank data file.

`packetsize` short

Stream packet size, in sectors, to use for each stream. The minimum value is 2.

Remarks

This constructor streams wave data as needed.

Note that packetsize is in sectors, which is 2048 bytes.

AudioEngine.Update() must be called at least once before using data from a streaming wave bank.

Properties

IsDisposed

Is true if the WaveBank has been disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsInUse

```
public bool IsInUse { get; }
```

Property Value

bool

IsPrepared

```
public bool IsPrepared { get; }
```

Property Value

bool

Methods

Dispose()

Disposes the WaveBank.

```
public void Dispose()
```

~WaveBank()

```
protected ~WaveBank()
```

Events

Disposing

This event is triggered when the WaveBank is disposed.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

Namespace Microsoft.Xna.Framework.Content

Classes

ArrayReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

ContentLoadException

The exception that's thrown when an error occurs when loading content.

ContentManager

The ContentManager is a run-time component which loads managed objects from .xnb binary files produced by the design time MonoGame Content Builder. It also manages the lifespan of the loaded objects, disposing the content manager will also dispose any assets which are themselves IDisposable.

ContentReader

Defines a worker object that implements most of ContentManger.Load. A new content reader is constructed for each asset loaded.

ContentReaderExtensions

Helper extension methods for ContentReader.

ContentSerializerAttribute

Defines a custom Attribute that marks a field or property to control how it is serialized or to indicate that protected or private data should be included in serialization.

ContentSerializerCollectionItemNameAttribute

This is used to specify the XML element name to use for each item in a collection.

ContentSerializerIgnoreAttribute

Defines a custom Attribute that marks a field or property to indicate that it should not be included in serialization.

ContentSerializerRuntimeTypeAttribute

This is used to specify the type to use when deserializing this object at runtime.

ContentSerializerTypeVersionAttribute

This is used to specify the version when deserializing this object at runtime.

ContentTypeReader

Defines the core behavior of content type readers used for reading a specific managed type from an .xnb binary format and provides a base for derived classes.

ContentTypeReaderManager

Defines a manager that constructs and keeps track of ContentTypeReader objects.

ContentTypeReader<T>

Defines the core behavior of content type readers used for reading a specific managed type from an .xnb binary format and provides a base for derived classes. Derive from this class to add new data types to the content pipeline system.

DictionaryReader<TKey, TValue>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

EnumReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

ListReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

MultiArrayReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

NullableReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

ReflectiveReader<T>

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If Load<T>(string) throws an NotSupportedException, the message should provide insights on how to fix it.

ResourceContentManager

Subclass of ContentManager, which is specialized to read from *.resx* resource files rather than directly from individual files on disk.

Class ArrayReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class ArrayReader<T> : ContentTypeReader<T[]>
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<T[]> ← ArrayReader<T>

Inherited Members

ContentTypeReader<T[]>.Read(ContentReader, object) ,
ContentTypeReader<T[]>.Read(ContentReader, T[]) ,
ContentTypeReader.CanDeserializeIntoExistingObject , ContentTypeReader.TargetType ,
ContentTypeReader.TypeVersion , ContentTypeReader.Read(ContentReader, object) ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ArrayReader()

```
public ArrayReader()
```

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, T[])

```
protected override T[] Read(ContentReader input, T[] existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` T[]

Returns

T[]

Class ContentLoadException

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

The exception that's thrown when an error occurs when loading content.

```
public class ContentLoadException : Exception, ISerializable
```

Inheritance

object ← Exception ← ContentLoadException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

ContentLoadException()

Create a new ContentLoadException instance.

```
public ContentLoadException()
```

ContentLoadException(string)

Create a new ContentLoadException instance with the specified message.

```
public ContentLoadException(string message)
```

Parameters

`message` string

The message of the exception.

ContentLoadException(string, Exception)

Create a new ContentLoadException instance with the specified message and inner exception.

```
public ContentLoadException(string message, Exception innerException)
```

Parameters

`message` string

The message of the exception.

`innerException` Exception

The inner exception.

Class ContentManager

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

The ContentManager is a run-time component which loads managed objects from .xnb binary files produced by the design time MonoGame Content Builder. It also manages the lifespan of the loaded objects, disposing the content manager will also dispose any assets which are themselves IDisposable .

```
public class ContentManager : IDisposable
```

Inheritance

object ← ContentManager

Implements

IDisposable

Derived

ResourceContentManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentManager(IServiceProvider)

Initializes a new instance of the ContentMangaer.

```
public ContentManager(IServiceProvider serviceProvider)
```

Parameters

`serviceProvider` IServiceProvider

The service provider that the ContentManager should use to locate services.

Remarks

By default, the ContentManager searches for content in the directory where the executable is located.

When creating a new ContentManager, if no instance of Game is otherwise required by the application, it is often better to create a new class that implements the IServiceProvider interface rather than creating an instance of Game just to create a new instance of GraphicsDeviceManager.

Exceptions

ArgumentNullException

The `serviceProvider` parameter is null.

ContentManager(IServiceProvider, string)

Initializes a new instance of the ContentMangaer.

```
public ContentManager(IServiceProvider serviceProvider, string rootDirectory)
```

Parameters

`serviceProvider` IServiceProvider

The service provider that the ContentManager should use to locate services.

`rootDirectory` string

The root directory the ContentManager will search for content in.

Remarks

By default, the ContentManager searches for content in the directory where the executable is located.

When creating a new ContentManager, if no instance of Game is otherwise required by the application, it is often better to create a new class that implements the IServiceProvider interface rather than creating an instance of Game just to create a new instance of GraphicsDeviceManager.

Exceptions

ArgumentNullException

The `serviceProvider` parameter is null.

Properties

LoadedAssets

```
protected virtual Dictionary<string, object> LoadedAssets { get; }
```

Property Value

Dictionary<string, object>

RootDirectory

Gets or Sets the root directory that this ContentManager will search for assets in.

```
public string RootDirectory { get; set; }
```

Property Value

string

ServiceProvider

Gets the service provider instance used by this ContentManager.

```
public IServiceProvider ServiceProvider { get; }
```

Property Value

IServiceProvider

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

true to release both managed and unmanaged resources; false to release only unmanaged resources.

~ContentManager()

```
protected ~ContentManager()
```

LoadLocalized<T>(string)

Loads an asset that has been processed by the Content Pipeline.

```
public virtual T LoadLocalized<T>(string assetName)
```

Parameters

`assetName` string

The asset name, relative to the ContentManager.RootDirectory, and not including the .xnb extension.

Returns

T

The loaded asset. Repeated calls to load the same asset will return the same object instance.

Type Parameters

T

The type of asset to load.

Effect, Model, SoundEffect, Song, SpriteFont, Texture, Texture2D, and TextureCube are all supported by default by the standard Content Pipeline processor, but additional types may be loaded by extending the processor.

Remarks

This method attempts to load the asset based on the CurrentCulture searching for the asset by name and appending it with with the culture name (e.g. "assetName.en-US") or two letter ISO language name (e.g. "assetName.en"). If unsuccessful in finding the asset with the culture information appended, it will fall back to loading the default asset.

Before a ContentManager can load an asset, you need to add the asset to your game project using the steps described in Adding Content - MonoGame.

Exceptions

ArgumentNullException

The `assetName` parameter is null or an empty string.

ObjectDisposedException

This was called after the ContentManger was disposed.

ContentLoadException

The type of the `assetName` in the file does not match the type of asset requested as specified by `T`.

-or-

A content file matching the `assetName` parameter could not be found.

-or-

The specified path in the `assetName` parameter is invalid (for example, a directory in the path does not exist).

-or-

An error occurred while opening the content file.

Load<T>(string)

Loads an asset that has been processed by the Content Pipeline.

```
public virtual T Load<T>(string assetName)
```

Parameters

`assetName` string

The asset name, relative to the ContentManager.RootDirectory, and not including the .xnb extension.

Returns

T

The loaded asset. Repeated calls to load the same asset will return the same object instance.

Type Parameters

`T`

The type of asset to load.

Effect, Model, SoundEffect, Song, SpriteFont, Texture, Texture2D, and TextureCube are all supported by default by the standard Content Pipeline processor, but additional types may be loaded by extending the processor.

Remarks

Before a ContentManager can load an asset, you need to add the asset to your game project using the steps described in Adding Content - MonoGame.
PNG, JPG/JPEG and BMP files can be loaded as Texture2D without using the content pipeline. The assetName must not contain extension.

Exceptions

ArgumentNullException

The `assetName` parameter is null or an empty string.

ObjectDisposedException

This was called after the ContentManger was disposed.

ContentLoadException

The type of the `assetName` in the file does not match the type of asset requested as specified by `T`.

-or-

A content file matching the `assetName` parameter could not be found.

-or-

The specified path in the `assetName` parameter is invalid (for example, a directory in the path does not exist).

-or-

An error occurred while opening the content file.

OpenStream(string)

```
protected virtual Stream OpenStream(string assetName)
```

Parameters

`assetName` string

Returns

Stream

ReadAsset<T>(string, Action<IDisposable>)

```
protected T ReadAsset<T>(string assetName, Action<IDisposable>
recordDisposableObject)
```

Parameters

`assetName` string

`recordDisposableObject` Action<IDisposable>

Returns

T

Type Parameters

`T`

ReloadAsset<T>(string, T)

```
protected virtual void ReloadAsset<T>(string originalAssetName, T currentAsset)
```

Parameters

`originalAssetName` string

`currentAsset` T

Type Parameters

`T`

ReloadGraphicsAssets()

```
protected virtual void ReloadGraphicsAssets()
```

Unload()

Unloads all assets that were loaded by this ContentManger.

```
public virtual void Unload()
```

Remarks

If an asset being unloaded implements the IDisposable interface, then the IDisposable.Dispose method will be called before unloading.

UnloadAsset(string)

Unloads a single asset that was loaded by this ContentManager.

```
public virtual void UnloadAsset(string assetName)
```

Parameters

`assetName` string

The asset name, relative to the ContentManager.RootDirectory, and not including the .xnb extension.

Remarks

If the asset being unloaded implements the IDisposable interface, then the IDisposable.Dispose method will be called before unloading.

Exceptions

ArgumentNullException

The `assetName` parameter is null or an empty string.

ObjectDisposedException

This was called after the ContentManger was disposed.

UnloadAssets(IList<string>)

Unloads a set of assets loaded by this ContentManager where each element in the provided collection represents the name of an asset to unload.

```
public virtual void UnloadAssets(IList<string> assetNames)
```

Parameters

`assetNames` IList<string>

The collection containing the names of assets to unload.

Remarks

If the asset being unloaded implements the IDisposable interface, then the IDisposable.Dispose method will be called before unloading.

Exceptions

ArgumentNullException

If the `assetNames` parameter is null.

-or-

If an element in the collection null or an empty string.

ObjectDisposedException

This was called after the ContentManger was disposed.

Class ContentReader

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines a worker object that implements most of ContentManger.Load. A new content reader is constructed for each asset loaded.

```
public sealed class ContentReader : BinaryReader, IDisposable
```

Inheritance

object ← BinaryReader ← ContentReader

Implements

IDisposable

Inherited Members

BinaryReader.Close() , BinaryReader.Dispose() , BinaryReader.PeekChar() , BinaryReader.Read() , BinaryReader.Read(byte[], int, int) , BinaryReader.Read(char[], int, int) , BinaryReader.Read(Span<byte>) , BinaryReader.Read(Span<char>) , BinaryReader.Read7BitEncodedInt() , BinaryReader.Read7BitEncodedInt64() , BinaryReader.ReadBoolean() , BinaryReader.ReadByte() , BinaryReader.ReadBytes(int) , BinaryReader.ReadChar() , BinaryReader.ReadChars(int) , BinaryReader.ReadDecimal() , BinaryReader.ReadDouble() , BinaryReader.ReadHalf() , BinaryReader.ReadInt16() , BinaryReader.ReadInt32() , BinaryReader.ReadInt64() , BinaryReader.ReadSByte() , BinaryReader.ReadSingle() , BinaryReader.ReadString() , BinaryReader.ReadUInt16() , BinaryReader.ReadUInt32() , BinaryReader.ReadUInt64() , BinaryReader.BaseStream , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

ContentReaderExtensions.GetGraphicsDevice(ContentReader)

Properties

AssetName

Gets the name of the asset currently being read by this content reader.

```
public string AssetName { get; }
```

Property Value

string

ContentManager

Gets a reference to the ContentManager instance that is using this content reader.

```
public ContentManager ContentManager { get; }
```

Property Value

ContentManager

Methods

ReadColor()

Reads a Color from the underlying stream.

```
public Color ReadColor()
```

Returns

Color

The Color that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadExternalReference<T>()

Reads a relative link to an external file from the underlying stream and returns an asset of type `T` loaded from the external file.

```
public T ReadExternalReference<T>()
```

Returns

T

The asset of type `T` loaded from the external file at the relative link read from the underlying stream. If the relative link read was null or an empty string, then the default implementation of type `T` is returned.

Type Parameters

T

The type of asset to expected to be loaded from the external file.

Exceptions

ArgumentNullException

The relative link to the external file read is null or an empty string.

ObjectDisposedException

This was called after the ContentReader.ContentManager was disposed.

-or

The stream is closed.

ContentLoadException

The type of asset in the external file does not match the type specified by `T`.

-or-

A content file matching the relative link to the external file could not be found.

-or-

The specified path in the relative link to the external file read is invalid (for example, a directory in the path does not exist).

-or-

An error occurred while opening the external file.

EndOfStreamException

The end of stream is reached.

IOException

An I/O error occurred.

ReadMatrix()

Reads a Matrix value from the underlying stream.

```
public Matrix ReadMatrix()
```

Returns

Matrix

The Matrix that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadObject<T>()

Reads a single managed object from the underlying stream as an instance of the specified type. Can be called recursively.

```
public T ReadObject<T>()
```

Returns

T

The object that was read.

Type Parameters

T

The type of object to read.

Exceptions

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

FormatException

The stream is corrupted.

ContentLoadException

Type reader index read from stream is out of bounds

ReadObject<T>(ContentTypeReader)

Reads a single managed object from the underlying stream as an instance of the specified type. Can be called recursively.

```
public T ReadObject<T>(ContentTypeReader typeReader)
```

Parameters

`typeReader` ContentTypeReader

The ContentTypeReader to use to read the object.

Returns

T

The object that was read.

Type Parameters

`T`

The type of object to read.

Exceptions

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

FormatException

The stream is corrupted.

ContentLoadException

Type reader index read from stream is out of bounds

ReadObject<T>(ContentTypeReader, T)

Reads a single managed object from the underlying stream as an instance of the specified type. Can be called recursively.

```
public T ReadObject<T>(ContentTypeReader typeReader, T existingInstance)
```

Parameters

`typeReader` ContentTypeReader

The ContentTypeReader to use to read the object.

`existingInstance` T

An existing object to write into.

Returns

T

The object that was read.

Type Parameters

`T`

The type of object to read.

Exceptions

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

FormatException

The stream is corrupted.

ContentLoadException

Type reader index read from stream is out of bounds

ReadObject<T>(T)

Reads a single managed object from the underlying stream as an instance of the specified type. Can be called recursively.

```
public T ReadObject<T>(T existingInstance)
```

Parameters

`existingInstance` T

An existing object to write into.

Returns

T

The object that was read.

Type Parameters

`T`

The type of object to read.

Exceptions

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

FormatException

The stream is corrupted.

ContentLoadException

Type reader index read from stream is out of bounds

ReadQuaternion()

Reads a Quaternion from the underlying stream.

```
public Quaternion ReadQuaternion()
```

Returns

Quaternion

The Quaternion that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadRawObject<T>()

Reads a single managed object from the current stream as an instance of the specified type. If you specify a base class of the actual object type, this method reads data only from the base type.

```
public T ReadRawObject<T>()
```

Returns

T

The object that was read.

Type Parameters

T

The type of object to read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadRawObject<T>(ContentTypeReader)

Reads a single managed object from the current stream as an instance of the specified type. If you specify a base class of the actual object type, this method reads data only from the base type.

```
public T ReadRawObject<T>(ContentTypeReader typeReader)
```

Parameters

`typeReader` ContentTypeReader

The ContentTypeReader to use to read the object.

Returns

T

The object that was read.

Type Parameters

T

The type of object to read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadRawObject<T>(ContentTypeReader, T)

Reads a single managed object from the current stream as an instance of the specified type. If you specify a base class of the actual object type, this method reads data only from the base type.

```
public T ReadRawObject<T>(ContentTypeReader typeReader, T existingInstance)
```

Parameters

`typeReader` ContentTypeReader

The ContentTypeReader to use to read the object.

`existingInstance` T

An existing object to write into.

Returns

T

The object that was read.

Type Parameters

`T`

The type of object to read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadRawObject<T>(T)

Reads a single managed object from the current stream as an instance of the specified type. If you specify a base class of the actual object type, this method reads data only from the base type.

```
public T ReadRawObject<T>(T existingInstance)
```

Parameters

`existingInstance` T

An existing object to write into.

Returns

T

The object that was read.

Type Parameters

`T`

The type of object to read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadSharedResource<T>(Action<T>)

Reads a shared resource ID and records it for subsequent fix-up.

```
public void ReadSharedResource<T>(Action<T> fixup)
```

Parameters

`fixup` Action<T>

The fix-up action to perform.

Type Parameters

`T`

The type of the shared resource.

Exceptions

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

FormatException

The stream is corrupted.

ContentLoadException

The type of the shared resource read does not match the specified `T` type.

ReadVector2()

Reads a Vector2 from the underlying stream.

```
public Vector2 ReadVector2()
```

Returns

Vector2

The Vector2 that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadVector3()

Reads a Vector3 from the underlying stream.

```
public Vector3 ReadVector3()
```

Returns

Vector3

The Vector3 that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

ReadVector4()

Reads a Vector4 from the underlying stream.

```
public Vector4 ReadVector4()
```

Returns

Vector4

The Vector4 that was read.

Exceptions

EndOfStreamException

The end of stream is reached.

ObjectDisposedException

The stream is closed.

IOException

An I/O error occurred.

Class ContentReaderExtensions

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Helper extension methods for ContentReader.

```
public static class ContentReaderExtensions
```

Inheritance

object ← ContentReaderExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetGraphicsDevice(ContentReader)

Returns the GraphicsDevice instance from the service provider of the ContentManager associated with this content reader.

```
public static GraphicsDevice GetGraphicsDevice(this ContentReader contentReader)
```

Parameters

`contentReader` ContentReader

Returns

GraphicsDevice

The GraphicsDevice.

Exceptions

InvalidOperationException

The ContentManager.ServiceProvider does not contain a GraphicsDevice instance.

Class ContentSerializerAttribute

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines a custom Attribute that marks a field or property to control how it is serialized or to indicate that protected or private data should be included in serialization.

```
[AttributeUsage(AttributeTargets.Property|AttributeTargets.Field)]
public sealed class ContentSerializerAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentSerializerAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,

Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentSerializerAttribute()

Initializes a new instance of the content serializer attribute.

```
public ContentSerializerAttribute()
```

Properties

AllowNull

Gets or Sets a value that indicates whether this member can have a null value (default is true).

```
public bool AllowNull { get; set; }
```

Property Value

bool

CollectionItemName

Gets or Sets the XML element name for each item in a collection (default is "Item").

```
public string CollectionItemName { get; set; }
```

Property Value

string

ElementName

Gets or Sets the XML element name (default is the name of the managed type member).

```
public string ElementName { get; set; }
```

Property Value

string

FlattenContent

Gets or Sets a value indicating whether to write member contents directly into the current XML context rather than wrapping the member in a new XML element (default is false).

```
public bool FlattenContent { get; set; }
```

Property Value

bool

HasCollectionItemName

Gets a value that indicates whether an explicit CollectionItemName is being used or the default value.

```
public bool HasCollectionItemName { get; }
```

Property Value

bool

Optional

Gets or Sets a value indicating whether to write this element if the member is null and skip past it if not found when deserializing XML (default is false).

```
public bool Optional { get; set; }
```

Property Value

bool

SharedResource

Gets or Sets a value indicating whether this member is reference from multiple parents and should be serialized as a unique ID reference (default is false).

```
public bool SharedResource { get; set; }
```

Property Value

bool

Methods

Clone()

Creates a copy of this content serializer attribute.

```
public ContentSerializerAttribute Clone()
```

Returns

ContentSerializerAttribute

The copy of this content serializer attribute.

Class ContentSerializerCollectionItemNameAttribute

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This is used to specify the XML element name to use for each item in a collection.

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class ContentSerializerCollectionItemNameAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentSerializerCollectionItemNameAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,

Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentSerializerCollectionItemNameAttribute(string)

Creates an instance of the attribute.

```
public ContentSerializerCollectionItemNameAttribute(string collectionItemName)
```

Parameters

`collectionItemName` string

The XML element name to use for each item in the collection.

Properties

CollectionItemName

The XML element name to use for each item in the collection.

```
public string CollectionItemName { get; }
```

Property Value

string

Class ContentSerializerIgnoreAttribute

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines a custom Attribute that marks a field or property to indicate that it should not be included in serialization.

```
[AttributeUsage(AttributeTargets.Property|AttributeTargets.Field)]
public sealed class ContentSerializerIgnoreAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentSerializerIgnoreAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,

Attribute.IsDefined(ParameterInfo, Type, bool), Attribute.Match(object), Attribute.TypeId, object.Equals(object, object), object.GetType(), object.ReferenceEquals(object, object), object.ToString()

Class ContentSerializerRuntimeTypeAttribute

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This is used to specify the type to use when deserializing this object at runtime.

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Struct)]
public sealed class ContentSerializerRuntimeTypeAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentSerializerRuntimeTypeAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentSerializerRuntimeTypeAttribute(string)

Creates an instance of the attribute.

```
public ContentSerializerRuntimeTypeAttribute(string runtimeType)
```

Parameters

`runtimeType` string

The name of the type to use at runtime.

Properties

RuntimeType

The name of the type to use at runtime.

```
public string RuntimeType { get; }
```

Property Value

string

Class ContentSerializerTypeVersionAttribute

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This is used to specify the version when deserializing this object at runtime.

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Struct)]
public sealed class ContentSerializerTypeVersionAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentSerializerTypeVersionAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentSerializerTypeVersionAttribute(int)

Creates an instance of the attribute.

```
public ContentSerializerTypeVersionAttribute(int typeVersion)
```

Parameters

`typeVersion` int

The version passed to the type at runtime.

Properties

TypeVersion

The version passed to the type at runtime.

```
public int TypeVersion { get; }
```

Property Value

int

Class ContentTypeReader

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines the core behavior of content type readers used for reading a specific managed type from an .xnb binary format and provides a base for derived classes.

```
public abstract class ContentTypeReader
```

Inheritance

object ← ContentTypeReader

Derived

ContentTypeReader<T>, ReflectiveReader<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeReader(Type)

```
protected ContentTypeReader(Type targetType)
```

Parameters

`targetType` Type

Properties

CanDeserializeIntoExistingObject

Gets a value that indicates whether this content type read can deserialize into an object with the same type ad defined in the TargetType property.

```
public virtual bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

TargetType

Gets the type handled by this reader component.

```
public Type TargetType { get; }
```

Property Value

Type

TypeVersion

Gets the format version number for this type.

```
public virtual int TypeVersion { get; }
```

Property Value

int

Methods

Initialize(ContentTypeReaderManager)

```
protected virtual void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, object)

```
protected abstract object Read(ContentReader input, object existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` object

Returns

object

Class ContentTypeReaderManager

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines a manager that constructs and keeps track of ContentTypeReader objects.

```
public sealed class ContentTypeReaderManager
```

Inheritance

object ← ContentTypeReaderManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AddTypeCreator(string, Func<ContentTypeReader>)

Registers a function to create a ContentTypeReader instance used to read an object of the type specified. Call this method to register content readers that may fail loading when trimming or PublishAot are used.

```
public static void AddTypeCreator(string typeString, Func<ContentTypeReader>
createFunction)
```

Parameters

`typeString` string

A string containing the fully-qualified type name of the object type.

`createFunction` Func<ContentTypeReader>

The function responsible for creating an instance of the ContentTypeReader class.

Exceptions

ArgumentNullException

If the `typeString` parameter is null or an empty string.

ClearTypeCreators()

Clears all content type creators that were registered with AddTypeCreator(string, Func<ContentTypeReader>)

```
public static void ClearTypeCreators()
```

GetTypeReader(Type)

Creates a new instance of the ContentReader class initialized for the specified type.

```
public ContentTypeReader GetTypeReader(Type targetType)
```

Parameters

`targetType` Type

The type the ContentReader will handle.

Returns

ContentTypeReader

The ContentReader created by this method if a content reader of the specified type has been registered with this content manager; otherwise, null.

Exceptions

ArgumentNullException

If the `targetType` parameter is null.

PrepareType(string)

Removes the Version, Culture, and PublicKeyToken from a fully-qualified type name string.

```
public static string PrepareType(string type)
```

Parameters

`type` string

A string containing the fully-qualified type name to prepare.

Returns

string

A new string with the Version, Culture and PublicKeyToken removed.

Remarks

Supports multiple generic types (e.g. Dictionary<TKey,TValue>) and nested generic types (e.g. List<List<int>>).

Class ContentTypeReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Defines the core behavior of content type readers used for reading a specific managed type from an .xnb binary format and provides a base for derived classes. Derive from this class to add new data types to the content pipeline system.

```
public abstract class ContentTypeReader<T> : ContentTypeReader
```

Type Parameters

T

The managed type to read.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<T>

Derived

ArrayReader<T>, DictionaryReader<TKey, TValue>, EnumReader<T>, ListReader<T>, MultiArrayReader<T>, NullableReader<T>

Inherited Members

ContentTypeReader.CanDeserializeIntoExistingObject , ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , ContentTypeReader.Initialize(ContentTypeReaderManager) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeReader()

```
protected ContentTypeReader()
```

Methods

Read(ContentReader, object)

```
protected override object Read(ContentReader input, object existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` object

Returns

object

Read(ContentReader, T)

```
protected abstract T Read(ContentReader input, T existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` T

Returns

T

Class DictionaryReader<TKey, TValue>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class DictionaryReader<TKey, TValue> : ContentTypeReader<Dictionary<TKey,
TValue>>
```

Type Parameters

`TKey`

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

`TValue`

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<Dictionary<TKey, TValue>> ← DictionaryReader<TKey, TValue>

Inherited Members

ContentTypeReader<Dictionary<TKey, TValue>>.Read(ContentReader, object) ,
ContentTypeReader<Dictionary<TKey, TValue>>.Read(ContentReader, Dictionary<TKey, TValue>) ,
ContentTypeReader.TargetType , ContentTypeReader.TypeVersion ,
ContentTypeReader.Read(ContentReader, object) , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DictionaryReader()

```
public DictionaryReader()
```

Properties

CanDeserializeIntoExistingObject

```
public override bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, Dictionary<TKey, TValue>)

```
protected override Dictionary<TKey, TValue> Read(ContentReader input,
Dictionary<TKey, TValue> existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` Dictionary<TKey, TValue>

Returns

Dictionary<TKey, TValue>

Class EnumReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class EnumReader<T> : ContentTypeReader<T>
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<T> ← EnumReader<T>

Inherited Members

ContentTypeReader<T>.Read(ContentReader, object) , ContentTypeReader.CanDeserializeIntoExistingObject , ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

EnumReader()

```
public EnumReader()
```

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, T)

```
protected override T Read(ContentReader input, T existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` T

Returns

T

Class ListReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class ListReader<T> : ContentTypeReader<List<T>>
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<List<T>> ← ListReader<T>

Inherited Members

ContentTypeReader<List<T>>.Read(ContentReader, object) , ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ListReader()

```
public ListReader()
```

Properties

CanDeserializeIntoExistingObject

```
public override bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, List<T>)

```
protected override List<T> Read(ContentReader input, List<T> existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` List<T>

Returns

List<T>

Class MultiArrayReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class MultiArrayReader<T> : ContentTypeReader<Array>
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<Array> ← MultiArrayReader<T>

Inherited Members

ContentTypeReader<Array>.Read(ContentReader, object) , ContentTypeReader.CanDeserializeIntoExistingObject , ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MultiArrayReader()

```
public MultiArrayReader()
```

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, Array)

```
protected override Array Read(ContentReader input, Array existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` Array

Returns

Array

Class NullableReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

```
public class NullableReader<T> : ContentTypeReader<T?> where T : struct
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If ContentManager.Load() throws an NotSupportedExeception, the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ContentTypeReader<T?> ← NullableReader<T>

Inherited Members

ContentTypeReader<T?>.Read(ContentReader, object) , ContentTypeReader.CanDeserializeIntoExistingObject , ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NullableReader()

```
public NullableReader()
```

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, T?)

```
protected override T? Read(ContentReader input, T? existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` T?

Returns

T?

Class ReflectiveReader<T>

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the ContentManager fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If Load<T>(string) throws an NotSupportedException, the message should provide insights on how to fix it.

```
public class ReflectiveReader<T> : ContentTypeReader
```

Type Parameters

T

This type is not meant to be used directly by MonoGame users. Its purpose is to allow to work-around AOT issues when loading assets with the fail due to the absence of runtime-reflection support in that context (i.e. missing types due to trimming and inability to statically discover them at compile-time). If throws an , the message should provide insights on how to fix it.

Inheritance

object ← ContentTypeReader ← ReflectiveReader<T>

Inherited Members

ContentTypeReader.TargetType , ContentTypeReader.TypeVersion , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ReflectiveReader()

```
public ReflectiveReader()
```

Properties

CanDeserializeIntoExistingObject

```
public override bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

Methods

Initialize(ContentTypeReaderManager)

```
protected override void Initialize(ContentTypeReaderManager manager)
```

Parameters

`manager` ContentTypeReaderManager

Read(ContentReader, object)

```
protected override object Read(ContentReader input, object existingInstance)
```

Parameters

`input` ContentReader

`existingInstance` object

Returns

object

Class ResourceContentManager

Namespace: Microsoft.Xna.Framework.Content

Assembly: MonoGame.Framework.dll

Subclass of ContentManager, which is specialized to read from *.resx* resource files rather than directly from individual files on disk.

```
public class ResourceContentManager : ContentManager, IDisposable
```

Inheritance

object ← ContentManager ← ResourceContentManager

Implements

IDisposable

Inherited Members

ContentManager.Dispose() , ContentManager.Dispose(bool) , ContentManager.LoadLocalized<T>(string) , ContentManager.Load<T>(string) , ContentManager.ReadAsset<T>(string, Action<IDisposable>) , ContentManager.LoadedAssets , ContentManager.ReloadGraphicsAssets() , ContentManager.ReloadAsset<T>(string, T) , ContentManager.Unload() , ContentManager.UnloadAsset(string) , ContentManager.UnloadAssets(IList<string>) , ContentManager.RootDirectory , ContentManager.ServiceProvider , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ResourceContentManager(IServiceProvider, ResourceManager)

Creates a new instance of ResourceContentManager.

```
public ResourceContentManager(IServiceProvider servicesProvider,
ResourceManager resource)
```

Parameters

`servicesProvider` IServiceProvider

The service provider the **ResourceContentManager** should use to locate services.

`resource` ResourceManager

The resource manager for the **ResourceContentManager** to read from.

Exceptions

ArgumentNullException

`resource` is null.

Methods

OpenStream(string)

Opens a stream for reading the specified resource. Derived classes can replace this to implement pack files or asset compression.

```
protected override Stream OpenStream(string assetName)
```

Parameters

`assetName` string

The name of the asset being read.

Returns

Stream

Exceptions

ContentLoadException

Error loading `assetName`. The resource was not a binary resource, or the resource was not found.

Namespace Microsoft.Xna.Framework.Content.Pipeline

Classes

ChildCollection<TParent, TChild>

Provides a collection of child objects for a content item.

Links from a child object to its parent are maintained as the collection contents are modified.

ContentBuildLogger

Provides methods for reporting informational messages or warnings from content importers and processors. Do not use this class to report errors. Instead, report errors by throwing a PipelineException or InvalidContentException.

ContentIdentity

Provides properties describing the origin of the game asset, such as the original source file and creation tool. This information is used for error reporting, and by processors that need to determine from what directory the asset was originally loaded.

ContentImporterAttribute

Provides properties that identify and provide metadata about the importer, such as supported file extensions and caching information. Importers are required to initialize this attribute.

ContentImporterContext

Provides properties that define logging behavior for the importer.

ContentImporter<T>

Implements a file format importer for use with game assets. Importers, either provided by the framework or written by a developer, must derive from ContentImporter, as well as being marked with a ContentImporterAttribute. An importer should produce results in the standard intermediate object model. If an asset has information not supported by the object model, the importer should output it as opaque data (key/value attributes attached to the relevant object). By following this procedure, a content pipeline can access specialized digital content creation (DCC) tool information, even when that information has not been fully standardized into the official object model. You can also design custom importers that accept and import types containing specific third-party extensions to the object model.

ContentItem

Provides properties that define various aspects of content stored using the intermediate file format of the XNA Framework.

ContentProcessorAttribute

Gets any existing content processor components.

ContentProcessorContext

Provides access to custom processor parameters, methods for converting member data, and triggering nested builds.

ContentProcessor<TInput, TOutput>

Provides a base class to use when developing custom processor components. All processors must derive from this class.

ContentStatsCollection

A collection of content building statistics for use in diagnosing content issues.

EffectImporter

Provides methods for reading effect (.fx) files for use in the Content Pipeline.

ExternalReference<T>

Specifies external references to a data file for the content item.

While the object model is instantiated, reference file names are absolute. When the file containing the external reference is serialized to disk, file names are relative to the file. This allows movement of the content tree to a different location without breaking internal links.

FbxImporter

Provides methods for reading AutoDesk (.fbx) files for use in the Content Pipeline.

FontDescriptionImporter

Provides methods for reading .spritefont files for use in the Content Pipeline.

FontImporter

Provides methods for reading font files for use in the Content Pipeline.

H264Importer

Provides methods for reading H.264 video files for use in the Content Pipeline

InvalidContentException

Thrown when errors are encountered in content during processing.

LoadedTypeCollection<T>

A helper for collecting instances of a particular type by scanning the types in loaded assemblies.

Mp3Importer

Provides methods for reading MP3 audio files for use in the Content Pipeline.

NamedValueDictionary<T>

Provides a named value dictionary for use in the Content Pipeline.

OggImporter

Provides methods for reading .ogg audio files for use in the Content Pipeline.

OpaqueDataDictionary

Provides an opaque data dictionary for use in the Content Pipeline.

OpenAssetImporter

Implementation of the content importer for common open 3D asset formats

PipelineComponentScanner

Implements a scanner object containing the available importers and processors for an application. Designed for internal use only.

PipelineException

Thrown when errors are encountered during a content pipeline build.

ProcessorParameter

Represents a processor parameter. Processor parameters are automatically discovered by the content pipeline. Therefore, only custom processor developers should use this class directly.

ProcessorParameterCollection

Represents a collection of processor parameters, usually for a single processor. This class is primarily designed for internal use or for custom processor developers.

TextureImporter

Provides methods for reading texture files for use in the Content Pipeline.

VideoContent

Provides a base class for all video objects.

WavImporter

Provides methods for reading .wav audio files for use in the Content Pipeline.

WmaImporter

Provides methods for reading Windows Media Audio (.wma) files for use in the Content Pipeline.

WmvImporter

Provides methods for reading Windows Media Video (.wmv) files for use in the Content Pipeline.

XImporter

Provides methods for reading DirectX Object (.x) files for use in the Content Pipeline.

XmlImporter

Implements an importer for reading intermediate XML files. This is a wrapper around IntermediateSerializer.

Structs

ContentStats

Content building statistics for a single source content file.

Interfaces

IContentImporter

Accesses a statically typed ContentImporter instance from generic code using dynamic typing.

IContentProcessor

Provides methods and properties for accessing a statically typed ContentProcessor subclass, using dynamically typed object data.

Enums

LogLevel

Specifies the importants of the message when logging the mssages from Log(LogLevel, string).

TargetPlatform

Identifiers for the target platform.

Class ChildCollection<TParent, TChild>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a collection of child objects for a content item.

Links from a child object to its parent are maintained as the collection contents are modified.

```
public abstract class ChildCollection<TParent, TChild> : Collection<TChild>,
IList<TChild>, ICollection<TChild>, IReadOnlyList<TChild>,
IReadOnlyCollection<TChild>, IEnumerable<TChild>, IList, ICollection, IEnumerable
where TParent : class where TChild : class
```

Type Parameters

`TParent`

`TChild`

Inheritance

object ← Collection<TChild> ← ChildCollection<TParent, TChild>

Implements

IList<TChild>, ICollection<TChild>, IReadOnlyList<TChild>, IReadOnlyCollection<TChild>, IEnumerable<TChild>, IList, ICollection, IEnumerable

Derived

GeometryContentCollection, NodeContentCollection

Inherited Members

Collection<TChild>.Add(TChild) , Collection<TChild>.Clear() , Collection<TChild>.Contains(TChild) , Collection<TChild>.CopyTo(TChild[], int) , Collection<TChild>.GetEnumerator() , Collection<TChild>.IndexOf(TChild) , Collection<TChild>.Insert(int, TChild) , Collection<TChild>.Remove(TChild) , Collection<TChild>.RemoveAt(int) , Collection<TChild>.Count , Collection<TChild>.this[int] , Collection<TChild>.Items , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ChildCollection(TParent)

Creates an instance of ChildCollection.

```
protected ChildCollection(TParent parent)
```

Parameters

`parent` TParent

Parent object of the child objects returned in the collection.

Methods

ClearItems()

Removes all children from the collection.

```
protected override void ClearItems()
```

GetParent(TChild)

Gets the parent of a child object.

```
protected abstract TParent GetParent(TChild child)
```

Parameters

`child` TChild

The child of the parent being retrieved.

Returns

TParent

The parent of the child object.

InsertItem(int, TChild)

Inserts a child object into the collection at the specified location.

```
protected override void InsertItem(int index, TChild item)
```

Parameters

`index` int

The position in the collection.

`item` TChild

The child object being inserted.

RemoveItem(int)

Removes a child object from the collection.

```
protected override void RemoveItem(int index)
```

Parameters

`index` int

The index of the item being removed.

SetItem(int, TChild)

Modifies the value of the child object at the specified location.

```
protected override void SetItem(int index, TChild item)
```

Parameters

`index` int

The index of the child object being modified.

`item` TChild

The new value for the child object.

SetParent(TChild, TParent)

Modifies the value of the parent object of the specified child object.

```
protected abstract void SetParent(TChild child, TParent parent)
```

Parameters

`child` TChild

The child of the parent being modified.

`parent` TParent

The new value for the parent object.

Class ContentBuildLogger

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reporting informational messages or warnings from content importers and processors. Do not use this class to report errors. Instead, report errors by throwing a PipelineException or InvalidContentException.

```
public class ContentBuildLogger
```

Inheritance

object ← ContentBuildLogger

Derived

PipelineBuildLogger

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

IndentString

Creates a string that has indentation characters equal to the current identation.

```
protected string IndentString { get; }
```

Property Value

string

LoggerLogLevel

Gets or sets the desired log level to report for.

```
public LogLevel LoggerLogLevel { get; set; }
```

Property Value

LogLevel

LoggerRootDirectory

Gets or sets the base reference path used when reporting errors during the content build process.

```
public string LoggerRootDirectory { get; set; }
```

Property Value

string

Methods

GetCurrentFilename(ContentIdentity?)

Gets the filename currently being processed, for use in warning and error messages.

```
public string GetCurrentFilename(ContentIdentity? contentIdentity = null)
```

Parameters

`contentIdentity` ContentIdentity

Identity of a content item. If specified, GetCurrentFilename uses this value to refine the search. If no value is specified, the current PushFile state is used.

Returns

string

Name of the file being processed.

Indent()

Adds an indent for all future Log calls.

```
public virtual void Indent()
```

Log(LogLevel, string)

Outputs a message from the content system with the specified log level.

```
public virtual void Log(LogLevel level, string message)
```

Parameters

`level` LogLevel

Log level of the message.

`message` string

Log(string)

Outputs a message from the content system with the Info log level.

```
public void Log(string message)
```

Parameters

`message` string

LogImportantMessage(string, params object[])

Outputs a high-priority status message from the content system.

```
[Obsolete("LogImportantMessage is deprecated, please use Log instead.")]
public virtual void LogImportantMessage(string message, params object[] messageArgs)
```

Parameters

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

LogMessage(string, params object[])

Outputs a low priority status message from the content system.

```
[Obsolete("LogMessage is deprecated, please use Log instead.")]
public virtual void LogMessage(string message, params object[] messageArgs)
```

Parameters

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

LogWarning(string, ContentIdentity, string, params object[])

Outputs a warning message from the content system.

```
[Obsolete("LogWarning is deprecated, please use Log instead.")]
public virtual void LogWarning(string helpLink, ContentIdentity contentIdentity,
string message, params object[] messageArgs)
```

Parameters

`helpLink` string

Link to an existing online help topic containing related information.

`contentIdentity` ContentIdentity

Identity of the content item that generated the message.

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

PopFile()

Outputs a message indicating that a content asset has completed processing.

```
public virtual void PopFile()
```

PushFile(string)

Outputs a message indicating that a content asset has begun processing. All logger warnings or error exceptions from this time forward to the next PopFile call refer to this file.

```
public virtual void PushFile(string filename)
```

Parameters

`filename` string

Name of the file containing future messages.

Unindent()

Removes an indent for all future Log calls.

```
public virtual void Unindent()
```

Class ContentIdentity

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties describing the origin of the game asset, such as the original source file and creation tool. This information is used for error reporting, and by processors that need to determine from what directory the asset was originally loaded.

```
[Serializable]
public class ContentIdentity
```

Inheritance

object ← ContentIdentity

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentIdentity()

Initializes a new instance of ContentIdentity.

```
public ContentIdentity()
```

ContentIdentity(string)

Initializes a new instance of ContentIdentity with the specified values.

```
public ContentIdentity(string sourceFilename)
```

Parameters

`sourceFilename` string

The absolute path to the file name of the asset source.

ContentIdentity(string, string)

Initializes a new instance of ContentIdentity with the specified values.

```
public ContentIdentity(string sourceFilename, string sourceTool)
```

Parameters

`sourceFilename` string

The absolute path to the file name of the asset source.

`sourceTool` string

The name of the digital content creation (DCC) tool that created the asset.

ContentIdentity(string, string, string)

Initializes a new instance of ContentIdentity with the specified values.

```
public ContentIdentity(string sourceFilename, string sourceTool,
string fragmentIdentifier)
```

Parameters

`sourceFilename` string

The absolute path to the file name of the asset source.

`sourceTool` string

The name of the digital content creation (DCC) tool that created the asset.

`fragmentIdentifier` string

Specific location of the content item within the larger source file. For example, this could be a line number in the file.

Properties

FragmentIdentifier

Gets or sets the specific location of the content item within the larger source file.

```
public string FragmentIdentifier { get; set; }
```

Property Value

string

SourceFilename

Gets or sets the file name of the asset source.

```
public string SourceFilename { get; set; }
```

Property Value

string

SourceTool

Gets or sets the creation tool of the asset.

```
public string SourceTool { get; set; }
```

Property Value

string

Class ContentImporterAttribute

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties that identify and provide metadata about the importer, such as supported file extensions and caching information. Importers are required to initialize this attribute.

```
public class ContentImporterAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentImporterAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentImporterAttribute(string)

Initializes a new instance of ContentImporterAttribute and sets the file name extension supported by the importer.

```
public ContentImporterAttribute(string fileExtension)
```

Parameters

`fileExtension` string

The list of file name extensions supported by the importer. Prefix each extension with a '.'.

ContentImporterAttribute(params string[])

Initializes a new instance of ContentImporterAttribute and sets the file name extensions supported by the importer.

```
public ContentImporterAttribute(params string[] fileExtensions)
```

Parameters

`fileExtensions` string[]

The list of file name extensions supported by the importer. Prefix each extension with a '.'.

Properties

CacheImportedData

Gets and sets the caching of the content during importation.

```
public bool CacheImportedData { get; set; }
```

Property Value

bool

DefaultProcessor

Gets or sets the name of the default processor for content read by this importer.

```
public string DefaultProcessor { get; set; }
```

Property Value

string

DisplayName

Gets or sets the string representing the importer in a user interface. This name is not used by the content pipeline and should not be passed to the BuildAssets task (a custom MSBuild task used by XNA Game Studio). It is used for display purposes only.

```
public virtual string DisplayName { get; set; }
```

Property Value

string

FileExtensions

Gets the supported file name extensions of the importer.

```
public IEnumerable<string> FileExtensions { get; }
```

Property Value

IEnumerable<string>

Class ContentImporterContext

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties that define logging behavior for the importer.

```
public abstract class ContentImporterContext
```

Inheritance

object ← ContentImporterContext

Derived

PipelineImporterContext

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentImporterContext()

Initializes a new instance of ContentImporterContext.

```
public ContentImporterContext()
```

Properties

IntermediateDirectory

The absolute path to the root of the build intermediate (object) directory.

```
public abstract string IntermediateDirectory { get; }
```

Property Value

string

Logger

Gets the logger for an importer.

```
public abstract ContentBuildLogger Logger { get; }
```

Property Value

ContentBuildLogger

OutputDirectory

The absolute path to the root of the build output (binaries) directory.

```
public abstract string OutputDirectory { get; }
```

Property Value

string

Methods

AddDependency(string)

Adds a dependency to the specified file. This causes a rebuild of the file, when modified, on subsequent incremental builds.

```
public abstract void AddDependency(string filename)
```

Parameters

`filename` string

Name of an asset file.

Class ContentImporter<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Implements a file format importer for use with game assets. Importers, either provided by the framework or written by a developer, must derive from ContentImporter, as well as being marked with a ContentImporterAttribute. An importer should produce results in the standard intermediate object model. If an asset has information not supported by the object model, the importer should output it as opaque data (key/value attributes attached to the relevant object). By following this procedure, a content pipeline can access specialized digital content creation (DCC) tool information, even when that information has not been fully standardized into the official object model. You can also design custom importers that accept and import types containing specific third-party extensions to the object model.

```
public abstract class ContentImporter<T> : IContentImporter
```

Type Parameters

T

Implements a file format importer for use with game assets. Importers, either provided by the framework or written by a developer, must derive from ContentImporter, as well as being marked with a ContentImporterAttribute. An importer should produce results in the standard intermediate object model. If an asset has information not supported by the object model, the importer should output it as opaque data (key/value attributes attached to the relevant object). By following this procedure, a content pipeline can access specialized digital content creation (DCC) tool information, even when that information has not been fully standardized into the official object model. You can also design custom importers that accept and import types containing specific third-party extensions to the object model.

Inheritance

object ← ContentImporter<T>

Implements

IContentImporter

Derived

EffectImporter, FbxImporter, FontDescriptionImporter, FontImporter, H264Importer, Mp3Importer, OggImporter, OpenAssetImporter, TextureImporter, WavImporter, WmaImporter, WmvImporter, XImporter, XmlImporter

Inherited Members
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentImporter()

Initializes a new instance of ContentImporter.

```
protected ContentImporter()
```

Properties

Version

Gets or sets the version of the current content importer that will be used to determien if the content needs to be rebuilt.

```
public virtual string Version { get; set; }
```

Property Value

string

Methods

Import(string, ContentImporterContext)

Called by the framework when importing a game asset. This is the method called by XNA when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public abstract T Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

T

Resulting game asset.

Class ContentItem

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties that define various aspects of content stored using the intermediate file format of the XNA Framework.

```
public class ContentItem
```

Inheritance

object ← ContentItem

Derived

AudioContent, ExternalReference<T>, AnimationContent, BitmapContent, EffectContent, FontDescription, GeometryContent, MaterialContent, NodeContent, TextureContent, CompiledEffectContent, VertexBufferContent, VertexDeclarationContent, VideoContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentItem()

Initializes a new instance of ContentItem.

```
public ContentItem()
```

Properties

Identity

Gets or sets the identity of the content item.

```
[ContentSerializer(Optional = true)]
public ContentIdentity Identity { get; set; }
```

Property Value

ContentIdentity

Name

Gets or sets the name of the content item.

```
[ContentSerializer(Optional = true)]
public string Name { get; set; }
```

Property Value

string

OpaqueData

Gets the opaque data of the content item.

```
[ContentSerializer(Optional = true)]
public OpaqueDataDictionary OpaqueData { get; }
```

Property Value

OpaqueDataDictionary

Class ContentProcessorAttribute

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Gets any existing content processor components.

```
public class ContentProcessorAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentProcessorAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentProcessorAttribute()

Initializes an instance of ContentProcessorAttribute.

```
public ContentProcessorAttribute()
```

Properties

DisplayName

Gets or sets the string representing the processor in a user interface. This name is not used by the content pipeline and should not be passed to the BuildAssets task (a custom MSBuild task used by XNA Game Studio). It is used for display purposes only.

```
public virtual string DisplayName { get; set; }
```

Property Value

string

Class ContentProcessorContext

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides access to custom processor parameters, methods for converting member data, and triggering nested builds.

```
public abstract class ContentProcessorContext
```

Inheritance

object ← ContentProcessorContext

Derived

PipelineProcessorContext

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentProcessorContext()

Initializes a new instance of ContentProcessorContext.

```
public ContentProcessorContext()
```

Properties

BuildConfiguration

Gets the name of the current content build configuration.

```
public abstract string BuildConfiguration { get; }
```

Property Value

string

IntermediateDirectory

Gets the path of the directory that will contain any intermediate files generated by the content processor.

```
public abstract string IntermediateDirectory { get; }
```

Property Value

string

Logger

Gets the logger interface used for status messages or warnings.

```
public abstract ContentBuildLogger Logger { get; }
```

Property Value

ContentBuildLogger

OutputDirectory

Gets the output path of the content processor.

```
public abstract string OutputDirectory { get; }
```

Property Value

string

OutputFilename

Gets the output file name of the content processor.

```
public abstract string OutputFilename { get; }
```

Property Value

string

Parameters

Gets the collection of parameters used by the content processor.

```
public abstract OpaqueDataDictionary Parameters { get; }
```

Property Value

OpaqueDataDictionary

ProjectDirectory

Gets the directory that contains the content project.

```
public abstract string ProjectDirectory { get; }
```

Property Value

string

SourceIdentity

Gets the ContentIdentity representing the source asset imported.

```
public abstract ContentIdentity SourceIdentity { get; }
```

Property Value

ContentIdentity

TargetPlatform

Gets the current content build target platform.

```
public abstract TargetPlatform TargetPlatform { get; }
```

Property Value

TargetPlatform

TargetProfile

Gets the current content build target profile.

```
public abstract GraphicsProfile TargetProfile { get; }
```

Property Value

GraphicsProfile

Methods

AddDependency(string)

Adds a dependency to the specified file. This causes a rebuild of the file, when modified, on subsequent incremental builds.

```
public abstract void AddDependency(string filename)
```

Parameters

`filename` string

Name of an asset file.

AddOutputFile(string)

Add a file name to the list of related output files maintained by the build item. This allows tracking build items that build multiple output files.

```
public abstract void AddOutputFile(string filename)
```

Parameters

`filename` string

The name of the file.

BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>, IContentImporter, IContentProcessor)

Initiates a nested build of the specified asset and then loads the result into memory.

```
public abstract TOutput BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>
sourceAsset, IContentImporter importer, IContentProcessor processor)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`importer` IContentImporter

Optional importer for this content.

`processor` IContentProcessor

Optional processor for this content.

Returns

TOutput

Copy of the final converted content.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the converted output.

Remarks

An example of usage would be a mesh processor calling BuildAndLoadAsset to build any associated textures and replace the original .tga file references with an embedded copy of the converted texture.

BuildAndLoadAsset<TInput, TOutput> (ExternalReference<TInput>, string)

Initiates a nested build of the specified asset and then loads the result into memory.

```
public TOutput BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>
sourceAsset, string processorName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

Returns

TOutput

Copy of the final converted content.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the converted output.

Remarks

An example of usage would be a mesh processor calling BuildAndLoadAsset to build any associated textures and replace the original .tga file references with an embedded copy of the converted texture.

BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>, string, OpaqueDataDictionary, string)

Initiates a nested build of the specified asset and then loads the result into memory.

```
[Obsolete("Please pass importer and processor as instances instead of just
their names.")]
public abstract TOutput BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>
sourceAsset, string processorName, OpaqueDataDictionary processorParameters,
string importerName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional collection of named values available as input to the content processor.

`importerName` string

Optional importer for this content.

Returns

TOutput

Copy of the final converted content.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the converted output.

Remarks

An example of usage would be a mesh processor calling BuildAndLoadAsset to build any associated textures and replace the original .tga file references with an embedded copy of the converted texture.

BuildAsset<TInput, TOutput>(ExternalReference<TInput>, IContentImporter, IContentProcessor, string?)

Initiates a nested build of an additional asset.

```
public abstract ExternalReference<TOutput> BuildAsset<TInput, TOutput>
(ExternalReference<TInput> sourceAsset, IContentImporter importer, IContentProcessor
processor, string? assetName = null)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`importer` IContentImporter

Optional importer for this content.

`processor` IContentProcessor

Optional processor for this content.

`assetName` string

Optional name of the final compiled content.

Returns

ExternalReference<TOutput>

Reference to the final compiled content. The build work is not required to complete before returning. Therefore, this file may not be up to date when BuildAsset returns but it will be available for loading by the game at runtime.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the output.

Remarks

An example of usage for BuildAsset is being called by a mesh processor to request that any related textures used are also built, replacing the original TGA file references with new references to the converted texture files.

BuildAsset<TInput, TOutput>(ExternalReference<TInput>, string)

Initiates a nested build of an additional asset.

```
public ExternalReference<TOutput> BuildAsset<TInput, TOutput>
(ExternalReference<TInput> sourceAsset, string processorName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

Returns

ExternalReference<TOutput>

Reference to the final compiled content. The build work is not required to complete before returning. Therefore, this file may not be up to date when BuildAsset returns but it will be available for loading by the game at runtime.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the output.

Remarks

An example of usage for BuildAsset is being called by a mesh processor to request that any related textures used are also built, replacing the original TGA file references with new references to the converted texture files.

BuildAsset<TInput, TOutput>(ExternalReference<TInput>, string, OpaqueDataDictionary, string, string)

Initiates a nested build of an additional asset.

```
[Obsolete("Please pass importer and processor as instances instead of just
their names.")]
public abstract ExternalReference<TOutput> BuildAsset<TInput, TOutput>
(ExternalReference<TInput> sourceAsset, string processorName, OpaqueDataDictionary
processorParameters, string importerName, string assetName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional collection of named values available as input to the content processor.

`importerName` string

Optional importer for this content.

`assetName` string

Optional name of the final compiled content.

Returns

ExternalReference<TOutput>

Reference to the final compiled content. The build work is not required to complete before returning. Therefore, this file may not be up to date when BuildAsset returns but it will be available for loading by the game at runtime.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the output.

Remarks

An example of usage for BuildAsset is being called by a mesh processor to request that any related textures used are also built, replacing the original TGA file references with new references to the converted texture files.

Convert<TInput, TOutput>(TInput, IContentProcessor)

Converts a content item object using the specified content processor.

```
public abstract TOutput Convert<TInput, TOutput>(TInput input,
IContentProcessor processor)
```

Parameters

`input` TInput

Source content to be converted.

`processor` IContentProcessor

Optional processor for this content.

Returns

TOutput

Reference of the final converted content.

Type Parameters

`TInput`

Type of the input content.

`TOutput`

Type of the converted output.

Convert<TInput, TOutput>(TInput, string)

Converts a content item object using the specified content processor.

```
public TOutput Convert<TInput, TOutput>(TInput input, string processorName)
```

Parameters

`input` TInput

Source content to be converted.

`processorName` string

Optional processor for this content.

Returns

TOutput

Reference of the final converted content.

Type Parameters

`TInput`

Type of the input content.

`TOutput`

Type of the converted output.

Convert<TInput, TOutput>(TInput, string, OpaqueDataDictionary)

Converts a content item object using the specified content processor.

```
[Obsolete("Please pass importer and processor as instances instead of just
their names.")]
public abstract TOutput Convert<TInput, TOutput>(TInput input, string processorName,
OpaqueDataDictionary processorParameters)
```

Parameters

`input` TInput

Source content to be converted.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional parameters for the processor.

Returns

TOutput

Reference of the final converted content.

Type Parameters

`TInput`

Type of the input content.

`TOutput`

Type of the converted output.

Class ContentProcessor<TInput, TOutput>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class to use when developing custom processor components. All processors must derive from this class.

```
public abstract class ContentProcessor<TInput, TOutput> : IContentProcessor
```

Type Parameters

`TInput`

Provides a base class to use when developing custom processor components. All processors must derive from this class.

`TOutput`

Provides a base class to use when developing custom processor components. All processors must derive from this class.

Inheritance

object ← ContentProcessor<TInput, TOutput>

Implements

IContentProcessor

Derived

EffectProcessor, FontDescriptionProcessor, FontTextureProcessor, LocalizedFontProcessor, MaterialProcessor, ModelProcessor, PassThroughProcessor, SongProcessor, SoundEffectProcessor, TextureProcessor, VideoProcessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentProcessor()

Initializes a new instance of the ContentProcessor class.

```
protected ContentProcessor()
```

Properties

Version

Gets or sets the version of the current content processor that will be used to determien if the content needs to be rebuilt.

```
public virtual string Version { get; set; }
```

Property Value

string

Methods

Process(TInput, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public abstract TOutput Process(TInput input, ContentProcessorContext context)
```

Parameters

`input` TInput

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

TOutput

A typed object representing the processed input.

Struct ContentStats

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Content building statistics for a single source content file.

```
public struct ContentStats
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

BuildSeconds

The content build time in seconds.

```
public float BuildSeconds
```

Field Value

float

ContentType

The content type name.

```
public string ContentType
```

Field Value

string

DestFile

The absolute path to the destination content file.

```
public string DestFile
```

Field Value

string

DestFileSize

The destination file size in bytes.

```
public long DestFileSize
```

Field Value

long

ProcessorType

The content processor type name.

```
public string ProcessorType
```

Field Value

string

SourceFile

The absolute path to the source content file.

```
public string SourceFile
```

Field Value

string

SourceFileSize

The source file size in bytes.

```
public long SourceFileSize
```

Field Value

long

Class ContentStatsCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

A collection of content building statistics for use in diagnosing content issues.

```
public class ContentStatsCollection
```

Inheritance

object ← ContentStatsCollection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

Extension

The file extension used by the content pipeline after building to store the content statistics.

Read only - cannot be assigned to

```
public static readonly string Extension
```

Field Value

string

Properties

PreviousStats

Optionally used for copying stats that were stored in another collection.

```
public ContentStatsCollection PreviousStats { get; set; }
```

Property Value

ContentStatsCollection

Stats

The internal content statistics dictionary.

```
public IReadOnlyDictionary<string, ContentStats> Stats { get; }
```

Property Value

IReadOnlyDictionary<string, ContentStats>

Methods

CopyPreviousStats(string)

Copy content building stats to the current collection from the PreviousStats.

```
public void CopyPreviousStats(string sourceFile)
```

Parameters

`sourceFile` string

The absolute path to the source asset file.

MergePreviousStats()

Merge in statistics from PreviousStats that do not exist in this collection.

```
public void MergePreviousStats()
```

Read(string)

Load the content statistics from a folder.

```
public static ContentStatsCollection Read(string outputPath)
```

Parameters

`outputPath` string

The folder where the .mgstats file can be found.

Returns

ContentStatsCollection

Returns the content statistics or an empty collection.

RecordStats(string, string, string, Type, float)

Store content building stats for a source file.

```
public void RecordStats(string sourceFile, string destFile, string processorType,
Type contentType, float buildSeconds)
```

Parameters

`sourceFile` string

The absolute path to the source asset file.

`destFile` string

The absolute path to the destination content file.

`processorType` string

The type name of the content processor.

`contentType` Type

The content type object.

`buildSeconds` float

The build time in seconds.

Reset()

Clears all the content statistics.

```
public void Reset()
```

TryGetStats(string, out ContentStats)

Get the content statistics for a source file and returns true if found.

```
public bool TryGetStats(string sourceFile, out ContentStats stats)
```

Parameters

`sourceFile` string

`stats` ContentStats

Returns

bool

Write(string)

Write the content statistics to a folder with the .mgstats file name.

```
public void Write(string outputPath)
```

Parameters

`outputPath` string

The folder to write the .mgstats file.

Class EffectImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading effect (.fx) files for use in the Content Pipeline.

```
[ContentImporter(".fx", DisplayName = "Effect Importer - MonoGame", DefaultProcessor
= "EffectProcessor")]
public class EffectImporter : ContentImporter<EffectContent>, IContentImporter
```

Inheritance

object ← ContentImporter<EffectContent> ← EffectImporter

Implements

IContentImporter

Inherited Members

ContentImporter<EffectContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

EffectImporter()

Initializes a new instance of EffectImporter.

```
public EffectImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing an .fx file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override EffectContent Import(string filename, ContentImporterContext
context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

EffectContent

Resulting game asset.

Class ExternalReference<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Specifies external references to a data file for the content item.

While the object model is instantiated, reference file names are absolute. When the file containing the external reference is serialized to disk, file names are relative to the file. This allows movement of the content tree to a different location without breaking internal links.

```
public sealed class ExternalReference<T> : ContentItem
```

Type Parameters

T

Inheritance

object ← ContentItem ← ExternalReference<T>

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ExternalReference()

Initializes a new instance of ExternalReference.

```
public ExternalReference()
```

ExternalReference(string)

Initializes a new instance of ExternalReference.

```
public ExternalReference(string filename)
```

Parameters

`filename` string

The name of the referenced file.

ExternalReference(string, ContentIdentity)

Initializes a new instance of ExternalReference, specifying the file path relative to another content item.

```
public ExternalReference(string filename, ContentIdentity relativeToContent)
```

Parameters

`filename` string

The name of the referenced file.

`relativeToContent` ContentIdentity

The content that the path specified in filename is relative to.

Properties

Filename

Gets and sets the file name of an ExternalReference.

```
public string Filename { get; set; }
```

Property Value

string

Class FbxImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading AutoDesk (.fbx) files for use in the Content Pipeline.

```
[ContentImporter(".fbx", DisplayName = "Fbx Importer - MonoGame", DefaultProcessor
= "ModelProcessor")]
public class FbxImporter : ContentImporter<NodeContent>, IContentImporter
```

Inheritance

object ← ContentImporter<NodeContent> ← FbxImporter

Implements

IContentImporter

Inherited Members

ContentImporter<NodeContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Import(string, ContentImporterContext)

Called by the framework when importing a game asset. This is the method called by XNA when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override NodeContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

NodeContent

Resulting game asset.

Class FontDescriptionImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading .spritefont files for use in the Content Pipeline.

```
[ContentImporter(".spritefont", DisplayName = "Sprite Font Importer - MonoGame",
DefaultProcessor = "FontDescriptionProcessor")]
public class FontDescriptionImporter : ContentImporter<FontDescription>,
IContentImporter
```

Inheritance

object ← ContentImporter<FontDescription> ← FontDescriptionImporter

Implements

IContentImporter

Inherited Members

ContentImporter<FontDescription>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FontDescriptionImporter()

Initializes a new instance of FontDescriptionImporter.

```
public FontDescriptionImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a .spritefont file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override FontDescription Import(string filename, ContentImporterContext
context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

FontDescription

Resulting game asset.

Class FontImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading font files for use in the Content Pipeline.

```
[ContentImporter(new string[] { ".ttf", ".otf" }, DisplayName = "Sprite Font
Importer - MonoGame", DefaultProcessor = "FontDescriptionProcessor")]
public class FontImporter : ContentImporter<FontDescription>, IContentImporter
```

Inheritance

object ← ContentImporter<FontDescription> ← FontImporter

Implements

IContentImporter

Inherited Members

ContentImporter<FontDescription>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

CharacterRegions

Retrieves the set of characters to include in the processor output.

```
public List<CharacterRegion> CharacterRegions { get; set; }
```

Property Value

List<CharacterRegion>

DefaultCharacter

Gets or sets the default character for the font.

```
public char? DefaultCharacter { get; set; }
```

Property Value

char?

ResourceFiles

Allows including extra characters as specified by the resource files.

```
public List<string> ResourceFiles { get; set; }
```

Property Value

List<string>

Size

Gets or sets the size, in points, of the font.

```
public float Size { get; set; }
```

Property Value

float

Spacing

Gets or sets the amount of space, in pixels, to insert between letters in a string.

```
public float Spacing { get; set; }
```

Property Value

float

Style

Gets or sets the style of the font, expressed as a combination of one or more FontDescriptionStyle flags.

```
public FontDescriptionStyle Style { get; set; }
```

Property Value

FontDescriptionStyle

UseKerning

Indicates if kerning information is used when drawing characters.

```
public bool UseKerning { get; set; }
```

Property Value

bool

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a font file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override FontDescription Import(string filename, ContentImporterContext
context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

FontDescription

Resulting game asset.

Class H264Importer

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading H.264 video files for use in the Content Pipeline

```
[ContentImporter(".mp4", DisplayName = "H.264 Video - MonoGame", DefaultProcessor
= "VideoProcessor")]
public class H264Importer : ContentImporter<VideoContent>, IContentImporter
```

Inheritance

object ← ContentImporter<VideoContent> ← H264Importer

Implements

IContentImporter

Inherited Members

ContentImporter<VideoContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

H264Importer()

Creates a new instance of H264Importer

```
public H264Importer()
```

Methods

Import(string, ContentImporterContext)

Called by the framework when importing a game asset. This is the method called by XNA when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override VideoContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

VideoContent

Resulting game asset.

Interface IContentImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Accesses a statically typed ContentImporter instance from generic code using dynamic typing.

```
public interface IContentImporter
```

Methods

Import(string, ContentImporterContext)

Imports an asset from the specified file.

```
object Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of the game asset file.

`context` ContentImporterContext

A ContentImporterContext class containing information for importing a game asset, such as a logger interface.

Returns

object

Resulting game asset.

Interface IContentProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for accessing a statically typed ContentProcessor subclass, using dynamically typed object data.

```
public interface IContentProcessor
```

Properties

InputType

Gets the expected object type of the input parameter to IContentProcessor.Process.

```
Type InputType { get; }
```

Property Value

Type

OutputType

Gets the object type returned by IContentProcessor.Process.

```
Type OutputType { get; }
```

Property Value

Type

Methods

Process(object, ContentProcessorContext)

Processes the specified input data and returns the result.

```
object Process(object input, ContentProcessorContext context)
```

Parameters

`input` object

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

object

An object representing the processed input.

Class InvalidContentException

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Thrown when errors are encountered in content during processing.

```
[Serializable]
public class InvalidContentException : Exception, ISerializable
```

Inheritance

object ← Exception ← InvalidContentException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

InvalidContentException()

Initializes a new instance of the InvalidContentException class

```
public InvalidContentException()
```

InvalidContentException(SerializationInfo, StreamingContext)

Initializes a new instance of the InvalidContentException class with information on serialization and streaming context for the related content item.

```
protected InvalidContentException(SerializationInfo serializationInfo,
StreamingContext streamingContext)
```

Parameters

`serializationInfo` SerializationInfo

Information necessary for serialization and deserialization of the content item.

`streamingContext` StreamingContext

Information necessary for the source and destination of a given serialized stream. Also provides an additional caller-defined context.

InvalidContentException(string)

Initializes a new instance of the InvalidContentException class with the specified error message.

```
public InvalidContentException(string message)
```

Parameters

`message` string

A message that describes the error.

InvalidContentException(string, ContentIdentity)

Initializes a new instance of the InvalidContentException class with the specified error message and the identity of the content throwing the exception.

```
public InvalidContentException(string message, ContentIdentity contentIdentity)
```

Parameters

`message` string

A message that describes the error.

`contentIdentity` ContentIdentity

Information about the content item that caused this error, including the file name. In some cases, a location within the file (of the problem) is specified.

InvalidContentException(string, ContentIdentity, Exception)

Initializes a new instance of the InvalidContentException class with the specified error message, the identity of the content throwing the exception, and a reference to the inner exception that is the cause of this exception.

```
public InvalidContentException(string message, ContentIdentity contentIdentity,
Exception innerException)
```

Parameters

`message` string

A message that describes the error.

`contentIdentity` ContentIdentity

Information about the content item that caused this error, including the file name. In some cases, a location within the file (of the problem) is specified.

`innerException` Exception

The exception that is the cause of the current exception. If innerException is not a null reference, the current exception is raised in a catch block that handles the inner exception.

InvalidContentException(string, Exception)

Initializes a new instance of the InvalidContentException class with the specified error message and a reference to the inner exception that is the cause of this exception.

```
public InvalidContentException(string message, Exception innerException)
```

Parameters

`message` string

A message that describes the error.

`innerException` Exception

The exception that is the cause of the current exception. If innerException is not a null reference, the current exception is raised in a catch block that handles the inner exception.

Properties

ContentIdentity

Gets or sets the identity of the content item that caused the exception.

```
public ContentIdentity ContentIdentity { get; set; }
```

Property Value

ContentIdentity

Class LoadedTypeCollection<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

A helper for collecting instances of a particular type by scanning the types in loaded assemblies.

```
public class LoadedTypeCollection<T> : IEnumerable<T>, IEnumerable
```

Type Parameters

T

A helper for collecting instances of a particular type by scanning the types in loaded assemblies.

Inheritance

object ← LoadedTypeCollection<T>

Implements

IEnumerable<T>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

LoadedTypeCollection()

Creates a new LoadedTypeCollection.

```
public LoadedTypeCollection()
```

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<T> GetEnumerator()
```

Returns

IEnumerator<T>

An enumerator that can be used to iterate through the collection.

Enum LogLevel

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Specifies the importants of the message when logging the mssages from Log(LogLevel, string).

```
public enum LogLevel
```

Fields

`Debug = 0`

Specifies that the message is used for debugging purposes only.

`Error = 3`

Specifies that the message is an error message.

`Info = 1`

Specifies that the message is just an informative message.

`Warning = 2`

Specifies that the message is a warning message.

Class Mp3Importer

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading MP3 audio files for use in the Content Pipeline.

```
[ContentImporter(".mp3", DisplayName = "Mp3 Importer - MonoGame", DefaultProcessor
= "SongProcessor")]
public class Mp3Importer : ContentImporter<AudioContent>, IContentImporter
```

Inheritance

object ← ContentImporter<AudioContent> ← Mp3Importer

Implements

IContentImporter

Inherited Members

ContentImporter<AudioContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Mp3Importer()

Initializes a new instance of Mp3Importer.

```
public Mp3Importer()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing an MP3 audio file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override AudioContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

AudioContent

Resulting game asset.

Class NamedValueDictionary<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a named value dictionary for use in the Content Pipeline.

```
public class NamedValueDictionary<T> : IDictionary<string, T>,
ICollection<KeyValuePair<string, T>>, IEnumerable<KeyValuePair<string,
T>>, IEnumerable
```

Type Parameters

T

Type of the value stored in the dictionary

Inheritance

object ← NamedValueDictionary<T>

Implements

IDictionary<string, T>, ICollection<KeyValuePair<string, T>>, IEnumerable<KeyValuePair<string, T>>, IEnumerable

Derived

AnimationChannelDictionary, AnimationContentDictionary, TextureReferenceDictionary, OpaqueDataDictionary

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NamedValueDictionary()

Initializes an instance of NamedValueDictionary.

```
public NamedValueDictionary()
```

Properties

Count

Gets the number of items in the dictionary.

```
public int Count { get; }
```

Property Value

int

DefaultSerializerType

Specifies the type hint for the intermediate serializer. Values of this type do not store an explicit type attribute in the related XML source.

```
protected virtual Type DefaultSerializerType { get; }
```

Property Value

Type

this[string]

Gets or sets the specified item.

```
public T this[string key] { get; set; }
```

Parameters

`key` string

Identity of a key.

Property Value

T

Keys

Gets all keys contained in the dictionary.

```
public ICollection<string> Keys { get; }
```

Property Value

ICollection<string>

Values

Gets all values contained in the dictionary.

```
public ICollection<T> Values { get; }
```

Property Value

ICollection<T>

Methods

Add(string, T)

Adds the specified key and value to the dictionary.

```
public void Add(string key, T value)
```

Parameters

`key` string

Identity of the key of the new data pair.

`value` T

The value of the new data pair.

AddItem(string, T)

Adds an element to the dictionary.

```
protected virtual void AddItem(string key, T value)
```

Parameters

`key` string

Identity of the key of the new element.

`value` T

The value of the new element.

Clear()

Removes all keys and values from the dictionary.

```
public void Clear()
```

ClearItems()

Removes all elements from the dictionary.

```
protected virtual void ClearItems()
```

ContainsKey(string)

Determines whether the specified key is present in the dictionary.

```
public bool ContainsKey(string key)
```

Parameters

`key` string

Identity of a key.

Returns

bool

GetEnumerator()

Gets an enumerator that iterates through items in a dictionary.

```
public IEnumerator<KeyValuePair<string, T>> GetEnumerator()
```

Returns

IEnumerator<KeyValuePair<string, T>>

Enumerator for iterating through the dictionary.

Remove(string)

Removes the specified key and value from the dictionary.

```
public bool Remove(string key)
```

Parameters

`key` string

Identity of the key to be removed.

Returns

bool

true if the value is present; false otherwise.

RemoveItem(string)

Removes the specified element from the dictionary.

```
protected virtual bool RemoveItem(string key)
```

Parameters

`key` string

Identity of the key of the data pair to be removed.

Returns

bool

true if the value is present; false otherwise.

SetItem(string, T)

Modifies the value of an existing element.

```
protected virtual void SetItem(string key, T value)
```

Parameters

`key` string

Identity of the element to be modified.

`value` T

The value to be set.

TryGetValue(string, out T)

Gets the value associated with the specified key.

```
public bool TryGetValue(string key, out T value)
```

Parameters

`key` string

Identity of the key of the element whose value is to be retrieved.

`value` T

The current value of the element.

Returns

bool

true if the value is present; false otherwise.

Class OggImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading .ogg audio files for use in the Content Pipeline.

```
[ContentImporter(".ogg", DisplayName = "Ogg Importer - MonoGame", DefaultProcessor
= "SongProcessor")]
public class OggImporter : ContentImporter<AudioContent>, IContentImporter
```

Inheritance

object ← ContentImporter<AudioContent> ← OggImporter

Implements

IContentImporter

Inherited Members

ContentImporter<AudioContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing an ogg audio file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override AudioContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

AudioContent

Resulting game asset.

Class OpaqueDataDictionary

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides an opaque data dictionary for use in the Content Pipeline.

```
[ContentSerializerCollectionItemName("Data")]
public sealed class OpaqueDataDictionary : NamedValueDictionary<object>,
IDictionary<string, object>, ICollection<KeyValuePair<string, object>>,
IEnumerable<KeyValuePair<string, object>>, IEnumerable
```

Inheritance

object ← NamedValueDictionary<object> ← OpaqueDataDictionary

Implements

IDictionary<string, object>, ICollection<KeyValuePair<string, object>>,
IEnumerable<KeyValuePair<string, object>>, IEnumerable

Inherited Members

NamedValueDictionary<object>.Add(string, object) ,
NamedValueDictionary<object>.ContainsKey(string) , NamedValueDictionary<object>.Keys ,
NamedValueDictionary<object>.Remove(string) ,
NamedValueDictionary<object>.TryGetValue(string, out object) ,
NamedValueDictionary<object>.Values , NamedValueDictionary<object>.this[string] ,
NamedValueDictionary<object>.Clear() , NamedValueDictionary<object>.Count ,
NamedValueDictionary<object>.GetEnumerator() , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Remarks

This is equivilant to a NamedValueDictionary where the type of the values is Object.

Methods

GetValue<T>(string, T)

Get the value for the specified key

```
public T GetValue<T>(string key, T defaultValue)
```

Parameters

`key` string

`defaultValue` T

Returns

T

The item cast as T, or the default value if the item is not present in the dictonary.

Type Parameters

`T`

Class OpenAssetImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Implementation of the content importer for common open 3D asset formats

```
[ContentImporter(new string[] { ".dae", ".gltf", "glb", ".blend", ".3ds", ".ase",
".obj", ".ifc", ".xgl", ".zgl", ".ply", ".dxf", ".lwo", ".lws", ".lxo", ".stl",
".ac", ".ms3d", ".cob", ".scn", ".bvh", ".csm", ".irrmesh", ".irr", ".mdl", ".md2",
".md3", ".pk3", ".mdc", ".md5", ".smd", ".vta", ".ogex", ".3d", ".b3d", ".q3d",
".q3s", ".nff", ".off", ".ter", ".hmp", ".ndo" }, DisplayName = "Open Asset Import
Library - MonoGame", DefaultProcessor = "ModelProcessor")]
public class OpenAssetImporter : ContentImporter<NodeContent>, IContentImporter
```

Inheritance

object ← ContentImporter<NodeContent> ← OpenAssetImporter

Implements

IContentImporter

Inherited Members

ContentImporter<NodeContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

OpenAssetImporter()

Default constructor.

```
public OpenAssetImporter()
```

Properties

XnaComptatible

This disables some Assimp model loading features so that the resulting content is the same as what the XNA FbxImporter

```
public bool XnaComptatible { get; set; }
```

Property Value

bool

Methods

Import(string, ContentImporterContext)

Called by the framework when importing a game asset. This is the method called by XNA when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override NodeContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

NodeContent

Resulting game asset.

Class PipelineComponentScanner

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Implements a scanner object containing the available importers and processors for an application. Designed for internal use only.

```
public sealed class PipelineComponentScanner
```

Inheritance

object ← PipelineComponentScanner

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PipelineComponentScanner()

Initializes a new instance of PipelineComponentScanner.

```
public PipelineComponentScanner()
```

Properties

Errors

Gets the list of error messages produced by the last call to Update.

```
public IList<string> Errors { get; }
```

Property Value

IList<string>

ImporterAttributes

Gets a dictionary that maps importer names to their associated metadata attributes.

```
public IDictionary<string, ContentImporterAttribute> ImporterAttributes { get; }
```

Property Value

IDictionary<string, ContentImporterAttribute>

ImporterNames

Gets the names of all available importers.

```
public IEnumerable<string> ImporterNames { get; }
```

Property Value

IEnumerable<string>

ImporterOutputTypes

Gets a dictionary that maps importer names to the fully qualified name of their return types.

```
public IDictionary<string, string> ImporterOutputTypes { get; }
```

Property Value

IDictionary<string, string>

ProcessorAttributes

Gets a dictionary that maps processor names to their associated metadata attributes.

```
public IDictionary<string, ContentProcessorAttribute> ProcessorAttributes { get; }
```

Property Value

IDictionary<string, ContentProcessorAttribute>

ProcessorInputTypes

Gets a dictionary that maps processor names to the fully qualified name of supported input types.

```
public IDictionary<string, string> ProcessorInputTypes { get; }
```

Property Value

IDictionary<string, string>

ProcessorNames

Gets the names of all available processors.

```
public IEnumerable<string> ProcessorNames { get; }
```

Property Value

IEnumerable<string>

ProcessorOutputTypes

Gets a dictionary that maps processor names to the fully qualified name of their output types.

```
public IDictionary<string, string> ProcessorOutputTypes { get; }
```

Property Value

IDictionary<string, string>

ProcessorParameters

A collection of supported processor parameters.

```
public IDictionary<string, ProcessorParameterCollection> ProcessorParameters {
get; }
```

Property Value

IDictionary<string, ProcessorParameterCollection>

Methods

Update(IEnumerable<string>)

Updates the scanner object with the latest available assembly states.

```
public bool Update(IEnumerable<string> pipelineAssemblies)
```

Parameters

`pipelineAssemblies` IEnumerable<string>

Enumerated list of available assemblies.

Returns

bool

true if an actual scan was required, indicating the collection contents may have changed. false if no assembly changes were detected since the previous call.

Update(IEnumerable<string>, IEnumerable<string>)

Updates the scanner object with the latest available assembly states.

```
public bool Update(IEnumerable<string> pipelineAssemblies, IEnumerable<string>
pipelineAssemblyDependencies)
```

Parameters

`pipelineAssemblies` IEnumerable<string>

Enumerated list of available assemblies.

`pipelineAssemblyDependencies` IEnumerable<string>

Enumerated list of dependent assemblies.

Returns

bool

true if an actual scan was required, indicating the collection contents may have changed. false if no assembly changes were detected since the previous call.

Class PipelineException

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Thrown when errors are encountered during a content pipeline build.

```
[Serializable]
public class PipelineException : Exception, ISerializable
```

Inheritance

object ← Exception ← PipelineException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PipelineException()

Creates an instance of PipelineException.

```
public PipelineException()
```

PipelineException(SerializationInfo, StreamingContext)

Creates an instance of PipelineException with information on serialization and streaming context for the related content item.

```
protected PipelineException(SerializationInfo serializationInfo,
StreamingContext streamingContext)
```

Parameters

`serializationInfo` SerializationInfo

Information necessary for serialization and deserialization of the content item.

`streamingContext` StreamingContext

Information necessary for the source and destination of a given serialized stream. Also provides an additional caller-defined context.

PipelineException(string)

Initializes a new instance of the PipelineException class with the specified error message.

```
public PipelineException(string message)
```

Parameters

`message` string

A message that describes the error.

PipelineException(string, Exception)

Initializes a new instance of the PipelineException class with the specified error message and a reference to the inner exception that is the cause of this exception.

```
public PipelineException(string message, Exception innerException)
```

Parameters

`message` string

A message that describes the error.

`innerException` Exception

The exception that is the cause of the current exception. If innerException is not a null reference, the current exception is raised in a catch block that handles the inner exception.

PipelineException(string, params object[])

Initializes a new instance of the PipelineException class with the specified error message.

```
public PipelineException(string message, params object[] messageArgs)
```

Parameters

`message` string

A message that describes the error.

`messageArgs` object[]

Array of strings specifying message-related arguments.

Class ProcessorParameter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents a processor parameter. Processor parameters are automatically discovered by the content pipeline. Therefore, only custom processor developers should use this class directly.

```
[Serializable]
public sealed class ProcessorParameter
```

Inheritance

object ← ProcessorParameter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

DefaultValue

Default value of the processor parameter.

```
public object DefaultValue { get; set; }
```

Property Value

object

Description

Description of the parameter, as specified by the [Description] attribute.

```
public string Description { get; set; }
```

Property Value

string

DisplayName

Name of the parameter displayed in the designer, as specified by the [DisplayName] attribute.

```
public string DisplayName { get; set; }
```

Property Value

string

IsEnum

Gets a value indicating whether the parameter is an enumeration.

```
public bool IsEnum { get; }
```

Property Value

bool

PossibleEnumValues

Available options for enumerated type parameters. For parameters of other types, this value is null.

```
public ReadOnlyCollection<string> PossibleEnumValues { get; }
```

Property Value

ReadOnlyCollection<string>

PropertyName

Name of the property, as defined in the C# code.

```
public string PropertyName { get; }
```

Property Value

string

PropertyType

Type of the parameter.

```
public string PropertyType { get; }
```

Property Value

string

Class ProcessorParameterCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents a collection of processor parameters, usually for a single processor. This class is primarily designed for internal use or for custom processor developers.

```
[Serializable]
public sealed class ProcessorParameterCollection :
ReadOnlyCollection<ProcessorParameter>, IList<ProcessorParameter>,
ICollection<ProcessorParameter>, IReadOnlyList<ProcessorParameter>,
IReadOnlyCollection<ProcessorParameter>, IEnumerable<ProcessorParameter>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ProcessorParameter> ← ProcessorParameterCollection

Implements

IList<ProcessorParameter>, ICollection<ProcessorParameter>,
IReadOnlyList<ProcessorParameter>, IReadOnlyCollection<ProcessorParameter>,
IEnumerable<ProcessorParameter>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ProcessorParameter>.Contains(ProcessorParameter) ,
ReadOnlyCollection<ProcessorParameter>.CopyTo(ProcessorParameter[], int) ,
ReadOnlyCollection<ProcessorParameter>.GetEnumerator() ,
ReadOnlyCollection<ProcessorParameter>.IndexOf(ProcessorParameter) ,
ReadOnlyCollection<ProcessorParameter>.Count ,
ReadOnlyCollection<ProcessorParameter>.Empty ,
ReadOnlyCollection<ProcessorParameter>.this[int] , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Enum TargetPlatform

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Identifiers for the target platform.

```
[TypeConverter(typeof(TargetPlatformTypeConverter))]
public enum TargetPlatform
```

Fields

`Android = 3`

Android-based devices (MonoGame)

`DesktopGL = 4`

All desktop versions using OpenGL. (MonoGame)

`DesktopVK = 13`

All desktop versions using Vulkan.

`MacOSX = 5`

Apple Mac OSX-based devices (iMac, MacBook, MacBook Air, etc) (MonoGame)

`NativeClient = 6`

Google Chrome Native Client (MonoGame)

`PlayStation4 = 8`

Sony PlayStation4

`PlayStation5 = 9`

Sony PlayStation5

`RaspberryPi = 7`

Raspberry Pi (MonoGame)

`Switch = 11`

Nintendo Switch

`Web = 12`

WebAssembly and Bridge.NET

`Windows = 0`

All desktop versions of Windows using DirectX.

`WindowsDX12 = 14`

Windows using DirectX 12

`Xbox360 = 1`

Xbox 360 video game and entertainment system

`XboxOne = 10`

Xbox One

`XboxSeries = 15`

Xbox Series S|X

`iOS = 2`

Apple iOS-based devices (iPod Touch, iPhone, iPad) (MonoGame)

Class TextureImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading texture files for use in the Content Pipeline.

```
[ContentImporter(new string[] { ".bmp", ".cut", ".dds", ".g3", ".hdr", ".gif",
".ico", ".iff", ".jbg", ".jbig", ".jng", ".jpg", ".jpeg", ".jpe", ".jif", ".jfif",
".jfi", ".jp2", ".j2k", ".jpf", ".jpx", ".jpm", ".mj2", ".jxr", ".hdp", ".wdp",
".koa", ".gg", ".pcd", ".mng", ".pcx", ".pbm", ".pgm", ".ppm", ".pnm", ".pfm",
".png", ".pict", ".pct", ".pic", ".psd", ".3fr", ".ari", ".arw", ".bay", ".crw",
".cr2", ".cap", ".dcs", ".dcr", ".dng", ".drf", ".eip", ".erf", ".fff", ".iiq",
".k25", ".kdc", ".mdc", ".mef", ".mos", ".mrw", ".nef", ".nrw", ".obm", ".orf",
".pef", ".ptx", ".pxn", ".r3d", ".raf", ".raw", ".rwl", ".rw2", ".rwz", ".sr2",
".srf", ".srw", ".x3f", ".ras", ".sun", ".sgi", ".rgba", ".bw", ".int", ".inta",
".tga", ".tiff", ".tif", ".wbmp", ".webp", ".xbm", ".xpm" }, DisplayName = "Texture
Importer - MonoGame", DefaultProcessor = "TextureProcessor")]
public class TextureImporter : ContentImporter<TextureContent>, IContentImporter
```

Inheritance

object ← ContentImporter<TextureContent> ← TextureImporter

Implements

IContentImporter

Inherited Members

ContentImporter<TextureContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TextureImporter()

Initializes a new instance of TextureImporter.

```
public TextureImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a texture file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override TextureContent Import(string filename, ContentImporterContext
context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

TextureContent

Resulting game asset.

Class VideoContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class for all video objects.

```
public class VideoContent : ContentItem, IDisposable
```

Inheritance

object ← ContentItem ← VideoContent

Implements

IDisposable

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

VideoContent(string)

Initializes a new copy of the VideoContent class for the specified video file.

```
public VideoContent(string filename)
```

Parameters

`filename` string

The file name of the video to import.

Properties

BitsPerSecond

Gets the bit rate for this video.

```
public int BitsPerSecond { get; }
```

Property Value

int

Duration

Gets the duration of this video.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

Filename

Gets or sets the file name for this video.

```
[ContentSerializer]
public string Filename { get; set; }
```

Property Value

string

FramesPerSecond

Gets the frame rate for this video.

```
public float FramesPerSecond { get; }
```

Property Value

float

Height

Gets the height of this video.

```
public int Height { get; }
```

Property Value

int

VideoSoundtrackType

Gets or sets the type of soundtrack accompanying the video.

```
[ContentSerializer]
public VideoSoundtrackType VideoSoundtrackType { get; set; }
```

Property Value

VideoSoundtrackType

Width

Gets the width of this video.

```
public int Width { get; }
```

Property Value

int

Methods

Dispose()

Immediately releases the unmanaged resources used by this object.

```
public void Dispose()
```

Dispose(bool)

Immediately releases the unmanaged resources used by this object.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

true to release both managed and unmanaged resources; false to release only unmanaged resources.

~VideoContent()

```
protected ~VideoContent()
```

Class WavImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading .wav audio files for use in the Content Pipeline.

```
[ContentImporter(".wav", DisplayName = "Wav Importer - MonoGame", DefaultProcessor
= "SoundEffectProcessor")]
public class WavImporter : ContentImporter<AudioContent>, IContentImporter
```

Inheritance

object ← ContentImporter<AudioContent> ← WavImporter

Implements

IContentImporter

Inherited Members

ContentImporter<AudioContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a .wav audio file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override AudioContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

AudioContent

Resulting game asset.

Remarks

This importer supports PCM in unsigned 8-bit, signed 16-bit, signed 24-bit, IEEE Float 32-bit, MS-ADPCM or IMA/ADPCM with sample rates from 8KHz up to 48KHz.

Class WmaImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading Windows Media Audio (.wma) files for use in the Content Pipeline.

```
[ContentImporter(".wma", DisplayName = "Wma Importer - MonoGame", DefaultProcessor
= "SongProcessor")]
public class WmaImporter : ContentImporter<AudioContent>, IContentImporter
```

Inheritance

object ← ContentImporter<AudioContent> ← WmaImporter

Implements

IContentImporter

Inherited Members

ContentImporter<AudioContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

WmaImporter()

Initializes a new instance of WmaImporter.

```
public WmaImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a .wma file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override AudioContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

AudioContent

Resulting game asset.

Class WmvImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading Windows Media Video (.wmv) files for use in the Content Pipeline.

```
[ContentImporter(".wmv", DisplayName = "Wmv Importer - MonoGame", DefaultProcessor
= "VideoProcessor")]
public class WmvImporter : ContentImporter<VideoContent>, IContentImporter
```

Inheritance

object ← ContentImporter<VideoContent> ← WmvImporter

Implements

IContentImporter

Inherited Members

ContentImporter<VideoContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

WmvImporter()

Initializes a new instance of WmvImporter.

```
public WmvImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a .wmv file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override VideoContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

VideoContent

Resulting game asset.

Class XImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading DirectX Object (.x) files for use in the Content Pipeline.

```
[ContentImporter(".x", DisplayName = "X Importer - MonoGame", DefaultProcessor
= "ModelProcessor")]
public class XImporter : ContentImporter<NodeContent>, IContentImporter
```

Inheritance

object ← ContentImporter<NodeContent> ← XImporter

Implements

IContentImporter

Inherited Members

ContentImporter<NodeContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

XImporter()

Initializes a new instance of XImporter.

```
public XImporter()
```

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing a .x file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override NodeContent Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

NodeContent

Resulting game asset.

Class XmlImporter

Namespace: Microsoft.Xna.Framework.Content.Pipeline

Assembly: MonoGame.Framework.Content.Pipeline.dll

Implements an importer for reading intermediate XML files. This is a wrapper around IntermediateSerializer.

```
[ContentImporter(".xml", DisplayName = "Xml Importer - MonoGame", DefaultProcessor
= "PassThroughProcessor")]
public class XmlImporter : ContentImporter<object>, IContentImporter
```

Inheritance

object ← ContentImporter<object> ← XmlImporter

Implements

IContentImporter

Inherited Members

ContentImporter<object>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Import(string, ContentImporterContext)

Called by the XNA Framework when importing an intermediate file to be used as a game asset. This is the method called by the XNA Framework when an asset is to be imported into an object that can be recognized by the Content Pipeline.

```
public override object Import(string filename, ContentImporterContext context)
```

Parameters

`filename` string

Name of a game asset file.

`context` ContentImporterContext

Contains information for importing a game asset, such as a logger interface.

Returns

object

The imported game asset.

Namespace Microsoft.Xna.Framework.Content.Pipeline.Audio

Classes

AudioContent

Encapsulates and provides operations, such as format conversions, on the source audio. This type is produced by the audio importers and used by audio processors to produce compiled audio assets.

AudioFormat

Encapsulates the native audio format (WAVEFORMATEX) information of the audio content.

AudioProfile

Provides a base class for handling audio profiles.

Enums

AudioFileType

Type of the audio file.

ConversionFormat

Target formats supported for audio source conversions.

ConversionQuality

Compression quality of the audio content.

Class AudioContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Encapsulates and provides operations, such as format conversions, on the source audio. This type is produced by the audio importers and used by audio processors to produce compiled audio assets.

```
public class AudioContent : ContentItem, IDisposable
```

Inheritance

object ← ContentItem ← AudioContent

Implements

IDisposable

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Note that AudioContent can load and process audio files that are not supported by the importers.

Constructors

AudioContent(string, AudioFileType)

Initializes a new instance of AudioContent.

```
public AudioContent(string audioFileName, AudioFileType audioFileType)
```

Parameters

`audioFileName` string

Name of the audio source file to be processed.

`audioFileType` AudioFileType

Type of the processed audio: WAV, MP3 or WMA.

Remarks

Constructs the object from the specified source file, in the format specified.

Properties

Data

The current raw audio data without header information.

```
public ReadOnlyCollection<byte> Data { get; }
```

Property Value

ReadOnlyCollection<byte>

Remarks

This changes from the source data to the output data after conversion. For MP3 and WMA files this throws an exception to match XNA behavior.

Duration

The duration of the audio data.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

FileName

The name of the original source audio file.

```
[ContentSerializer(AllowNull = false)]
public string FileName { get; }
```

Property Value

string

FileType

The type of the original source audio file.

```
public AudioFileType FileType { get; }
```

Property Value

AudioFileType

Format

The current format of the audio data.

```
public AudioFormat Format { get; }
```

Property Value

AudioFormat

Remarks

This changes from the source format to the output format after conversion.

LoopLength

The current loop length in samples.

```
public int LoopLength { get; }
```

Property Value

int

Remarks

This changes from the source loop length to the output loop length after conversion.

LoopStart

The current loop start location in samples.

```
public int LoopStart { get; }
```

Property Value

int

Remarks

This changes from the source loop start to the output loop start after conversion.

Methods

ConvertFormat(ConversionFormat, ConversionQuality, string)

Transcodes the source audio to the target format and quality.

```
[Obsolete("You should prefer to use AudioProfile.")]
public void ConvertFormat(ConversionFormat formatType, ConversionQuality quality,
string saveToFile)
```

Parameters

`formatType` ConversionFormat

Format to convert this audio to.

`quality` ConversionQuality

Quality of the processed output audio. For streaming formats, it can be one of the following: Low (96 kbps), Medium (128 kbps), Best (192 kbps). For WAV formats, it can be one of the following: Low (11kHz ADPCM), Medium (22kHz ADPCM), Best (44kHz PCM)

`saveToFile` string

The name of the file that the converted audio should be saved into. This is used for SongContent, where the audio is stored external to the XNB file. If this is null, then the converted audio is stored in the Data property.

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

SetData(byte[], AudioFormat, TimeSpan, int, int)

Sets the data and properties for the audio content.

```
public void SetData(byte[] data, AudioFormat format, TimeSpan duration, int
loopStart, int loopLength)
```

Parameters

`data` byte[]

The audio data to be set.

`format` AudioFormat

The audio format.

`duration` TimeSpan

The duration of the audio.

`loopStart` int

The start position of the loop.

`loopLength` int

The length of the loop.

Remarks

The values for LoopStart and LoopLength are consumed as a number of samples and not a concrete unit of time.

Exceptions

ArgumentNullException

Thrown when the data or format is null.

Enum AudioFileType

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Type of the audio file.

```
public enum AudioFileType
```

Fields

`Mp3 = 0`

The MP3 format

`Ogg = 3`

The Ogg format

`Wav = 1`

The WAV format

`Wma = 2`

The WMA format

Class AudioFormat

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Encapsulates the native audio format (WAVEFORMATEX) information of the audio content.

```
public sealed class AudioFormat
```

Inheritance

object ← AudioFormat

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

AverageBytesPerSecond

Gets the average bytes processed per second.

```
public int AverageBytesPerSecond { get; }
```

Property Value

int

Average bytes processed per second.

BitsPerSample

Gets the bit depth of the audio content.

```
public int BitsPerSample { get; }
```

Property Value

int

If the audio has not been processed, the source bit depth; otherwise, the bit depth of the new format.

BlockAlign

Gets the number of bytes per sample block, taking channels into consideration. For example, for 16-bit stereo audio (PCM format), the size of each sample block is 4 bytes.

```
public int BlockAlign { get; }
```

Property Value

int

Number of bytes, per sample block.

ChannelCount

Gets the number of channels.

```
public int ChannelCount { get; }
```

Property Value

int

If the audio has not been processed, the source channel count; otherwise, the new channel count.

Format

Gets the format of the audio content.

```
public int Format { get; }
```

Property Value

int

If the audio has not been processed, the format tag of the source content; otherwise, the new format tag.

NativeWaveFormat

Gets the raw byte buffer for the format. For non-PCM formats, this buffer contains important format-specific information beyond the basic format information exposed in other properties of the AudioFormat type.

```
public ReadOnlyCollection<byte> NativeWaveFormat { get; }
```

Property Value

ReadOnlyCollection<byte>

The raw byte buffer represented in a collection.

SampleRate

Gets the sample rate of the audio content.

```
public int SampleRate { get; }
```

Property Value

int

If the audio has not been processed, the source sample rate; otherwise, the new sample rate.

Class AudioProfile

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class for handling audio profiles.

```
public abstract class AudioProfile
```

Inheritance

object ← AudioProfile

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ConvertAudio(TargetPlatform, ConversionQuality, AudioContent)

Converts the audio content to work on targeted platform.

```
public abstract ConversionQuality ConvertAudio(TargetPlatform platform,
ConversionQuality quality, AudioContent content)
```

Parameters

`platform` TargetPlatform

The platform to build the audio content for.

`quality` ConversionQuality

The suggested audio quality level.

`content` AudioContent

The audio content to convert.

Returns

ConversionQuality

The quality used for conversion which could be different from the suggested quality.

ConvertStreamingAudio(TargetPlatform, ConversionQuality, AudioContent, ref string)

Converts the audio content to a streaming format that works on targeted platform.

```
public abstract ConversionQuality ConvertStreamingAudio(TargetPlatform platform,
ConversionQuality quality, AudioContent content, ref string outputFileName)
```

Parameters

`platform` TargetPlatform

The platform to build the audio content for.

`quality` ConversionQuality

The suggested audio quality level.

`content` AudioContent

he audio content to convert.

`outputFileName` string

Returns

ConversionQuality

The quality used for conversion which could be different from the suggested quality.

ForPlatform(TargetPlatform)

Find the profile for this target platform.

```
public static AudioProfile ForPlatform(TargetPlatform platform)
```

Parameters

`platform` TargetPlatform

The platform target for audio.

Returns

AudioProfile

QualityToBitRate(ConversionQuality)

```
protected static int QualityToBitRate(ConversionQuality quality)
```

Parameters

`quality` ConversionQuality

Returns

int

QualityToSampleRate(ConversionQuality, int)

Calculates the sample rate based on the conversion quality and the source sample rate.

```
protected static int QualityToSampleRate(ConversionQuality quality,
int sourceSampleRate)
```

Parameters

`quality` ConversionQuality

The target ConversionQuality

`sourceSampleRate` int

The source sample rate.

Returns

int

The calculated sample rate.

Supports(TargetPlatform)

Returns true if this profile supports audio processing for this platform.

```
public abstract bool Supports(TargetPlatform platform)
```

Parameters

`platform` TargetPlatform

Returns

bool

Enum ConversionFormat

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Target formats supported for audio source conversions.

```
public enum ConversionFormat
```

Fields

`Aac = 5`

Advanced Audio Coding

`Adpcm = 0`

Microsoft ADPCM encoding technique using 4 bits

`ImaAdpcm = 4`

QuickTime ADPCM format

`Mp3 = 7`

mp3 audio format

`Pcm = 1`

8/16-bit mono/stereo PCM audio 8KHz-48KHz

`Vorbis = 6`

Vorbis open, patent-free audio encoding

`WindowsMedia = 2`

Windows Media CBR formats (64 kbps, 128 kbps, 192 kbps)

`Xma = 3`

The Xbox compression format

Enum ConversionQuality

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Audio

Assembly: MonoGame.Framework.Content.Pipeline.dll

Compression quality of the audio content.

```
public enum ConversionQuality
```

Fields

`Best = 2`

Lowest compression, but the best audio quality

`Low = 0`

High compression yielding lower file size, but could compromise audio quality

`Medium = 1`

Moderate compression resulting in a compromise between audio quality and file size

Namespace Microsoft.Xna.Framework.Content.Pipeline.Builder.Convertors

Classes

StringToColorConverter

Class that provides methods to convert a Color to and from a string.

Class StringToColorConverter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Builder.Convertors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class that provides methods to convert a Color to and from a string.

```
public class StringToColorConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← StringToColorConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) , TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetConvertFromException(object) ,
TypeConverter.GetConvertToException(object, Type) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,
TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,

TypeConverter.GetStandardValuesSupported() ,
TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
TypeConverter.SortProperties(PropertyDescriptorCollection, string[]) , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

CanConvertTo(ITypeDescriptorContext, Type)

Returns whether this converter can convert the object to the specified type, using the specified context.

```
public override bool CanConvertTo(ITypeDescriptorContext context,
Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type you want to convert to.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type)

Converts the given value object to the specified type, using the specified context and culture information.

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo
culture, object value, Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If null is passed, the current culture is assumed.

`value` object

The object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

object

An object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is null.

NotSupportedException

The conversion cannot be performed.

Namespace Microsoft.Xna.Framework.Content.Pipeline.Graphics

Classes

AlphaTestMaterialContent

Class to provide properties for processing basic materials with alpha.

AnimationChannel

Provides methods and properties for maintaining an animation channel. An animation channel is a collection of keyframes describing the movement of a single bone or rigid object.

AnimationChannelDictionary

Collection of animation data channels, one per bone or rigid object.

AnimationContent

Provides properties for maintaining an animation.

AnimationContentDictionary

Collection of named animations.

AnimationKeyframe

Provides methods and properties for managing a keyframe. A keyframe describes the position of an animation channel at a single point in time.

AstcBitmapContent

AtcBitmapContent

Provides properties and methods for creating and maintaining an ATC compressed bitmap resource.

AtcExplicitBitmapContent

Supports the processing of a texture compressed using ATC Explicit Alpha.

AtcInterpolatedBitmapContent

Provides properties and methods for creating and maintaining an ATC interpolated compressed bitmap resource.

BasicMaterialContent

Class to provide properties for processing basic materials.

BitmapContent

Provides properties and methods for creating and maintaining a bitmap resource.

BoneContent

Represents an animation skeleton.

BoneWeightCollection

Collection of bone weights of a vertex.

CharacterRegionTypeConverter

DualTextureMaterialContent

Dxt1BitmapContent

Provides properties and methods for creating and maintaining a Dxt1 compressed bitmap resource.

Dxt3BitmapContent

Provides properties and methods for creating and maintaining a Dxt3 compressed bitmap resource.

Dxt5BitmapContent

Provides properties and methods for creating and maintaining a Dxt5 compressed bitmap resource.

DxtBitmapContent

EffectContent

Contains the source code for a DirectX Effect, loaded from a .fx file.

EffectMaterialContent

Provides methods and properties for maintaining a collection of named effect texture references.

EnvironmentMapMaterialContent

Class to provide properties for processing environmental map materials.

Etc1BitmapContent

Supports the processing of a texture compressed using ETC1.

Etc2BitmapContent

Supports the processing of a texture compressed using ETC2.

FontDescription

Provides information to the FontDescriptionProcessor describing which font to rasterize, which font size to utilize, and which Unicode characters to include in the processor output.

GeometryContent

Provides properties that define various aspects of a geometry batch.

GeometryContentCollection

Provides methods for maintaining a collection of geometry batches that make up a mesh.

GraphicsUtil

IndexCollection

Provides methods for maintaining a list of index values.

IndirectPositionCollection

Provides methods for maintaining a list of vertex positions.

LocalizedFontDescription

Normally, when you add a .spritefont file to your project, this data is deserialized into a FontDescription object, which is then built into a SpriteFontContent by the FontDescriptionProcessor. But to localize the font, we want to add some additional data, so our custom processor can know what .resx files it needs to scan. We do this by defining our own custom font description class, deriving from the built in FontDescription type, and adding a new property to store the resource filenames.

MaterialContent

Provides methods and properties for maintaining a collection of named texture references.

MaxRectsBin

A bin that can pack rectangles using the MaxRects algorithm with several heuristics.

MeshBuilder

Provides methods for building a mesh for use in the Content Pipeline.

MeshContent

Provides properties and methods that define various aspects of a mesh.

MeshHelper

Class to provide helper methods for processing mesh data.

MipmapChain

Provides methods for accessing a mipmap chain.

MipmapChainCollection

Provides methods for maintaining a mipmap chain.

NodeContent

Provides a base class for graphics types that define local coordinate systems.

NodeContentCollection

Provides a collection of child NodeContent objects for a content item.

PixelBitmapContent<T>

Provides properties and methods for creating and maintaining a bitmap resource.

PositionCollection

Provides a collection of vertex position values.

PvrtcBitmapContent

Provides properties and methods for creating and maintaining a compressed bitmap resource.

PvrtcRgb2BitmapContent

PvrtcRgb4BitmapContent

Provides properties and methods for creating and maintaining a compressed bitmap resource.

PvrtcRgba2BitmapContent

PvrtcRgba4BitmapContent

Provides properties and methods for creating and maintaining a compressed bitmap resource.

SkinnedMaterialContent

Class to provide properties for processing skinned materials.

SpriteFontContent

Texture2DContent

Provides properties for maintaining a 2D texture

Texture3DContent

Provides properties for maintaining a 3D texture

TextureContent

Provides a base class for all texture objects.

TextureCubeContent

Provides a base class for texture cube content.

TextureProfile

Base class for handling texture profiles.

TextureReferenceDictionary

Provides a collection of named references to texture files.

VertexChannel

Provides methods and properties for maintaining a vertex channel. A vertex channel is a list of arbitrary data with one value for each vertex. Channels are stored inside a GeometryContent and identified by name.

VertexChannelCollection

Provides methods and properties for managing a list of vertex data channels.

VertexChannelNames

Provides properties for managing a collection of vertex channel names.

VertexChannel<T>

Provides methods and properties for maintaining a vertex channel. This is a generic implementation of VertexChannel and, therefore, can handle strongly typed content data.

VertexContent

Provides methods and properties for maintaining the vertex data of a GeometryContent.

Structs

ABCFloat

Structure used to store float values sequentially

BoneWeight

Provides properties for managing a bone weight.

CharacterRegion

Enums

FontDescriptionStyle

Flags that describe style information to be applied to text. You can combine these flags by using a bitwise OR operator (|).

GrowRule

How to grow the bin when a rectangel can't be placed.

MaxRectsHeuristic

Heuristic method to use to choose the rectangles positions.

Struct ABCFloat

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Structure used to store float values sequentially

```
public struct ABCFloat
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

A

```
public float A
```

Field Value

float

B

```
public float B
```

Field Value

float

C

```
public float C
```

Field Value

float

Class AlphaTestMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide properties for processing basic materials with alpha.

```
public class AlphaTestMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← AlphaTestMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

AlphaFunctionKey

String key of the alpha function.

```
public const string AlphaFunctionKey = "AlphaFunction"
```

Field Value

string

AlphaKey

Gets or sets the alpha property.

```
public const string AlphaKey = "Alpha"
```

Field Value

string

DiffuseColorKey

String key of the diffuse color.

```
public const string DiffuseColorKey = "DiffuseColor"
```

Field Value

string

ReferenceAlphaKey

String key of the alpha reference.

```
public const string ReferenceAlphaKey = "ReferenceAlpha"
```

Field Value

string

TextureKey

String key of the texture.

```
public const string TextureKey = "Texture"
```

Field Value

string

VertexColorEnabledKey

String key of the vertex color enable flag.

```
public const string VertexColorEnabledKey = "VertexColorEnabled"
```

Field Value

string

Properties

Alpha

Gets or sets the alpha property.

```
public float? Alpha { get; set; }
```

Property Value

float?

AlphaFunction

Gets or Sets the compare function for alpha tests.

```
public CompareFunction? AlphaFunction { get; set; }
```

Property Value

CompareFunction?

DiffuseColor

String key of the diffuse color.

```
public Vector3? DiffuseColor { get; set; }
```

Property Value

Vector3?

ReferenceAlpha

Gets or sets the alpha reference property.

```
public int? ReferenceAlpha { get; set; }
```

Property Value

int?

Texture

Gets or sets the texture property.

```
public ExternalReference<TextureContent> Texture { get; set; }
```

Property Value

ExternalReference<TextureContent>

VertexColorEnabled

String key of the vertex color enable flag.

```
public bool? VertexColorEnabled { get; set; }
```

Property Value

bool?

Class AnimationChannel

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining an animation channel. An animation channel is a collection of keyframes describing the movement of a single bone or rigid object.

```
public sealed class AnimationChannel : ICollection<AnimationKeyframe>,
IEnumerable<AnimationKeyframe>, IEnumerable
```

Inheritance

object ← AnimationChannel

Implements

ICollection<AnimationKeyframe>, IEnumerable<AnimationKeyframe>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AnimationChannel()

Initializes a new instance of AnimationChannel.

```
public AnimationChannel()
```

Properties

Count

Gets the number of keyframes in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Gets the keyframe at the specified index position.

```
public AnimationKeyframe this[int index] { get; }
```

Parameters

`index` int

Property Value

AnimationKeyframe

Methods

Add(AnimationKeyframe)

Adds a new keyframe to the collection, automatically sorting the contents according to keyframe times.

```
public int Add(AnimationKeyframe item)
```

Parameters

`item` AnimationKeyframe

Keyframe to be added to the channel.

Returns

int

Index of the new keyframe.

Clear()

Removes all keyframes from the collection.

```
public void Clear()
```

Contains(AnimationKeyframe)

Searches the collection for the specified keyframe.

```
public bool Contains(AnimationKeyframe item)
```

Parameters

`item` AnimationKeyframe

Keyframe being searched for.

Returns

bool

true if the keyframe exists; false otherwise.

GetEnumerator()

Returns an enumerator that iterates through the keyframes.

```
public IEnumerator<AnimationKeyframe> GetEnumerator()
```

Returns

IEnumerator<AnimationKeyframe>

Enumerator for the keyframe collection.

IndexOf(AnimationKeyframe)

Determines the index for the specified keyframe.

```
public int IndexOf(AnimationKeyframe item)
```

Parameters

`item` AnimationKeyframe

Identity of a keyframe.

Returns

int

Index of the specified keyframe.

Remove(AnimationKeyframe)

Removes the specified keyframe from the collection.

```
public bool Remove(AnimationKeyframe item)
```

Parameters

`item` AnimationKeyframe

Keyframe being removed.

Returns

bool

true if the keyframe was removed; false otherwise.

RemoveAt(int)

Removes the keyframe at the specified index position.

```
public void RemoveAt(int index)
```

Parameters

`index` int

Index of the keyframe being removed.

Class AnimationChannelDictionary

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Collection of animation data channels, one per bone or rigid object.

```
public sealed class AnimationChannelDictionary :
NamedValueDictionary<AnimationChannel>, IDictionary<string, AnimationChannel>,
ICollection<KeyValuePair<string, AnimationChannel>>,
IEnumerable<KeyValuePair<string, AnimationChannel>>, IEnumerable
```

Inheritance

object ← NamedValueDictionary<AnimationChannel> ← AnimationChannelDictionary

Implements

IDictionary<string, AnimationChannel>,
ICollection<KeyValuePair<string, AnimationChannel>>,
IEnumerable<KeyValuePair<string, AnimationChannel>>, IEnumerable

Inherited Members

NamedValueDictionary<AnimationChannel>.Add(string, AnimationChannel) ,
NamedValueDictionary<AnimationChannel>.ContainsKey(string) ,
NamedValueDictionary<AnimationChannel>.Keys ,
NamedValueDictionary<AnimationChannel>.Remove(string) ,
NamedValueDictionary<AnimationChannel>.TryGetValue(string, out AnimationChannel) ,
NamedValueDictionary<AnimationChannel>.Values ,
NamedValueDictionary<AnimationChannel>.this[string] ,
NamedValueDictionary<AnimationChannel>.Clear() ,
NamedValueDictionary<AnimationChannel>.Count ,
NamedValueDictionary<AnimationChannel>.GetEnumerator() , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

AnimationChannelDictionary()

Initializes a new instance of AnimationChannelDictionary.

```
public AnimationChannelDictionary()
```

Class AnimationContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties for maintaining an animation.

```
public class AnimationContent : ContentItem
```

Inheritance

object ← ContentItem ← AnimationContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AnimationContent()

Initializes a new instance of AnimationContent.

```
public AnimationContent()
```

Properties

Channels

Gets the collection of animation data channels. Each channel describes the movement of a single bone or rigid object.

```
public AnimationChannelDictionary Channels { get; }
```

Property Value

AnimationChannelDictionary

Duration

Gets or sets the total length of the animation.

```
public TimeSpan Duration { get; set; }
```

Property Value

TimeSpan

Class AnimationContentDictionary

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Collection of named animations.

```
public sealed class AnimationContentDictionary :
NamedValueDictionary<AnimationContent>, IDictionary<string, AnimationContent>,
ICollection<KeyValuePair<string, AnimationContent>>,
IEnumerable<KeyValuePair<string, AnimationContent>>, IEnumerable
```

Inheritance

object ← NamedValueDictionary<AnimationContent> ← AnimationContentDictionary

Implements

IDictionary<string, AnimationContent>, ICollection<KeyValuePair<string, AnimationContent>>, IEnumerable<KeyValuePair<string, AnimationContent>>, IEnumerable

Inherited Members

NamedValueDictionary<AnimationContent>.Add(string, AnimationContent) ,
NamedValueDictionary<AnimationContent>.ContainsKey(string) ,
NamedValueDictionary<AnimationContent>.Keys ,
NamedValueDictionary<AnimationContent>.Remove(string) ,
NamedValueDictionary<AnimationContent>.TryGetValue(string, out AnimationContent) ,
NamedValueDictionary<AnimationContent>.Values ,
NamedValueDictionary<AnimationContent>.this[string] ,
NamedValueDictionary<AnimationContent>.Clear() , NamedValueDictionary<AnimationContent>.Count ,
NamedValueDictionary<AnimationContent>.GetEnumerator() , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

AnimationContentDictionary()

Initializes a new instance of AnimationContentDictionary.

```
public AnimationContentDictionary()
```

Class AnimationKeyframe

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for managing a keyframe. A keyframe describes the position of an animation channel at a single point in time.

```
public sealed class AnimationKeyframe : IComparable<AnimationKeyframe>
```

Inheritance

object ← AnimationKeyframe

Implements

IComparable<AnimationKeyframe>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

AnimationKeyframe(TimeSpan, Matrix)

Initializes a new instance of AnimationKeyframe with the specified time offsetand transform.

```
public AnimationKeyframe(TimeSpan time, Matrix transform)
```

Parameters

`time` TimeSpan

Time offset of the keyframe.

`transform` Matrix

Position of the keyframe.

Properties

Time

Gets the time offset from the start of the animation to the position described by this keyframe.

```
public TimeSpan Time { get; }
```

Property Value

TimeSpan

Transform

Gets or sets the position described by this keyframe.

```
public Matrix Transform { get; set; }
```

Property Value

Matrix

Methods

CompareTo(AnimationKeyframe)

Compares this instance of a keyframe to another.

```
public int CompareTo(AnimationKeyframe other)
```

Parameters

`other` AnimationKeyframe

Keyframe being compared to.

Returns

int

Indication of their relative values.

Class AstcBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class AstcBitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← AstcBitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ToString() , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

AstcBitmapContent()

Initializes a new instance of AstcBitmapContent.

```
protected AstcBitmapContent()
```

AstcBitmapContent(int, int)

Initializes a new instance of AstcBitmapContent with the specified width or height.

```
public AstcBitmapContent(int width, int height)
```

Parameters

`width` int

Width in pixels of the bitmap resource.

`height` int

Height in pixels of the bitmap resource.

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class AtcBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining an ATC compressed bitmap resource.

```
public abstract class AtcBitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← AtcBitmapContent

Derived

AtcExplicitBitmapContent, AtcInterpolatedBitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ToString() , BitmapContent.TryGetFormat(out SurfaceFormat) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

AtcBitmapContent()

Initializes a new instance of AtcBitmapContent.

```
public AtcBitmapContent()
```

AtcBitmapContent(int, int)

Initializes a new instance of AtcBitmapContent with the specified width or height.

```
public AtcBitmapContent(int width, int height)
```

Parameters

`width` int

Width, in pixels, of the bitmap resource.

`height` int

Height, in pixels, of the bitmap resource.

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

Class AtcExplicitBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Supports the processing of a texture compressed using ATC Explicit Alpha.

```
public class AtcExplicitBitmapContent : AtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← AtcBitmapContent ← AtcExplicitBitmapContent

Inherited Members

AtcBitmapContent.GetPixelData() , AtcBitmapContent.SetPixelData(byte[]) , AtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , AtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

AtcExplicitBitmapContent()

Creates an instance of AtcExplicitBitmapContent.

```
public AtcExplicitBitmapContent()
```

AtcExplicitBitmapContent(int, int)

Creates an instance of AtcExplicitBitmapContent with the specified width and height.

```
public AtcExplicitBitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class AtcInterpolatedBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining an ATC interpolated compressed bitmap resource.

```
public class AtcInterpolatedBitmapContent : AtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← AtcBitmapContent ← AtcInterpolatedBitmapContent

Inherited Members

AtcBitmapContent.GetPixelData() , AtcBitmapContent.SetPixelData(byte[]) , AtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , AtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

AtcInterpolatedBitmapContent()

Creates an instance of AtcInterpolatedBitmapContent.

```
public AtcInterpolatedBitmapContent()
```

AtcInterpolatedBitmapContent(int, int)

Creates an instance of AtcInterpolatedBitmapContent with the specified width and height.

```
public AtcInterpolatedBitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class BasicMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide properties for processing basic materials.

```
public class BasicMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← BasicMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

AlphaKey

String key of the alpha value.

```
public const string AlphaKey = "Alpha"
```

Field Value

string

DiffuseColorKey

String key of the diffuse color.

```
public const string DiffuseColorKey = "DiffuseColor"
```

Field Value

string

EmissiveColorKey

String key of the emissive color.

```
public const string EmissiveColorKey = "EmissiveColor"
```

Field Value

string

SpecularColorKey

String key of the specular color.

```
public const string SpecularColorKey = "SpecularColor"
```

Field Value

string

SpecularPowerKey

String key of the specular power.

```
public const string SpecularPowerKey = "SpecularPower"
```

Field Value

string

TextureKey

String key of the texture.

```
public const string TextureKey = "Texture"
```

Field Value

string

VertexColorEnabledKey

String key of the vertex color enable flag.

```
public const string VertexColorEnabledKey = "VertexColorEnabled"
```

Field Value

string

Properties

Alpha

Gets or sets the alpha property.

```
public float? Alpha { get; set; }
```

Property Value

float?

DiffuseColor

Gets or sets the diffuse color property.

```
public Vector3? DiffuseColor { get; set; }
```

Property Value

Vector3?

EmissiveColor

Gets or sets the emissive color property.

```
public Vector3? EmissiveColor { get; set; }
```

Property Value

Vector3?

SpecularColor

Gets or sets the specular color property.

```
public Vector3? SpecularColor { get; set; }
```

Property Value

Vector3?

SpecularPower

Gets or sets the specular power property.

```
public float? SpecularPower { get; set; }
```

Property Value

float?

Texture

Gets or sets the texture property.

```
public ExternalReference<TextureContent> Texture { get; set; }
```

Property Value

ExternalReference<TextureContent>

VertexColorEnabled

Gets or sets the vertex color enabled property.

```
public bool? VertexColorEnabled { get; set; }
```

Property Value

bool?

Class BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a bitmap resource.

```
public abstract class BitmapContent : ContentItem
```

Inheritance

object ← ContentItem ← BitmapContent

Derived

AstcBitmapContent, AtcBitmapContent, DxtBitmapContent, Etc1BitmapContent, Etc2BitmapContent, PixelBitmapContent<T>, PvrtcBitmapContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

BitmapContent()

Initializes a new instance of BitmapContent.

```
protected BitmapContent()
```

BitmapContent(int, int)

Initializes a new instance of BitmapContent with the specified width or height.

```
protected BitmapContent(int width, int height)
```

Parameters

`width` int

Width, in pixels, of the bitmap resource.

`height` int

Height, in pixels, of the bitmap resource.

Properties

Height

Gets or sets the height of the bitmap, in pixels.

```
public int Height { get; protected set; }
```

Property Value

int

Width

Gets or sets the width of the bitmap, in pixels.

```
public int Width { get; protected set; }
```

Property Value

int

Methods

Copy(BitmapContent, BitmapContent)

Copies one bitmap into another. The destination bitmap can be in any format and size. If the destination is larger or smaller, the source bitmap is scaled accordingly.

```
public static void Copy(BitmapContent sourceBitmap, BitmapContent destinationBitmap)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

Copy(BitmapContent, Rectangle, BitmapContent, Rectangle)

Copies one bitmap into another. The destination bitmap can be in any format and size. If the destination is larger or smaller, the source bitmap is scaled accordingly.

```
public static void Copy(BitmapContent sourceBitmap, Rectangle sourceRegion,
BitmapContent destinationBitmap, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Region of sourceBitmap.

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`destinationRegion` Rectangle

Region of bitmap to be overwritten.

GetPixelData()

Reads encoded bitmap content.

```
public abstract byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public abstract void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

ToString()

Returns a string description of the bitmap resource.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected abstract bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected abstract bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public abstract bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle)

Validates the arguments to the Copy function.

```
protected static void ValidateCopyArguments(BitmapContent sourceBitmap, Rectangle
sourceRegion, BitmapContent destinationBitmap, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`destinationRegion` Rectangle

Region of bitmap to be overwritten.

Class BoneContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents an animation skeleton.

```
public class BoneContent : NodeContent
```

Inheritance

object ← ContentItem ← NodeContent ← BoneContent

Inherited Members

NodeContent.AbsoluteTransform , NodeContent.Animations , NodeContent.Children , NodeContent.Parent , NodeContent.Transform , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BoneContent()

Initializes a new instance of BoneContent.

```
public BoneContent()
```

Struct BoneWeight

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties for managing a bone weight.

```
public struct BoneWeight
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

BoneWeight(string, float)

Initializes a new instance of BoneWeight with the specified name and weight.

```
public BoneWeight(string boneName, float weight)
```

Parameters

`boneName` string

Name of the bone.

`weight` float

Amount of influence, ranging from zero to one.

Properties

BoneName

Gets the name of the bone.

```
public string BoneName { get; }
```

Property Value

string

Weight

Gets the amount of bone influence, ranging from zero to one. The complete set of weights in a BoneWeightCollection should sum to one.

```
public float Weight { get; }
```

Property Value

float

Class BoneWeightCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Collection of bone weights of a vertex.

```
public sealed class BoneWeightCollection : Collection<BoneWeight>,
IList<BoneWeight>, ICollection<BoneWeight>, IReadOnlyList<BoneWeight>,
IReadOnlyCollection<BoneWeight>, IEnumerable<BoneWeight>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<BoneWeight> ← BoneWeightCollection

Implements

IList<BoneWeight>, ICollection<BoneWeight>, IReadOnlyList<BoneWeight>, IReadOnlyCollection<BoneWeight>, IEnumerable<BoneWeight>, IList, ICollection, IEnumerable

Inherited Members

Collection<BoneWeight>.Add(BoneWeight) , Collection<BoneWeight>.Clear() , Collection<BoneWeight>.Contains(BoneWeight) , Collection<BoneWeight>.CopyTo(BoneWeight[], int) , Collection<BoneWeight>.GetEnumerator() , Collection<BoneWeight>.IndexOf(BoneWeight) , Collection<BoneWeight>.Insert(int, BoneWeight) , Collection<BoneWeight>.Remove(BoneWeight) , Collection<BoneWeight>.RemoveAt(int) , Collection<BoneWeight>.Count , Collection<BoneWeight>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BoneWeightCollection()

Initializes a new instance of BoneWeightCollection.

```
public BoneWeightCollection()
```

Methods

NormalizeWeights()

Normalizes the contents of the weights list.

```
public void NormalizeWeights()
```

NormalizeWeights(int)

Normalizes the contents of the bone weights list.

```
public void NormalizeWeights(int maxWeights)
```

Parameters

`maxWeights` int

Maximum number of weights allowed.

Struct CharacterRegion

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[TypeConverter(typeof(CharacterRegionTypeConverter))]
public struct CharacterRegion
```

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

CharacterRegion(char, char)

```
public CharacterRegion(char start, char end)
```

Parameters

`start` char

`end` char

Fields

Default

```
public static CharacterRegion Default
```

Field Value

CharacterRegion

End

```
public char End
```

Field Value

char

Start

```
public char Start
```

Field Value

char

Methods

Any<T>(IEnumerable<T>)

Test if there is an element in this enumeration.

```
public static bool Any<T>(IEnumerable<T> source)
```

Parameters

`source` IEnumerable<T>

The enumerable source.

Returns

bool

`true` if there is an element in this enumeration, `false` otherwise

Type Parameters

`T`

Type of the element

Characters()

```
public IEnumerable<char> Characters()
```

Returns

IEnumerable<char>

Distinct<TSource>(IEnumerable<TSource>, IEqualityComparer<TSource>)

Selects distinct elements from an enumeration.

```
public static IEnumerable<TSource> Distinct<TSource>(IEnumerable<TSource> source,
IEqualityComparer<TSource> comparer = null)
```

Parameters

`source` IEnumerable<TSource>

The source.

`comparer` IEqualityComparer<TSource>

The comparer.

Returns

IEnumerable<TSource>

A enumeration of selected values

Type Parameters

`TSource`

The type of the T source.

SelectMany<TSource, TResult>(IEnumerable<TSource>, Func<TSource, IEnumerable<TResult>>)

Select elements from an enumeration.

```
public static IEnumerable<TResult> SelectMany<TSource, TResult>(IEnumerable<TSource>
source, Func<TSource, IEnumerable<TResult>> selector)
```

Parameters

`source` IEnumerable<TSource>

The source.

`selector` Func<TSource, IEnumerable<TResult>>

The selector.

Returns

IEnumerable<TResult>

A enumeration of selected values

Type Parameters

`TSource`

The type of the T source.

`TResult`

The type of the T result.

Class CharacterRegionTypeConverter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class CharacterRegionTypeConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← CharacterRegionTypeConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) ,
TypeConverter.CanConvertTo(ITypeDescriptorContext, Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) ,
TypeConverter.ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type) ,
TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetConvertFromException(object) ,
TypeConverter.GetConvertToException(object, Type) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,

TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesSupported() ,
TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
TypeConverter.SortProperties(PropertyDescriptorCollection, string[]) , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

Class DualTextureMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class DualTextureMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← DualTextureMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

AlphaKey

String key of the alpha value.

```
public const string AlphaKey = "Alpha"
```

Field Value

string

DiffuseColorKey

String key of the diffuse color.

```
public const string DiffuseColorKey = "DiffuseColor"
```

Field Value

string

Texture2Key

String key of the second texture.

```
public const string Texture2Key = "Texture2"
```

Field Value

string

TextureKey

String key of the texture.

```
public const string TextureKey = "Texture"
```

Field Value

string

VertexColorEnabledKey

String key of the vertex color enable flag.

```
public const string VertexColorEnabledKey = "VertexColorEnabled"
```

Field Value

string

Properties

Alpha

Gets or sets the alpha property.

```
public float? Alpha { get; set; }
```

Property Value

float?

DiffuseColor

Gets or sets the diffuse color property.

```
public Vector3? DiffuseColor { get; set; }
```

Property Value

Vector3?

Texture

Gets or sets the texture property.

```
public ExternalReference<TextureContent> Texture { get; set; }
```

Property Value

ExternalReference<TextureContent>

Texture2

Gets or sets the second texture property.

```
public ExternalReference<TextureContent> Texture2 { get; set; }
```

Property Value

ExternalReference<TextureContent>

VertexColorEnabled

Gets or sets the vertex color enabled property.

```
public bool? VertexColorEnabled { get; set; }
```

Property Value

bool?

Class Dxt1BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a Dxt1 compressed bitmap resource.

```
public class Dxt1BitmapContent : DxtBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← DxtBitmapContent ← Dxt1BitmapContent

Inherited Members

DxtBitmapContent.GetPixelData() , DxtBitmapContent.SetPixelData(byte[]) , DxtBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , DxtBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Dxt1BitmapContent(int, int)

Creates an instance of Dxt1BitmapContent with the specified width and height.

```
public Dxt1BitmapContent(int width, int height)
```

Parameters

`width` int

The width, in pixels, of the bitmap.

`height` int

The height, in pixels, of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class Dxt3BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a Dxt3 compressed bitmap resource.

```
public class Dxt3BitmapContent : DxtBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← DxtBitmapContent ← Dxt3BitmapContent

Inherited Members

DxtBitmapContent.GetPixelData() , DxtBitmapContent.SetPixelData(byte[]) , DxtBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , DxtBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Dxt3BitmapContent(int, int)

Creates an instance of Dxt3BitmapContent with the specified width and height.

```
public Dxt3BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class Dxt5BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a Dxt5 compressed bitmap resource.

```
public class Dxt5BitmapContent : DxtBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← DxtBitmapContent ← Dxt5BitmapContent

Inherited Members

DxtBitmapContent.GetPixelData() , DxtBitmapContent.SetPixelData(byte[]) , DxtBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , DxtBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Dxt5BitmapContent(int, int)

Creates an instance of Dxt5BitmapContent with the specified width and height.

```
public Dxt5BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class DxtBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public abstract class DxtBitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← DxtBitmapContent

Derived

Dxt1BitmapContent, Dxt3BitmapContent, Dxt5BitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ToString() , BitmapContent.TryGetFormat(out SurfaceFormat) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

DxtBitmapContent(int)

```
protected DxtBitmapContent(int blockSize)
```

Parameters

`blockSize` int

DxtBitmapContent(int, int, int)

```
protected DxtBitmapContent(int blockSize, int width, int height)
```

Parameters

`blockSize` int

`width` int

`height` int

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

Class EffectContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Contains the source code for a DirectX Effect, loaded from a .fx file.

```
public class EffectContent : ContentItem
```

Inheritance

object ← ContentItem ← EffectContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

EffectContent()

Initializes a new instance of EffectContent.

```
public EffectContent()
```

Properties

EffectCode

Gets or sets the effect program source code.

```
public string EffectCode { get; set; }
```

Property Value

string

Class EffectMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining a collection of named effect texture references.

```
public class EffectMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← EffectMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

CompiledEffectKey

Key to use in the external reference dictionary for the compiled effect content.

```
public const string CompiledEffectKey = "CompiledEffect"
```

Field Value

string

EffectKey

Key to use in the external reference dictionary for the effect content.

```
public const string EffectKey = "Effect"
```

Field Value

string

Properties

CompiledEffect

Returns or sets the external reference to the effects compiled content.

```
public ExternalReference<CompiledEffectContent> CompiledEffect { get; set; }
```

Property Value

ExternalReference<CompiledEffectContent>

Effect

Returns or sets the external reference to the effects content.

```
public ExternalReference<EffectContent> Effect { get; set; }
```

Property Value

ExternalReference<EffectContent>

Class EnvironmentMapMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide properties for processing environmental map materials.

```
public class EnvironmentMapMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← EnvironmentMapMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

AlphaKey

String key of the alpha value.

```
public const string AlphaKey = "Alpha"
```

Field Value

string

DiffuseColorKey

String key of the diffuse color.

```
public const string DiffuseColorKey = "DiffuseColor"
```

Field Value

string

EmissiveColorKey

String key of the emissive color.

```
public const string EmissiveColorKey = "EmissiveColor"
```

Field Value

string

EnvironmentMapAmountKey

string key of environment map amount.

```
public const string EnvironmentMapAmountKey = "EnvironmentMapAmount"
```

Field Value

string

EnvironmentMapKey

String key of the environment map.

```
public const string EnvironmentMapKey = "EnvironmentMap"
```

Field Value

string

EnvironmentMapSpecularKey

String key of the specular color.

```
public const string EnvironmentMapSpecularKey = " EnvironmentMapSpecular"
```

Field Value

string

FresnelFactorKey

String key of the fresnel factor.

```
public const string FresnelFactorKey = "FresnelFactor"
```

Field Value

string

TextureKey

String key of the texture.

```
public const string TextureKey = "Texture"
```

Field Value

string

Properties

Alpha

Gets or sets the alpha property.

```
public float? Alpha { get; set; }
```

Property Value

float?

DiffuseColor

Gets or sets the diffuse color property.

```
public Vector3? DiffuseColor { get; set; }
```

Property Value

Vector3?

EmissiveColor

Gets or sets the emissive color property.

```
public Vector3? EmissiveColor { get; set; }
```

Property Value

Vector3?

EnvironmentMap

Gets or sets the environment map property.

```
public ExternalReference<TextureContent> EnvironmentMap { get; set; }
```

Property Value

ExternalReference<TextureContent>

EnvironmentMapAmount

Gets or sets the environment map amount property.

```
public float? EnvironmentMapAmount { get; set; }
```

Property Value

float?

EnvironmentMapSpecular

Gets or sets the specular color property.

```
public Vector3? EnvironmentMapSpecular { get; set; }
```

Property Value

Vector3?

FresnelFactor

Gets or sets the fresnel factor property.

```
public float? FresnelFactor { get; set; }
```

Property Value

float?

Texture

Gets or sets the texture property.

```
public ExternalReference<TextureContent> Texture { get; set; }
```

Property Value

ExternalReference<TextureContent>

Class Etc1BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Supports the processing of a texture compressed using ETC1.

```
public class Etc1BitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← Etc1BitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Etc1BitmapContent()

Initializes a new instance of Etc1BitmapContent.

```
protected Etc1BitmapContent()
```

Etc1BitmapContent(int, int)

Initializes a new instance of Etc1BitmapContent with the specified width or height.

```
public Etc1BitmapContent(int width, int height)
```

Parameters

`width` int

Width in pixels of the bitmap resource.

`height` int

Height in pixels of the bitmap resource.

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class Etc2BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Supports the processing of a texture compressed using ETC2.

```
public class Etc2BitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← Etc2BitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Etc2BitmapContent()

Initializes a new instance of Etc2BitmapContent.

```
protected Etc2BitmapContent()
```

Etc2BitmapContent(int, int)

Initializes a new instance of Etc2BitmapContent with the specified width or height.

```
public Etc2BitmapContent(int width, int height)
```

Parameters

`width` int

Width in pixels of the bitmap resource.

`height` int

Height in pixels of the bitmap resource.

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class FontDescription

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides information to the FontDescriptionProcessor describing which font to rasterize, which font size to utilize, and which Unicode characters to include in the processor output.

```
public class FontDescription : ContentItem
```

Inheritance

object ← ContentItem ← FontDescription

Derived

LocalizedFontDescription

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FontDescription(string, float, float)

Initializes a new instance of FontDescription and initializes its members to the specified font, size, and spacing, using FontDescriptionStyle.Regular as the default value for Style.

```
public FontDescription(string fontName, float size, float spacing)
```

Parameters

`fontName` string

The name of the font, such as Times New Roman.

`size` float

The size, in points, of the font.

`spacing` float

The amount of space, in pixels, to insert between letters in a string.

FontDescription(string, float, float, FontDescriptionStyle)

Initializes a new instance of FontDescription and initializes its members to the specified font, size, spacing, and style.

```
public FontDescription(string fontName, float size, float spacing,
FontDescriptionStyle fontStyle)
```

Parameters

`fontName` string

The name of the font, such as Times New Roman.

`size` float

The size, in points, of the font.

`spacing` float

The amount of space, in pixels, to insert between letters in a string.

`fontStyle` FontDescriptionStyle

The font style for the font.

FontDescription(string, float, float, FontDescriptionStyle, bool)

Initializes a new instance of FontDescription using the specified values.

```
public FontDescription(string fontName, float size, float spacing,
FontDescriptionStyle fontStyle, bool useKerning)
```

Parameters

`fontName` string

The name of the font, such as Times New Roman.

`size` float

The size, in points, of the font.

`spacing` float

The amount of space, in pixels, to insert between letters in a string.

`fontStyle` FontDescriptionStyle

The font style for the font.

`useKerning` bool

true if kerning information is used when drawing characters; false otherwise.

Properties

Characters

Retrieves the set of characters to include in the processor output.

```
public ICollection<char> Characters { get; }
```

Property Value

ICollection<char>

DefaultCharacter

Gets or sets the default character for the font.

```
[ContentSerializer(Optional = true)]
public char? DefaultCharacter { get; set; }
```

Property Value

char?

FontName

Gets or sets the name of the font, such as "Times New Roman" or "Arial". This value cannot be null or empty.

```
[ContentSerializer(AllowNull = false)]
public string FontName { get; set; }
```

Property Value

string

Size

Gets or sets the size, in points, of the font.

```
public float Size { get; set; }
```

Property Value

float

Spacing

Gets or sets the amount of space, in pixels, to insert between letters in a string.

```
[ContentSerializer(Optional = true)]
public float Spacing { get; set; }
```

Property Value

float

Style

Gets or sets the style of the font, expressed as a combination of one or more FontDescriptionStyle flags.

```
public FontDescriptionStyle Style { get; set; }
```

Property Value

FontDescriptionStyle

UseKerning

Indicates if kerning information is used when drawing characters.

```
[ContentSerializer(Optional = true)]
public bool UseKerning { get; set; }
```

Property Value

bool

Enum FontDescriptionStyle

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Flags that describe style information to be applied to text. You can combine these flags by using a bitwise OR operator (|).

```
[Flags]
public enum FontDescriptionStyle
```

Fields

`Bold = 0`

Bold text.

`Italic = 1`

Italic text.

`Regular = 2`

Normal text.

Class GeometryContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties that define various aspects of a geometry batch.

```
public class GeometryContent : ContentItem
```

Inheritance

object ← ContentItem ← GeometryContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GeometryContent()

Creates an instance of GeometryContent.

```
public GeometryContent()
```

Properties

Indices

Gets the list of triangle indices for this geometry batch. Geometry is stored as an indexed triangle list, where each group of three indices defines a single triangle.

```
public IndexCollection Indices { get; }
```

Property Value

IndexCollection

Material

Gets or sets the material of the parent mesh.

```
public MaterialContent Material { get; set; }
```

Property Value

MaterialContent

Parent

Gets or sets the parent MeshContent for this object.

```
public MeshContent Parent { get; set; }
```

Property Value

MeshContent

Vertices

Gets the set of vertex batches for the geometry batch.

```
public VertexContent Vertices { get; }
```

Property Value

VertexContent

Class GeometryContentCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for maintaining a collection of geometry batches that make up a mesh.

```
public sealed class GeometryContentCollection : ChildCollection<MeshContent,
GeometryContent>, IList<GeometryContent>, ICollection<GeometryContent>,
IReadOnlyList<GeometryContent>, IReadOnlyCollection<GeometryContent>,
IEnumerable<GeometryContent>, IList, ICollection, IEnumerable
```

Inheritance

object ← Collection<GeometryContent> ← ChildCollection<MeshContent, GeometryContent> ← GeometryContentCollection

Implements

IList<GeometryContent>, ICollection<GeometryContent>, IReadOnlyList<GeometryContent>, IReadOnlyCollection<GeometryContent>, IEnumerable<GeometryContent>, IList, ICollection, IEnumerable

Inherited Members

Collection<GeometryContent>.Add(GeometryContent) , Collection<GeometryContent>.Clear() , Collection<GeometryContent>.Contains(GeometryContent) , Collection<GeometryContent>.CopyTo(GeometryContent[], int) , Collection<GeometryContent>.GetEnumerator() , Collection<GeometryContent>.IndexOf(GeometryContent) , Collection<GeometryContent>.Insert(int, GeometryContent) , Collection<GeometryContent>.Remove(GeometryContent) , Collection<GeometryContent>.RemoveAt(int) , Collection<GeometryContent>.Count , Collection<GeometryContent>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetParent(GeometryContent)

Gets the parent of a child object.

```
protected override MeshContent GetParent(GeometryContent child)
```

Parameters

`child` GeometryContent

The child of the parent being retrieved.

Returns

MeshContent

The parent of the child object.

SetParent(GeometryContent, MeshContent)

Sets the parent of the specified child object.

```
protected override void SetParent(GeometryContent child, MeshContent parent)
```

Parameters

`child` GeometryContent

The child of the parent being set.

`parent` MeshContent

The parent of the child object.

Class GraphicsUtil

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public static class GraphicsUtil
```

Inheritance

object ← GraphicsUtil

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CompressAstc(ContentProcessorContext, TextureContent, bool)

```
public static void CompressAstc(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

CompressAti(ContentProcessorContext, TextureContent, bool)

```
public static void CompressAti(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

CompressColor16Bit(ContentProcessorContext, TextureContent)

```
public static void CompressColor16Bit(ContentProcessorContext context,
TextureContent content)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

CompressDxt(ContentProcessorContext, TextureContent, bool)

```
public static void CompressDxt(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

CompressEtc(ContentProcessorContext, TextureContent, bool)

```
public static void CompressEtc(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

CompressEtc1(ContentProcessorContext, TextureContent, bool)

```
public static void CompressEtc1(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

CompressFontDXT3(TextureContent)

```
public static void CompressFontDXT3(TextureContent content)
```

Parameters

`content` TextureContent

CompressPvrtc(ContentProcessorContext, TextureContent, bool)

```
public static void CompressPvrtc(ContentProcessorContext context, TextureContent
content, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`isSpriteFont` bool

GetNextPowerOfTwo(int)

Returns the next power of two. Returns same value if already is PoT.

```
public static int GetNextPowerOfTwo(int value)
```

Parameters

`value` int

Returns

int

IsPowerOfTwo(int)

```
public static bool IsPowerOfTwo(int x)
```

Parameters

`x` int

Returns

bool

Enum GrowRule

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

How to grow the bin when a rectangel can't be placed.

```
public enum GrowRule
```

Fields

`Both = 3`

Alternate growing the bins width and height. Starts with width.

`Height = 2`

Grow the bins height.

`None = 0`

Don't grow the bin, throw an exception when full.

`Width = 1`

Grow the bins width.

Class IndexCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for maintaining a list of index values.

```
public sealed class IndexCollection : Collection<int>, IList<int>, ICollection<int>,
IReadOnlyList<int>, IReadOnlyCollection<int>, IEnumerable<int>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<int> ← IndexCollection

Implements

IList<int>, ICollection<int>, IReadOnlyList<int>, IReadOnlyCollection<int>, IEnumerable<int>, IList, ICollection, IEnumerable

Inherited Members

Collection<int>.Add(int) , Collection<int>.Clear() , Collection<int>.Contains(int) , Collection<int>.CopyTo(int[], int) , Collection<int>.GetEnumerator() , Collection<int>.IndexOf(int) , Collection<int>.Insert(int, int) , Collection<int>.Remove(int) , Collection<int>.RemoveAt(int) , Collection<int>.Count , Collection<int>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IndexCollection()

Initializes a new instance of IndexCollection.

```
public IndexCollection()
```

Methods

AddRange(IEnumerable<int>)

Add a range of indices to the collection.

```
public void AddRange(IEnumerable<int> indices)
```

Parameters

`indices` IEnumerable<int>

A collection of indices to add.

Class IndirectPositionCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for maintaining a list of vertex positions.

```
public sealed class IndirectPositionCollection : IList<Vector3>,
ICollection<Vector3>, IEnumerable<Vector3>, IEnumerable
```

Inheritance

object ← IndirectPositionCollection

Implements

IList<Vector3>, ICollection<Vector3>, IEnumerable<Vector3>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class is designed to collect the vertex positions for a VertexContent object. Use the contents of the PositionIndices property (of the contained VertexContent object) to index into the Positions property of the parent mesh.

Properties

Count

Number of positions in the collection.

```
public int Count { get; }
```

Property Value

int

Number of positions.

this[int]

Gets or sets the position at the specified index.

```
public Vector3 this[int index] { get; set; }
```

Parameters

`index` int

Property Value

Vector3

Position located at index.

Methods

Contains(Vector3)

Determines whether the specified position is in the collection.

```
public bool Contains(Vector3 item)
```

Parameters

`item` Vector3

Position being searched for in the collection.

Returns

bool

true if the position was found; false otherwise.

CopyTo(Vector3[], int)

Copies the specified positions to an array, starting at the specified index.

```
public void CopyTo(Vector3[] array, int arrayIndex)
```

Parameters

`array` Vector3[]

Array of positions to be copied.

`arrayIndex` int

Index of the first copied position.

GetEnumerator()

Gets an enumerator interface for reading the position values.

```
public IEnumerator<Vector3> GetEnumerator()
```

Returns

IEnumerator<Vector3>

Interface for enumerating the collection of position values.

IndexOf(Vector3)

Gets the index of the specified position in a collection.

```
public int IndexOf(Vector3 item)
```

Parameters

`item` Vector3

Position being searched for.

Returns

int

Index of the specified position or -1 if not found.

Class LocalizedFontDescription

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Normally, when you add a .spritefont file to your project, this data is deserialized into a FontDescription object, which is then built into a SpriteFontContent by the FontDescriptionProcessor. But to localize the font, we want to add some additional data, so our custom processor can know what .resx files it needs to scan. We do this by defining our own custom font description class, deriving from the built in FontDescription type, and adding a new property to store the resource filenames.

```
public class LocalizedFontDescription : FontDescription
```

Inheritance

object ← ContentItem ← FontDescription ← LocalizedFontDescription

Inherited Members

FontDescription.FontName , FontDescription.Size , FontDescription.Spacing , FontDescription.UseKerning , FontDescription.Style , FontDescription.DefaultCharacter , FontDescription.Characters , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

LocalizedFontDescription()

Constructor.

```
public LocalizedFontDescription()
```

Properties

ResourceFiles

Add a new property to our font description, which will allow us to include a ResourceFiles element in the .spritefont XML. We use the ContentSerializer attribute to mark this as optional, so existing .spritefont files that do not include this ResourceFiles element can be imported as well.

```
[ContentSerializer(Optional = true, CollectionItemName = "Resx")]
public List<string> ResourceFiles { get; }
```

Property Value

List<string>

Class MaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining a collection of named texture references.

```
public class MaterialContent : ContentItem
```

Inheritance

object ← ContentItem ← MaterialContent

Derived

AlphaTestMaterialContent, BasicMaterialContent, DualTextureMaterialContent, EffectMaterialContent, EnvironmentMapMaterialContent, SkinnedMaterialContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

In addition to texture references, opaque data values are stored in the OpaqueData property of the base class.

Constructors

MaterialContent()

Initializes a new instance of MaterialContent.

```
public MaterialContent()
```

Properties

Textures

Gets the texture collection of the material.

```
public TextureReferenceDictionary Textures { get; }
```

Property Value

TextureReferenceDictionary

Collection of textures used by the material.

Methods

Clone()

Helper method to make a copy of a material.

```
public MaterialContent Clone()
```

Returns

MaterialContent

A clone of the material.

GetReferenceTypeProperty<T>(string)

Gets a reference type from the OpaqueDataDictionary collection.

```
protected T GetReferenceTypeProperty<T>(string key) where T : class
```

Parameters

`key` string

Key of the property being retrieved.

Returns

T

The related opaque data.

Type Parameters

T

Type of the related opaque data.

GetTexture(string)

Gets a value from the Textures collection.

```
protected ExternalReference<TextureContent> GetTexture(string key)
```

Parameters

key string

Key of the texture being retrieved.

Returns

ExternalReference<TextureContent>

Reference to a texture from the collection.

GetValueTypeProperty<T>(string)

Gets a value type from the OpaqueDataDictionary collection.

```
protected T? GetValueTypeProperty<T>(string key) where T : struct
```

Parameters

`key` string

Key of the value type being retrieved.

Returns

T?

Index of the value type beng retrieved.

Type Parameters

`T`

Type of the value being retrieved.

SetProperty<T>(string, T)

Sets a value in the contained OpaqueDataDictionary object. If null is passed, the value is removed.

```
protected void SetProperty<T>(string key, T value)
```

Parameters

`key` string

Name of the key being modified.

`value` T

Value being set.

Type Parameters

`T`

Type of the element being set.

SetTexture(string, ExternalReference<TextureContent>)

Sets a value in the contained TextureReferenceDictionary object. If null is passed, the value is removed.

```
protected void SetTexture(string key, ExternalReference<TextureContent> value)
```

Parameters

`key` string

Name of the key being modified.

`value` ExternalReference<TextureContent>

Value being set.

Remarks

The key value differs depending on the type of attached dictionary. If attached to a BasicMaterialContent dictionary (which becomes a BasicEffect object at run time), the value for the Texture key is used as the texture for the BasicEffect runtime object. Other keys are ignored. If attached to a EffectMaterialContent dictionary, key names are the texture names used by the effect. These names are dependent upon the author of the effect object.

Class MaxRectsBin

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

A bin that can pack rectangles using the MaxRects algorithm with several heuristics.

```
public class MaxRectsBin
```

Inheritance

object ← MaxRectsBin

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MaxRectsBin(int, int, GrowRule, float, bool)

Create a MaxRectsBin instance.

```
public MaxRectsBin(int width = 64, int height = 64, GrowRule grow = GrowRule.Both,
float growIncrement = -2, bool allowRotation = false)
```

Parameters

`width` int

Initial available width. Defaults to 64.

`height` int

Initial available height. Defaults to 64.

`grow` GrowRule

Whether to allow growing the available space when a rectangle doesn't fit.

`growIncrement` float

Determines how the bin is grown. A positive value means a fixed size increase, a negative value means a factor increase. E.g. -2 will grow the bin by a factor of 2 when it needs to grow.

`allowRotation` bool

Whether to allow rotating the rectangles.

Properties

AllowRotations

Indicates if rectangle can be rotated.

```
public bool AllowRotations { get; }
```

Property Value

bool

BinHeight

Height of the bin.

```
public int BinHeight { get; }
```

Property Value

int

BinWidth

Width of the bin.

```
public int BinWidth { get; }
```

Property Value

int

Grow

Rule to choose in which dimension to grow.

```
public GrowRule Grow { get; set; }
```

Property Value

GrowRule

GrowIncrement

Determines how the bin is grown. A positive value means a fixed size increase, a negative value means a factor increase. E.g. -2 will grow the bin by a factor of 2 when it needs to grow.

```
public float GrowIncrement { get; set; }
```

Property Value

float

GrowLimit

The number of times to grow the bin for a single rectangle when it doesn't fit. Defaults to 3.

```
public int GrowLimit { get; set; }
```

Property Value

int

IncludePadding

Determines if padding is included in returned rectangles. Default is `false`.

```
public bool IncludePadding { get; set; }
```

Property Value

bool

PaddingHeight

Padding to add to both sides in the vertical dimension. Default is 0.

```
public int PaddingHeight { get; set; }
```

Property Value

int

PaddingWidth

Padding to add to both sides in the horizontal dimension. Default is 0.

```
public int PaddingWidth { get; set; }
```

Property Value

int

UsedHeight

The actually used height of the bin.

```
public int UsedHeight { get; }
```

Property Value

int

UsedWidth

The actually used width of the bin.

```
public int UsedWidth { get; }
```

Property Value

int

Methods

GetOccupancy(bool)

Computes the ratio of used surface area.

```
public float GetOccupancy(bool crop = false)
```

Parameters

`crop` bool

If `true` this uses UsedWidth and UsedHeight instead of BinWidth and BinHeight for available area.

Returns

float

Insert(Point[], MaxRectsHeuristic)

Place an array of rectangles in the bin.

```
public Rectangle[] Insert(Point[] sizes, MaxRectsHeuristic heuristic
= MaxRectsHeuristic.Bssf)
```

Parameters

`sizes` Point[]

Sizes of the rectangles.

`heuristic` MaxRectsHeuristic

Heuristic to use.

Returns

Rectangle[]

Bounds of all rectangles after placement.

Exceptions

ArgumentNullException

If `sizes` is `null`.

Insert(Point[], Rectangle[], int, MaxRectsHeuristic)

Place an array of rectangles in the bin.

```
public void Insert(Point[] sizes, Rectangle[] bounds, int indexOffset = 0,
MaxRectsHeuristic heuristic = MaxRectsHeuristic.Bssf)
```

Parameters

`sizes` Point[]

Sizes of the rectangles.

`bounds` Rectangle[]

Array to fill with rectangle bounds.

`indexOffset` int

Index into `bounds` to start filling.

`heuristic` MaxRectsHeuristic

Heuristic to use.

Exceptions

ArgumentNullException

If `sizes` or `bounds` is `null`.

ArgumentException

If `indexOffset` is less than 0.

ArgumentOutOfRangeException

If `bounds` is not large enough to hold all bounds. I.e. `bounds.Length - indexOffset < sizes.Length`

Insert(int, int, MaxRectsHeuristic)

Insert a rectangle into the bin.

```
public Rectangle Insert(int width, int height, MaxRectsHeuristic heuristic)
```

Parameters

`width` int

Width of the rectangle.

`height` int

Height of the rectangle.

`heuristic` MaxRectsHeuristic

Heuristic to use.

Returns

Rectangle

The bounds of the rectangle after placement.

Exceptions

Exception

If the rectangle did not fit in the bin.

Enum MaxRectsHeuristic

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Heuristic method to use to choose the rectangles positions.

```
public enum MaxRectsHeuristic
```

Fields

`Baf = 2`

Best Area Fit: Positions the rectangle into the smallest free rect into which it fits.

`Bl = 3`

Bottom Left (Tetris method): Each rectangle is placed to a position (possibly rotating it) where its top side lies as low as possible.

`Blsf = 1`

Best Long Side Fit: Positions the rectangle against the long side of a free rectangle into which it fits the best.

`Bssf = 0`

Best Short Side Fit: Positions the rectangle against the short side of a free rectangle into which it fits the best.

`RectContactPointRule = 4`

Contact Point: Choosest the placement where the rectangle touches other rects as much as possible.

Class MeshBuilder

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for building a mesh for use in the Content Pipeline.

```
public sealed class MeshBuilder
```

Inheritance

object ← MeshBuilder

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

MergeDuplicatePositions

Gets or sets the current value for position merging of the mesh.

```
public bool MergeDuplicatePositions { get; set; }
```

Property Value

bool

MergePositionTolerance

Gets or sets the tolerance for MergeDuplicatePositions.

```
public float MergePositionTolerance { get; set; }
```

Property Value

float

Name

Gets or sets the name of the current MeshContent object being processed.

```
public string Name { get; set; }
```

Property Value

string

SwapWindingOrder

Reverses the triangle winding order of the specified mesh.

```
public bool SwapWindingOrder { get; set; }
```

Property Value

bool

Methods

AddTriangleVertex(int)

Adds a vertex into the index collection.

```
public void AddTriangleVertex(int indexIntoVertexCollection)
```

Parameters

`indexIntoVertexCollection` int

Index of the inserted vertex, in the collection. This corresponds to the value returned by Create Position(float, float, float).

CreatePosition(Vector3)

Inserts the specified vertex position into the vertex channel at the specified index.

```
public int CreatePosition(Vector3 pos)
```

Parameters

`pos` Vector3

Value of the vertex being inserted.

Returns

int

Index of the vertex being inserted.

CreatePosition(float, float, float)

Inserts the specified vertex position into the vertex channel.

```
public int CreatePosition(float x, float y, float z)
```

Parameters

`x` float

Value of the x component of the vector.

`y` float

Value of the y component of the vector.

`z` float

Value of the z component of the vector.

Returns

int

Index of the inserted vertex.

CreateVertexChannel<T>(string)

Creates a new vertex channel with the specified usage and adds it to the mesh builder.

```
public int CreateVertexChannel<T>(string usage)
```

Parameters

`usage` string

The usage of the vertex channel.

Returns

int

The new count of vertex channels.

Type Parameters

`T`

The type of data to be stored in the vertex channel.

Exceptions

InvalidOperationException

Thrown if the mesh builder has already been finished.

FinishMesh()

Ends the creation of a mesh.

```
public MeshContent FinishMesh()
```

Returns

MeshContent

Resultant mesh.

SetMaterial(MaterialContent)

Sets the material for the next triangles.

```
public void SetMaterial(MaterialContent material)
```

Parameters

`material` MaterialContent

Material for the next triangles.

Remarks

Sets the material for the triangles being defined next. This material and the opaque data dictionary, set with SetOpaqueData(OpaqueDataDictionary) define the GeometryContent object containing the next triangles. When you set a new material or opaque data dictionary the triangles you add afterwards will belong to a new GeometryContent object.

SetOpaqueData(OpaqueDataDictionary)

Sets the opaque data for the next triangles.

```
public void SetOpaqueData(OpaqueDataDictionary opaqueData)
```

Parameters

`opaqueData` OpaqueDataDictionary

Opaque data dictionary for the next triangles.

Remarks

Sets the opaque data dictionary for the triangles being defined next. This dictionary and the material, set with SetMaterial(MaterialContent), define the GeometryContent object containing the next triangles. When you set a new material or opaque data dictionary the triangles you add afterwards will belong to a new GeometryContent object.

SetVertexChannelData(int, object)

Sets the specified vertex data with new data.

```
public void SetVertexChannelData(int vertexDataIndex, object channelData)
```

Parameters

`vertexDataIndex` int

Index of the vertex data channel being set. This should match the index returned by CreateVertexChannel.

`channelData` object

New data values for the vertex data. The data type being set must match the data type for the vertex channel specified by vertexDataIndex.

StartMesh(string)

Initializes the creation of a mesh.

```
public static MeshBuilder StartMesh(string name)
```

Parameters

`name` string

Name of the mesh.

Returns

MeshBuilder

Object used when building the mesh.

Class MeshContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods that define various aspects of a mesh.

```
public class MeshContent : NodeContent
```

Inheritance

object ← ContentItem ← NodeContent ← MeshContent

Inherited Members

NodeContent.AbsoluteTransform , NodeContent.Animations , NodeContent.Children , NodeContent.Parent , NodeContent.Transform , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MeshContent()

Initializes a new instance of MeshContent.

```
public MeshContent()
```

Properties

Geometry

Gets the list of geometry batches for the mesh.

```
public GeometryContentCollection Geometry { get; }
```

Property Value

GeometryContentCollection

Positions

Gets the list of vertex position values.

```
public PositionCollection Positions { get; }
```

Property Value

PositionCollection

Class MeshHelper

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide helper methods for processing mesh data.

```
public static class MeshHelper
```

Inheritance

object ← MeshHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CalculateNormals(GeometryContent, bool)

Generates vertex normals by accumulation of triangle face normals.

```
public static void CalculateNormals(GeometryContent geom,
bool overwriteExistingNormals)
```

Parameters

`geom` GeometryContent

The geometry which will receive the normals.

`overwriteExistingNormals` bool

Overwrite or skip over geometry with existing normals.

Remarks

We use a "Mean Weighted Equally" method generate vertex normals from triangle face normals. If normal cannot be calculated from the geometry we set it to zero.

CalculateNormals(MeshContent, bool)

Generates vertex normals by accumulation of triangle face normals.

```
public static void CalculateNormals(MeshContent mesh, bool overwriteExistingNormals)
```

Parameters

`mesh` MeshContent

The mesh which will receive the normals.

`overwriteExistingNormals` bool

Overwrite or skip over geometry with existing normals.

Remarks

This calls CalculateNormals(GeometryContent, bool) to do the work.

CalculateTangentFrames(GeometryContent, string, string, string)

Generate the tangents and binormals (tangent frames) for each vertex in the mesh geometry.

```
public static void CalculateTangentFrames(GeometryContent geom, string
textureCoordinateChannelName, string tangentChannelName, string binormalChannelName)
```

Parameters

`geom` GeometryContent

The mesh geometry which will have add tangent and binormal channels added.

`textureCoordinateChannelName` string

The Vector2 texture coordinate channel used to generate tangent frames.

`tangentChannelName` string

`binormalChannelName` string

CalculateTangentFrames(MeshContent, string, string, string)

Generate the tangents and binormals (tangent frames) for each vertex in the mesh.

```
public static void CalculateTangentFrames(MeshContent mesh, string
textureCoordinateChannelName, string tangentChannelName, string binormalChannelName)
```

Parameters

`mesh` MeshContent

The mesh which will have add tangent and binormal channels added.

`textureCoordinateChannelName` string

The Vector2 texture coordinate channel used to generate tangent frames.

`tangentChannelName` string

`binormalChannelName` string

CalculateTangentFrames(IList<Vector3>, IList<int>, IList<Vector3>, IList<Vector2>, out Vector3[], out Vector3[])

Calculates the tangent and bitangent vectors for a mesh.

```
public static void CalculateTangentFrames(IList<Vector3> positions, IList<int>
indices, IList<Vector3> normals, IList<Vector2> textureCoords, out Vector3[]
tangents, out Vector3[] bitangents)
```

Parameters

`positions` IList<Vector3>

The list of vertex positions.

`indices` IList<int>

The list of indices that define the triangles in the mesh.

`normals` IList<Vector3>

The list of vertex normals.

`textureCoords` IList<Vector2>

The list of texture coordinates.

`tangents` Vector3[]

The output list of tangent vectors.

`bitangents` Vector3[]

The output list of bitangent vectors.

FindSkeleton(NodeContent)

Search for the root bone of the skeletion.

```
public static BoneContent FindSkeleton(NodeContent node)
```

Parameters

`node` NodeContent

The node from which to begin the search for the skeleton.

Returns

BoneContent

The root bone of the skeletion or null if none is found.

FlattenSkeleton(BoneContent)

Traverses a skeleton depth-first and builds a list of its bones.

```
public static IList<BoneContent> FlattenSkeleton(BoneContent skeleton)
```

Parameters

`skeleton` BoneContent

Returns

IList<BoneContent>

MergeDuplicatePositions(MeshContent, float)

Merge any positions in the PositionCollection of the specified mesh that are at a distance less than the specified tolerance from each other.

```
public static void MergeDuplicatePositions(MeshContent mesh, float tolerance)
```

Parameters

`mesh` MeshContent

Mesh to be processed.

`tolerance` float

Tolerance value that determines how close positions must be to each other to be merged.

Remarks

This method will also update the PositionIndices in the GeometryContent of the specified mesh.

MergeDuplicateVertices(GeometryContent)

Merge vertices with the same PositionIndices and VertexChannel data within the specified GeometryContent.

```
public static void MergeDuplicateVertices(GeometryContent geometry)
```

Parameters

`geometry` GeometryContent

Geometry to be processed.

MergeDuplicateVertices(MeshContent)

Merge vertices with the same PositionIndices and VertexChannel data within the Geometry of this mesh. If you want to merge positions too, call MergeDuplicatePositions(MeshContent, float) on your mesh before this function.

```
public static void MergeDuplicateVertices(MeshContent mesh)
```

Parameters

`mesh` MeshContent

Mesh to be processed

OptimizeForCache(MeshContent)

Optimizes the mesh for caching.

```
public static void OptimizeForCache(MeshContent mesh)
```

Parameters

`mesh` MeshContent

Mesh data to optimize.

Remarks

This method is not currently implemented.

SwapWindingOrder(MeshContent)

Reverses the triangle winding order of the mesh.

```
public static void SwapWindingOrder(MeshContent mesh)
```

Parameters

`mesh` MeshContent

The mesh which will be modified.

Remarks

This method is useful when changing the direction of backface culling like when switching between left/right handed coordinate systems.

TransformScene(NodeContent, Matrix)

Transforms the contents of a node and its descendants.

```
public static void TransformScene(NodeContent scene, Matrix transform)
```

Parameters

`scene` NodeContent

The root node of the scene to transform.

`transform` Matrix

The transform matrix to apply to the scene.

Remarks

The node transforms themselves are unaffected.

Class MipmapChain

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for accessing a mipmap chain.

```
public sealed class MipmapChain : Collection<BitmapContent>,
IList<BitmapContent>, ICollection<BitmapContent>, IReadOnlyList<BitmapContent>,
IReadOnlyCollection<BitmapContent>, IEnumerable<BitmapContent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<BitmapContent> ← MipmapChain

Implements

IList<BitmapContent>, ICollection<BitmapContent>, IReadOnlyList<BitmapContent>, IReadOnlyCollection<BitmapContent>, IEnumerable<BitmapContent>, IList, ICollection, IEnumerable

Inherited Members

Collection<BitmapContent>.Add(BitmapContent) , Collection<BitmapContent>.Clear() , Collection<BitmapContent>.Contains(BitmapContent) , Collection<BitmapContent>.CopyTo(BitmapContent[], int) , Collection<BitmapContent>.GetEnumerator() , Collection<BitmapContent>.IndexOf(BitmapContent) , Collection<BitmapContent>.Insert(int, BitmapContent) , Collection<BitmapContent>.Remove(BitmapContent) , Collection<BitmapContent>.RemoveAt(int) , Collection<BitmapContent>.Count , Collection<BitmapContent>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MipmapChain()

Initializes a new instance of MipmapChain.

```
public MipmapChain()
```

MipmapChain(BitmapContent)

Initializes a new instance of MipmapChain with the specified mipmap.

```
public MipmapChain(BitmapContent bitmap)
```

Parameters

`bitmap` BitmapContent

Operators

implicit operator MipmapChain(BitmapContent)

Constructs a new mipmap chain containing the specified bitmap.

```
public static implicit operator MipmapChain(BitmapContent bitmap)
```

Parameters

`bitmap` BitmapContent

Bitmap used for the mipmap chain.

Returns

MipmapChain

Resultant mipmap chain.

Class MipmapChainCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for maintaining a mipmap chain.

```
public sealed class MipmapChainCollection : Collection<MipmapChain>,
IList<MipmapChain>, ICollection<MipmapChain>, IReadOnlyList<MipmapChain>,
IReadOnlyCollection<MipmapChain>, IEnumerable<MipmapChain>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<MipmapChain> ← MipmapChainCollection

Implements

IList<MipmapChain>, ICollection<MipmapChain>, IReadOnlyList<MipmapChain>, IReadOnlyCollection<MipmapChain>, IEnumerable<MipmapChain>, IList, ICollection, IEnumerable

Inherited Members

Collection<MipmapChain>.Add(MipmapChain) , Collection<MipmapChain>.Clear() , Collection<MipmapChain>.Contains(MipmapChain) , Collection<MipmapChain>.CopyTo(MipmapChain[], int) , Collection<MipmapChain>.GetEnumerator() , Collection<MipmapChain>.IndexOf(MipmapChain) , Collection<MipmapChain>.Insert(int, MipmapChain) , Collection<MipmapChain>.Remove(MipmapChain) , Collection<MipmapChain>.RemoveAt(int) , Collection<MipmapChain>.Count , Collection<MipmapChain>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ClearItems()

Removes all elements from the Collection<T>.

```
protected override void ClearItems()
```

InsertItem(int, MipmapChain)

Inserts an element into the Collection<T> at the specified index.

```
protected override void InsertItem(int index, MipmapChain item)
```

Parameters

`index` int

The zero-based index at which `item` should be inserted.

`item` MipmapChain

The object to insert. The value can be null for reference types.

Exceptions

ArgumentOutOfRangeException

`index` is less than zero.

-or-

`index` is greater than Count.

RemoveItem(int)

Removes the element at the specified index of the Collection<T>.

```
protected override void RemoveItem(int index)
```

Parameters

`index` int

The zero-based index of the element to remove.

Exceptions

ArgumentOutOfRangeException

`index` is less than zero.

-or-

`index` is equal to or greater than Count.

Class NodeContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class for graphics types that define local coordinate systems.

```
public class NodeContent : ContentItem
```

Inheritance

object ← ContentItem ← NodeContent

Derived

BoneContent, MeshContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NodeContent()

Creates an instance of NodeContent.

```
public NodeContent()
```

Properties

AbsoluteTransform

Gets the value of the local Transform property, multiplied by the AbsoluteTransform of the parent.

```
public Matrix AbsoluteTransform { get; }
```

Property Value

Matrix

Animations

Gets the set of animations belonging to this node.

```
public AnimationContentDictionary Animations { get; }
```

Property Value

AnimationContentDictionary

Children

Gets the children of the NodeContent object.

```
public NodeContentCollection Children { get; }
```

Property Value

NodeContentCollection

Parent

Gets the parent of this NodeContent object.

```
public NodeContent Parent { get; set; }
```

Property Value

NodeContent

Transform

Gets the transform matrix of the scene. The transform matrix defines a local coordinate system for the content in addition to any children of this object.

```
public Matrix Transform { get; set; }
```

Property Value

Matrix

Class NodeContentCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a collection of child NodeContent objects for a content item.

```
public class NodeContentCollection : ChildCollection<NodeContent, NodeContent>,
IList<NodeContent>, ICollection<NodeContent>, IReadOnlyList<NodeContent>,
IReadOnlyCollection<NodeContent>, IEnumerable<NodeContent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← Collection<NodeContent> ← ChildCollection<NodeContent, NodeContent> ← NodeContentCollection

Implements

IList<NodeContent>, ICollection<NodeContent>, IReadOnlyList<NodeContent>, IReadOnlyCollection<NodeContent>, IEnumerable<NodeContent>, IList, ICollection, IEnumerable

Inherited Members

ChildCollection<NodeContent, NodeContent>.ClearItems() ,
ChildCollection<NodeContent, NodeContent>.GetParent(NodeContent) ,
ChildCollection<NodeContent, NodeContent>.InsertItem(int, NodeContent) ,
ChildCollection<NodeContent, NodeContent>.RemoveItem(int) ,
ChildCollection<NodeContent, NodeContent>.SetItem(int, NodeContent) ,
ChildCollection<NodeContent, NodeContent>.SetParent(NodeContent, NodeContent) ,
Collection<NodeContent>.Add(NodeContent) , Collection<NodeContent>.Clear() ,
Collection<NodeContent>.ClearItems() , Collection<NodeContent>.Contains(NodeContent) ,
Collection<NodeContent>.CopyTo(NodeContent[], int) , Collection<NodeContent>.GetEnumerator() ,
Collection<NodeContent>.IndexOf(NodeContent) ,
Collection<NodeContent>.Insert(int, NodeContent) ,
Collection<NodeContent>.InsertItem(int, NodeContent) ,
Collection<NodeContent>.Remove(NodeContent) , Collection<NodeContent>.RemoveAt(int) ,
Collection<NodeContent>.RemoveItem(int) , Collection<NodeContent>.SetItem(int, NodeContent) ,
Collection<NodeContent>.Count , Collection<NodeContent>.this[int] ,
Collection<NodeContent>.Items , object.Equals(object) , object.Equals(object, object) ,

object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Links from a child object to its parent are maintained as the collection contents are modified.

Methods

GetParent(NodeContent)

Gets the parent of a child object.

```
protected override NodeContent GetParent(NodeContent child)
```

Parameters

`child` NodeContent

The child of the parent being retrieved.

Returns

NodeContent

The parent of the child object.

SetParent(NodeContent, NodeContent)

Modifies the value of the parent object of the specified child object.

```
protected override void SetParent(NodeContent child, NodeContent parent)
```

Parameters

`child` NodeContent

The child of the parent being modified.

`parent` NodeContent

The new value for the parent object.

Class PixelBitmapContent<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a bitmap resource.

```
public class PixelBitmapContent<T> : BitmapContent where T : struct, IEquatable<T>
```

Type Parameters

T

Provides properties and methods for creating and maintaining a bitmap resource.

Inheritance

object ← ContentItem ← BitmapContent ← PixelBitmapContent<T>

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ToString() , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PixelBitmapContent(int, int)

Initializes a new instance of PixelBitmapContent with the specified width or height.

```
public PixelBitmapContent(int width, int height)
```

Parameters

`width` int

Width, in pixels, of the bitmap resource.

`height` int

Height, in pixels, of the bitmap resource.

Methods

GetPixel(int, int)

Gets the pixel at the specified location.

```
public T GetPixel(int x, int y)
```

Parameters

`x` int

Pixel x coordinate.

`y` int

Pixel y coordinate.

Returns

T

The pixel for the specified location.

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

GetRow(int)

Returns pixel data for the given row.

```
public T[] GetRow(int y)
```

Parameters

y int

The row index.

Returns

T[]

Array containing the pixel data for the given row.

Exceptions

ArgumentOutOfRangeException

Row is outside the bounds of the bitmap.

ReplaceColor(T, T)

Replaces all pixels of the specified color with the new color.

```
public void ReplaceColor(T originalColor, T newColor)
```

Parameters

originalColor T

The color to replace.

`newColor` T

The color to replace it with.

SetPixel(int, int, T)

Sets the pixel at the specified location.

```
public void SetPixel(int x, int y, T value)
```

Parameters

`x` int

Pixel x coordinate.

`y` int

Pixel y coordinate.

`value` T

The new pixel value.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class PositionCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a collection of vertex position values.

```
public sealed class PositionCollection : Collection<Vector3>, IList<Vector3>,
ICollection<Vector3>, IReadOnlyList<Vector3>, IReadOnlyCollection<Vector3>,
IEnumerable<Vector3>, IList, ICollection, IEnumerable
```

Inheritance

object ← Collection<Vector3> ← PositionCollection

Implements

IList<Vector3>, ICollection<Vector3>, IReadOnlyList<Vector3>, IReadOnlyCollection<Vector3>, IEnumerable<Vector3>, IList, ICollection, IEnumerable

Inherited Members

Collection<Vector3>.Add(Vector3) , Collection<Vector3>.Clear() , Collection<Vector3>.Contains(Vector3) , Collection<Vector3>.CopyTo(Vector3[], int) , Collection<Vector3>.GetEnumerator() , Collection<Vector3>.IndexOf(Vector3) , Collection<Vector3>.Insert(int, Vector3) , Collection<Vector3>.Remove(Vector3) , Collection<Vector3>.RemoveAt(int) , Collection<Vector3>.Count , Collection<Vector3>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PositionCollection()

Initializes a new instance of PositionCollection.

```
public PositionCollection()
```

Class PvrtcBitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a compressed bitmap resource.

```
public abstract class PvrtcBitmapContent : BitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← PvrtcBitmapContent

Derived

PvrtcRgb2BitmapContent, PvrtcRgb4BitmapContent, PvrtcRgba2BitmapContent, PvrtcRgba4BitmapContent

Inherited Members

BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ToString() , BitmapContent.TryGetFormat(out SurfaceFormat) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PvrtcBitmapContent(int, int)

Initializes a new instance of BitmapContent with the specified width or height.

```
public PvrtcBitmapContent(int width, int height)
```

Parameters

`width` int

Width, in pixels, of the bitmap resource.

`height` int

Height, in pixels, of the bitmap resource.

Methods

GetPixelData()

Reads encoded bitmap content.

```
public override byte[] GetPixelData()
```

Returns

byte[]

Array containing encoded bitmap data.

SetPixelData(byte[])

Writes encoded bitmap content.

```
public override void SetPixelData(byte[] sourceData)
```

Parameters

`sourceData` byte[]

Array containing encoded bitmap data to be set.

TryCopyFrom(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region from a specified bitmap.

```
protected override bool TryCopyFrom(BitmapContent sourceBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`sourceBitmap` BitmapContent

BitmapContent being copied.

`sourceRegion` Rectangle

Location of sourceBitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

TryCopyTo(BitmapContent, Rectangle, Rectangle)

Attempts to copy a region of the specified bitmap onto another.

```
protected override bool TryCopyTo(BitmapContent destinationBitmap, Rectangle
sourceRegion, Rectangle destinationRegion)
```

Parameters

`destinationBitmap` BitmapContent

BitmapContent being overwritten.

`sourceRegion` Rectangle

Location of the source bitmap.

`destinationRegion` Rectangle

Region of destination bitmap to be overwritten.

Returns

bool

true if region copy is supported; false otherwise.

Class PvrtcRgb2BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[Obsolete("PVRTC 2BPP formats are no longer supported, and this will be removed in a
future version.")]
public class PvrtcRgb2BitmapContent : PvrtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← PvrtcBitmapContent ← PvrtcRgb2BitmapContent

Inherited Members

PvrtcBitmapContent.GetPixelData() , PvrtcBitmapContent.SetPixelData(byte[]) , PvrtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , PvrtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PvrtcRgb2BitmapContent(int, int)

Creates an instance of PvrtcRgb2BitmapContent with the specified width and height.

```
public PvrtcRgb2BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class PvrtcRgb4BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a compressed bitmap resource.

```
public class PvrtcRgb4BitmapContent : PvrtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← PvrtcBitmapContent ← PvrtcRgb4BitmapContent

Inherited Members

PvrtcBitmapContent.GetPixelData() , PvrtcBitmapContent.SetPixelData(byte[]) , PvrtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , PvrtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PvrtcRgb4BitmapContent(int, int)

Creates an instance of PvrtcRgb4BitmapContent with the specified width and height.

```
public PvrtcRgb4BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class PvrtcRgba2BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[Obsolete("PVRTC 2BPP formats are no longer supported, and this will be removed in a
future version.")]
public class PvrtcRgba2BitmapContent : PvrtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← PvrtcBitmapContent ← PvrtcRgba2BitmapContent

Inherited Members

PvrtcBitmapContent.GetPixelData() , PvrtcBitmapContent.SetPixelData(byte[]) , PvrtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , PvrtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PvrtcRgba2BitmapContent(int, int)

Creates an instance of PvrtcRgba2BitBitmapContent with the specified width and height.

```
public PvrtcRgba2BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class PvrtcRgba4BitmapContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for creating and maintaining a compressed bitmap resource.

```
public class PvrtcRgba4BitmapContent : PvrtcBitmapContent
```

Inheritance

object ← ContentItem ← BitmapContent ← PvrtcBitmapContent ← PvrtcRgba4BitmapContent

Inherited Members

PvrtcBitmapContent.GetPixelData() , PvrtcBitmapContent.SetPixelData(byte[]) , PvrtcBitmapContent.TryCopyFrom(BitmapContent, Rectangle, Rectangle) , PvrtcBitmapContent.TryCopyTo(BitmapContent, Rectangle, Rectangle) , BitmapContent.Height , BitmapContent.Width , BitmapContent.Copy(BitmapContent, BitmapContent) , BitmapContent.Copy(BitmapContent, Rectangle, BitmapContent, Rectangle) , BitmapContent.ValidateCopyArguments(BitmapContent, Rectangle, BitmapContent, Rectangle) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

PvrtcRgba4BitmapContent(int, int)

Creates an instance of PvrtcRgba4BitBitmapContent with the specified width and height.

```
public PvrtcRgba4BitmapContent(int width, int height)
```

Parameters

`width` int

The width in pixels of the bitmap.

`height` int

The height in pixels of the bitmap.

Methods

ToString()

Returns a string description of the bitmap.

```
public override string ToString()
```

Returns

string

Description of the bitmap.

TryGetFormat(out SurfaceFormat)

Gets the corresponding GPU texture format for the specified bitmap type.

```
public override bool TryGetFormat(out SurfaceFormat format)
```

Parameters

`format` SurfaceFormat

Format being retrieved.

Returns

bool

The GPU texture format of the bitmap type.

Class SkinnedMaterialContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide properties for processing skinned materials.

```
public class SkinnedMaterialContent : MaterialContent
```

Inheritance

object ← ContentItem ← MaterialContent ← SkinnedMaterialContent

Inherited Members

MaterialContent.Textures , MaterialContent.GetReferenceTypeProperty<T>(string) , MaterialContent.GetTexture(string) , MaterialContent.GetValueTypeProperty<T>(string) , MaterialContent.SetProperty<T>(string, T) , MaterialContent.SetTexture(string, ExternalReference<TextureContent>) , MaterialContent.Clone() , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

AlphaKey

String key of the alpha value.

```
public const string AlphaKey = "Alpha"
```

Field Value

string

DiffuseColorKey

String key of the diffuse color.

```
public const string DiffuseColorKey = "DiffuseColor"
```

Field Value

string

EmissiveColorKey

String key of the emissive color.

```
public const string EmissiveColorKey = "EmissiveColor"
```

Field Value

string

SpecularColorKey

String key of the specular color.

```
public const string SpecularColorKey = "SpecularColor"
```

Field Value

string

SpecularPowerKey

String key of the specular power.

```
public const string SpecularPowerKey = "SpecularPower"
```

Field Value

string

TextureKey

String key of the texture.

```
public const string TextureKey = "Texture"
```

Field Value

string

WeightsPerVertexKey

String key of the vertex weights.

```
public const string WeightsPerVertexKey = "WeightsPerVertex"
```

Field Value

string

Properties

Alpha

Gets or sets the alpha property.

```
public float? Alpha { get; set; }
```

Property Value

float?

DiffuseColor

Gets or sets the diffuse color property.

```
public Vector3? DiffuseColor { get; set; }
```

Property Value

Vector3?

EmissiveColor

Gets or sets the emissive color property.

```
public Vector3? EmissiveColor { get; set; }
```

Property Value

Vector3?

SpecularColor

Gets or sets the specular color property.

```
public Vector3? SpecularColor { get; set; }
```

Property Value

Vector3?

SpecularPower

Gets or sets the specular power property.

```
public float? SpecularPower { get; set; }
```

Property Value

float?

Texture

Gets or sets the texture property.

```
public ExternalReference<TextureContent> Texture { get; set; }
```

Property Value

ExternalReference<TextureContent>

WeightsPerVertex

Gets or sets the number of weights per vertex.

```
public int? WeightsPerVertex { get; set; }
```

Property Value

int?

Class SpriteFontContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class SpriteFontContent
```

Inheritance

object ← SpriteFontContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SpriteFontContent()

```
public SpriteFontContent()
```

SpriteFontContent(FontDescription)

```
public SpriteFontContent(FontDescription desc)
```

Parameters

`desc` FontDescription

Fields

CharacterMap

```
public List<char> CharacterMap
```

Field Value

List<char>

Cropping

```
public List<Rectangle> Cropping
```

Field Value

List<Rectangle>

DefaultCharacter

```
public char? DefaultCharacter
```

Field Value

char?

FontName

```
public string FontName
```

Field Value

string

FontSize

```
public float FontSize
```

Field Value

float

Glyphs

```
public List<Rectangle> Glyphs
```

Field Value

List<Rectangle>

HorizontalSpacing

```
public float HorizontalSpacing
```

Field Value

float

Kerning

```
public List<Vector3> Kerning
```

Field Value

List<Vector3>

Texture

```
public Texture2DContent Texture
```

Field Value

Texture2DContent

VerticalLineSpacing

```
public int VerticalLineSpacing
```

Field Value

int

Class Texture2DContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties for maintaining a 2D texture

```
public class Texture2DContent : TextureContent
```

Inheritance

object ← ContentItem ← TextureContent ← Texture2DContent

Inherited Members

TextureContent.Faces , TextureContent.ConvertBitmapType(Type) , TextureContent.GenerateMipmaps(bool) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Texture2DContent()

Initializes a new instance of the Texture2DContent class.

```
public Texture2DContent()
```

Properties

Mipmaps

Get or set the mipmap chain.

```
public MipmapChain Mipmaps { get; set; }
```

Property Value

MipmapChain

Methods

Validate(GraphicsProfile?)

Verifies that all contents of this texture are present, correct and match the capabilities of the device.

```
public override void Validate(GraphicsProfile? targetProf)
```

Parameters

`targetProf` GraphicsProfile?

Class Texture3DContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties for maintaining a 3D texture

```
public class Texture3DContent : TextureContent
```

Inheritance

object ← ContentItem ← TextureContent ← Texture3DContent

Inherited Members

TextureContent.Faces , TextureContent.ConvertBitmapType(Type) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Texture3DContent()

Initializes a new instance of the Texture3DContent class.

```
public Texture3DContent()
```

Methods

GenerateMipmaps(bool)

Generates a full set of mipmaps for the texture.

```
public override void GenerateMipmaps(bool overwriteExistingMipmaps)
```

Parameters

`overwriteExistingMipmaps` bool

true if the existing mipmap set is replaced with the new set; false otherwise.

Validate(GraphicsProfile?)

Verifies that all contents of this texture are present, correct and match the capabilities of the device.

```
public override void Validate(GraphicsProfile? targetProf)
```

Parameters

`targetProf` GraphicsProfile?

Class TextureContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class for all texture objects.

```
public abstract class TextureContent : ContentItem
```

Inheritance

object ← ContentItem ← TextureContent

Derived

Texture2DContent, Texture3DContent, TextureCubeContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TextureContent(MipmapChainCollection)

Initializes a new instance of TextureContent with the specified face collection.

```
protected TextureContent(MipmapChainCollection faces)
```

Parameters

`faces` MipmapChainCollection

Mipmap chain containing the face collection.

Properties

Faces

Collection of image faces that hold a single mipmap chain for a regular 2D texture, six chains for a cube map, or an arbitrary number for volume and array textures.

```
public MipmapChainCollection Faces { get; }
```

Property Value

MipmapChainCollection

Methods

ConvertBitmapType(Type)

Converts all bitmaps for this texture to a different format.

```
public void ConvertBitmapType(Type newBitmapType)
```

Parameters

`newBitmapType` Type

Type being converted to. The new type must be a subclass of BitmapContent, such as PixelBitmapContent or DxtBitmapContent.

GenerateMipmaps(bool)

Generates a full set of mipmaps for the texture.

```
public virtual void GenerateMipmaps(bool overwriteExistingMipmaps)
```

Parameters

`overwriteExistingMipmaps` bool

true if the existing mipmap set is replaced with the new set; false otherwise.

Validate(GraphicsProfile?)

Verifies that all contents of this texture are present, correct and match the capabilities of the device.

```
public abstract void Validate(GraphicsProfile? targetProfile)
```

Parameters

`targetProfile` GraphicsProfile?

The profile identifier that defines the capabilities of the device.

Class TextureCubeContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a base class for texture cube content.

```
public class TextureCubeContent : TextureContent
```

Inheritance

object ← ContentItem ← TextureContent ← TextureCubeContent

Inherited Members

TextureContent.Faces , TextureContent.ConvertBitmapType(Type) , TextureContent.GenerateMipmaps(bool) , ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TextureCubeContent()

Creates a new TextureCubeContent object.

```
public TextureCubeContent()
```

Methods

Validate(GraphicsProfile?)

Verifies that all contents of this texture are present, correct and match the capabilities of the device.

```
public override void Validate(GraphicsProfile? targetProf)
```

Parameters

`targetProf` GraphicsProfile?

Remarks

Validate is not implemented for this texture content type.

Class TextureProfile

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Base class for handling texture profiles.

```
public abstract class TextureProfile
```

Inheritance

object ← TextureProfile

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ConvertTexture(ContentProcessorContext, TextureContent, TextureProcessorOutputFormat, bool)

Performs conversion of the texture content to the correct format.

```
public void ConvertTexture(ContentProcessorContext context, TextureContent content,
TextureProcessorOutputFormat format, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

The processor context.

`content` TextureContent

The content to be compressed.

`format` TextureProcessorOutputFormat

The user requested format for compression.

`isSpriteFont` bool

If the texture has represents a sprite font, i.e. is greyscale and has sharp black/white contrast.

ForPlatform(TargetPlatform)

Find the profile for this target platform.

```
public static TextureProfile ForPlatform(TargetPlatform platform)
```

Parameters

`platform` TargetPlatform

The platform target for textures.

Returns

TextureProfile

PlatformCompressTexture(ContentProcessorContext, TextureContent, TextureProcessorOutputFormat, bool)

```
protected abstract void PlatformCompressTexture(ContentProcessorContext context,
TextureContent content, TextureProcessorOutputFormat format, bool isSpriteFont)
```

Parameters

`context` ContentProcessorContext

`content` TextureContent

`format` TextureProcessorOutputFormat

`isSpriteFont` bool

Requirements(ContentProcessorContext, TextureProcessorOutputFormat, out bool, out bool)

Determines if the texture format will require power-of-two dimensions and/or equal width and height.

```
public abstract void Requirements(ContentProcessorContext context,
TextureProcessorOutputFormat format, out bool requiresPowerOfTwo, out
bool requiresSquare)
```

Parameters

`context` ContentProcessorContext

The processor context.

`format` TextureProcessorOutputFormat

The desired texture format.

`requiresPowerOfTwo` bool

True if the texture format requires power-of-two dimensions.

`requiresSquare` bool

True if the texture format requires equal width and height.

Supports(TargetPlatform)

Returns true if this profile supports texture processing for this platform.

```
public abstract bool Supports(TargetPlatform platform)
```

Parameters

`platform` TargetPlatform

Returns

bool

Class TextureReferenceDictionary

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a collection of named references to texture files.

```
public sealed class TextureReferenceDictionary :
NamedValueDictionary<ExternalReference<TextureContent>>, IDictionary<string,
ExternalReference<TextureContent>>, ICollection<KeyValuePair<string,
ExternalReference<TextureContent>>>, IEnumerable<KeyValuePair<string,
ExternalReference<TextureContent>>>, IEnumerable
```

Inheritance

object ← NamedValueDictionary<ExternalReference<TextureContent>> ← TextureReferenceDictionary

Implements

IDictionary<string, ExternalReference<TextureContent>>,
ICollection<KeyValuePair<string, ExternalReference<TextureContent>>>,
IEnumerable<KeyValuePair<string, ExternalReference<TextureContent>>>, IEnumerable

Inherited Members

NamedValueDictionary<ExternalReference<TextureContent>>.Add(string,
ExternalReference<TextureContent>) ,
NamedValueDictionary<ExternalReference<TextureContent>>.ContainsKey(string) ,
NamedValueDictionary<ExternalReference<TextureContent>>.Keys ,
NamedValueDictionary<ExternalReference<TextureContent>>.Remove(string) ,
NamedValueDictionary<ExternalReference<TextureContent>>.TryGetValue(string, out
ExternalReference<TextureContent>) ,
NamedValueDictionary<ExternalReference<TextureContent>>.Values ,
NamedValueDictionary<ExternalReference<TextureContent>>.this[string] ,
NamedValueDictionary<ExternalReference<TextureContent>>.Clear() ,
NamedValueDictionary<ExternalReference<TextureContent>>.Count ,
NamedValueDictionary<ExternalReference<TextureContent>>.GetEnumerator() ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

TextureReferenceDictionary()

Initializes a new instance of TextureReferenceDictionary.

```
public TextureReferenceDictionary()
```

Class VertexChannel

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining a vertex channel. A vertex channel is a list of arbitrary data with one value for each vertex. Channels are stored inside a GeometryContent and identified by name.

```
public abstract class VertexChannel : IList, ICollection, IEnumerable
```

Inheritance

object ← VertexChannel

Implements

IList, ICollection, IEnumerable

Derived

VertexChannel<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of elements in the vertex channel

```
public int Count { get; }
```

Property Value

int

ElementType

Gets the type of data contained in this channel.

```
public abstract Type ElementType { get; }
```

Property Value

Type

this[int]

Gets or sets the element at the specified index.

```
public object this[int index] { get; set; }
```

Parameters

`index` int

Property Value

object

Name

Gets the name of the vertex channel.

```
public string Name { get; }
```

Property Value

string

Methods

Contains(object)

Determines whether the specified element is in the channel.

```
public bool Contains(object value)
```

Parameters

`value` object

Element being searched for.

Returns

bool

true if the element is present; false otherwise.

CopyTo(Array, int)

Copies the elements of the channel to an array, starting at the specified index.

```
public void CopyTo(Array array, int index)
```

Parameters

`array` Array

Array that will receive the copied channel elements.

`index` int

Starting index for copy operation.

GetEnumerator()

Gets an enumerator interface for reading channel content.

```
public IEnumerator GetEnumerator()
```

Returns

IEnumerator

Enumeration of the channel content.

IndexOf(object)

Gets the index of the specified item.

```
public int IndexOf(object value)
```

Parameters

`value` object

Item whose index is to be retrieved.

Returns

int

Index of specified item.

ReadConvertedContent<TargetType>()

Reads channel content and automatically converts it to the specified vector format.

```
public abstract IEnumerable<TargetType> ReadConvertedContent<TargetType>()
```

Returns

IEnumerable<TargetType>

The converted data.

Type Parameters

`TargetType`

Target vector format of the converted data.

Class VertexChannelCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for managing a list of vertex data channels.

```
public sealed class VertexChannelCollection : IList<VertexChannel>,
ICollection<VertexChannel>, IEnumerable<VertexChannel>, IEnumerable
```

Inheritance

object ← VertexChannelCollection

Implements

IList<VertexChannel>, ICollection<VertexChannel>, IEnumerable<VertexChannel>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of vertex channels in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Gets or sets the vertex channel at the specified index position.

```
public VertexChannel this[int index] { get; set; }
```

Parameters

`index` int

Property Value

VertexChannel

this[string]

Gets or sets the vertex channel with the specified name.

```
public VertexChannel this[string name] { get; set; }
```

Parameters

`name` string

Property Value

VertexChannel

Methods

Add(string, Type, IEnumerable)

Adds a new vertex channel to the end of the collection.

```
public VertexChannel Add(string name, Type elementType, IEnumerable channelData)
```

Parameters

`name` string

Name of the new channel.

`elementType` Type

Type of data to be contained in the new channel.

`channelData` IEnumerable

Initial data for the new channel. If null, the channel is filled with the default value for that type.

Returns

VertexChannel

The newly added vertex channel.

Add<ElementType>(string, IEnumerable<ElementType>)

Adds a new vertex channel to the end of the collection.

```
public VertexChannel<ElementType> Add<ElementType>(string name,
IEnumerable<ElementType> channelData)
```

Parameters

`name` string

Name of the new channel.

`channelData` IEnumerable<ElementType>

Initial data for the new channel. If null, the channel is filled with the default value for that type.

Returns

VertexChannel<ElementType>

The newly added vertex channel.

Type Parameters

`ElementType`

Type of the channel.

Clear()

Removes all vertex channels from the collection.

```
public void Clear()
```

Contains(VertexChannel)

Determines whether the collection contains the specified vertex channel.

```
public bool Contains(VertexChannel item)
```

Parameters

`item` VertexChannel

The channel being searched for.

Returns

bool

true if the channel was found; false otherwise.

Contains(string)

Determines whether the collection contains the specified vertex channel.

```
public bool Contains(string name)
```

Parameters

`name` string

Name of the channel being searched for.

Returns

bool

true if the channel was found; false otherwise.

ConvertChannelContent<TargetType>(int)

Converts the channel, at the specified index, to another vector format.

```
public VertexChannel<TargetType> ConvertChannelContent<TargetType>(int index)
```

Parameters

`index` int

Index of the channel to be converted.

Returns

VertexChannel<TargetType>

New channel in the specified format.

Type Parameters

`TargetType`

Type of the target format. Can be one of the following: Single, Vector2, Vector3, Vector4, IPackedVector

ConvertChannelContent<TargetType>(string)

Converts the channel, specified by name to another vector format.

```
public VertexChannel<TargetType> ConvertChannelContent<TargetType>(string name)
```

Parameters

`name` string

Name of the channel to be converted.

Returns

VertexChannel<TargetType>

New channel in the specified format.

Type Parameters

`TargetType`

Type of the target format. Can be one of the following: Single, Vector2, Vector3, Vector4, IPackedVector

GetEnumerator()

Gets an enumerator that iterates through the vertex channels of a collection.

```
public IEnumerator<VertexChannel> GetEnumerator()
```

Returns

IEnumerator<VertexChannel>

Enumerator for the collection.

Get<T>(int)

Gets the vertex channel with the specified index and content type.

```
public VertexChannel<T> Get<T>(int index)
```

Parameters

`index` int

Index of a vertex channel.

Returns

VertexChannel<T>

The vertex channel.

Type Parameters

`T`

Type of a vertex channel.

Get<T>(string)

Gets the vertex channel with the specified name and content type.

```
public VertexChannel<T> Get<T>(string name)
```

Parameters

`name` string

Name of a vertex channel.

Returns

VertexChannel<T>

The vertex channel.

Type Parameters

`T`

Type of the vertex channel.

IndexOf(VertexChannel)

Determines the index of the specified vertex channel.

```
public int IndexOf(VertexChannel item)
```

Parameters

`item` VertexChannel

Vertex channel being searched for.

Returns

int

Index of the vertex channel.

IndexOf(string)

Determines the index of a vertex channel with the specified name.

```
public int IndexOf(string name)
```

Parameters

`name` string

Name of the vertex channel being searched for.

Returns

int

Index of the vertex channel.

Insert(int, string, Type, IEnumerable)

Inserts a new vertex channel at the specified position.

```
public VertexChannel Insert(int index, string name, Type elementType,
IEnumerable channelData)
```

Parameters

`index` int

Index for channel insertion.

`name` string

Name of the new channel.

`elementType` Type

Type of the new channel.

`channelData` IEnumerable

Initial data for the new channel. If null, it is filled with the default value.

Returns

VertexChannel

The inserted vertex channel.

Insert<ElementType>(int, string, IEnumerable<ElementType>)

Inserts a new vertex channel at the specified position.

```
public VertexChannel<ElementType> Insert<ElementType>(int index, string name,
IEnumerable<ElementType> channelData)
```

Parameters

`index` int

Index for channel insertion.

`name` string

Name of the new channel.

`channelData` IEnumerable<ElementType>

The new channel.

Returns

VertexChannel<ElementType>

The inserted vertex channel.

Type Parameters

`ElementType`

Type of the new channel.

Remove(VertexChannel)

Removes the specified vertex channel from the collection.

```
public bool Remove(VertexChannel item)
```

Parameters

`item` VertexChannel

The vertex channel being removed.

Returns

bool

true if the channel was removed; false otherwise.

Remove(string)

Removes the specified vertex channel from the collection.

```
public bool Remove(string name)
```

Parameters

`name` string

Name of the vertex channel being removed.

Returns

bool

true if the channel was removed; false otherwise.

RemoveAt(int)

Removes the vertex channel at the specified index position.

```
public void RemoveAt(int index)
```

Parameters

`index` int

Index of the vertex channel being removed.

Class VertexChannelNames

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties for managing a collection of vertex channel names.

```
public static class VertexChannelNames
```

Inheritance

object ← VertexChannelNames

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Binormal(int)

Gets the name of a binormal vector channel with the specified index. This will typically contain Vector3 data.

```
public static string Binormal(int usageIndex)
```

Parameters

`usageIndex` int

Zero-based index of the vector channel being retrieved.

Returns

string

Name of the retrieved vector channel.

Color(int)

Gets the name of a color channel with the specified index. This will typically contain Vector3 data.

```
public static string Color(int usageIndex)
```

Parameters

`usageIndex` int

Zero-based index of the color channel being retrieved.

Returns

string

Name of the retrieved color channel.

DecodeBaseName(string)

Gets a channel base name stub from the encoded string format.

```
public static string DecodeBaseName(string encodedName)
```

Parameters

`encodedName` string

Encoded string to be decoded.

Returns

string

Extracted base name.

DecodeUsageIndex(string)

Gets a channel usage index from the encoded format.

```
public static int DecodeUsageIndex(string encodedName)
```

Parameters

`encodedName` string

Encoded name to be decoded.

Returns

int

Resulting channel usage index.

EncodeName(VertexElementUsage, int)

Combines a vertex declaration usage and usage index into a string name.

```
public static string EncodeName(VertexElementUsage vertexElementUsage,
int usageIndex)
```

Parameters

`vertexElementUsage` VertexElementUsage

A vertex declaration.

`usageIndex` int

An index for the vertex declaration.

Returns

string

Resulting encoded name.

EncodeName(string, int)

Combines a channel name stub and usage index into a string name.

```
public static string EncodeName(string baseName, int usageIndex)
```

Parameters

`baseName` string

A channel base name stub.

`usageIndex` int

A channel usage index.

Returns

string

Resulting encoded name.

Normal()

Gets the name of the primary normal channel. This will typically contain Vector3 data.

```
public static string Normal()
```

Returns

string

Primary normal channel name.

Normal(int)

Gets the name of a normal channel with the specified index. This will typically contain Vector3 data.

```
public static string Normal(int usageIndex)
```

Parameters

`usageIndex` int

Zero-based index of the normal channel being retrieved.

Returns

string

Normal channel at the specified index.

Tangent(int)

Gets the name of a tangent vector channel with the specified index. This will typically contain Vector3 data.

```
public static string Tangent(int usageIndex)
```

Parameters

`usageIndex` int

Zero-based index of the tangent vector channel being retrieved.

Returns

string

Name of the retrieved tangent vector channel.

TextureCoordinate(int)

Gets the name of a texture coordinate channel with the specified index. This will typically contain Vector3 data.

```
public static string TextureCoordinate(int usageIndex)
```

Parameters

`usageIndex` int

Zero-based index of the texture coordinate channel being retrieved.

Returns

string

Name of the retrieved texture coordinate channel.

TryDecodeUsage(string, out VertexElementUsage)

Gets a vertex declaration usage enumeration from the encoded string format.

```
public static bool TryDecodeUsage(string encodedName, out VertexElementUsage usage)
```

Parameters

`encodedName` string

Encoded name of a vertex declaration.

`usage` VertexElementUsage

Value of the declaration usage for the vertex declaration.

Returns

bool

true if the encoded name maps to a VertexElementUsage enumeration value; false otherwise.

Weights()

Gets the name of the primary animation weights channel. This will typically contain data on the bone weights for a vertex channel. For more information, see BoneWeightCollection.

```
public static string Weights()
```

Returns

string

Name of the primary animation weights channel.

Weights(int)

Gets the name of an animation weights channel at the specified index. This will typically contain data on the bone weights for a vertex channel. For more information, see BoneWeightCollection.

```
public static string Weights(int usageIndex)
```

Parameters

`usageIndex` int

Index of the animation weight channel to be retrieved.

Returns

string

Name of the retrieved animation weights channel.

Class VertexChannel<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining a vertex channel. This is a generic implementation of VertexChannel and, therefore, can handle strongly typed content data.

```
public sealed class VertexChannel<T> : VertexChannel, IList, ICollection, IList<T>,
ICollection<T>, IEnumerable<T>, IEnumerable
```

Type Parameters

T

Provides methods and properties for maintaining a vertex channel. This is a generic implementation of VertexChannel and, therefore, can handle strongly typed content data.

Inheritance

object ← VertexChannel ← VertexChannel<T>

Implements

IList, ICollection, IList<T>, ICollection<T>, IEnumerable<T>, IEnumerable

Inherited Members

VertexChannel.Count , VertexChannel.Name , VertexChannel.Contains(object) , VertexChannel.CopyTo(Array, int) , VertexChannel.IndexOf(object) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ElementType

Gets the type of data contained in this channel.

```
public override Type ElementType { get; }
```

Property Value

Type

this[int]

Gets or sets the element at the specified index.

```
public T this[int index] { get; set; }
```

Parameters

`index` int

Property Value

T

Methods

Contains(T)

Determines whether the specified element is in the channel.

```
public bool Contains(T item)
```

Parameters

`item` T

Element being searched for.

Returns

bool

true if the element is present; false otherwise.

CopyTo(T[], int)

Copies the elements of the channel to an array, starting at the specified index.

```
public void CopyTo(T[] array, int arrayIndex)
```

Parameters

`array` T[]

Array that will receive the copied channel elements.

`arrayIndex` int

Starting index for copy operation.

GetEnumerator()

Gets an enumerator interface for reading channel content.

```
public IEnumerator<T> GetEnumerator()
```

Returns

IEnumerator<T>

Enumeration of the channel content.

IndexOf(T)

Gets the index of the specified item.

```
public int IndexOf(T item)
```

Parameters

`item` T

Item whose index is to be retrieved.

Returns

int

Index of specified item.

ReadConvertedContent<TargetType>()

Reads channel content and automatically converts it to the specified vector format.

```
public override IEnumerable<TargetType> ReadConvertedContent<TargetType>()
```

Returns

IEnumerable<TargetType>

The converted channel data.

Type Parameters

`TargetType`

Target vector format for the converted channel data.

Class VertexContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Graphics

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining the vertex data of a GeometryContent.

```
public sealed class VertexContent
```

Inheritance

object ← VertexContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This class combines a collection of arbitrarily named data channels with a list of position indices that reference the Positions collection of the parent MeshContent.

Properties

Channels

Gets the list of named vertex data channels in the VertexContent.

```
public VertexChannelCollection Channels { get; }
```

Property Value

VertexChannelCollection

Collection of vertex data channels.

PositionIndices

Gets the list of position indices.

```
public VertexChannel<int> PositionIndices { get; }
```

Property Value

VertexChannel<int>

Position of the position index being retrieved.

Remarks

This list adds a level of indirection between the actual triangle indices and the Positions member of the parent. This indirection preserves the topological vertex identity in cases where a single vertex position is used by triangles that straddle a discontinuity in some other data channel. For example, the following code gets the position of the first vertex of the first triangle in a GeometryContent object: parent.Positions[Vertices.PositionIndices[Indices[0]]]

Positions

Gets position data from the parent mesh object.

```
public IndirectPositionCollection Positions { get; }
```

Property Value

IndirectPositionCollection

Collection of vertex positions for the mesh.

Remarks

The collection returned from this call provides a virtualized view of the vertex positions for this batch. The collection uses the contents of the PositionIndices property to index into the parent Positions. This collection is read-only. If you need to modify any contained values, edit the PositionIndices or Positions members directly.

VertexCount

Number of vertices for the content.

```
public int VertexCount { get; }
```

Property Value

int

Number of vertices.

Methods

Add(int)

Appends a new vertex index to the end of the PositionIndices collection. Other vertex channels will automatically be extended and the new indices populated with default values.

```
public int Add(int positionIndex)
```

Parameters

`positionIndex` int

Index into the MeshContent.Positions member of the parent.

Returns

int

Index of the new entry. This can be added to the Indices member of the parent.

AddRange(IEnumerable<int>)

Appends multiple vertex indices to the end of the PositionIndices collection. Other vertex channels will automatically be extended and the new indices populated with default values.

```
public void AddRange(IEnumerable<int> positionIndexCollection)
```

Parameters

`positionIndexCollection` IEnumerable<int>

Index into the Positions member of the parent.

CreateVertexBuffer()

Converts design-time vertex position and channel data into a vertex buffer format that a graphics device can recognize.

```
public VertexBufferContent CreateVertexBuffer()
```

Returns

VertexBufferContent

A packed vertex buffer.

Exceptions

InvalidContentException

One or more of the vertex channel types are invalid or an unrecognized name was passed to VertexElementUsage.

Insert(int, int)

Inserts a new vertex index to the PositionIndices collection. Other vertex channels will automatically be extended and the new indices populated with default values.

```
public void Insert(int index, int positionIndex)
```

Parameters

`index` int

Vertex index to be inserted.

`positionIndex` int

Position of new vertex index in the collection.

InsertRange(int, IEnumerable<int>)

Inserts multiple vertex indices to the PositionIndices collection. Other vertex channels will automatically be extended and the new indices populated with default values.

```
public void InsertRange(int index, IEnumerable<int> positionIndexCollection)
```

Parameters

`index` int

Vertex index to be inserted.

`positionIndexCollection` IEnumerable<int>

Position of the first element of the inserted range in the collection.

RemoveAt(int)

Removes a vertex index from the specified location in both PositionIndices and VertexChannel<T>.

```
public void RemoveAt(int index)
```

Parameters

`index` int

Index of the vertex to be removed.

RemoveRange(int, int)

Removes a range of vertex indices from the specified location in both PositionIndices and VertexChannel<T>.

```
public void RemoveRange(int index, int count)
```

Parameters

`index` int

Index of the first vertex index to be removed.

`count` int

Number of indices to remove.

Namespace Microsoft.Xna.Framework.Content.Pipeline.Processors

Classes

CompiledEffectContent

Represents a compiled Effect.

EffectProcessor

Processes a string representation to a platform-specific compiled effect.

FontDescriptionProcessor

FontTextureProcessor

Class to provide methods to handle processing of font textures.

LocalizedFontProcessor

Custom processor extends the SpriteFont build process to scan over the resource strings used by the game, automatically adding whatever characters it finds in them to the font. This makes sure the game will always have all the characters it needs, no matter what languages it is localized into, while still producing an efficient font that does not waste space on unnecessary characters. This is especially useful for languages such as Japanese and Korean, which have potentially thousands of different characters, although games typically only use a small fraction of these. Building only the characters we need is far more efficient than if we tried to include the entire CJK character region.

MaterialProcessor

Provides methods and properties for maintaining a collection of named texture references.

ModelBoneContent

Provides properties and methods for managing model bone content.

ModelBoneContentCollection

A collection of ModelBoneContent instances.

ModelContent

Provides properties and methods for managing model content.

ModelMeshContent

Provides methods and properties for loading model mesh data.

ModelMeshContentCollection

A collection of ModelMeshPartContent objects. This class cannot be inherited.

ModelMeshPartContent

Stores design-time data for a ModelMeshPart asset.

ModelMeshPartContentCollection

ModelProcessor

Processes a game asset mesh to a model content that is optimal for runtime.

PassThroughProcessor

As the name implies, this processor simply passes data through as-is.

SongContent

Represents a processed Song object.

SongProcessor

A custom song processor that processes an intermediate AudioContent type. This type encapsulates the source audio content, producing a Song type that can be used in the game.

SoundEffectContent

Represents a processed sound effect.

SoundEffectProcessor

A sound effect processor that processes an intermediate AudioContent type. This type encapsulates the source audio content, producing a SoundEffect type that can be used in the game.

TextureProcessor

Class to provide methods and properties for processing textures.

VertexBufferContent

Provides methods and properties for managing a design-time vertex buffer that holds packed vertex data.

VertexDeclarationContent

Provides methods and properties for maintaining the vertex declaration data of a VertexContent.

VideoProcessor

Provides methods and properties for maintaining video content.

Enums

EffectProcessorDebugMode

Specifies how debugging of effects is to be supported in PIX.

MaterialProcessorDefaultEffect

Specifies the default effect type.

TextureProcessorOutputFormat

Specifies the target output (of type SurfaceFormat) of the texture processor. Used by TextureProcessor.TextureFormat.

Class CompiledEffectContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents a compiled Effect.

```
public class CompiledEffectContent : ContentItem
```

Inheritance

object ← ContentItem ← CompiledEffectContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CompiledEffectContent(byte[])

Creates a new instance of the CompiledEffectContent class

```
public CompiledEffectContent(byte[] effectCode)
```

Parameters

`effectCode` byte[]

The compiled effect code.

Methods

GetEffectCode()

Retrieves the compiled byte code for this shader.

```
public byte[] GetEffectCode()
```

Returns

byte[]

The compiled bytecode.

Class EffectProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Processes a string representation to a platform-specific compiled effect.

```
[ContentProcessor(DisplayName = "Effect - MonoGame")]
public class EffectProcessor : ContentProcessor<EffectContent,
CompiledEffectContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<EffectContent, CompiledEffectContent> ← EffectProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<EffectContent, CompiledEffectContent>.Process(EffectContent, ContentProcessorContext) , ContentProcessor<EffectContent, CompiledEffectContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

EffectProcessor()

Processes a string representation to a platform-specific compiled effect.

```
public EffectProcessor()
```

Properties

DebugMode

The debug mode for compiling effects.

```
public virtual EffectProcessorDebugMode DebugMode { get; set; }
```

Property Value

EffectProcessorDebugMode

The debug mode to use when compiling effects.

Defines

Define assignments for the effect.

```
public virtual string Defines { get; set; }
```

Property Value

string

A list of define assignments delimited by semicolons.

Methods

Process(EffectContent, ContentProcessorContext)

Processes the string representation of the specified effect into a platform-specific binary format using the specified context.

```
public override CompiledEffectContent Process(EffectContent input,
ContentProcessorContext context)
```

Parameters

`input` EffectContent

The effect string to be processed.

`context` ContentProcessorContext

Context for the specified processor.

Returns

CompiledEffectContent

A platform-specific compiled binary effect.

Remarks

If you get an error during processing, compilation stops immediately. The effect processor displays an error message. Once you fix the current error, it is possible you may get more errors on subsequent compilation attempts.

Enum EffectProcessorDebugMode

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Specifies how debugging of effects is to be supported in PIX.

```
public enum EffectProcessorDebugMode
```

Fields

`Auto = 0`

Enables effect debugging when built with Debug profile.

`Debug = 1`

Enables effect debugging for all profiles. Will produce unoptimized shaders.

`Optimize = 2`

Disables debugging for all profiles, produce optimized shaders.

Class FontDescriptionProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[ContentProcessor(DisplayName = "Sprite Font Description - MonoGame")]
public class FontDescriptionProcessor : ContentProcessor<FontDescription,
SpriteFontContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<FontDescription, SpriteFontContent> ← FontDescriptionProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<FontDescription, SpriteFontContent>.Process(FontDescription, ContentProcessorContext) , ContentProcessor<FontDescription, SpriteFontContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FontDescriptionProcessor()

```
public FontDescriptionProcessor()
```

Properties

PremultiplyAlpha

```
public virtual bool PremultiplyAlpha { get; set; }
```

Property Value

bool

TextureFormat

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

Methods

Process(FontDescription, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public override SpriteFontContent Process(FontDescription input,
ContentProcessorContext context)
```

Parameters

`input` FontDescription

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

SpriteFontContent

A typed object representing the processed input.

Class FontTextureProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide methods to handle processing of font textures.

```
[ContentProcessor(DisplayName = "Font Texture - MonoGame")]
public class FontTextureProcessor : ContentProcessor<Texture2DContent,
SpriteFontContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<Texture2DContent, SpriteFontContent> ← FontTextureProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<Texture2DContent, SpriteFontContent>.Process(Texture2DContent, ContentProcessorContext) , ContentProcessor<Texture2DContent, SpriteFontContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FontTextureProcessor()

Creates a new FontTextureProcessor.

```
public FontTextureProcessor()
```

Properties

FirstCharacter

Gets or sets the first character of the font.

```
public virtual char FirstCharacter { get; set; }
```

Property Value

char

PremultiplyAlpha

Gets or sets the flag that indicates if the alpha channel should be premultiplied.

```
public virtual bool PremultiplyAlpha { get; set; }
```

Property Value

bool

TextureFormat

Specifies the target output (of type SurfaceFormat) of the texture processor. Used by TextureProcessor.TextureFormat.

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

Methods

GetCharacterForIndex(int)

Gets the character for the specified index relative to the first character.

```
protected virtual char GetCharacterForIndex(int index)
```

Parameters

`index` int

Character index.

Returns

char

Character at index.

Process(Texture2DContent, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public override SpriteFontContent Process(Texture2DContent input,
ContentProcessorContext context)
```

Parameters

`input` Texture2DContent

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

SpriteFontContent

A typed object representing the processed input.

Class LocalizedFontProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Custom processor extends the SpriteFont build process to scan over the resource strings used by the game, automatically adding whatever characters it finds in them to the font. This makes sure the game will always have all the characters it needs, no matter what languages it is localized into, while still producing an efficient font that does not waste space on unnecessary characters. This is especially useful for languages such as Japanese and Korean, which have potentially thousands of different characters, although games typically only use a small fraction of these. Building only the characters we need is far more efficient than if we tried to include the entire CJK character region.

```
[Obsolete("Please use FontImporter for importing and FontDescriptionProcessor for
processing instead.")]
[ContentProcessor]
public class LocalizedFontProcessor : ContentProcessor<LocalizedFontDescription,
SpriteFontContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<LocalizedFontDescription, SpriteFontContent> ← LocalizedFontProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<LocalizedFontDescription, SpriteFontContent>.Process(LocalizedFontDescription, ContentProcessorContext) , ContentProcessor<LocalizedFontDescription, SpriteFontContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

LocalizedFontProcessor()

Initializes a new instance of the LocalizedFontProcessor class. Sets the PremultiplyAlpha property to true and the TextureFormat property to Compressed.

```
public LocalizedFontProcessor()
```

Properties

PremultiplyAlpha

Gets or Sets the premultiply alpha flag.

```
public virtual bool PremultiplyAlpha { get; set; }
```

Property Value

bool

TextureFormat

Gets or Sets the target texture output format.

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

Methods

Process(LocalizedFontDescription, ContentProcessorContext)

Converts a font description into SpriteFont format.

```
public override SpriteFontContent Process(LocalizedFontDescription input,
ContentProcessorContext context)
```

Parameters

`input` LocalizedFontDescription

`context` ContentProcessorContext

Returns

SpriteFontContent

Class MaterialProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining a collection of named texture references.

```
[ContentProcessor(DisplayName = "Material - MonoGame")]
public class MaterialProcessor : ContentProcessor<MaterialContent,
MaterialContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<MaterialContent, MaterialContent> ← MaterialProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<MaterialContent, MaterialContent>.Process(MaterialContent, ContentProcessorContext) , ContentProcessor<MaterialContent, MaterialContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

In addition to texture references, opaque data values are stored in the OpaqueData property of the base class.

Constructors

MaterialProcessor()

Initializes a new instance of the MaterialProcessor class.

```
public MaterialProcessor()
```

Properties

ColorKeyColor

Gets or sets the color value to replace with transparent black.

```
public virtual Color ColorKeyColor { get; set; }
```

Property Value

Color

Color value of the material to replace with transparent black.

ColorKeyEnabled

Specifies whether color keying of a texture is enabled.

```
public virtual bool ColorKeyEnabled { get; set; }
```

Property Value

bool

true if color keying is enabled; false otherwise.

DefaultEffect

The default effect type for this instance of MaterialProcessor.

```
public virtual MaterialProcessorDefaultEffect DefaultEffect { get; set; }
```

Property Value

MaterialProcessorDefaultEffect

The default effect type.

Remarks

When MaterialProcessor is instantiated, DefaultEffect is set to default to BasicEffect Class.

GenerateMipmaps

Specifies if a full chain of mipmaps are generated from the source material. Existing mipmaps of the material are not replaced.

```
public virtual bool GenerateMipmaps { get; set; }
```

Property Value

bool

true if mipmap generation is enabled; false otherwise.

PremultiplyTextureAlpha

Specifies whether alpha premultiply of textures is enabled.

```
public virtual bool PremultiplyTextureAlpha { get; set; }
```

Property Value

bool

true if alpha premultiply is enabled; false otherwise.

ResizeTexturesToPowerOfTwo

Specifies whether resizing of a material is enabled. Typically used to maximize compatability with a graphics card because many graphics cards do not support a material size that is not a power of two. If ResizeTexturesToPowerOfTwo is enabled, the material is resized to the next largest power of two.

```
public virtual bool ResizeTexturesToPowerOfTwo { get; set; }
```

Property Value

bool

true if resizing is enabled; false otherwise.

TextureFormat

Specifies the texture format of output materials. Materials can either be left unchanged from the source asset, converted to a corresponding Color, or compressed using the appropriate DxtCompressed format.

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

The texture format of the output.

Methods

BuildEffect(ExternalReference<EffectContent>, ContentProcessorContext)

Builds effect content.

```
protected virtual ExternalReference<CompiledEffectContent>
BuildEffect(ExternalReference<EffectContent> effect, ContentProcessorContext
context)
```

Parameters

`effect` ExternalReference<EffectContent>

An external reference to the effect content.

`context` ContentProcessorContext

Context for the specified processor.

Returns

ExternalReference<CompiledEffectContent>

A platform-specific compiled binary effect.

Remarks

If the input to process is of type EffectMaterialContent, this function will be called to request that the EffectContent be built. The EffectProcessor is used to process the EffectContent. Subclasses of MaterialProcessor can override this function to modify the parameters used to build EffectContent. For example, a different version of this function could request a different processor for the EffectContent.

BuildTexture(string, ExternalReference<TextureContent>, ContentProcessorContext)

Builds texture content.

```
protected virtual ExternalReference<TextureContent> BuildTexture(string textureName,
ExternalReference<TextureContent> texture, ContentProcessorContext context)
```

Parameters

`textureName` string

The name of the texture. This should correspond to the key used to store the texture in Textures.

`texture` ExternalReference<TextureContent>

The asset to build. This should be a member of Textures.

`context` ContentProcessorContext

Context for the specified processor.

Returns

ExternalReference<TextureContent>

The built texture content.

Remarks

textureName can be used to determine which processor to use. For example, if a texture is being used as a normal map, the user may not want to use the ModelTextureProcessor on it, which compresses textures.

CreateDefaultMaterial(MaterialProcessorDefaultEffect)

Helper method which returns the material for a default effect.

```
public static MaterialContent CreateDefaultMaterial(MaterialProcessorDefaultEffect
effect)
```

Parameters

`effect` MaterialProcessorDefaultEffect

Returns

MaterialContent

A material.

GetDefaultEffect(MaterialContent)

Helper method which returns the default effect for a material.

```
public static MaterialProcessorDefaultEffect GetDefaultEffect(MaterialContent
content)
```

Parameters

`content` MaterialContent

Returns

MaterialProcessorDefaultEffect

The default effect.

Process(MaterialContent, ContentProcessorContext)

Builds the texture and effect content for the material.

```
public override MaterialContent Process(MaterialContent input,
ContentProcessorContext context)
```

Parameters

`input` MaterialContent

The material content to build.

`context` ContentProcessorContext

Context for the specified processor.

Returns

MaterialContent

The built material.

Remarks

If the MaterialContent is of type EffectMaterialContent, a build is requested for Effect, and validation will be performed on the OpaqueData to ensure that all parameters are valid input to SetValue or SetValueTranspose. If the MaterialContent is a BasicMaterialContent, no validation will be performed on OpaqueData. Process requests builds for all textures in Textures, unless the MaterialContent is of type BasicMaterialContent, in which case a build will only be requested for DiffuseColor. The textures in Textures will be ignored.

Enum MaterialProcessorDefaultEffect

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Specifies the default effect type.

```
public enum MaterialProcessorDefaultEffect
```

Fields

`AlphaTestEffect = 4`

An AlphaTestEffect Class effect.

`BasicEffect = 0`

A BasicEffect Class effect.

`DualTextureEffect = 3`

A DualTextureEffect Class effect.

`EnvironmentMapEffect = 2`

An EnvironmentMapEffect Class effect.

`SkinnedEffect = 1`

A SkinnedEffect Class effect.

Class ModelBoneContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for managing model bone content.

```
public sealed class ModelBoneContent
```

Inheritance

object ← ModelBoneContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Children

Stores the children of this bone content.

```
public ModelBoneContentCollection Children { get; }
```

Property Value

ModelBoneContentCollection

Index

Returns the index of this bone content.

```
public int Index { get; }
```

Property Value

int

Name

Returns the name of this bone content.

```
public string Name { get; }
```

Property Value

string

Parent

Returns teh parent of this bone content.

```
public ModelBoneContent Parent { get; }
```

Property Value

ModelBoneContent

Transform

Returns or sets the transform matrix of this bone content.

```
public Matrix Transform { get; set; }
```

Property Value

Matrix

Class ModelBoneContentCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

A collection of ModelBoneContent instances.

```
public sealed class ModelBoneContentCollection :
ReadOnlyCollection<ModelBoneContent>, IList<ModelBoneContent>,
ICollection<ModelBoneContent>, IReadOnlyList<ModelBoneContent>,
IReadOnlyCollection<ModelBoneContent>, IEnumerable<ModelBoneContent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelBoneContent> ← ModelBoneContentCollection

Implements

IList<ModelBoneContent>, ICollection<ModelBoneContent>, IReadOnlyList<ModelBoneContent>, IReadOnlyCollection<ModelBoneContent>, IEnumerable<ModelBoneContent>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelBoneContent>.Contains(ModelBoneContent) ,
ReadOnlyCollection<ModelBoneContent>.CopyTo(ModelBoneContent[], int) ,
ReadOnlyCollection<ModelBoneContent>.GetEnumerator() ,
ReadOnlyCollection<ModelBoneContent>.IndexOf(ModelBoneContent) ,
ReadOnlyCollection<ModelBoneContent>.Count ,
ReadOnlyCollection<ModelBoneContent>.Empty ,
ReadOnlyCollection<ModelBoneContent>.this[int] , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Class ModelContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides properties and methods for managing model content.

```
public sealed class ModelContent
```

Inheritance

object ← ModelContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Bones

Returns the bone content collection.

```
public ModelBoneContentCollection Bones { get; }
```

Property Value

ModelBoneContentCollection

Meshes

Returns the bone mesh collection.

```
public ModelMeshContentCollection Meshes { get; }
```

Property Value

ModelMeshContentCollection

Root

Returns the root bone.

```
public ModelBoneContent Root { get; }
```

Property Value

ModelBoneContent

Tag

Gets or sets an object that can be used to tag this model content.

```
public object Tag { get; set; }
```

Property Value

object

Class ModelMeshContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for loading model mesh data.

```
public sealed class ModelMeshContent
```

Inheritance

object ← ModelMeshContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

BoundingSphere

Gets the bounding sphere of the mesh.

```
public BoundingSphere BoundingSphere { get; }
```

Property Value

BoundingSphere

MeshParts

Gets the collection of mesh parts contained in this mesh.

```
public ModelMeshPartContentCollection MeshParts { get; }
```

Property Value

ModelMeshPartContentCollection

Name

Gets the name of the mesh.

```
public string Name { get; }
```

Property Value

string

ParentBone

Gets the parent bone model.

```
public ModelBoneContent ParentBone { get; }
```

Property Value

ModelBoneContent

SourceMesh

Gets the source mesh.

```
public MeshContent SourceMesh { get; }
```

Property Value

MeshContent

Tag

Gets or sets the tag associated with the mesh.

```
public object Tag { get; set; }
```

Property Value

object

Class ModelMeshContentCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

A collection of ModelMeshPartContent objects. This class cannot be inherited.

```
public sealed class ModelMeshContentCollection :
ReadOnlyCollection<ModelMeshContent>, IList<ModelMeshContent>,
ICollection<ModelMeshContent>, IReadOnlyList<ModelMeshContent>,
IReadOnlyCollection<ModelMeshContent>, IEnumerable<ModelMeshContent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelMeshContent> ← ModelMeshContentCollection

Implements

IList<ModelMeshContent>, ICollection<ModelMeshContent>,
IReadOnlyList<ModelMeshContent>, IReadOnlyCollection<ModelMeshContent>,
IEnumerable<ModelMeshContent>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelMeshContent>.Contains(ModelMeshContent) ,
ReadOnlyCollection<ModelMeshContent>.CopyTo(ModelMeshContent[], int) ,
ReadOnlyCollection<ModelMeshContent>.GetEnumerator() ,
ReadOnlyCollection<ModelMeshContent>.IndexOf(ModelMeshContent) ,
ReadOnlyCollection<ModelMeshContent>.Count ,
ReadOnlyCollection<ModelMeshContent>.Empty ,
ReadOnlyCollection<ModelMeshContent>.this[int] , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Class ModelMeshPartContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Stores design-time data for a ModelMeshPart asset.

```
public sealed class ModelMeshPartContent
```

Inheritance

object ← ModelMeshPartContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

IndexBuffer

Gets the collection of indices for this mesh part.

```
public IndexCollection IndexBuffer { get; }
```

Property Value

IndexCollection

Material

Gets the material of this mesh part.

```
public MaterialContent Material { get; set; }
```

Property Value

MaterialContent

NumVertices

Gets the number of vertices used in this mesh part.

```
public int NumVertices { get; }
```

Property Value

int

PrimitiveCount

Gets the number of primitives to render for this mesh part.

```
public int PrimitiveCount { get; }
```

Property Value

int

StartIndex

Gets the location in the index buffer at which to start reading vertices.

```
public int StartIndex { get; }
```

Property Value

int

Tag

Gets a user-defined tag object.

```
public object Tag { get; set; }
```

Property Value

object

VertexBuffer

Gets the collection of vertices for this mesh part.

```
public VertexBufferContent VertexBuffer { get; }
```

Property Value

VertexBufferContent

VertexOffset

Gets the offset from the start of the index buffer to the first vertex index.

```
public int VertexOffset { get; }
```

Property Value

int

Class ModelMeshPartContentCollection

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public sealed class ModelMeshPartContentCollection :
ReadOnlyCollection<ModelMeshPartContent>, IList<ModelMeshPartContent>,
ICollection<ModelMeshPartContent>, IReadOnlyList<ModelMeshPartContent>,
IReadOnlyCollection<ModelMeshPartContent>, IEnumerable<ModelMeshPartContent>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelMeshPartContent> ← ModelMeshPartContentCollection

Implements

IList<ModelMeshPartContent>, ICollection<ModelMeshPartContent>,
IReadOnlyList<ModelMeshPartContent>, IReadOnlyCollection<ModelMeshPartContent>,
IEnumerable<ModelMeshPartContent>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelMeshPartContent>.Contains(ModelMeshPartContent) ,
ReadOnlyCollection<ModelMeshPartContent>.CopyTo(ModelMeshPartContent[], int) ,
ReadOnlyCollection<ModelMeshPartContent>.GetEnumerator() ,
ReadOnlyCollection<ModelMeshPartContent>.IndexOf(ModelMeshPartContent) ,
ReadOnlyCollection<ModelMeshPartContent>.Count ,
ReadOnlyCollection<ModelMeshPartContent>.Empty ,
ReadOnlyCollection<ModelMeshPartContent>.this[int] , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Class ModelProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Processes a game asset mesh to a model content that is optimal for runtime.

```
[ContentProcessor(DisplayName = "Model - MonoGame")]
public class ModelProcessor : ContentProcessor<NodeContent,
ModelContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<NodeContent, ModelContent> ← ModelProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<NodeContent, ModelContent>.Process(NodeContent, ContentProcessorContext) , ContentProcessor<NodeContent, ModelContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModelProcessor()

Initializes a new instance of **ModelProcessor**.

```
public ModelProcessor()
```

Properties

ColorKeyColor

Gets or Sets the background color used for transparency

```
public virtual Color ColorKeyColor { get; set; }
```

Property Value

Color

ColorKeyEnabled

Gets or Sets a value that indicates whether colors identified by the ColorKeyColor will be marked as transparent (Alpha 0). The default value is **false**.

```
public virtual bool ColorKeyEnabled { get; set; }
```

Property Value

bool

DefaultEffect

Gets or Sets the default effect class to apply to this model. The default value is **BasicEffect**

```
public virtual MaterialProcessorDefaultEffect DefaultEffect { get; set; }
```

Property Value

MaterialProcessorDefaultEffect

GenerateMipmaps

Gets or Sets a value that indicates whether this will generate MipMaps for the selected texture. The default value is **false**.

```
public virtual bool GenerateMipmaps { get; set; }
```

Property Value

bool

GenerateTangentFrames

Gets or Sets a value that indicates whether model tangents should be generated for use in Normal Mapping. The default value is **false**.

```
public virtual bool GenerateTangentFrames { get; set; }
```

Property Value

bool

PremultiplyTextureAlpha

Gets or Sets a value that indicates whether the selected texture will be treated as an Alpha Mask. The default value is **true**.

```
public virtual bool PremultiplyTextureAlpha { get; set; }
```

Property Value

bool

PremultiplyVertexColors

Gets or Sets a value that indicates whether the selected texture will be treated as an Vertex Color Mask. The default value is **true**.

```
public virtual bool PremultiplyVertexColors { get; set; }
```

Property Value

bool

ResizeTexturesToPowerOfTwo

Gets or Sets a value that indicates if the model texture(s) dimensions should be resized to the next largest power of 2 size. (e.g. 128x128 or 512x512). The default value is **false**.

```
public virtual bool ResizeTexturesToPowerOfTwo { get; set; }
```

Property Value

bool

RotationX

Gets or Sets the default model x-axis rotation number, in degrees of rotation. The default value is **0.0f**.

```
public virtual float RotationX { get; set; }
```

Property Value

float

RotationY

Gets or Sets the default model y-axis rotation number, in degrees of rotation. The default value is **0.0f**.

```
public virtual float RotationY { get; set; }
```

Property Value

float

RotationZ

Gets or Sets the default model z-axis rotation number, in degrees of rotation. The default value is **0.0f**.

```
public virtual float RotationZ { get; set; }
```

Property Value

float

Scale

Gets or Sets the default model scale. The default value is **1.0f**.

```
public virtual float Scale { get; set; }
```

Property Value

float

SwapWindingOrder

Gets or Sets a value indicating whether the winding order is swapped. This is useful for models that appear to be drawn inside out. The default value is **false**.

```
public virtual bool SwapWindingOrder { get; set; }
```

Property Value

bool

TextureFormat

Gets or Sets the format of the texture to process (e.g. raw color, DXT). The default value is **compressed**.

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

Methods

ConvertMaterial(MaterialContent, ContentProcessorContext)

```
protected virtual MaterialContent ConvertMaterial(MaterialContent material,
ContentProcessorContext context)
```

Parameters

`material` MaterialContent

`context` ContentProcessorContext

Returns

MaterialContent

Process(NodeContent, ContentProcessorContext)

Converts mesh content to model content.

The root node content. Context for the specified processor.

```
public override ModelContent Process(NodeContent input, ContentProcessorContext
context)
```

Parameters

`input` NodeContent

`context` ContentProcessorContext

Returns

ModelContent

The model content.

ProcessGeometryUsingMaterial(MaterialContent, IEnumerable<GeometryContent>, ContentProcessorContext)

```
protected virtual void ProcessGeometryUsingMaterial(MaterialContent material,
IEnumerable<GeometryContent> geometryCollection, ContentProcessorContext context)
```

Parameters

`material` MaterialContent

`geometryCollection` IEnumerable<GeometryContent>

`context` ContentProcessorContext

ProcessVertexChannel(GeometryContent, int, ContentProcessorContext)

```
protected virtual void ProcessVertexChannel(GeometryContent geometry, int
vertexChannelIndex, ContentProcessorContext context)
```

Parameters

`geometry` GeometryContent

`vertexChannelIndex` int

`context` ContentProcessorContext

Class PassThroughProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

As the name implies, this processor simply passes data through as-is.

```
[ContentProcessor(DisplayName = "No Processing Required")]
public class PassThroughProcessor : ContentProcessor<object,
object>, IContentProcessor
```

Inheritance

object ← ContentProcessor<object, object> ← PassThroughProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<object, object>.Process(object, ContentProcessorContext) , ContentProcessor<object, object>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Process(object, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public override object Process(object input, ContentProcessorContext context)
```

Parameters

`input` object

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

object

A typed object representing the processed input.

Class SongContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents a processed Song object.

```
public sealed class SongContent
```

Inheritance

object ← SongContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Class SongProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

A custom song processor that processes an intermediate AudioContent type. This type encapsulates the source audio content, producing a Song type that can be used in the game.

```
[ContentProcessor(DisplayName = "Song - MonoGame")]
public class SongProcessor : ContentProcessor<AudioContent,
SongContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<AudioContent, SongContent> ← SongProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<AudioContent, SongContent>.Process(AudioContent, ContentProcessorContext) , ContentProcessor<AudioContent, SongContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SongProcessor()

Initializes a new instance of SongProcessor.

```
public SongProcessor()
```

Properties

Quality

Gets or sets the target format quality of the audio content.

```
public ConversionQuality Quality { get; set; }
```

Property Value

ConversionQuality

The ConversionQuality of this audio data.

Methods

Process(AudioContent, ContentProcessorContext)

Builds the content for the source audio.

```
public override SongContent Process(AudioContent input, ContentProcessorContext
context)
```

Parameters

`input` AudioContent

The audio content to build.

`context` ContentProcessorContext

Context for the specified processor.

Returns

SongContent

The built audio.

Class SoundEffectContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Represents a processed sound effect.

```
public sealed class SoundEffectContent
```

Inheritance

object ← SoundEffectContent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Class SoundEffectProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

A sound effect processor that processes an intermediate AudioContent type. This type encapsulates the source audio content, producing a SoundEffect type that can be used in the game.

```
[ContentProcessor(DisplayName = "Sound Effect - MonoGame")]
public class SoundEffectProcessor : ContentProcessor<AudioContent,
SoundEffectContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<AudioContent, SoundEffectContent> ← SoundEffectProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<AudioContent, SoundEffectContent>.Process(AudioContent, ContentProcessorContext) , ContentProcessor<AudioContent, SoundEffectContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SoundEffectProcessor()

Initializes a new instance of SoundEffectProcessor.

```
public SoundEffectProcessor()
```

Properties

Quality

Gets or sets the target format quality of the audio content.

```
public ConversionQuality Quality { get; set; }
```

Property Value

ConversionQuality

The ConversionQuality of this audio data.

Methods

Process(AudioContent, ContentProcessorContext)

Builds the content for the source audio.

```
public override SoundEffectContent Process(AudioContent input,
ContentProcessorContext context)
```

Parameters

`input` AudioContent

The audio content to build.

`context` ContentProcessorContext

Context for the specified processor.

Returns

SoundEffectContent

The built audio.

Class TextureProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide methods and properties for processing textures.

```
[ContentProcessor(DisplayName = "Texture - MonoGame")]
public class TextureProcessor : ContentProcessor<TextureContent,
TextureContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<TextureContent, TextureContent> ← TextureProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<TextureContent, TextureContent>.Process(TextureContent, ContentProcessorContext) , ContentProcessor<TextureContent, TextureContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

TextureProcessor()

Creates a new instance of the TextureProcessor class.

```
public TextureProcessor()
```

Properties

ColorKeyColor

Gets or sets the color key color. Typically used to make the background color of a texture transparent.

```
public virtual Color ColorKeyColor { get; set; }
```

Property Value

Color

ColorKeyEnabled

Gets or sets the color key flag. Must be set to true to use the color key.

```
public virtual bool ColorKeyEnabled { get; set; }
```

Property Value

bool

GenerateMipmaps

Gets or sets the generate mipmaps flag.

```
public virtual bool GenerateMipmaps { get; set; }
```

Property Value

bool

MakeSquare

Gets or sets the make square flag.

```
public virtual bool MakeSquare { get; set; }
```

Property Value

bool

PremultiplyAlpha

Gets or sets the premultiply alpha flag.

```
public virtual bool PremultiplyAlpha { get; set; }
```

Property Value

bool

ResizeToPowerOfTwo

Gets or sets the resize to power of two flag.

```
public virtual bool ResizeToPowerOfTwo { get; set; }
```

Property Value

bool

TextureFormat

Gets or sets the texture processor for the output format.

```
public virtual TextureProcessorOutputFormat TextureFormat { get; set; }
```

Property Value

TextureProcessorOutputFormat

Methods

Process(TextureContent, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public override TextureContent Process(TextureContent input,
ContentProcessorContext context)
```

Parameters

`input` TextureContent

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

TextureContent

A typed object representing the processed input.

Enum TextureProcessorOutputFormat

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Specifies the target output (of type SurfaceFormat) of the texture processor. Used by TextureProcessor.TextureFormat.

```
public enum TextureProcessorOutputFormat
```

Fields

`AstcCompressed = 9`

The input texture is compressed using Adaptive Scalable Texture Compression. Used on some Mobile and Intel platforms.

`AtcCompressed = 7`

The input texture is compressed using ATI texture compression. Used on some Android platforms.

`Color = 0`

The SurfaceFormat value, of the input TextureContent object, is converted to Color by the processor. Typically used for 2D graphics and overlays.

`Color16Bit = 4`

The pixel depth of the input texture is reduced to BGR565 for opaque textures, otherwise it uses BGRA4444.

`Compressed = 3`

The SurfaceFormat value, of the input TextureContent object, is converted to an appropriate compressed format for the target platform. This can include PVRTC for iOS, DXT for desktop, Windows 8 and Windows Phone 8, and ETC1/2 or BGRA4444 for Android.

`DxtCompressed = 1`

The SurfaceFormat value, of the input TextureContent object, is converted to an appropriate DXT compression by the processor. If the input texture contains fractional alpha values, it is converted to DXT5 format (8 bits per texel); otherwise it is converted to DXT1 (4 bits per texel). This conversion reduces the resource's size on the graphics card. Typically used for 3D textures such as 3D model textures.

```
[Obsolete("Prefer EtcCompressed when possible.")] Etc1Compressed = 5
```

The input texture is compressed using ETC1 texture compression. Used on Android platforms.

```
EtcCompressed = 8
```

The input texture is compressed using ETC texture compression. The best ETC option will be selected given the image data. Used on Android platforms. This option should be preferred over Etc1 Compressed whenever possible.

```
NoChange = 2
```

The SurfaceFormat value, of the input TextureContent object, is not changed by the processor. Typically used for textures processed by an external tool.

```
PvrCompressed = 6
```

The input texture is compressed using PVR texture compression. Used on iOS and some Android platforms.

Class VertexBufferContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for managing a design-time vertex buffer that holds packed vertex data.

```
public class VertexBufferContent : ContentItem
```

Inheritance

object ← ContentItem ← VertexBufferContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This type directly corresponds to the runtime VertexBuffer class, and when a VertexBufferContent object is passed to the content compiler, the vertex data deserializes directly into a VertexBuffer at runtime. VertexBufferContent objects are not directly created by importers. The preferred method is to store vertex data in the more flexible VertexContent class.

Constructors

VertexBufferContent()

Initializes a new instance of VertexBufferContent.

```
public VertexBufferContent()
```

VertexBufferContent(int)

Initializes a new instance of VertexBufferContent of the specified size.

```
public VertexBufferContent(int size)
```

Parameters

`size` int

The size of the vertex buffer content, in bytes.

Properties

VertexData

Gets the array containing the raw bytes of the packed vertex data. Use this method to get and set the contents of the vertex buffer.

```
public byte[] VertexData { get; }
```

Property Value

byte[]

Raw data of the packed vertex data.

VertexDeclaration

Gets and sets the associated VertexDeclarationContent object.

```
public VertexDeclarationContent VertexDeclaration { get; set; }
```

Property Value

VertexDeclarationContent

The associated VertexDeclarationContent object.

Methods

SizeOf(Type)

Gets the size of the specified type, in bytes.

```
public static int SizeOf(Type type)
```

Parameters

`type` Type

The type.

Returns

int

The size of the specified type, in bytes.

Remarks

Call this method to compute offset parameters for the Write method. If the specified data type cannot be packed into a vertex buffer—for example, if type is not a valid value type—a NotSupportedException is thrown.

Exceptions

NotSupportedException

type is not a valid value type

Write(int, int, Type, IEnumerable)

Writes additional data into the vertex buffer. Writing begins at the specified byte offset, and each value is spaced according to the specified stride value (in bytes).

```
public void Write(int offset, int stride, Type dataType, IEnumerable data)
```

Parameters

`offset` int

Offset at which to begin writing.

`stride` int

Stride of the data being written, in bytes.

`dataType` Type

The type of data to be written.

`data` IEnumerable

The data to write.

Exceptions

NotSupportedException

The specified data type cannot be packed into a vertex buffer.

Write<T>(int, int, IEnumerable<T>)

Writes additional data into the vertex buffer. Writing begins at the specified byte offset, and each value is spaced according to the specified stride value (in bytes).

```
public void Write<T>(int offset, int stride, IEnumerable<T> data)
```

Parameters

`offset` int

Offset to begin writing at.

`stride` int

Stride of the data being written, in bytes.

`data` IEnumerable<T>

Enumerated collection of data.

Type Parameters

T

Type being written.

Exceptions

NotSupportedException

The specified data type cannot be packed into a vertex buffer.

Class VertexDeclarationContent

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining the vertex declaration data of a VertexContent.

```
public class VertexDeclarationContent : ContentItem
```

Inheritance

object ← ContentItem ← VertexDeclarationContent

Inherited Members

ContentItem.Identity , ContentItem.Name , ContentItem.OpaqueData , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

VertexDeclarationContent()

Initializes a new instance of VertexDeclarationContent.

```
public VertexDeclarationContent()
```

Properties

VertexElements

Gets the VertexElement object of the vertex declaration.

```
public Collection<VertexElement> VertexElements { get; }
```

Property Value

Collection<VertexElement>

The VertexElement object of the vertex declaration.

VertexStride

The number of bytes from one vertex to the next.

```
public int? VertexStride { get; set; }
```

Property Value

int?

The stride (in bytes).

Class VideoProcessor

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Processors

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for maintaining video content.

```
[ContentProcessor(DisplayName = "Video - MonoGame")]
public class VideoProcessor : ContentProcessor<VideoContent,
VideoContent>, IContentProcessor
```

Inheritance

object ← ContentProcessor<VideoContent, VideoContent> ← VideoProcessor

Implements

IContentProcessor

Inherited Members

ContentProcessor<VideoContent, VideoContent>.Process(VideoContent, ContentProcessorContext) , ContentProcessor<VideoContent, VideoContent>.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Process(VideoContent, ContentProcessorContext)

Processes the specified input data and returns the result.

```
public override VideoContent Process(VideoContent input, ContentProcessorContext
context)
```

Parameters

`input` VideoContent

Existing content object being processed.

`context` ContentProcessorContext

Contains any required custom process parameters.

Returns

VideoContent

A typed object representing the processed input.

Namespace Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Classes

ContentCompiler

Provides methods for writing compiled binary format.

ContentTypeWriter

Provides methods and properties for compiling a specific managed type into a binary format.

ContentTypeWriterAttribute

Identifies the components of a type writer. Custom content writers must apply this attribute to their class as well as extend the ContentTypeWriter class.

ContentTypeWriter<T>

Provides a generic implementation of ContentTypeWriter methods and properties for compiling a specific managed type into a binary format.

ContentWriter

Provides an implementation for many of the ContentCompiler methods including compilation, state tracking for shared resources and creation of the header type manifest.

SpriteFontContentWriter

Provides methods and properties to write a SpriteFontContent to binary format.

Class ContentCompiler

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for writing compiled binary format.

```
public sealed class ContentCompiler
```

Inheritance

object ← ContentCompiler

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentCompiler()

Initializes a new instance of ContentCompiler.

```
public ContentCompiler()
```

Methods

Compile(Stream, object, TargetPlatform, GraphicsProfile, bool, string, string)

Write the content to a XNB file.

```
public void Compile(Stream stream, object content, TargetPlatform targetPlatform,
GraphicsProfile targetProfile, bool compressContent, string rootDirectory,
string referenceRelocationPath)
```

Parameters

`stream` Stream

The stream to write the XNB file to.

`content` object

The content to write to the XNB file.

`targetPlatform` TargetPlatform

The platform the XNB is intended for.

`targetProfile` GraphicsProfile

The graphics profile of the target.

`compressContent` bool

True if the content should be compressed.

`rootDirectory` string

The root directory of the content.

`referenceRelocationPath` string

The path of the XNB file, used to calculate relative paths for external references.

GetTypeWriter(Type)

Retrieves the worker writer for the specified type.

```
public ContentTypeWriter GetTypeWriter(Type type)
```

Parameters

`type` Type

The type.

Returns

ContentTypeWriter

The worker writer.

Remarks

This should be called from the ContentTypeWriter.Initialize method.

Class ContentTypeWriter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties for compiling a specific managed type into a binary format.

```
public abstract class ContentTypeWriter
```

Inheritance

object ← ContentTypeWriter

Derived

ContentTypeWriter<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeWriter(Type)

Initializes a new instance of the ContentTypeWriter class.

```
protected ContentTypeWriter(Type targetType)
```

Parameters

`targetType` Type

Fields

_typeVersion

```
protected int _typeVersion
```

Field Value

int

Properties

CanDeserializeIntoExistingObject

Determines if deserialization into an existing object is possible.

```
public virtual bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

true if the object can be deserialized into; false otherwise.

TargetType

Gets the type handled by this compiler component.

```
public Type TargetType { get; }
```

Property Value

Type

The type handled by this compiler component.

TypeVersion

Gets a format version number for this type.

```
public virtual int TypeVersion { get; }
```

Property Value

int

A format version number for this type.

Methods

GetRuntimeReader(TargetPlatform)

Gets the assembly qualified name of the runtime loader for this type.

```
public abstract string GetRuntimeReader(TargetPlatform targetPlatform)
```

Parameters

`targetPlatform` TargetPlatform

Name of the platform.

Returns

string

Name of the runtime loader.

GetRuntimeType(TargetPlatform)

Gets the assembly qualified name of the runtime target type. The runtime target type often matches the design time type, but may differ.

```
public virtual string GetRuntimeType(TargetPlatform targetPlatform)
```

Parameters

`targetPlatform` TargetPlatform

The target platform.

Returns

string

The qualified name.

Initialize(ContentCompiler)

Retrieves and caches nested type writers and allows for reflection over the target data type. Called by the framework at creation time.

```
protected virtual void Initialize(ContentCompiler compiler)
```

Parameters

`compiler` ContentCompiler

The content compiler.

ShouldCompressContent(TargetPlatform, object)

Indicates whether a given type of content should be compressed.

```
protected virtual bool ShouldCompressContent(TargetPlatform targetPlatform,
object value)
```

Parameters

`targetPlatform` TargetPlatform

The target platform of the content build.

`value` object

The object about to be serialized, or null if a collection of objects is to be serialized.

Returns

bool

true if the content of the requested type should be compressed; false otherwise.

Remarks

This base class implementation of this method always returns true. It should be overridden to return false if there would be little or no useful reduction in size of the content type's data from a general-purpose lossless compression algorithm. The implementations for Song Class and SoundEffect Class data return false because data for these content types is already in compressed form.

Write(ContentWriter, object)

Compiles an object into binary format.

```
protected abstract void Write(ContentWriter output, object value)
```

Parameters

`output` ContentWriter

The content writer serializing the value.

`value` object

The resultant object.

Class ContentTypeWriterAttribute

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Identifies the components of a type writer. Custom content writers must apply this attribute to their class as well as extend the ContentTypeWriter class.

```
[AttributeUsage(AttributeTargets.Class)]
public sealed class ContentTypeWriterAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentTypeWriterAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,

Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeWriterAttribute()

Initializes a new instance of the ContentTypeWriterAttribute class.

```
public ContentTypeWriterAttribute()
```

Class ContentTypeWriter<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a generic implementation of ContentTypeWriter methods and properties for compiling a specific managed type into a binary format.

```
public abstract class ContentTypeWriter<T> : ContentTypeWriter
```

Type Parameters

T

The type to write

Inheritance

object ← ContentTypeWriter ← ContentTypeWriter<T>

Derived

SpriteFontContentWriter

Inherited Members

ContentTypeWriter._typeVersion , ContentTypeWriter.CanDeserializeIntoExistingObject , ContentTypeWriter.TargetType , ContentTypeWriter.TypeVersion , ContentTypeWriter.GetRuntimeReader(TargetPlatform) , ContentTypeWriter.GetRuntimeType(TargetPlatform) , ContentTypeWriter.Initialize(ContentCompiler) , ContentTypeWriter.ShouldCompressContent(TargetPlatform, object) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

This is a generic implementation of ContentTypeWriter and, therefore, can handle strongly typed content data.

Constructors

ContentTypeWriter()

Initializes a new instance of the ContentTypeWriter class.

```
protected ContentTypeWriter()
```

Methods

Write(ContentWriter, object)

Compiles a strongly typed object into binary format.

```
protected override void Write(ContentWriter output, object value)
```

Parameters

`output` ContentWriter

The content writer serializing the value.

`value` object

The value to write.

Write(ContentWriter, T)

Compiles a strongly typed object into binary format.

```
protected abstract void Write(ContentWriter output, T value)
```

Parameters

`output` ContentWriter

The content writer serializing the value.

`value` T

The value to write.

Class ContentWriter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides an implementation for many of the ContentCompiler methods including compilation, state tracking for shared resources and creation of the header type manifest.

```
public sealed class ContentWriter : BinaryWriter, IAsyncDisposable, IDisposable
```

Inheritance

object ← BinaryWriter ← ContentWriter

Implements

IAsyncDisposable, IDisposable

Inherited Members

BinaryWriter.Null , BinaryWriter.Close() , BinaryWriter.Dispose() , BinaryWriter.DisposeAsync() , BinaryWriter.Flush() , BinaryWriter.Seek(int, SeekOrigin) , BinaryWriter.Write(bool) , BinaryWriter.Write(byte) , BinaryWriter.Write(byte[]) , BinaryWriter.Write(byte[], int, int) , BinaryWriter.Write(char) , BinaryWriter.Write(char[]) , BinaryWriter.Write(char[], int, int) , BinaryWriter.Write(decimal) , BinaryWriter.Write(double) , BinaryWriter.Write(Half) , BinaryWriter.Write(short) , BinaryWriter.Write(int) , BinaryWriter.Write(long) , BinaryWriter.Write(ReadOnlySpan<byte>) , BinaryWriter.Write(ReadOnlySpan<char>) , BinaryWriter.Write(sbyte) , BinaryWriter.Write(float) , BinaryWriter.Write(string) , BinaryWriter.Write(ushort) , BinaryWriter.Write(uint) , BinaryWriter.Write(ulong) , BinaryWriter.Write7BitEncodedInt(int) , BinaryWriter.Write7BitEncodedInt64(long) , BinaryWriter.BaseStream , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

A new ContentWriter is constructed for each compilation operation.

Properties

TargetPlatform

Gets the content build target platform.

```
public TargetPlatform TargetPlatform { get; }
```

Property Value

TargetPlatform

TargetProfile

Gets or sets the target graphics profile.

```
public GraphicsProfile TargetProfile { get; }
```

Property Value

GraphicsProfile

Methods

Dispose(bool)

Releases the resources used by the IDisposable class.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

true to release both managed and unmanaged resources; false to release only unmanaged resources.

Write(Color)

Writes a Color value.

```
public void Write(Color value)
```

Parameters

`value` Color

Value of a color using Red, Green, Blue, and Alpha values to write.

Write(Matrix)

Writes a Matrix value.

```
public void Write(Matrix value)
```

Parameters

`value` Matrix

Value to write.

Write(Quaternion)

Writes a Matrix value.

```
public void Write(Quaternion value)
```

Parameters

`value` Quaternion

Value to write.

Write(Vector2)

Writes a Vector2 value.

```
public void Write(Vector2 value)
```

Parameters

`value` Vector2

Value to write.

Write(Vector3)

Writes a Vector3 value.

```
public void Write(Vector3 value)
```

Parameters

`value` Vector3

Value to write.

Write(Vector4)

Writes a Vector4 value.

```
public void Write(Vector4 value)
```

Parameters

`value` Vector4

Value to write.

WriteExternalReference<T>(ExternalReference<T>)

Writes the name of an external file to the output binary.

```
public void WriteExternalReference<T>(ExternalReference<T> reference)
```

Parameters

`reference` ExternalReference<T>

External reference to a data file for the content item.

Type Parameters

`T`

The type of reference.

WriteObject<T>(T)

Writes a single object preceded by a type identifier to the output binary.

```
public void WriteObject<T>(T value)
```

Parameters

`value` T

The value to write.

Type Parameters

`T`

The type of value.

Remarks

This method can be called recursively with a null value.

WriteObject<T>(T, ContentTypeWriter)

Writes a single object to the output binary, using the specified type hint and writer worker.

```
public void WriteObject<T>(T value, ContentTypeWriter typeWriter)
```

Parameters

`value` T

The value to write.

`typeWriter` ContentTypeWriter

The content type writer.

Type Parameters

`T`

The type of value.

Remarks

The type hint should be retrieved from the Initialize method of the ContentTypeWriter that is calling WriteObject, by calling GetTypeWriter and passing it the type of the field used to hold the value being serialized.

WriteRawObject<T>(T)

Writes a single object to the output binary as an instance of the specified type.

```
public void WriteRawObject<T>(T value)
```

Parameters

`value` T

The value to write.

Type Parameters

`T`

The type of value.

Remarks

If you specify a base class of the actual object value only data from this base type will be written. This method does not write any type identifier so it cannot support null or polymorphic values, and the reader must specify an identical type while loading the compiled data.

WriteRawObject<T>(T, ContentTypeWriter)

Writes a single object to the output binary using the specified writer worker.

```
public void WriteRawObject<T>(T value, ContentTypeWriter typeWriter)
```

Parameters

`value` `T`

The value to write.

`typeWriter` ContentTypeWriter

The writer worker. This should be looked up from the Initialize method of the ContentTypeWriter that is calling WriteRawObject, by calling GetTypeWriter.

Type Parameters

`T`

The type of value.

Remarks

WriteRawObject does not write any type identifier, so it cannot support null or polymorphic values, and the reader must specify an identical type while loading the compiled data.

WriteSharedResource<T>(T)

Adds a shared reference to the output binary and records the object to be serialized later.

```
public void WriteSharedResource<T>(T value)
```

Parameters

`value` T

The object to record.

Type Parameters

`T`

The type of value.

Class SpriteFontContentWriter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compiler

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods and properties to write a SpriteFontContent to binary format.

```
[ContentTypeWriter]
public class SpriteFontContentWriter : ContentTypeWriter<SpriteFontContent>
```

Inheritance

object ← ContentTypeWriter ← ContentTypeWriter<SpriteFontContent> ← SpriteFontContentWriter

Inherited Members

ContentTypeWriter<SpriteFontContent>.Write(ContentWriter, object) , ContentTypeWriter._typeVersion , ContentTypeWriter.CanDeserializeIntoExistingObject , ContentTypeWriter.TargetType , ContentTypeWriter.TypeVersion , ContentTypeWriter.Initialize(ContentCompiler) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetRuntimeReader(TargetPlatform)

Gets the assembly qualified name of the runtime loader for this type.

```
public override string GetRuntimeReader(TargetPlatform targetPlatform)
```

Parameters

`targetPlatform` TargetPlatform

Name of the platform.

Returns

string

Name of the runtime loader.

GetRuntimeType(TargetPlatform)

Gets the assembly qualified name of the runtime target type. The runtime target type often matches the design time type, but may differ.

```
public override string GetRuntimeType(TargetPlatform targetPlatform)
```

Parameters

`targetPlatform` TargetPlatform

The target platform.

Returns

string

The qualified name.

ShouldCompressContent(TargetPlatform, object)

Indicates whether a given type of content should be compressed.

```
protected override bool ShouldCompressContent(TargetPlatform targetPlatform,
object value)
```

Parameters

`targetPlatform` TargetPlatform

The target platform of the content build.

`value` object

The object about to be serialized, or null if a collection of objects is to be serialized.

Returns

bool

true if the content of the requested type should be compressed; false otherwise.

Remarks

This base class implementation of this method always returns true. It should be overridden to return false if there would be little or no useful reduction in size of the content type's data from a general-purpose lossless compression algorithm. The implementations for Song Class and SoundEffect Class data return false because data for these content types is already in compressed form.

Write(ContentWriter, SpriteFontContent)

Compiles a strongly typed object into binary format.

```
protected override void Write(ContentWriter output, SpriteFontContent value)
```

Parameters

`output` ContentWriter

The content writer serializing the value.

`value` SpriteFontContent

The value to write.

Namespace Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Classes

ContentTypeSerializer

Provides methods for serializing and deserializing a specific managed type.

ContentTypeSerializerAttribute

Used to identify custom ContentTypeSerializer classes.

ContentTypeSerializer<T>

Provides a generic implementation of ContentTypeSerializer methods and properties for serializing and deserializing a specific managed type.

IntermediateReader

IntermetiateReader is used to read content from the intermediate format.

IntermediateSerializer

Provides methods for reading and writing XNA intermediate XML format.

IntermediateWriter

IntermediateWriter is used to write content to the intermediate format.

Delegates

ContentTypeSerializer.ChildCallback

Class ContentTypeSerializer

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for serializing and deserializing a specific managed type.

```
public abstract class ContentTypeSerializer
```

Inheritance

object ← ContentTypeSerializer

Derived

ContentTypeSerializer<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeSerializer(Type, string)

Initializes a new instance of the ContentTypeSerializer class.

```
protected ContentTypeSerializer(Type targetType, string xmlTypeName)
```

Parameters

`targetType` Type

The target type.

`xmlTypeName` string

The XML shortcut name.

Properties

CanDeserializeIntoExistingObject

Gets a value indicating whether this component may load data into an existing object or if it must it construct a new instance of the object before loading the data.

```
public virtual bool CanDeserializeIntoExistingObject { get; }
```

Property Value

bool

TargetType

Gets the target type.

```
public Type TargetType { get; }
```

Property Value

Type

XmlTypeName

Gets the XML shortcut name (XmlTypeName).

```
public string XmlTypeName { get; }
```

Property Value

string

Methods

Deserialize(IntermediateReader, ContentSerializerAttribute, object)

```
protected abstract object Deserialize(IntermediateReader input,
ContentSerializerAttribute format, object existingInstance)
```

Parameters

`input` IntermediateReader

`format` ContentSerializerAttribute

`existingInstance` object

Returns

object

Initialize(IntermediateSerializer)

```
protected virtual void Initialize(IntermediateSerializer serializer)
```

Parameters

`serializer` IntermediateSerializer

ObjectIsEmpty(object)

Queries whether an object contains data to be serialized.

```
public virtual bool ObjectIsEmpty(object value)
```

Parameters

`value` object

The object to test.

Returns

bool

`true` if the object is empty; otherwise, `false`.

ScanChildren(IntermediateSerializer, ChildCallback, object)

```
protected virtual void ScanChildren(IntermediateSerializer serializer,
ContentTypeSerializer.ChildCallback callback, object value)
```

Parameters

`serializer` IntermediateSerializer

`callback` ContentTypeSerializer.ChildCallback

`value` object

Serialize(IntermediateWriter, object, ContentSerializerAttribute)

```
protected abstract void Serialize(IntermediateWriter output, object value,
ContentSerializerAttribute format)
```

Parameters

`output` IntermediateWriter

`value` object

`format` ContentSerializerAttribute

Delegate ContentTypeSerializer.ChildCallback

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
protected delegate void ContentTypeSerializer.ChildCallback(ContentTypeSerializer
typeSerializer, object value)
```

Parameters

`typeSerializer` ContentTypeSerializer

`value` object

Class ContentTypeSerializerAttribute

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

Used to identify custom ContentTypeSerializer classes.

```
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public sealed class ContentTypeSerializerAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentTypeSerializerAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeSerializerAttribute()

Initializes an instance of the ContentTypeSerializerAttribute.

```
public ContentTypeSerializerAttribute()
```

Class ContentTypeSerializer<T>

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides a generic implementation of ContentTypeSerializer methods and properties for serializing and deserializing a specific managed type.

```
public abstract class ContentTypeSerializer<T> : ContentTypeSerializer
```

Type Parameters

T

Inheritance

object ← ContentTypeSerializer ← ContentTypeSerializer<T>

Inherited Members

ContentTypeSerializer.CanDeserializeIntoExistingObject , ContentTypeSerializer.TargetType , ContentTypeSerializer.XmlTypeName , ContentTypeSerializer.Initialize(IntermediateSerializer) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentTypeSerializer()

Initializes a new instance of the ContentTypeSerializer class.

```
protected ContentTypeSerializer()
```

ContentTypeSerializer(string)

Initializes a new instance of the ContentTypeSerializer class using the specified XML shortcut name.

```
protected ContentTypeSerializer(string xmlTypeName)
```

Parameters

`xmlTypeName` string

Xml shortcut name.

Methods

Deserialize(IntermediateReader, ContentSerializerAttribute, object)

```
protected override object Deserialize(IntermediateReader input,
ContentSerializerAttribute format, object existingInstance)
```

Parameters

`input` IntermediateReader

`format` ContentSerializerAttribute

`existingInstance` object

Returns

object

Deserialize(IntermediateReader, ContentSerializerAttribute, T)

```
protected abstract T Deserialize(IntermediateReader input,
ContentSerializerAttribute format, T existingInstance)
```

Parameters

`input` IntermediateReader

`format` ContentSerializerAttribute

`existingInstance` T

Returns

T

ObjectIsEmpty(object)

Queries whether an object contains data to be serialized.

```
public override bool ObjectIsEmpty(object value)
```

Parameters

`value` object

The object to test.

Returns

bool

`true` if the object is empty; otherwise, `false`.

ObjectIsEmpty(T)

Queries whether an object contains data to be serialized.

```
public virtual bool ObjectIsEmpty(T value)
```

Parameters

`value` T

The object to test.

Returns

bool

`true` if the object is empty; otherwise, `false`.

ScanChildren(IntermediateSerializer, ChildCallback, object)

```
protected override void ScanChildren(IntermediateSerializer serializer,
ContentTypeSerializer.ChildCallback callback, object value)
```

Parameters

`serializer` IntermediateSerializer

`callback` ContentTypeSerializer.ChildCallback

`value` object

ScanChildren(IntermediateSerializer, ChildCallback, T)

```
protected virtual void ScanChildren(IntermediateSerializer serializer,
ContentTypeSerializer.ChildCallback callback, T value)
```

Parameters

`serializer` IntermediateSerializer

`callback` ContentTypeSerializer.ChildCallback

`value` T

Serialize(IntermediateWriter, object, ContentSerializerAttribute)

```
protected override void Serialize(IntermediateWriter output, object value,
ContentSerializerAttribute format)
```

Parameters

`output` IntermediateWriter

`value` object

`format` ContentSerializerAttribute

Serialize(IntermediateWriter, T, ContentSerializerAttribute)

```
protected abstract void Serialize(IntermediateWriter output, T value,
ContentSerializerAttribute format)
```

Parameters

`output` IntermediateWriter

`value` T

`format` ContentSerializerAttribute

Class IntermediateReader

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

IntermetiateReader is used to read content from the intermediate format.

```
public sealed class IntermediateReader
```

Inheritance

object ← IntermediateReader

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Serializer

Gets the serializer.

```
public IntermediateSerializer Serializer { get; }
```

Property Value

IntermediateSerializer

Xml

Gets the instances XML reader.

```
public XmlReader Xml { get; }
```

Property Value

XmlReader

Methods

MoveToElement(string)

Moves the XML reader to the specified element.

```
public bool MoveToElement(string elementName)
```

Parameters

`elementName` string

The name of the element to move to.

Returns

bool

`true` if the element is found, `false` otherwise.

ReadExternalReference<T>(ExternalReference<T>)

Reads an external reference of a given type.

```
public void ReadExternalReference<T>(ExternalReference<T> existingInstance)
```

Parameters

`existingInstance` ExternalReference<T>

The existing instance of the ExternalReference.

Type Parameters

`T`

The type of the external reference.

Exceptions

InvalidContentException

Thrown if the external reference type is invalid.

ReadObject<T>(ContentSerializerAttribute)

Reads an object from the XML.

```
public T ReadObject<T>(ContentSerializerAttribute format)
```

Parameters

`format` ContentSerializerAttribute

The format attribute to use.

Returns

T

The deserialized object of type T.

Type Parameters

T

The type of the object.

ReadObject<T>(ContentSerializerAttribute, ContentTypeSerializer)

Reads an object from the XML.

```
public T ReadObject<T>(ContentSerializerAttribute format, ContentTypeSerializer
typeSerializer)
```

Parameters

`format` ContentSerializerAttribute

The format attribute to use.

`typeSerializer` ContentTypeSerializer

The type serializer to use.

Returns

T

The deserialized object of type T.

Type Parameters

`T`

The type of the object.

ReadObject<T>(ContentSerializerAttribute, ContentTypeSerializer, T)

Reads an object from the XML.

```
public T ReadObject<T>(ContentSerializerAttribute format, ContentTypeSerializer
typeSerializer, T existingInstance)
```

Parameters

`format` ContentSerializerAttribute

The format attribute of the object.

`typeSerializer` ContentTypeSerializer

The type serializer for the object.

`existingInstance` T

An existing instance of the object.

Returns

T

The deserialized object of type T.

Type Parameters

T

The type of the object.

Exceptions

InvalidContentException

Thrown when the element can not be found, is null or cannot be assigned.

ReadObject<T>(ContentSerializerAttribute, T)

Reads an object from the XML.

```
public T ReadObject<T>(ContentSerializerAttribute format, T existingInstance)
```

Parameters

`format` ContentSerializerAttribute

The format attribute of the object.

`existingInstance` T

An existing instance of the object.

Returns

T

The deserialized object of type T.

Type Parameters

T

The type of the object.

ReadRawObject<T>(ContentSerializerAttribute)

Reads a raw object from the XML using the specified format and type serializer.

```
public T ReadRawObject<T>(ContentSerializerAttribute format)
```

Parameters

format ContentSerializerAttribute

The format attribute of the object.

Returns

T

The deserialized object of type T.

Type Parameters

T

The type of the object.

Exceptions

InvalidContentException

Thrown when the element can not be found or is null.

ReadRawObject<T>(ContentSerializerAttribute, ContentTypeSerializer)

Reads an object from the XML.

```
public T ReadRawObject<T>(ContentSerializerAttribute format,
ContentTypeSerializer typeSerializer)
```

Parameters

`format` ContentSerializerAttribute

The format attribute of the object.

`typeSerializer` ContentTypeSerializer

The type serializer for the object.

Returns

T

The deserialized object of type T.

Type Parameters

`T`

The type of the object.

ReadRawObject<T>(ContentSerializerAttribute, ContentTypeSerializer, T)

Reads a raw object from the XML using the specified format and type serializer.

```
public T ReadRawObject<T>(ContentSerializerAttribute format, ContentTypeSerializer
typeSerializer, T existingInstance)
```

Parameters

`format` ContentSerializerAttribute

The format attribute of the object.

`typeSerializer` ContentTypeSerializer

The type serializer for the object.

`existingInstance` T

An existing instance of the object.

Returns

T

The deserialized object of type T.

Type Parameters

`T`

The type of the object.

Exceptions

InvalidContentException

Thrown when the element can not be found or is null.

ReadRawObject<T>(ContentSerializerAttribute, T)

Reads a raw object from the XML using the specified format and type serializer.

```
public T ReadRawObject<T>(ContentSerializerAttribute format, T existingInstance)
```

Parameters

`format` ContentSerializerAttribute

The format attribute of the object.

`existingInstance` T

An existing instance of the object.

Returns

T

The deserialized object of type T.

Type Parameters

T

The type of the object.

Exceptions

InvalidContentException

Thrown when the element can not be found or is null.

ReadSharedResource<T>(ContentSerializerAttribute, Action<T>)

Reads a shared resource from the XML using the specified format and fixup action.

```
public void ReadSharedResource<T>(ContentSerializerAttribute format,
Action<T> fixup)
```

Parameters

format ContentSerializerAttribute

The format attribute of the resource.

fixup Action<T>

The fixup action to apply to the resource.

Type Parameters

T

The type of the resource.

Exceptions

InvalidContentException

Thrown if the element specified by the format attribute is not found.

ReadTypeName()

Reads the next type in the

```
public Type ReadTypeName()
```

Returns

Type

Class IntermediateSerializer

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reading and writing XNA intermediate XML format.

```
public class IntermediateSerializer
```

Inheritance

object ← IntermediateSerializer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Deserialize<T>(XmlReader, string)

Deserializes an object using the IntermediateSerializer and IntermediateReader.

```
public static T Deserialize<T>(XmlReader input, string referenceRelocationPath)
```

Parameters

`input` XmlReader

The XmlReader to read from.

`referenceRelocationPath` string

The path to relocate any relative references to.

Returns

T

The deserialized object of type T.

Type Parameters

`T`

The type of object to deserialize.

Exceptions

InvalidContentException

Thrown if no content is found.

XmlException

Thrown if an error occurs parsing the XML.

GetTypeSerializer(Type)

Retrieves the serializer for the specified type.

```
public ContentTypeSerializer GetTypeSerializer(Type type)
```

Parameters

`type` Type

The type for which to retrieve the serializer.

Returns

ContentTypeSerializer

The serializer for the specified type.

Exceptions

RankException

Thrown if the type is not a single dimensional array.

NotImplementedException

Thrown if there is no implementation for the type.

Serialize<T>(XmlWriter, T, string)

Serializes the given value of type T to an XML writer.

```
public static void Serialize<T>(XmlWriter output, T value,
string referenceRelocationPath)
```

Parameters

`output` XmlWriter

The XML writer to write the serialized value to.

`value` T

The value to serialize.

`referenceRelocationPath` string

The path to relocate any relative references to.

Type Parameters

`T`

The type of the value to serialize.

Class IntermediateWriter

Namespace: Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate

Assembly: MonoGame.Framework.Content.Pipeline.dll

IntermediateWriter is used to write content to the intermediate format.

```
public sealed class IntermediateWriter
```

Inheritance

object ← IntermediateWriter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Serializer

Gets the serializer.

```
public IntermediateSerializer Serializer { get; }
```

Property Value

IntermediateSerializer

Xml

Gets the instances XML writer.

```
public XmlWriter Xml { get; }
```

Property Value

XmlWriter

Methods

WriteExternalReference<T>(ExternalReference<T>)

Writes an external reference to the XML writer.

```
public void WriteExternalReference<T>(ExternalReference<T> value)
```

Parameters

`value` ExternalReference<T>

The external reference to write.

Type Parameters

`T`

The type of the external reference.

WriteObject(object, ContentSerializerAttribute, ContentTypeSerializer, Type)

Writes an object to the XML writer using the specified format and type serializer.

```
public void WriteObject(object value, ContentSerializerAttribute format,
ContentTypeSerializer typeSerializer, Type declaredType)
```

Parameters

`value` object

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

`typeSerializer` ContentTypeSerializer

The type serializer to use.

`declaredType` Type

The declared type of the object.

WriteObject<T>(T, ContentSerializerAttribute)

Writes an object to the XML writer using the specified format and type serializer.

```
public void WriteObject<T>(T value, ContentSerializerAttribute format)
```

Parameters

`value` T

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

Type Parameters

`T`

The type of the object.

WriteObject<T>(T, ContentSerializerAttribute, ContentTypeSerializer)

Writes an object to the XML writer using the specified format and type serializer.

```
public void WriteObject<T>(T value, ContentSerializerAttribute format,
ContentTypeSerializer typeSerializer)
```

Parameters

`value` T

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

`typeSerializer` ContentTypeSerializer

The type serializer to use.

Type Parameters

`T`

The type of the object.

WriteRawObject<T>(T, ContentSerializerAttribute)

Writes a raw object to the XML writer.

```
public void WriteRawObject<T>(T value, ContentSerializerAttribute format)
```

Parameters

`value` T

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

Type Parameters

`T`

The type of the object.

WriteRawObject<T>(T, ContentSerializerAttribute, ContentTypeSerializer)

Writes a raw object to the XML writer.

```
public void WriteRawObject<T>(T value, ContentSerializerAttribute format,
ContentTypeSerializer typeSerializer)
```

Parameters

`value` T

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

`typeSerializer` ContentTypeSerializer

The type serializer to use.

Type Parameters

`T`

The type of the object.

WriteSharedResource<T>(T, ContentSerializerAttribute)

Writes a raw object to the XML writer.

```
public void WriteSharedResource<T>(T value, ContentSerializerAttribute format)
```

Parameters

`value` T

The object to write.

`format` ContentSerializerAttribute

The format attribute to use.

Type Parameters

`T`

The type of the object.

WriteTypeName(Type)

Writes the fully qualified name of the specified type to the XML writer.

```
public void WriteTypeName(Type type)
```

Parameters

`type` Type

The type whose fully qualified name will be written.

Namespace Microsoft.Xna.Framework.Design

Classes

Byte4TypeConverter

Provides a unified way of converting Byte4 value to other types, as well as for accessing standard values and subproperties.

Vector2TypeConverter

Provides a unified way of converting Vector2 values to other types, as well as for accessing standard values and subproperties.

Vector3TypeConverter

Provides a unified way of converting Vector3 values to other types, as well as for accessing standard values and subproperties.

Vector4TypeConverter

Provides a unified way of converting Vector4 values to other types, as well as for accessing standard values and subproperties.

Class Byte4TypeConverter

Namespace: Microsoft.Xna.Framework.Design

Assembly: MonoGame.Framework.dll

Provides a unified way of converting Byte4 value to other types, as well as for accessing standard values and subproperties.

```
public sealed class Byte4TypeConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← Byte4TypeConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) , TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,
TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesSupported() ,

TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

CanConvertTo(ITypeDescriptorContext, Type)

Returns whether this converter can convert the object to the specified type, using the specified context.

```
public override bool CanConvertTo(ITypeDescriptorContext context,
Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type you want to convert to.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type)

Converts the given value object to the specified type, using the specified context and culture information.

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo
culture, object value, Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If null is passed, the current culture is assumed.

`value` object

The object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

object

An object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is null.

NotSupportedException

The conversion cannot be performed.

Class Vector2TypeConverter

Namespace: Microsoft.Xna.Framework.Design

Assembly: MonoGame.Framework.dll

Provides a unified way of converting Vector2 values to other types, as well as for accessing standard values and subproperties.

```
public sealed class Vector2TypeConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← Vector2TypeConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) , TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,
TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesSupported() ,

TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

CanConvertTo(ITypeDescriptorContext, Type)

Returns whether this converter can convert the object to the specified type, using the specified context.

```
public override bool CanConvertTo(ITypeDescriptorContext context,
Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type you want to convert to.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type)

Converts the given value object to the specified type, using the specified context and culture information.

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo
culture, object value, Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If null is passed, the current culture is assumed.

`value` object

The object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

object

An object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is null.

NotSupportedException

The conversion cannot be performed.

Class Vector3TypeConverter

Namespace: Microsoft.Xna.Framework.Design

Assembly: MonoGame.Framework.dll

Provides a unified way of converting Vector3 values to other types, as well as for accessing standard values and subproperties.

```
public sealed class Vector3TypeConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← Vector3TypeConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) , TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,
TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesSupported() ,

TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

CanConvertTo(ITypeDescriptorContext, Type)

Returns whether this converter can convert the object to the specified type, using the specified context.

```
public override bool CanConvertTo(ITypeDescriptorContext context,
Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type you want to convert to.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type)

Converts the given value object to the specified type, using the specified context and culture information.

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo
culture, object value, Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If null is passed, the current culture is assumed.

`value` object

The object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

object

An object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is null.

NotSupportedException

The conversion cannot be performed.

Class Vector4TypeConverter

Namespace: Microsoft.Xna.Framework.Design

Assembly: MonoGame.Framework.dll

Provides a unified way of converting Vector4 values to other types, as well as for accessing standard values and subproperties.

```
public sealed class Vector4TypeConverter : TypeConverter
```

Inheritance

object ← TypeConverter ← Vector4TypeConverter

Inherited Members

TypeConverter.CanConvertFrom(Type) , TypeConverter.CanConvertTo(Type) ,
TypeConverter.ConvertFrom(object) ,
TypeConverter.ConvertFromInvariantString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromInvariantString(string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, CultureInfo, string) ,
TypeConverter.ConvertFromString(ITypeDescriptorContext, string) ,
TypeConverter.ConvertFromString(string) , TypeConverter.ConvertTo(object, Type) ,
TypeConverter.ConvertToInvariantString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToInvariantString(object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, CultureInfo, object) ,
TypeConverter.ConvertToString(ITypeDescriptorContext, object) ,
TypeConverter.ConvertToString(object) , TypeConverter.CreateInstance(IDictionary) ,
TypeConverter.CreateInstance(ITypeDescriptorContext, IDictionary) ,
TypeConverter.GetCreateInstanceSupported() ,
TypeConverter.GetCreateInstanceSupported(ITypeDescriptorContext) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object) ,
TypeConverter.GetProperties(ITypeDescriptorContext, object, Attribute[]) ,
TypeConverter.GetProperties(object) , TypeConverter.GetPropertiesSupported() ,
TypeConverter.GetPropertiesSupported(ITypeDescriptorContext) ,
TypeConverter.GetStandardValues() , TypeConverter.GetStandardValues(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesExclusive() ,
TypeConverter.GetStandardValuesExclusive(ITypeDescriptorContext) ,
TypeConverter.GetStandardValuesSupported() ,

TypeConverter.GetStandardValuesSupported(ITypeDescriptorContext) ,
TypeConverter.IsValid(ITypeDescriptorContext, object) , TypeConverter.IsValid(object) ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Methods

CanConvertFrom(ITypeDescriptorContext, Type)

Returns whether this converter can convert an object of the given type to the type of this converter, using the specified context.

```
public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`sourceType` Type

A Type that represents the type you want to convert from.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

CanConvertTo(ITypeDescriptorContext, Type)

Returns whether this converter can convert the object to the specified type, using the specified context.

```
public override bool CanConvertTo(ITypeDescriptorContext context,
Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`destinationType` Type

A Type that represents the type you want to convert to.

Returns

bool

true if this converter can perform the conversion; otherwise, false.

ConvertFrom(ITypeDescriptorContext, CultureInfo, object)

Converts the given object to the type of this converter, using the specified context and culture information.

```
public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

The CultureInfo to use as the current culture.

`value` object

The object to convert.

Returns

object

An object that represents the converted value.

Exceptions

NotSupportedException

The conversion cannot be performed.

ConvertTo(ITypeDescriptorContext, CultureInfo, object, Type)

Converts the given value object to the specified type, using the specified context and culture information.

```
public override object ConvertTo(ITypeDescriptorContext context, CultureInfo
culture, object value, Type destinationType)
```

Parameters

`context` ITypeDescriptorContext

An ITypeDescriptorContext that provides a format context.

`culture` CultureInfo

A CultureInfo. If null is passed, the current culture is assumed.

`value` object

The object to convert.

`destinationType` Type

The Type to convert the `value` parameter to.

Returns

object

An object that represents the converted value.

Exceptions

ArgumentNullException

The `destinationType` parameter is null.

NotSupportedException

The conversion cannot be performed.

Namespace Microsoft.Xna.Framework.Graphics

Classes

AlphaTestEffect

Built-in effect that supports alpha testing.

BasicEffect

Built-in effect that supports optional texturing, vertex coloring, fog, and lighting.

BlendState

Contains blend state for the device.

DefaultColorProcessors

Represents the default processors used in processing color values. This class cannot be inherited.

DepthStencilState

Contains depth-stencil state for the device.

DeviceLostException

The exception that is thrown when the device has been lost, but cannot be reset at this time. Therefore, rendering is not possible.

DeviceNotResetException

The exception that is thrown when the device has been lost, but can be reset at this time.

DirectionalLight

Represents a directional light. This class cannot be inherited.

DisplayMode

Describes the display mode.

DisplayModeCollection

A collection that manipulates DisplayMode instances.

DualTextureEffect

Built-in effect that supports two-layer multitexturing.

DynamicIndexBuffer

Describes the rendering order of the vertices in a vertex buffer. Use **DynamicIndexBuffer** for storing indices for dynamic vertices and IndexBuffer for indices of non-dynamic arrays.

DynamicVertexBuffer

Represents a list of 3D vertices to be streamed to the graphics device. Use **DynamicVertexBuffer** for dynamic vertex arrays and VertexBuffer for non-dynamic vertex arrays.

Effect

Used to set and query shader effects, and to choose techniques.

EffectAnnotation

Represents an annotation to an EffectParameter.

EffectAnnotationCollection

Represents a collection of EffectAnnotation objects.

EffectMaterial

Contains an effect subclass which is used to load data for an EffectMaterialContent type.

EffectParameter

Represents an Effect parameter.

EffectParameterCollection

Represents a collection of EffectParameter objects.

EffectPass

Contains the rendering state for drawing with an effect.

An effect can contain one or more passes.

EffectPassCollection

Represents a collection of EffectPass objects.

EffectTechnique

Represents a technique used in a shader effect.

EffectTechniqueCollection

Represents a collection of EffectTechnique objects.

EnvironmentMapEffect

Built-in effect that supports environment mapping.

GraphicsAdapter

Provides methods to retrieve and manipulate graphics adapters.

GraphicsDebug

Represents a class for debugging graphics.

GraphicsDebugMessage

Represents a debug message from the graphics subsystem.

GraphicsDevice

Performs primitive-based rendering, creates resources, handles system-level variables, adjusts gamma ramp levels, and creates shaders.

GraphicsResource

Queries and prepares resources.

IndexBuffer

Describes the rendering order of the vertices in a vertex buffer.

Model

A basic 3D model with per mesh parent bones.

ModelBone

Represents bone data for a model.

ModelBoneCollection

Represents a set of bones associated with a model.

ModelEffectCollection

Represents a collection of effects associated with a model.

ModelMesh

Represents a mesh that is part of a Model.

ModelMeshCollection

Represents a collection of ModelMesh objects.

ModelMeshPart

Represents a batch of geometry information to submit to the graphics device during rendering. Each **ModelMeshPart** is a subdivision of a ModelMesh object. The ModelMesh class is split into multiple **ModelMeshPart** objects, typically based on material information.

ModelMeshPartCollection

Represents a collection of ModelMeshPart objects.

NoSuitableGraphicsDeviceException

Thrown when no available graphics device fits the given device preferences.

OcclusionQuery

Used to perform an occlusion query against the latest drawn objects.

PresentationParameters

Contains graphics presentation parameters.

RasterizerState

Contains rasterizer state, which determines how to convert vector data (shapes) into raster data (pixels).

RenderTarget2D

Represents a 2D texture resource that will be written to at the end of a render pass.

RenderTargetCube

Represents a texture cube that can be used as a render target.

ResourceCreatedEventArgs

Provides data for the ResourceCreated event. This class cannot be inherited.

ResourceDestroyedEventArgs

Provides data for the ResourceDestroyed event. This class cannot be inherited.

SamplerState

Contains sampler state, which determines how to sample texture data.

SamplerStateCollection

Represents a collection of SamplerState objects,

ShaderCompilerException

This exception occurs when a shader compiled at runtime has errors. It should provide enough information to help diagnose the bug.

SkinnedEffect

Built-in effect for rendering skinned character models.

SpriteBatch

Helper class for drawing text strings and sprites in one or more optimized batches.

SpriteEffect

The default effect used by SpriteBatch.

SpriteFont

Represents a font texture.

TargetBlendState

Defines the blend state for a single render target.

Texture

Represents a texture resource

Texture2D
Represents a 2D grid of texels.

Texture3D
Represents a 3D volume of texels.

TextureCollection
Represents a collection of Texture objects. This class cannot be inherited.

TextureCube
Represents a set of six 2D textures, one for each face of a cube.

VertexBuffer
Represents a list of 3D vertices to be streamed to the graphics device.

VertexDeclaration
Defines per-vertex data of a vertex buffer.

Structs

EffectPassCollection.Enumerator
Enumerator to iterate through the EffectPassCollection

GraphicsMetrics
A snapshot of rendering statistics from Metrics to be used for runtime debugging and profiling.

ModelBoneCollection.Enumerator
Provides the ability to iterate through the bones in an ModelBoneCollection.

ModelEffectCollection.Enumerator
Enumerator to iterate through the ModelEffectCollection

ModelMeshCollection.Enumerator
Provides the ability to iterate through the bones in an ModelMeshCollection.

RenderTargetBinding
Represents the binding of a RenderTarget2D used in an array of bindings when setting multiple render targets on the graphics device.

SpriteFont.Glyph
Struct that defines the spacing, Kerning, and bounds of a character.

VertexBufferBinding

Defines how a vertex buffer is bound to the graphics device for rendering.

VertexElement

Defines a single element in a vertex.

VertexPosition

Describes a custom vertex format structure that contains position.

VertexPositionColor

Describes a custom vertex format structure that contains position and color.

VertexPositionColorNormal

Describes a custom vertex format structure that contains position, color and normal data.

VertexPositionColorNormalTexture

Describes a custom vertex format structure that contains position, color, normal data, and one set of texture coordinates.

VertexPositionColorTexture

Describes a custom vertex format structure that contains position, color, and one set of texture coordinates.

VertexPositionNormalTexture

Describes a custom vertex format structure that contains position, normal data, and one set of texture coordinates.

VertexPositionTexture

Describes a custom vertex format structure that contains position and one set of texture coordinates.

Viewport

Describes the view bounds for render-target surface.

Interfaces

IEffectBones

Interface for Effects that support bone transforms.

IEffectFog

The common effect fog rendering parameters.

IEffectLights

The common effect light rendering parameters.

IEffectMatrices

Gets or sets transformation matrix parameters for the current effect.

IGraphicsDeviceService

Provider of a GraphicsDevice.

IVertexType

Vertex type interface which is implemented by a custom vertex type structure.

Enums

Blend

Defines a blend mode.

BlendFunction

Defines a function for color blending.

BufferUsage

A usage hint for optimizing memory placement of graphics buffers.

ClearOptions

Defines the buffers for clearing when calling Clear(ClearOptions, Color, float, int) operation.

ColorWriteChannels

Defines the color channels for render target blending operations.

CompareFunction

The comparison function used for depth, stencil, and alpha tests.

CubeMapFace

Defines the faces in a cube map for the TextureCube class.

CullMode

Defines a culling mode for faces in rasterization process.

DepthFormat

Defines formats for depth-stencil buffer.

EffectParameterClass

Defines classes for effect parameters and shader constants.

EffectParameterType

Defines types for effect parameters and shader constants.

FillMode

Defines options for filling the primitive.

GraphicsAdapter.DriverType

Defines the driver type for graphics adapter. Usable only on DirectX platforms for now.

GraphicsDeviceStatus

Describes the status of the GraphicsDevice.

GraphicsProfile

Defines a set of graphic capabilities.

IndexElementSize

Defines size for index in IndexBuffer and DynamicIndexBuffer.

PresentInterval

Defines how Present() updates the game window.

PrimitiveType

Defines how vertex data is ordered.

RenderTargetUsage

Defines if the previous content in a render target is preserved when it set on the graphics device.

SetDataOptions

Defines how vertex or index buffer data will be flushed during a SetData operation.

ShaderStage

SpriteEffects

Defines sprite visual options for mirroring.

SpriteSortMode

Defines sprite sort rendering options.

StencilOperation

Defines stencil buffer operations.

SurfaceFormat

Defines types of surface formats.

Texture2D.SurfaceType

Defines the values for the surface type of a **Texture2D**

TextureAddressMode

Defines modes for addressing texels using texture coordinates that are outside of the range of 0.0 to 1.0.

TextureFilter

Defines filtering types for texture sampler.

TextureFilterMode

Represents the texture filtering mode used for texture sampling.

VertexElementFormat

Defines vertex element formats.

VertexElementUsage

Defines usage for vertex elements.

Class AlphaTestEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Built-in effect that supports alpha testing.

```
public class AlphaTestEffect : Effect, IDisposable, IEffectMatrices, IEffectFog
```

Inheritance

object ← GraphicsResource ← Effect ← AlphaTestEffect

Implements

IDisposable, IEffectMatrices, IEffectFog

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

AlphaTestEffect(AlphaTestEffect)

Creates a new AlphaTestEffect by cloning parameter settings from an existing instance.

```
protected AlphaTestEffect(AlphaTestEffect cloneSource)
```

Parameters

`cloneSource` AlphaTestEffect

AlphaTestEffect(GraphicsDevice)

Creates a new AlphaTestEffect with default parameter settings.

```
public AlphaTestEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

Properties

Alpha

Gets or sets the material alpha.

```
public float Alpha { get; set; }
```

Property Value

float

AlphaFunction

Gets or sets the alpha compare function (default Greater).

```
public CompareFunction AlphaFunction { get; set; }
```

Property Value

CompareFunction

DiffuseColor

Gets or sets the material diffuse color (range 0 to 1).

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

FogColor

Gets or sets the fog color.

```
public Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Gets or sets the fog enable flag.

```
public bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

Gets or sets the fog end distance.

```
public float FogEnd { get; set; }
```

Property Value

float

FogStart

Gets or sets the fog start distance.

```
public float FogStart { get; set; }
```

Property Value

float

Projection

Gets or sets the projection matrix.

```
public Matrix Projection { get; set; }
```

Property Value

Matrix

ReferenceAlpha

Gets or sets the reference alpha value (default 0).

```
public int ReferenceAlpha { get; set; }
```

Property Value

int

Texture

Gets or sets the current texture.

```
public Texture2D Texture { get; set; }
```

Property Value

Texture2D

VertexColorEnabled

Gets or sets whether vertex color is enabled.

```
public bool VertexColorEnabled { get; set; }
```

Property Value

bool

View

Gets or sets the view matrix.

```
public Matrix View { get; set; }
```

Property Value

Matrix

World

Gets or sets the world matrix.

```
public Matrix World { get; set; }
```

Property Value

Matrix

Methods

Clone()

Creates a clone of the current AlphaTestEffect instance.

```
public override Effect Clone()
```

Returns

Effect

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

Class BasicEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Built-in effect that supports optional texturing, vertex coloring, fog, and lighting.

```
public class BasicEffect : Effect, IDisposable, IEffectMatrices,
IEffectLights, IEffectFog
```

Inheritance

object ← GraphicsResource ← Effect ← BasicEffect

Implements

IDisposable, IEffectMatrices, IEffectLights, IEffectFog

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

BasicEffect(BasicEffect)

Creates a new BasicEffect by cloning parameter settings from an existing instance.

```
protected BasicEffect(BasicEffect cloneSource)
```

Parameters

`cloneSource` BasicEffect

BasicEffect(GraphicsDevice)

Creates a new BasicEffect with default parameter settings.

```
public BasicEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

Properties

Alpha

Gets or sets the material alpha.

```
public float Alpha { get; set; }
```

Property Value

float

AmbientLightColor

The floating point ambient light color.

```
public Vector3 AmbientLightColor { get; set; }
```

Property Value

Vector3

DiffuseColor

Gets or sets the material diffuse color (range 0 to 1).

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

DirectionalLight0

Returns the first directional light.

```
public DirectionalLight DirectionalLight0 { get; }
```

Property Value

DirectionalLight

DirectionalLight1

Returns the second directional light.

```
public DirectionalLight DirectionalLight1 { get; }
```

Property Value

DirectionalLight

DirectionalLight2

Returns the third directional light.

```
public DirectionalLight DirectionalLight2 { get; }
```

Property Value

DirectionalLight

EmissiveColor

Gets or sets the material emissive color (range 0 to 1).

```
public Vector3 EmissiveColor { get; set; }
```

Property Value

Vector3

FogColor

The floating point fog color.

```
public Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Used to toggle the rendering of fog.

```
public bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

The world space distance from the camera at which fogging is fully applied.

```
public float FogEnd { get; set; }
```

Property Value

float

Remarks

FogEnd should be greater than FogStart. If FogEnd and FogStart are the same value everything is fully fogged.

FogStart

The world space distance from the camera at which fogging begins.

```
public float FogStart { get; set; }
```

Property Value

float

Remarks

FogStart should be less than FogEnd. If FogEnd and FogStart are the same value everything is fully fogged.

LightingEnabled

Toggles the rendering of lighting.

```
public bool LightingEnabled { get; set; }
```

Property Value

bool

PreferPerPixelLighting

Gets or sets the per-pixel lighting prefer flag.

```
public bool PreferPerPixelLighting { get; set; }
```

Property Value

bool

Projection

Gets or sets the projection matrix.

```
public Matrix Projection { get; set; }
```

Property Value

Matrix

SpecularColor

Gets or sets the material specular color (range 0 to 1).

```
public Vector3 SpecularColor { get; set; }
```

Property Value

Vector3

SpecularPower

Gets or sets the material specular power.

```
public float SpecularPower { get; set; }
```

Property Value

float

Texture

Gets or sets the current texture.

```
public Texture2D Texture { get; set; }
```

Property Value

Texture2D

TextureEnabled

Gets or sets whether texturing is enabled.

```
public bool TextureEnabled { get; set; }
```

Property Value

bool

VertexColorEnabled

Gets or sets whether vertex color is enabled.

```
public bool VertexColorEnabled { get; set; }
```

Property Value

bool

View

Gets or sets the view matrix.

```
public Matrix View { get; set; }
```

Property Value

Matrix

World

Gets or sets the world matrix.

```
public Matrix World { get; set; }
```

Property Value

Matrix

Methods

Clone()

Creates a clone of the current BasicEffect instance.

```
public override Effect Clone()
```

Returns

Effect

EnableDefaultLighting()

Initializes the lights to the standard key/fill/back lighting rig.

```
public void EnableDefaultLighting()
```

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

Enum Blend

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines a blend mode.

```
public enum Blend
```

Fields

`BlendFactor = 10`

Each component of the color is multiplied by a constant in the BlendFactor.

`DestinationAlpha = 8`

Each component of the color is multiplied by the alpha value of the destination. {Ad, Ad, Ad, Ad}, where Ad is the destination alpha value.

`DestinationColor = 6`

Each component color is multiplied by the destination color. {Rd, Gd, Bd, Ad}, where Rd, Gd, Bd, Ad are color destination values.

`InverseBlendFactor = 11`

Each component of the color is multiplied by a inversed constant in the BlendFactor.

`InverseDestinationAlpha = 9`

Each component of the color is multiplied by the inversed alpha value of the destination. {1 − Ad, 1 − Ad, 1 − Ad, 1 − Ad}, where Ad is the destination alpha value.

`InverseDestinationColor = 7`

Each component of the color is multiplied by the inversed destination color. {1 − Rd, 1 − Gd, 1 − Bd, 1 − Ad}, where Rd, Gd, Bd, Ad are color destination values.

`InverseSourceAlpha = 5`

Each component of the color is multiplied by the inverse of the alpha value of the source. {1 − As, 1 − As, 1 − As, 1 − As}, where As is the source alpha value.

`InverseSourceColor = 3`

Each component of the color is multiplied by the inverse of the source color. {1 − Rs, 1 − Gs, 1 − Bs, 1 − As}, where Rs, Gs, Bs, As are color source values.

`One = 0`

Each component of the color is multiplied by {1, 1, 1, 1}.

`SourceAlpha = 4`

Each component of the color is multiplied by the alpha value of the source. {As, As, As, As}, where As is the source alpha value.

`SourceAlphaSaturation = 12`

Each component of the color is multiplied by either the alpha of the source color, or the inverse of the alpha of the source color, whichever is greater. {f, f, f, 1}, where f = min(As, 1 − As), where As is the source alpha value.

`SourceColor = 2`

Each component of the color is multiplied by the source color. {Rs, Gs, Bs, As}, where Rs, Gs, Bs, As are color source values.

`Zero = 1`

Each component of the color is multiplied by {0, 0, 0, 0}.

Enum BlendFunction

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines a function for color blending.

```
public enum BlendFunction
```

Fields

`Add = 0`

The function will adds destination to the source. (srcColor * srcBlend) + (destColor * destBlend)

`Max = 4`

The function will extracts maximum of the source and destination. max((srcColor * srcBlend), (destColor * destBlend))

`Min = 3`

The function will extracts minimum of the source and destination. min((srcColor * srcBlend),(destColor * destBlend))

`ReverseSubtract = 2`

The function will subtracts source from destination. (destColor * destBlend) - (srcColor * srcBlend)

`Subtract = 1`

The function will subtracts destination from source. (srcColor * srcBlend) − (destColor * destBlend)

Class BlendState

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains blend state for the device.

```
public class BlendState : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← BlendState

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

BlendState()

Initializes a new instance of the BlendState class with the default values, using additive color and alpha blending.

```
public BlendState()
```

Fields

Additive

A built-in state object with settings for additive blend that is adding the destination data to the source data without using alpha.

```
public static readonly BlendState Additive
```

Field Value

BlendState

Remarks

This built-in state object has the following settings:

Property	Value
ColorSourceBlend	SourceAlpha
AlphaSourceBlend	SourceAlpha
ColorDestinationBlend	One
AlphaDestinationBlend	One

AlphaBlend

A built-in state object with settings for alpha blend that is blending the source and destination data using alpha.

```
public static readonly BlendState AlphaBlend
```

Field Value

BlendState

Remarks

This built-in state object has the following settings:

Property	Value
ColorSourceBlend	One
AlphaSourceBlend	One
ColorDestinationBlend	InverseSourceAlpha
AlphaDestinationBlend	InverseSourceAlpha

NonPremultiplied

A built-in state object with settings for blending with non-premultipled alpha that is blending source and destination data by using alpha while assuming the color data contains no alpha information.

```
public static readonly BlendState NonPremultiplied
```

Field Value

BlendState

Remarks

This built-in state object has the following settings:

Property	Value
ColorSourceBlend	SourceAlpha
AlphaSourceBlend	SourceAlpha
ColorDestinationBlend	InverseSourceAlpha
AlphaDestinationBlend	InverseSourceAlpha

Opaque

A built-in state object with settings for opaque blend that is overwriting the source with the destination data.

```
public static readonly BlendState Opaque
```

Field Value

BlendState

Remarks

This built-in state object has the following settings:

Property	Value
ColorSourceBlend	One
AlphaSourceBlend	One
ColorDestinationBlend	Zero
AlphaDestinationBlend	Zero

Properties

AlphaBlendFunction

Gets or sets the arithmetic operation when blending alpha values. The default is Add.

```
public BlendFunction AlphaBlendFunction { get; set; }
```

Property Value

BlendFunction

A value from the BlendFunction enumeration.

AlphaDestinationBlend

Gets or sets the blend factor for the destination alpha, which is the percentage of the destination alpha included in the blended result. The default is One.

```
public Blend AlphaDestinationBlend { get; set; }
```

Property Value

Blend

A value from the Blend enumeration.

AlphaSourceBlend

Gets or sets the alpha blend factor. The default is One.

```
public Blend AlphaSourceBlend { get; set; }
```

Property Value

Blend

A value from the Blend enumeration.

BlendFactor

The color used as blend factor when alpha blending.

```
public Color BlendFactor { get; set; }
```

Property Value

Color

Remarks

BlendFactor is set to this value when this BlendState is bound to a GraphicsDevice.

ColorBlendFunction

Gets or sets the arithmetic operation when blending color values. The default is Add.

```
public BlendFunction ColorBlendFunction { get; set; }
```

Property Value

BlendFunction

A value from the BlendFunction enumeration.

ColorDestinationBlend

Gets or sets the blend factor for the destination color. The default is One.

```
public Blend ColorDestinationBlend { get; set; }
```

Property Value

Blend

A value from the Blend enumeration.

ColorSourceBlend

Gets or sets the blend factor for the source color. The default is One.

```
public Blend ColorSourceBlend { get; set; }
```

Property Value

Blend

A value from the Blend enumeration.

ColorWriteChannels

Gets or sets which color channels (RGBA) are enabled for writing during color blending. The default value is None.

```
public ColorWriteChannels ColorWriteChannels { get; set; }
```

Property Value

ColorWriteChannels

A value from the ColorWriteChannels enumeration.

ColorWriteChannels1

Gets or sets which color channels (RGBA) are enabled for writing during color blending. The default value is None.

```
public ColorWriteChannels ColorWriteChannels1 { get; set; }
```

Property Value

ColorWriteChannels

A value from the ColorWriteChannels enumeration.

ColorWriteChannels2

Gets or sets which color channels (RGBA) are enabled for writing during color blending. The default value is None.

```
public ColorWriteChannels ColorWriteChannels2 { get; set; }
```

Property Value

ColorWriteChannels

A value from the ColorWriteChannels enumeration.

ColorWriteChannels3

Gets or sets which color channels (RGBA) are enabled for writing during color blending. The default value is None.

```
public ColorWriteChannels ColorWriteChannels3 { get; set; }
```

Property Value

ColorWriteChannels

A value from the ColorWriteChannels enumeration.

IndependentBlendEnable

Enables use of the per-target blend states.

```
public bool IndependentBlendEnable { get; set; }
```

Property Value

bool

this[int]

Returns the target specific blend state.

```
public TargetBlendState this[int index] { get; }
```

Parameters

`index` int

The 0 to 3 target blend state index.

Property Value

TargetBlendState

A target blend state.

MultiSampleMask

Gets or sets a bitmask which defines which samples can be written during multisampling. The default is `0xffffffff`.

```
public int MultiSampleMask { get; set; }
```

Property Value

int

Methods

Dispose(bool)

The method that derived classes should override to implement disposing of managed and native resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if managed objects should be disposed.

Remarks

Native resources should always be released regardless of the value of the disposing parameter.

Enum BufferUsage

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

A usage hint for optimizing memory placement of graphics buffers.

```
public enum BufferUsage
```

Fields

`None = 0`

No special usage.

`WriteOnly = 1`

The buffer will not be readable and will be optimized for rendering and writing.

Enum ClearOptions

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the buffers for clearing when calling Clear(ClearOptions, Color, float, int) operation.

```
[Flags]
public enum ClearOptions
```

Fields

`DepthBuffer = 2`

Depth buffer.

`Stencil = 4`

Stencil buffer.

`Target = 1`

Color buffer.

Enum ColorWriteChannels

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the color channels for render target blending operations.

```
[Flags]
public enum ColorWriteChannels
```

Fields

`All = Red | Green | Blue | Alpha`

All channels selected.

`Alpha = 8`

Alpha channel selected.

`Blue = 4`

Blue channel selected.

`Green = 2`

Green channel selected.

`None = 0`

No channels selected.

`Red = 1`

Red channel selected.

Enum CompareFunction

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The comparison function used for depth, stencil, and alpha tests.

```
public enum CompareFunction
```

Fields

`Always = 0`

Always passes the test.

`Equal = 4`

Passes the test when the new pixel value is equal to current pixel value.

`Greater = 6`

Passes the test when the new pixel value is greater than current pixel value.

`GreaterEqual = 5`

Passes the test when the new pixel value is greater than or equal to current pixel value.

`Less = 2`

Passes the test when the new pixel value is less than current pixel value.

`LessEqual = 3`

Passes the test when the new pixel value is less than or equal to current pixel value.

`Never = 1`

Never passes the test.

`NotEqual = 7`

Passes the test when the new pixel value does not equal to current pixel value.

Enum CubeMapFace

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the faces in a cube map for the TextureCube class.

```
public enum CubeMapFace
```

Fields

`NegativeX = 1`

Negative X face in the cube map.

`NegativeY = 3`

Negative Y face in the cube map.

`NegativeZ = 5`

Negative Z face in the cube map.

`PositiveX = 0`

Positive X face in the cube map.

`PositiveY = 2`

Positive Y face in the cube map.

`PositiveZ = 4`

Positive Z face in the cube map.

Enum CullMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines a culling mode for faces in rasterization process.

```
public enum CullMode
```

Fields

`CullClockwiseFace = 1`

Cull faces with clockwise order.

`CullCounterClockwiseFace = 2`

Cull faces with counter clockwise order.

`None = 0`

Do not cull faces.

Class DefaultColorProcessors

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents the default processors used in processing color values. This class cannot be inherited.

```
public static class DefaultColorProcessors
```

Inheritance

object ← DefaultColorProcessors

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

PremultiplyAlpha

Premultiplies the RGB component value of each color by the its alpha component.

```
public static readonly Action<byte[]> PremultiplyAlpha
```

Field Value

Action<byte[]>

ZeroTransparentPixels

Sets the RGB component values of each color to zero if the alpha component is zero.

```
public static readonly Action<byte[]> ZeroTransparentPixels
```

Field Value

Action<byte[]>

Remarks

This is standard XNA behavior.

Enum DepthFormat

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines formats for depth-stencil buffer.

```
public enum DepthFormat
```

Fields

`Depth16 = 1`

16-bit depth buffer.

`Depth24 = 2`

24-bit depth buffer. Equivalent of Depth24Stencil8 for DirectX platforms.

`Depth24Stencil8 = 3`

32-bit depth-stencil buffer. Where 24-bit depth and 8-bit for stencil used.

`None = 0`

Depth-stencil buffer will not be created.

Class DepthStencilState

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains depth-stencil state for the device.

```
public class DepthStencilState : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← DepthStencilState

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

DepthStencilState()

Creates a new instance of the DepthStencilState class with default values.

```
public DepthStencilState()
```

Fields

Default

A built-in state object with default settings for using a depth stencil buffer.

```
public static readonly DepthStencilState Default
```

Field Value

DepthStencilState

Remarks

This built-in state object has the following settings:

```
DepthBufferEnable = true,
DepthBufferWriteEnable = true
```

DepthRead

A built-int state object with settings for enabling a read-only depth stencil buffer.

```
public static readonly DepthStencilState DepthRead
```

Field Value

DepthStencilState

Remarks

This built-in state object has the following settings:

```
DepthBufferEnable = true,
DepthBufferWriteEnable = false
```

None

A built in state object with settings for not using a depth stencil buffer.

```
public static readonly DepthStencilState None
```

Field Value

DepthStencilState

Remarks

This built-int state object has the following settings:

```
DepthBufferEnable = false,
DepthBufferWriteEnable = false
```

Properties

CounterClockwiseStencilDepthBufferFail

Gets or Sets the stencil operation to perform if the stencil test passes and the depth-buffer test fails for a counterclockwise triangle. The default value is StencilOperation.Keep.

```
public StencilOperation CounterClockwiseStencilDepthBufferFail { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

CounterClockwiseStencilFail

Gets or Sets the stencil operation to perform if the stencil test fails for the counterclockwise triangle. The default value is StencilOperation.Keep.

```
public StencilOperation CounterClockwiseStencilFail { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

CounterClockwiseStencilFunction

Gets or Sets the comparison function to use for counterclockwise stencil tests. The default value is CompareFunction.Always

```
public CompareFunction CounterClockwiseStencilFunction { get; set; }
```

Property Value

CompareFunction

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

CounterClockwiseStencilPass

Gets or Sets the stencil operation to perform if the stencil and depth-tests pass for a counterclockwise triangle. The default value is StencilOperation.Keep.

```
public StencilOperation CounterClockwiseStencilPass { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

DepthBufferEnable

Gets or Sets a value that indicates if depth buffering is enabled. The default value is true.

```
public bool DepthBufferEnable { get; set; }
```

Property Value

bool

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

DepthBufferFunction

Gets or Sets the comparison function for the depth-buffer test. The default value is CompareFunction.LessEqual.

```
public CompareFunction DepthBufferFunction { get; set; }
```

Property Value

CompareFunction

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

DepthBufferWriteEnable

Gets or Sets a value that indicates if writing to the depth buffer is enabled. The default value is true.

```
public bool DepthBufferWriteEnable { get; set; }
```

Property Value

bool

Remarks

This property enables an application to prevent the system from updating the depth buffer with new values.

if false, depth comparisons are still made according to the render state DepthBufferFunction, assuming that depth buffering is taking place, but depth values are not written to the buffer.

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

ReferenceStencil

Gets or Sets a reference value to use for the stencil test. The default is 0.

```
public int ReferenceStencil { get; set; }
```

Property Value

int

Remarks

The reference value is compared, by the comparison function specified by the StencilFunction property, to the stencil buffer entry of a pixel. This can be illustrated by a simple equation:

```
ReferenceStencil StencilFunction (stencil buffer entry)
```

This comparison applies only to the bits in the reference value and stencil buffer entry that are set in the stencil mask by this property. If the comparison is true, the stencil test passes and the pass operation (specified by the StencilPass property) is performed.

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilDepthBufferFail

Gets or Sets the stencil operation to perform if the stencil test passes and the depth-test fails. The default is StencilOperation.Keep.

```
public StencilOperation StencilDepthBufferFail { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilEnable

Gets or Sets a value that indicates whether stenciling is enabled. The default is false.

```
public bool StencilEnable { get; set; }
```

Property Value

bool

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilFail

Gets or Sets the stencil operation to perform if the stencil test fails. The default is StencilOperation.Keep.

```
public StencilOperation StencilFail { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilFunction

Gets or Sets the comparison function for the stencil test. The default is CompareFunction.Always.

```
public CompareFunction StencilFunction { get; set; }
```

Property Value

CompareFunction

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilMask

Gets or Sets the mask applied to the reference value and each stencil buffer entry to determine the significant bits for the stencil test. The default mask is Int32.MaxValue.

```
public int StencilMask { get; set; }
```

Property Value

int

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilPass

Gets or Sets the stencil operation to perform if the stencil test passes. The default value is StencilOperation.Keep.

```
public StencilOperation StencilPass { get; set; }
```

Property Value

StencilOperation

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

StencilWriteMask

Gets or Sets the write mask applied to values written into the stencil buffer. The default mask is Int32.MaxValue.

```
public int StencilWriteMask { get; set; }
```

Property Value

int

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

TwoSidedStencilMode

Gets or Sets a value that indicates whether two-sided stenciling is enabled. The default value is false.

```
public bool TwoSidedStencilMode { get; set; }
```

Property Value

bool

Exceptions

InvalidOperationException

When setting this value for one of the default DepthStencilState instances; Default, DepthRead, or None.

-or-

When setting this value after this DepthStencilState has already been bound to the graphics device.

Methods

Dispose(bool)

The method that derived classes should override to implement disposing of managed and native resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if managed objects should be disposed.

Remarks

Native resources should always be released regardless of the value of the disposing parameter.

Class DeviceLostException

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The exception that is thrown when the device has been lost, but cannot be reset at this time. Therefore, rendering is not possible.

```
[DataContract]
public sealed class DeviceLostException : Exception, ISerializable
```

Inheritance

object ← Exception ← DeviceLostException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

DeviceLostException()

Initializes a new instance of the DeviceLostException class.

```
public DeviceLostException()
```

DeviceLostException(string)

Initializes a new instance of the DeviceLostException class with a specified error message.

```
public DeviceLostException(string message)
```

Parameters

`message` string

The message that describes the error.

DeviceLostException(string, Exception)

Initializes a new instance of the DeviceLostException with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public DeviceLostException(string message, Exception inner)
```

Parameters

`message` string

The message that describes the error.

`inner` Exception

The exception that is the cause of the current exception, or a null reference if no inner exception is specified.

Class DeviceNotResetException

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The exception that is thrown when the device has been lost, but can be reset at this time.

```
[DataContract]
public sealed class DeviceNotResetException : Exception, ISerializable
```

Inheritance

object ← Exception ← DeviceNotResetException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

DeviceNotResetException()

Initializes a new instance of the DeviceNotResetException class.

```
public DeviceNotResetException()
```

DeviceNotResetException(string)

Initializes a new instance of the DeviceNotResetException class with a specified error message.

```
public DeviceNotResetException(string message)
```

Parameters

`message` string

The message that describes the error.

DeviceNotResetException(string, Exception)

Initializes a new instance of the DeviceNotResetException with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public DeviceNotResetException(string message, Exception inner)
```

Parameters

`message` string

The message that describes the error.

`inner` Exception

The exception that is the cause of the current exception, or a null reference if no inner exception is specified.

Class DirectionalLight

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a directional light. This class cannot be inherited.

```
public sealed class DirectionalLight
```

Inheritance

object ← DirectionalLight

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DirectionalLight(EffectParameter, EffectParameter, EffectParameter, DirectionalLight)

Creates a new instance of the DirectionalLight class with or without a copy of a DirectionalLight instance.

```
public DirectionalLight(EffectParameter directionParameter, EffectParameter
diffuseColorParameter, EffectParameter specularColorParameter,
DirectionalLight cloneSource)
```

Parameters

`directionParameter` EffectParameter

The light direction.

`diffuseColorParameter` EffectParameter

The diffuse color.

`specularColorParameter` EffectParameter

The specular color.

`cloneSource` DirectionalLight

The cloned source to copy from.

Remarks

The initial parameter values are either copied from the cloned object or set to default values (if the coned object is null). The three Effect Parameters are updated whenever the direction, diffuse color, or specular color properties are changed; or you can set these to null if you are using the cloned object.

Properties

DiffuseColor

Gets or Sets the diffuse color of the light.

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

Direction

Gets or Sets the light direction.

```
public Vector3 Direction { get; set; }
```

Property Value

Vector3

Remarks

This value must be a unit vector.

Enabled

Gets or Sets a value indicating whether light is enabled.

```
public bool Enabled { get; set; }
```

Property Value

bool

SpecularColor

Gets or Sets the specular color of the light.

```
public Vector3 SpecularColor { get; set; }
```

Property Value

Vector3

Class DisplayMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes the display mode.

```
[DataContract]
public class DisplayMode
```

Inheritance

object ← DisplayMode

Inherited Members

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

AspectRatio

Gets a value indicating the aspect ratio of the display mode

```
public float AspectRatio { get; }
```

Property Value

float

Format

Gets a value indicating the surface format of the display mode.

```
public SurfaceFormat Format { get; }
```

Property Value

SurfaceFormat

Height

Gets a value indicating the screen height, in pixels.

```
public int Height { get; }
```

Property Value

int

TitleSafeArea

Gets the bounds of the display that is guaranteed to be visible by the users screen.

```
public Rectangle TitleSafeArea { get; }
```

Property Value

Rectangle

Width

Gets a value indicating the screen width, in pixels.

```
public int Width { get; }
```

Property Value

int

Methods

Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to the current object; otherwise, false.

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

int

A hash code for the current object.

ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

string

A string that represents the current object.

Operators

operator ==(DisplayMode, DisplayMode)

Compares the current instance of the DisplayMode class to another instance to determine whether they are equal.

```
public static bool operator ==(DisplayMode left, DisplayMode right)
```

Parameters

`left` DisplayMode

The DisplayMode object on the left of the equality operator.

`right` DisplayMode

The DisplayMode object on the right of the equality operator.

Returns

bool

true if the objects are equal; otherwise, false.

operator !=(DisplayMode, DisplayMode)

Compares the current instance of the DisplayMode class to another instance to determine whether they are different.

```
public static bool operator !=(DisplayMode left, DisplayMode right)
```

Parameters

`left` DisplayMode

The DisplayMode object on the left of the inequality operator.

`right` DisplayMode

The DisplayMode object on the right of the inequality operator.

Returns

bool

true if the objects are different; otherwise, false.

Class DisplayModeCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

A collection that manipulates DisplayMode instances.

```
public class DisplayModeCollection : IEnumerable<DisplayMode>, IEnumerable
```

Inheritance

object ← DisplayModeCollection

Implements

IEnumerable<DisplayMode>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

this[SurfaceFormat]

Gets the DisplayMode instance with the specified format.

```
public IEnumerable<DisplayMode> this[SurfaceFormat format] { get; }
```

Parameters

`format` SurfaceFormat

Property Value

IEnumerable<DisplayMode>

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<DisplayMode> GetEnumerator()
```

Returns

IEnumerator<DisplayMode>

An enumerator that can be used to iterate through the collection.

Class DualTextureEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Built-in effect that supports two-layer multitexturing.

```
public class DualTextureEffect : Effect, IDisposable, IEffectMatrices, IEffectFog
```

Inheritance

object ← GraphicsResource ← Effect ← DualTextureEffect

Implements

IDisposable, IEffectMatrices, IEffectFog

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

DualTextureEffect(DualTextureEffect)

Creates a new DualTextureEffect by cloning parameter settings from an existing instance.

```
protected DualTextureEffect(DualTextureEffect cloneSource)
```

Parameters

`cloneSource` DualTextureEffect

DualTextureEffect(GraphicsDevice)

Creates a new DualTextureEffect with default parameter settings.

```
public DualTextureEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

Properties

Alpha

Gets or sets the material alpha.

```
public float Alpha { get; set; }
```

Property Value

float

DiffuseColor

Gets or sets the material diffuse color (range 0 to 1).

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

FogColor

Gets or sets the fog color.

```
public Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Gets or sets the fog enable flag.

```
public bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

Gets or sets the fog end distance.

```
public float FogEnd { get; set; }
```

Property Value

float

FogStart

Gets or sets the fog start distance.

```
public float FogStart { get; set; }
```

Property Value

float

Projection

Gets or sets the projection matrix.

```
public Matrix Projection { get; set; }
```

Property Value

Matrix

Texture

Gets or sets the current base texture.

```
public Texture2D Texture { get; set; }
```

Property Value

Texture2D

Texture2

Gets or sets the current overlay texture.

```
public Texture2D Texture2 { get; set; }
```

Property Value

Texture2D

VertexColorEnabled

Gets or sets whether vertex color is enabled.

```
public bool VertexColorEnabled { get; set; }
```

Property Value

bool

View

Gets or sets the view matrix.

```
public Matrix View { get; set; }
```

Property Value

Matrix

World

Gets or sets the world matrix.

```
public Matrix World { get; set; }
```

Property Value

Matrix

Methods

Clone()

Creates a clone of the current DualTextureEffect instance.

```
public override Effect Clone()
```

Returns

Effect

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

Class DynamicIndexBuffer

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes the rendering order of the vertices in a vertex buffer. Use **DynamicIndexBuffer** for storing indices for dynamic vertices and IndexBuffer for indices of non-dynamic arrays.

```
public class DynamicIndexBuffer : IndexBuffer, IDisposable
```

Inheritance

object ← GraphicsResource ← IndexBuffer ← DynamicIndexBuffer

Implements

IDisposable

Inherited Members

IndexBuffer.BufferUsage , IndexBuffer.IndexCount , IndexBuffer.IndexElementSize , IndexBuffer.GraphicsDeviceResetting() , IndexBuffer.GetData<T>(int, T[], int, int) , IndexBuffer.GetData<T>(T[], int, int) , IndexBuffer.GetData<T>(T[]) , IndexBuffer.SetData<T>(int, T[], int, int) , IndexBuffer.SetData<T>(T[], int, int) , IndexBuffer.SetData<T>(T[]) , IndexBuffer.SetDataInternal<T>(int, T[], int, int, SetDataOptions) , IndexBuffer.Dispose(bool) , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

For more information on drawing with dynamic buffers, see DynamicVertexBuffer Class

Constructors

DynamicIndexBuffer(GraphicsDevice, IndexElementSize, int,

BufferUsage)

Initializes a new instance of **DynamicIndexBuffer** with the specified parameters.

```
public DynamicIndexBuffer(GraphicsDevice graphicsDevice, IndexElementSize
indexElementSize, int indexCount, BufferUsage usage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexElementSize` IndexElementSize

The size, in bits, of an index element.

`indexCount` int

Number of indices in the buffer.

`usage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentOutOfRangeException

One of the following conditions is true

- The `indexCount` parameter is invalid. Index buffers can only be created for indices which are sixteen or thirty-two bits in length
- The `indexElementSize` parameter is invalid. The resource size must be greater than zero.
- The `indexElementSize` parameter is invalid. The total size of the index buffer must be an even multiple of the index element size (either 16 or 32 bits).

DynamicIndexBuffer(GraphicsDevice, Type, int, BufferUsage)

Initializes a new instance of **DynamicIndexBuffer** with the specified parameters.

```
public DynamicIndexBuffer(GraphicsDevice graphicsDevice, Type indexType, int
indexCount, BufferUsage usage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexType` Type

Type to use for index values.

`indexCount` int

Number of indices in the buffer.

`usage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentOutOfRangeException

The `indexCount` parameter is invalid. It must be greater than zero, and the index must be sixteen or thirty-two bits in length. Index buffers can only be created for indices which are sixteen or thirty-two bits in length.

InvalidOperationException

The resource could not be created

Properties

IsContentLost

Gets a value that indicates whether the index buffer data has been lost due to a lost device event.

```
public bool IsContentLost { get; }
```

Property Value

bool

Methods

SetData<T>(int, T[], int, int, SetDataOptions)

Copies array data to the index buffer.

```
public void SetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, SetDataOptions options) where T : struct
```

Parameters

`offsetInBytes` int

Number of bytes into the index buffer where copying will start.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

`options` SetDataOptions

Specifies whether existing data in the buffer will be kept after this operation.

Type Parameters

`T`

The type of elements in the array.

Remarks

An InvalidOperationException is thrown if an attempt is made to modify (for example, calls to the **SetData** method) a resource that is currently set on a graphics device.

SetData<T>(T[], int, int, SetDataOptions)

Copies array data to the index buffer.

```
public void SetData<T>(T[] data, int startIndex, int elementCount, SetDataOptions
options) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

`options` SetDataOptions

Specifies whether existing data in the buffer will be kept after this operation.

Type Parameters

`T`

The type of elements in the array.

Remarks

An InvalidOperationException is thrown if an attempt is made to modify (for example, calls to the **SetData** method) a resource that is currently set on a graphics device.

Class DynamicVertexBuffer

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a list of 3D vertices to be streamed to the graphics device. Use **DynamicVertexBuffer** for dynamic vertex arrays and VertexBuffer for non-dynamic vertex arrays.

```
public class DynamicVertexBuffer : VertexBuffer, IDisposable
```

Inheritance

object ← GraphicsResource ← VertexBuffer ← DynamicVertexBuffer

Implements

IDisposable

Inherited Members

VertexBuffer.VertexCount , VertexBuffer.VertexDeclaration , VertexBuffer.BufferUsage , VertexBuffer.GraphicsDeviceResetting() , VertexBuffer.GetData<T>(int, T[], int, int, int) , VertexBuffer.GetData<T>(T[], int, int) , VertexBuffer.GetData<T>(T[]) , VertexBuffer.SetData<T>(int, T[], int, int, int) , VertexBuffer.SetData<T>(T[], int, int) , VertexBuffer.SetData<T>(T[]) , VertexBuffer.SetDataInternal<T>(int, T[], int, int, int, SetDataOptions) , VertexBuffer.Dispose(bool) , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

In situations where your game frequently modifies a vertex buffer, it is recommended that the buffer be instantiated or derived from **DynamicVertexBuffer** instead of the VertexBuffer class. **DynamicVertexBuffer** is optimized for frequent vertex data modification.

Constructors

DynamicVertexBuffer(GraphicsDevice, VertexDeclaration, int, BufferUsage)

Initializes a new instance of **DynamicVertexBuffer** with the specified parameters

```
public DynamicVertexBuffer(GraphicsDevice graphicsDevice, VertexDeclaration
vertexDeclaration, int vertexCount, BufferUsage bufferUsage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the vertex buffer.

`vertexDeclaration` VertexDeclaration

The vertex declaration, which describes per-vertex data.

`vertexCount` int

The number of vertices in this vertex buffer.

`bufferUsage` BufferUsage

A set of options identifying the behaviors of this vertex buffer resource.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentOutOfRangeException

`vertexCount` must be greater than zero.

InvalidOperationException

This resource could not be created.

DynamicVertexBuffer(GraphicsDevice, Type, int, BufferUsage)

Initializes a new instance of **DynamicVertexBuffer** with the specified parameters

```
public DynamicVertexBuffer(GraphicsDevice graphicsDevice, Type type, int
vertexCount, BufferUsage bufferUsage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the vertex buffer.

`type` Type

The type used that describes the per-vertex data. Must derive from IVertexType.

`vertexCount` int

The number of vertices in this vertex buffer.

`bufferUsage` BufferUsage

A set of options identifying the behaviors of this vertex buffer resource.

Exceptions

ArgumentException

`type` /// One of the following conditions is true:

- The `type` parameter is not a value type.
- The `type` parameter does not derive from IVertexType.

ArgumentNullException

One of the following conditions is true:

- The `graphicsDevice` parameter is null.
- The `type` parameter is null.

ArgumentOutOfRangeException

The `vertexCount` parameter must be greater than zero.

Exception

A new VertexDeclaration instance cannot be created from the type provided by the `type` parameter.

InvalidOperationException

This resource could not be created.

Properties

IsContentLost

Gets a value that indicates if the vertex buffer data has been lost due to a lost device event.

```
[Obsolete("This is provided for XNA compatibility only and will always
return false")]
public bool IsContentLost { get; }
```

Property Value

bool

Remarks

This property will always return **false**. It is included for XNA compatibility.

Methods

SetData<T>(int, T[], int, int, int, SetDataOptions)

Copies array data to the vertex buffer.

```
public void SetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, int vertexStride, SetDataOptions options) where T : struct
```

Parameters

`offsetInBytes` int

Number of bytes into the vertex buffer where copying will start.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

`vertexStride` int

The size, in bytes, of the elements in the vertex buffer.

`options` SetDataOptions

Specifies whether existing data in the buffer will be kept after this operation.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

The `data` is null.

ArgumentOutOfRangeException

One of the following conditions is true:

- The `vertexStride` parameter larger than the vertex buffer size.
- The `vertexStride` parameter is less than the size of the specified data.
- The `data` parameter is not the correct size for the amount of data requested.

SetData<T>(T[], int, int, SetDataOptions)

Copies array data to the vertex buffer.

```
public void SetData<T>(T[] data, int startIndex, int elementCount, SetDataOptions
options) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

`options` SetDataOptions

Specifies whether existing data in the buffer will be kept after this operation.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

The `data` is null.

ArgumentOutOfRangeException

The `data` parameter is not the correct size for the amount of data requested.

Class Effect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Used to set and query shader effects, and to choose techniques.

```
public class Effect : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← Effect

Implements

IDisposable

Derived

AlphaTestEffect, BasicEffect, DualTextureEffect, EffectMaterial, EnvironmentMapEffect, SkinnedEffect, SpriteEffect

Inherited Members

GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Effect(Effect)

Creates a clone of the Effect.

```
protected Effect(Effect cloneSource)
```

Parameters

`cloneSource` Effect

Effect to clone.

Effect(GraphicsDevice, byte[])

Creates a new instance of Effect.

```
public Effect(GraphicsDevice graphicsDevice, byte[] effectCode)
```

Parameters

`graphicsDevice` GraphicsDevice

Graphics device

`effectCode` byte[]

The effect code.

Exceptions

ArgumentException

This `effectCode` is invalid.

Effect(GraphicsDevice, byte[], int, int)

Creates a new instance of Effect.

```
public Effect(GraphicsDevice graphicsDevice, byte[] effectCode, int index,
int count)
```

Parameters

`graphicsDevice` GraphicsDevice

Graphics device

`effectCode` byte[]

The effect code.

`index` int

`count` int

Exceptions

ArgumentException

This `effectCode` is invalid.

Properties

CurrentTechnique

Gets or sets the active technique.

```
public EffectTechnique CurrentTechnique { get; set; }
```

Property Value

EffectTechnique

Remarks

If there are multiple techiques in an effect and you want to use a new technique in the next pass, you must set **CurrentTechnique** to the new technique before making the rendering pass.

Parameters

Gets a collection of shader parameters used for this effect.

```
public EffectParameterCollection Parameters { get; }
```

Property Value

EffectParameterCollection

Techniques

Gets a collection of shader techniques that are defined for this effect.

```
public EffectTechniqueCollection Techniques { get; }
```

Property Value

EffectTechniqueCollection

Methods

Clone()

Returns a deep copy of the effect where immutable types are shared and mutable data is duplicated.

```
public virtual Effect Clone()
```

Returns

Effect

The cloned effect.

Remarks

See "Cloning an Effect" in MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/ff476138(v=vs.85).aspx

Dispose(bool)

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GraphicsDeviceResetting()

The GraphicsDevice is resetting, so GPU resources must be recreated.

```
protected override void GraphicsDeviceResetting()
```

OnApply()

Applies the effect state just prior to rendering the effect.

```
protected virtual void OnApply()
```

Class EffectAnnotation

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents an annotation to an EffectParameter.

```
public class EffectAnnotation
```

Inheritance

object ← EffectAnnotation

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ColumnCount

Gets the number of columns in this effect annotation.

```
public int ColumnCount { get; }
```

Property Value

int

Name

Gets the name of the effect annotation.

```
public string Name { get; }
```

Property Value

string

ParameterClass

Gets the parameter class of this effect annotation.

```
public EffectParameterClass ParameterClass { get; }
```

Property Value

EffectParameterClass

ParameterType

Gets the parameter type of this effect annotation.

```
public EffectParameterType ParameterType { get; }
```

Property Value

EffectParameterType

RowCount

Gets the row count of this effect annotation.

```
public int RowCount { get; }
```

Property Value

int

Semantic

Gets the semantic of this effect annotation.

```
public string Semantic { get; }
```

Property Value

string

Class EffectAnnotationCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of EffectAnnotation objects.

```
public class EffectAnnotationCollection : IEnumerable<EffectAnnotation>, IEnumerable
```

Inheritance

object ← EffectAnnotationCollection

Implements

IEnumerable<EffectAnnotation>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of elements contained in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Retrieves the EffectAnnotation at the specified index in the collection.

```
public EffectAnnotation this[int index] { get; }
```

Parameters

`index` int

Property Value

EffectAnnotation

this[string]

Retrieves a EffectAnnotation from the collection, given the name of the annotation.

```
public EffectAnnotation this[string name] { get; }
```

Parameters

`name` string

The name of the annotation to retrieve.

Property Value

EffectAnnotation

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<EffectAnnotation> GetEnumerator()
```

Returns

IEnumerator<EffectAnnotation>

An enumerator that can be used to iterate through the collection.

Class EffectMaterial

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains an effect subclass which is used to load data for an EffectMaterialContent type.

```
public class EffectMaterial : Effect, IDisposable
```

Inheritance

object ← GraphicsResource ← Effect ← EffectMaterial

Implements

IDisposable

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Clone() , Effect.OnApply() , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

For most purposes, this type can be ignored, and treated exactly like a regular effect. When an EffectMaterial type is loaded from .xnb format, its parameter values and textures are also loaded and automatically set on the effect, in addition to the HLSL shader code. Use this class to write a content pipeline extension to store materials inside a custom data type.

Constructors

EffectMaterial(Effect)

Creates a new instance of EffectMaterial.

```
public EffectMaterial(Effect cloneSource)
```

Parameters

`cloneSource` Effect

An instance of an object to copy initialization data from.

Class EffectParameter

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents an Effect parameter.

```
public class EffectParameter
```

Inheritance

object ← EffectParameter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

1. Create a **EffectParameter** for each parameter in your Effect that you will be setting in Draw(GameTime) or Update(GameTime).

```
public EffectParameter mWorld;
public EffectParameter mCameraView;
public EffectParameter CameraPos;
public EffectParameter mCameraProj;
```

2. Assign an Effect parameter to your **EffectParameter**.

```
mWorld = effect.Parameters["g_mWorld"];
mCameraView = effect.Parameters["g_mCameraView"];
CameraPos = effect.Parameters["g_CameraPos"];
mCameraProj = effect.Parameters["g_mCameraProj"];
```

3. Call **SetValue()** on your **EffectParameter** to change the parameter value.

Remarks

Creating and assigning a **EffectParameter** instance for each technique in your Effect is significantly faster than using the Parameters indexed property on Effect.

Properties

Annotations

Gets the collection of EffectAnnotation objects for this parameter.

```
public EffectAnnotationCollection Annotations { get; }
```

Property Value

EffectAnnotationCollection

ColumnCount

Gets the number of columns in the parameter description.

```
public int ColumnCount { get; }
```

Property Value

int

Elements

Gets the collection of effect parameters.

```
public EffectParameterCollection Elements { get; }
```

Property Value

EffectParameterCollection

Name

Gets the name of the parameter.

```
public string Name { get; }
```

Property Value

string

ParameterClass

Gets the class of the parameter.

```
public EffectParameterClass ParameterClass { get; }
```

Property Value

EffectParameterClass

ParameterType

Gets the type of the parameter.

```
public EffectParameterType ParameterType { get; }
```

Property Value

EffectParameterType

RowCount

Gets the number of rows in the parameter description.

```
public int RowCount { get; }
```

Property Value

int

Semantic

Gets the semantic meaning, or usage, of the parameter.

```
public string Semantic { get; }
```

Property Value

string

StructureMembers

Gets the collection of structure members.

```
public EffectParameterCollection StructureMembers { get; }
```

Property Value

EffectParameterCollection

Methods

GetValueBoolean()

Gets the value of the EffectParameter as a bool.

```
public bool GetValueBoolean()
```

Returns

bool

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type bool.

GetValueInt32()

Gets the value of the EffectParameter as an int.

```
public int GetValueInt32()
```

Returns

int

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type int.

GetValueInt32Array()

Gets the value of the EffectParameter as an array of int.

```
public int[] GetValueInt32Array()
```

Returns

int[]

GetValueMatrix()

Gets the value of the EffectParameter as an Matrix.

```
public Matrix GetValueMatrix()
```

Returns

Matrix

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Matrix.

GetValueMatrixArray(int)

Gets the value of the EffectParameter as an array of Matrix.

```
public Matrix[] GetValueMatrixArray(int count)
```

Parameters

`count` int

The number of elements in the array.

Returns

Matrix[]

GetValueQuaternion()

Gets the value of the EffectParameter as an Quaternion.

```
public Quaternion GetValueQuaternion()
```

Returns

Quaternion

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Quaternion.

GetValueSingle()

Gets the value of the EffectParameter as an float.

```
public float GetValueSingle()
```

Returns

float

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type float.

GetValueSingleArray()

Gets the value of the EffectParameter as an array of float.

```
public float[] GetValueSingleArray()
```

Returns

float[]

GetValueString()

Gets the value of the EffectParameter as an string.

```
public string GetValueString()
```

Returns

string

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type string.

GetValueTexture2D()

Gets the value of the EffectParameter as an Texture2D.

```
public Texture2D GetValueTexture2D()
```

Returns

Texture2D

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Texture2D.

GetValueTexture3D()

Gets the value of the EffectParameter as an Texture3D.

```
public Texture3D GetValueTexture3D()
```

Returns

Texture3D

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Texture3D.

GetValueTextureCube()

Gets the value of the EffectParameter as an TextureCube.

```
public TextureCube GetValueTextureCube()
```

Returns

TextureCube

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type TextureCube.

GetValueVector2()

Gets the value of the EffectParameter as an Vector2.

```
public Vector2 GetValueVector2()
```

Returns

Vector2

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector2.

GetValueVector2Array()

Gets the value of the EffectParameter as an array of Vector2.

```
public Vector2[] GetValueVector2Array()
```

Returns

Vector2[]

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector2.

GetValueVector3()

Gets the value of the EffectParameter as an Vector3.

```
public Vector3 GetValueVector3()
```

Returns

Vector3

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector3.

GetValueVector3Array()

Gets the value of the EffectParameter as an array of Vector3.

```
public Vector3[] GetValueVector3Array()
```

Returns

Vector3[]

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector3.

GetValueVector4()

Gets the value of the EffectParameter as an Vector4.

```
public Vector4 GetValueVector4()
```

Returns

Vector4

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector4.

GetValueVector4Array()

Gets the value of the EffectParameter as an array of Vector4.

```
public Vector4[] GetValueVector4Array()
```

Returns

Vector4[]

Exceptions

InvalidCastException

Unable to cast this EffectParameter to type Vector4.

SetValue(Texture)

Sets the value of the EffectParameter.

```
public void SetValue(Texture value)
```

Parameters

`value` Texture

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Matrix)

Sets the value of the EffectParameter.

```
public void SetValue(Matrix value)
```

Parameters

`value` Matrix

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Matrix[])

Sets the value of the EffectParameter.

```
public void SetValue(Matrix[] value)
```

Parameters

`value` Matrix[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Quaternion)

Sets the value of the EffectParameter.

```
public void SetValue(Quaternion value)
```

Parameters

`value` Quaternion

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector2)

Sets the value of the EffectParameter.

```
public void SetValue(Vector2 value)
```

Parameters

`value` Vector2

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector2[])

Sets the value of the EffectParameter.

```
public void SetValue(Vector2[] value)
```

Parameters

`value` Vector2[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector3)

Sets the value of the EffectParameter.

```
public void SetValue(Vector3 value)
```

Parameters

`value` Vector3

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector3[])

Sets the value of the EffectParameter.

```
public void SetValue(Vector3[] value)
```

Parameters

`value` Vector3[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector4)

Sets the value of the EffectParameter.

```
public void SetValue(Vector4 value)
```

Parameters

`value` Vector4

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(Vector4[])

Sets the value of the EffectParameter.

```
public void SetValue(Vector4[] value)
```

Parameters

`value` Vector4[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(bool)

Sets the value of the EffectParameter.

```
public void SetValue(bool value)
```

Parameters

`value` bool

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(int)

Sets the value of the EffectParameter.

```
public void SetValue(int value)
```

Parameters

`value` int

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(int[])

Sets the value of the EffectParameter.

```
public void SetValue(int[] value)
```

Parameters

`value` int[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(float)

Sets the value of the EffectParameter.

```
public void SetValue(float value)
```

Parameters

`value` float

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValue(float[])

Sets the value of the EffectParameter.

```
public void SetValue(float[] value)
```

Parameters

`value` float[]

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

SetValueTranspose(Matrix)

Sets the value of the EffectParameter to the transpose of a Matrix.

```
public void SetValueTranspose(Matrix value)
```

Parameters

`value` Matrix

The value to assign to the EffectParameter.

Remarks

Setting the value of an effect parameter is a slow operation. Avoid high-frequency calls.

Exceptions

InvalidCastException

Unable to cast this EffectParameter to target type.

Enum EffectParameterClass

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines classes for effect parameters and shader constants.

```
public enum EffectParameterClass
```

Fields

`Matrix = 2`

Matrix class type.

`Object = 3`

Class type for textures, shaders or strings.

`Scalar = 0`

Scalar class type.

`Struct = 4`

Structure class type.

`Vector = 1`

Vector class type.

Class EffectParameterCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of EffectParameter objects.

```
public class EffectParameterCollection : IEnumerable<EffectParameter>, IEnumerable
```

Inheritance

object ← EffectParameterCollection

Implements

IEnumerable<EffectParameter>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of elements contained in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Retrieves the EffectParameter at the specified index in the collection.

```
public EffectParameter this[int index] { get; }
```

Parameters

`index` int

Property Value

EffectParameter

this[string]

Retrieves a EffectParameter from the collection, given the name of the parameter.

```
public EffectParameter this[string name] { get; }
```

Parameters

`name` string

The name of the parameter to retrieve.

Property Value

EffectParameter

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<EffectParameter> GetEnumerator()
```

Returns

IEnumerator<EffectParameter>

An enumerator that can be used to iterate through the collection.

Enum EffectParameterType

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines types for effect parameters and shader constants.

```
public enum EffectParameterType
```

Fields

`Bool = 1`

Boolean type. Any non-zero will be `true`; `false` otherwise.

`Int32 = 2`

32-bit integer type.

`Single = 3`

Float type.

`String = 4`

String type.

`Texture = 5`

Any texture type.

`Texture1D = 6`

1D-texture type.

`Texture2D = 7`

2D-texture type.

`Texture3D = 8`

3D-texture type.

`TextureCube = 9`

Cubic texture type.

`Void = 0`

Pointer to void type.

Class EffectPass

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains the rendering state for drawing with an effect.

An effect can contain one or more passes.

```
public class EffectPass
```

Inheritance

object ← EffectPass

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Annotations

Gets the collection of EffectAnnotation objects for this EffectPass.

```
public EffectAnnotationCollection Annotations { get; }
```

Property Value

EffectAnnotationCollection

Name

Gets the name of this pass.

```
public string Name { get; }
```

Property Value

string

Methods

Apply()

Begins this pass.

```
public void Apply()
```

Class EffectPassCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of EffectPass objects.

```
public class EffectPassCollection : IEnumerable<EffectPass>, IEnumerable
```

Inheritance

object ← EffectPassCollection

Implements

IEnumerable<EffectPass>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of elements contained in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Retrieves the EffectPass at the specified index in the collection.

```
public EffectPass this[int index] { get; }
```

Parameters

`index` int

Property Value

EffectPass

this[string]

Retrieves a EffectPass from the collection, given the name of the pass.

```
public EffectPass this[string name] { get; }
```

Parameters

`name` string

The name of the pass to retrieve.

Property Value

EffectPass

Methods

GetEnumerator()

Returns a EffectPassCollection.Enumerator that can iterate through a collection.

```
public EffectPassCollection.Enumerator GetEnumerator()
```

Returns

EffectPassCollection.Enumerator

Struct EffectPassCollection.Enumerator

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Enumerator to iterate through the EffectPassCollection

```
public struct EffectPassCollection.Enumerator : IEnumerator<EffectPass>,
IEnumerator, IDisposable
```

Implements

IEnumerator<EffectPass>, IEnumerator, IDisposable

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Current

Gets the element in the collection at the current position of the enumerator.

```
public EffectPass Current { get; }
```

Property Value

EffectPass

The element in the collection at the current position of the enumerator.

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

MoveNext()

Advances the enumerator to the next element of the collection.

```
public bool MoveNext()
```

Returns

bool

true if the enumerator was successfully advanced to the next element; false if the enumerator has passed the end of the collection.

Exceptions

InvalidOperationException

The collection was modified after the enumerator was created.

Class EffectTechnique

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a technique used in a shader effect.

```
public class EffectTechnique
```

Inheritance

object ← EffectTechnique

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

1. Create a **EffectTechnique** for each technique in your Effect.

```
public EffectTechnique texture;
public EffectTechnique shadows;
public EffectTechnique shadowMap;
```

2. Assign an Effect technique to your **EffectTechnique**.

```
texture = effect.Techniques["TextureRender"];
shadowMap = effect.Techniques["ShadowMapRender"];
shadows = effect.Techniques["ShadowRender"];
```

3. Assign your **EffectTechnique** to the CurrentTechnique of your Effect before drawing.

```
private void DrawScene(EffectTechnique technique)
{
   MyEffect.mWorld.SetValue(terrainWorld);
   MyEffect.MeshTexture.SetValue(terrainTex);
   foreach (ModelMesh mesh in terrain.Meshes)
   {
       foreach (Effect effect in mesh.Effects)
       {
           effect.CurrentTechnique = technique;
           mesh.Draw();
       }
   }
}
```

Remarks

Creating and assigning a **EffectTechnique** instance for each technique in your Effect is significantly faster than using the Techniques indexed property on Effect.

Properties

Annotations

Gets the EffectAnnotation objects associated with this technique.

```
public EffectAnnotationCollection Annotations { get; }
```

Property Value

EffectAnnotationCollection

Name

Gets the name of this technique.

```
public string Name { get; }
```

Property Value

string

Passes

Gets the collection of EffectPass objects this rendering technique requires.

```
public EffectPassCollection Passes { get; }
```

Property Value

EffectPassCollection

Class EffectTechniqueCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of EffectTechnique objects.

```
public class EffectTechniqueCollection : IEnumerable<EffectTechnique>, IEnumerable
```

Inheritance

object ← EffectTechniqueCollection

Implements

IEnumerable<EffectTechnique>, IEnumerable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of elements contained in the collection.

```
public int Count { get; }
```

Property Value

int

this[int]

Retrieves the EffectTechnique at the specified index in the collection.

```
public EffectTechnique this[int index] { get; }
```

Parameters

`index` int

Property Value

EffectTechnique

this[string]

Retrieves a EffectTechnique from the collection, given the name of the technique.

```
public EffectTechnique this[string name] { get; }
```

Parameters

`name` string

The name of the technique to retrieve.

Property Value

EffectTechnique

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<EffectTechnique> GetEnumerator()
```

Returns

IEnumerator<EffectTechnique>

An enumerator that can be used to iterate through the collection.

Class EnvironmentMapEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Built-in effect that supports environment mapping.

```
public class EnvironmentMapEffect : Effect, IDisposable, IEffectMatrices,
IEffectLights, IEffectFog
```

Inheritance

object ← GraphicsResource ← Effect ← EnvironmentMapEffect

Implements

IDisposable, IEffectMatrices, IEffectLights, IEffectFog

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

EnvironmentMapEffect(EnvironmentMapEffect)

Creates a new EnvironmentMapEffect by cloning parameter settings from an existing instance.

```
protected EnvironmentMapEffect(EnvironmentMapEffect cloneSource)
```

Parameters

`cloneSource` EnvironmentMapEffect

EnvironmentMapEffect(GraphicsDevice)

Creates a new EnvironmentMapEffect with default parameter settings.

```
public EnvironmentMapEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

Properties

Alpha

Gets or sets the material alpha.

```
public float Alpha { get; set; }
```

Property Value

float

AmbientLightColor

Gets or sets the ambient light color (range 0 to 1).

```
public Vector3 AmbientLightColor { get; set; }
```

Property Value

Vector3

DiffuseColor

Gets or sets the material diffuse color (range 0 to 1).

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

DirectionalLight0

Gets the first directional light.

```
public DirectionalLight DirectionalLight0 { get; }
```

Property Value

DirectionalLight

DirectionalLight1

Gets the second directional light.

```
public DirectionalLight DirectionalLight1 { get; }
```

Property Value

DirectionalLight

DirectionalLight2

Gets the third directional light.

```
public DirectionalLight DirectionalLight2 { get; }
```

Property Value

DirectionalLight

EmissiveColor

Gets or sets the material emissive color (range 0 to 1).

```
public Vector3 EmissiveColor { get; set; }
```

Property Value

Vector3

EnvironmentMap

Gets or sets the current environment map texture.

```
public TextureCube EnvironmentMap { get; set; }
```

Property Value

TextureCube

EnvironmentMapAmount

Gets or sets the amount of the environment map RGB that will be blended over the base texture. Range 0 to 1, default 1. If set to zero, the RGB channels of the environment map will completely ignored (but the environment map alpha may still be visible if EnvironmentMapSpecular is greater than zero).

```
public float EnvironmentMapAmount { get; set; }
```

Property Value

float

EnvironmentMapSpecular

Gets or sets the amount of the environment map alpha channel that will be added to the base texture. Range 0 to 1, default 0. This can be used to implement cheap specular lighting, by encoding one or more specular highlight patterns into the environment map alpha channel, then setting EnvironmentMapSpecular to the desired specular light color.

```
public Vector3 EnvironmentMapSpecular { get; set; }
```

Property Value

Vector3

FogColor

Gets or sets the fog color.

```
public Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Gets or sets the fog enable flag.

```
public bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

Gets or sets the fog end distance.

```
public float FogEnd { get; set; }
```

Property Value

float

FogStart

Gets or sets the fog start distance.

```
public float FogStart { get; set; }
```

Property Value

float

FresnelFactor

Gets or sets the Fresnel factor used for the environment map blending. Higher values make the environment map only visible around the silhouette edges of the object, while lower values make it visible everywhere. Setting this property to 0 disables Fresnel entirely, making the environment map equally visible regardless of view angle. The default is 1. Fresnel only affects the environment map RGB (the intensity of which is controlled by EnvironmentMapAmount). The alpha contribution (controlled by EnvironmentMapSpecular) is not affected by the Fresnel setting.

```
public float FresnelFactor { get; set; }
```

Property Value

float

Projection

Gets or sets the projection matrix.

```
public Matrix Projection { get; set; }
```

Property Value

Matrix

Texture

Gets or sets the current texture.

```
public Texture2D Texture { get; set; }
```

Property Value

Texture2D

View

Gets or sets the view matrix.

```
public Matrix View { get; set; }
```

Property Value

Matrix

World

Gets or sets the world matrix.

```
public Matrix World { get; set; }
```

Property Value

Matrix

Methods

Clone()

Creates a clone of the current EnvironmentMapEffect instance.

```
public override Effect Clone()
```

Returns

Effect

EnableDefaultLighting()

Sets up the standard key/fill/back lighting rig.

```
public void EnableDefaultLighting()
```

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

Enum FillMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines options for filling the primitive.

```
public enum FillMode
```

Fields

`Solid = 0`

Draw solid faces for each primitive.

`WireFrame = 1`

Draw lines for each primitive.

Class GraphicsAdapter

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provides methods to retrieve and manipulate graphics adapters.

```
public sealed class GraphicsAdapter : IDisposable
```

Inheritance

object ← GraphicsAdapter

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Adapters

Gets a read-only collection of available adapters on the system.

```
public static ReadOnlyCollection<GraphicsAdapter> Adapters { get; }
```

Property Value

ReadOnlyCollection<GraphicsAdapter>

CurrentDisplayMode

Gets the current display mode.

```
public DisplayMode CurrentDisplayMode { get; }
```

Property Value

DisplayMode

DefaultAdapter

Gets the default adapter.

```
public static GraphicsAdapter DefaultAdapter { get; }
```

Property Value

GraphicsAdapter

Description

Gets a string used for presentation to the user.

```
public string Description { get; }
```

Property Value

string

IsWideScreen

Gets a bool indicating whether CurrentDisplayMode has a Width:Height ratio corresponding to a widescreen DisplayMode. Common widescreen modes include 16:9, 16:10 and 2:1.

```
public bool IsWideScreen { get; }
```

Property Value

bool

SupportedDisplayModes

Gets a collection of supported display modes for the current adapter.

```
public DisplayModeCollection SupportedDisplayModes { get; }
```

Property Value

DisplayModeCollection

UseDriverType

Used to request creation of a specific kind of driver.

```
public static GraphicsAdapter.DriverType UseDriverType { get; set; }
```

Property Value

GraphicsAdapter.DriverType

Remarks

These values only work on DirectX platforms and must be defined before the graphics device is created. Hardware by default.

UseReferenceDevice

Used to request creation of the reference graphics device, or the default hardware accelerated device (when set to false).

```
public static bool UseReferenceDevice { get; set; }
```

Property Value

bool

Remarks

This only works on DirectX platforms where a reference graphics device is available and must be defined before the graphics device is created. It defaults to false.

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

IsProfileSupported(GraphicsProfile)

Returns a value that indicates whether the specified graphics profile is supported by the current adapter.

```
public bool IsProfileSupported(GraphicsProfile graphicsProfile)
```

Parameters

`graphicsProfile` GraphicsProfile

The graphics profile to check for support.

Returns

bool

Exceptions

InvalidOperationException

The `graphicsProfile` parameter is not a valid GraphicsProfile enum value.

QueryRenderTargetFormat(GraphicsProfile, SurfaceFormat, DepthFormat, int, out SurfaceFormat, out DepthFormat, out int)

Queries for support of the requested render target format on the adaptor.

```
public bool QueryRenderTargetFormat(GraphicsProfile graphicsProfile,
SurfaceFormat format, DepthFormat depthFormat, int multiSampleCount, out
SurfaceFormat selectedFormat, out DepthFormat selectedDepthFormat, out
int selectedMultiSampleCount)
```

Parameters

`graphicsProfile` GraphicsProfile

The graphics profile.

`format` SurfaceFormat

The requested surface format.

`depthFormat` DepthFormat

The requested depth stencil format.

`multiSampleCount` int

The requested multisample count.

`selectedFormat` SurfaceFormat

Set to the best format supported by the adaptor for the requested surface format.

`selectedDepthFormat` DepthFormat

Set to the best format supported by the adaptor for the requested depth stencil format.

`selectedMultiSampleCount` int

Set to the best count supported by the adaptor for the requested multisample count.

Returns

bool

True if the requested format is supported by the adaptor. False if one or more of the values was changed.

Enum GraphicsAdapter.DriverType

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the driver type for graphics adapter. Usable only on DirectX platforms for now.

```
public enum GraphicsAdapter.DriverType
```

Fields

`FastSoftware = 2`

Useful when Hardware acceleration does not work.

`Hardware = 0`

Hardware device been used for rendering. Maximum speed and performance.

`Reference = 1`

Emulates the hardware device on CPU. Slowly, only for testing.

Class GraphicsDebug

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a class for debugging graphics.

```
public class GraphicsDebug
```

Inheritance

object ← GraphicsDebug

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

TryDequeueMessage(out GraphicsDebugMessage)

Attempt to dequeue a debugging message from the graphics subsystem.

```
public bool TryDequeueMessage(out GraphicsDebugMessage message)
```

Parameters

`message` GraphicsDebugMessage

The graphics debugging message if retrieved, null otherwise.

Returns

bool

True if a graphics debugging message was retrieved, false otherwise.

Remarks

When running on a graphics device with debugging enabled, this allows you to retrieve subsystem-specific (e.g. DirectX, OpenGL, etc.) debugging messages including information about improper usage of shaders and APIs.

Class GraphicsDebugMessage

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a debug message from the graphics subsystem.

```
public class GraphicsDebugMessage
```

Inheritance

object ← GraphicsDebugMessage

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Category

Gets or Sets the category of the debug message.

```
public string Category { get; set; }
```

Property Value

string

Id

Gets or Sets the ID of the debug message.

```
public int Id { get; set; }
```

Property Value

int

IdName

Gets or Sets the string representation of the debug message ID.

```
public string IdName { get; set; }
```

Property Value

string

Message

Gets or Sets the message.

```
public string Message { get; set; }
```

Property Value

string

Severity

Gets or Sets the debug severity level of the message.

```
public string Severity { get; set; }
```

Property Value

string

Class GraphicsDevice

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Performs primitive-based rendering, creates resources, handles system-level variables, adjusts gamma ramp levels, and creates shaders.

```
public class GraphicsDevice : IDisposable
```

Inheritance

object ← GraphicsDevice

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

GraphicsDevice(GraphicsAdapter, GraphicsProfile, PresentationParameters)

Initializes a new instance of the GraphicsDevice class.

```
public GraphicsDevice(GraphicsAdapter adapter, GraphicsProfile graphicsProfile,
PresentationParameters presentationParameters)
```

Parameters

`adapter` GraphicsAdapter

The graphics adapter.

`graphicsProfile` GraphicsProfile

The graphics profile.

`presentationParameters` PresentationParameters

The presentation options.

Exceptions

ArgumentNullException

`presentationParameters` is null.

GraphicsDevice(GraphicsAdapter, GraphicsProfile, bool, PresentationParameters)

Initializes a new instance of the GraphicsDevice class.

```
public GraphicsDevice(GraphicsAdapter adapter, GraphicsProfile graphicsProfile, bool
preferHalfPixelOffset, PresentationParameters presentationParameters)
```

Parameters

`adapter` GraphicsAdapter

The graphics adapter.

`graphicsProfile` GraphicsProfile

The graphics profile.

`preferHalfPixelOffset` bool

Indicates if DX9 style pixel addressing or current standard pixel addressing should be used. This value is passed to UseHalfPixelOffset

`presentationParameters` PresentationParameters

The presentation options.

Exceptions

ArgumentNullException

`presentationParameters` is null.

Properties

Adapter

Gets the graphics adapter.

```
public GraphicsAdapter Adapter { get; }
```

Property Value

GraphicsAdapter

BlendFactor

The color used as blend factor when alpha blending.

```
public Color BlendFactor { get; set; }
```

Property Value

Color

Remarks

When only changing BlendFactor, use this rather than BlendFactor to only update BlendFactor so the whole BlendState does not have to be updated.

BlendState

Gets or sets a system-defined instance of a blend state object initialized for alpha blending. The default value is Opaque.

```
public BlendState BlendState { get; set; }
```

Property Value

BlendState

DepthStencilState

Gets or sets a system-defined instance of a depth-stencil state object. The default value is Default.

```
public DepthStencilState DepthStencilState { get; set; }
```

Property Value

DepthStencilState

DiscardColor

Get or set the color a RenderTarget2D is cleared to when it is set.

```
public static Color DiscardColor { get; set; }
```

Property Value

Color

DisplayMode

Retrieves the display mode's spatial resolution, color resolution, and refresh frequency.

```
public DisplayMode DisplayMode { get; }
```

Property Value

DisplayMode

GraphicsDebug

Access debugging APIs for the graphics subsystem.

```
public GraphicsDebug GraphicsDebug { get; set; }
```

Property Value

GraphicsDebug

GraphicsDeviceStatus

Retrieves the status of the device.

```
public GraphicsDeviceStatus GraphicsDeviceStatus { get; }
```

Property Value

GraphicsDeviceStatus

GraphicsProfile

Gets the graphics profile. The default value is Reach.

```
public GraphicsProfile GraphicsProfile { get; }
```

Property Value

GraphicsProfile

Indices

Gets or sets index data. The default value is null.

```
public IndexBuffer Indices { get; set; }
```

Property Value

IndexBuffer

IsContentLost

Gets a value that indicates whether the associated content was lost.

```
public bool IsContentLost { get; }
```

Property Value

bool

IsDisposed

Gets a value that indicates whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Metrics

The rendering information for debugging and profiling. The metrics are reset every frame after draw within Present().

```
public GraphicsMetrics Metrics { get; set; }
```

Property Value

GraphicsMetrics

PresentationParameters

Gets the presentation parameters associated with this graphics device.

```
public PresentationParameters PresentationParameters { get; }
```

Property Value

PresentationParameters

RasterizerState

Gets or sets rasterizer state. The default value is CullCounterClockwise.

```
public RasterizerState RasterizerState { get; set; }
```

Property Value

RasterizerState

RenderTargetCount

Gets the amount of render targets bound to this device.

```
public int RenderTargetCount { get; }
```

Property Value

int

ResourcesLost

Gets a value that indicates whether the resources were lost.

```
public bool ResourcesLost { get; set; }
```

Property Value

bool

SamplerStates

Retrieves a collection of SamplerState objects for the current GraphicsDevice.

```
public SamplerStateCollection SamplerStates { get; }
```

Property Value

SamplerStateCollection

ScissorRectangle

Gets or sets the rectangle used for scissor testing. By default, the size matches the render target size.

```
public Rectangle ScissorRectangle { get; set; }
```

Property Value

Rectangle

Textures

Returns the collection of textures that have been assigned to the texture stages of the device.

```
public TextureCollection Textures { get; }
```

Property Value

TextureCollection

UseHalfPixelOffset

Indicates if DX9 style pixel addressing or current standard pixel addressing should be used. This flag is set to `false` by default. If `UseHalfPixelOffset` is `true` you have to add half-pixel offset to a Projection matrix. See also PreferHalfPixelOffset.

```
public bool UseHalfPixelOffset { get; }
```

Property Value

bool

Remarks

XNA uses DirectX9 for its graphics. DirectX9 interprets UV coordinates differently from other graphics API's. This is typically referred to as the half-pixel offset. MonoGame replicates XNA behavior if this flag is set to `true`.

VertexSamplerStates

Returns the collection of vertex sampler states.

```
public SamplerStateCollection VertexSamplerStates { get; }
```

Property Value

SamplerStateCollection

VertexTextures

Gets the collection of vertex textures that support texture lookup in the vertex shader using the texldl statement. The vertex engine contains four texture sampler stages.

```
public TextureCollection VertexTextures { get; }
```

Property Value

TextureCollection

Viewport

Gets or sets a viewport identifying the portion of the render target to receive draw calls.

```
public Viewport Viewport { get; set; }
```

Property Value

Viewport

Methods

Clear(Color)

Clears resource buffers.

```
public void Clear(Color color)
```

Parameters

`color` Color

Set this color value in all buffers.

Clear(ClearOptions, Color, float, int)

Clears resource buffers.

```
public void Clear(ClearOptions options, Color color, float depth, int stencil)
```

Parameters

`options` ClearOptions

Options for clearing a buffer.

`color` Color

Set this color value in all buffers.

`depth` float

Set this depth value in the buffer.

`stencil` int

Set this stencil value in the buffer.

Clear(ClearOptions, Vector4, float, int)

Clears resource buffers.

```
public void Clear(ClearOptions options, Vector4 color, float depth, int stencil)
```

Parameters

`options` ClearOptions

Options for clearing a buffer.

`color` Vector4

Set this color value in all buffers.

`depth` float

Set this depth value in the buffer.

`stencil` int

Set this stencil value in the buffer.

Dispose()

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

DrawIndexedPrimitives(PrimitiveType, int, int, int)

Draw geometry by indexing into the vertex buffer.

```
public void DrawIndexedPrimitives(PrimitiveType primitiveType, int baseVertex, int
startIndex, int primitiveCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives in the index buffer.

`baseVertex` int

Used to offset the vertex range indexed from the vertex buffer.

`startIndex` int

The index within the index buffer to start drawing from.

`primitiveCount` int

The number of primitives to render from the index buffer.

DrawIndexedPrimitives(PrimitiveType, int, int, int, int, int)

Draw geometry by indexing into the vertex buffer.

```
[Obsolete("Use DrawIndexedPrimitives(PrimitiveType primitiveType, int baseVertex,
int startIndex, int primitiveCount) instead. In future versions this method can
be removed.")]
public void DrawIndexedPrimitives(PrimitiveType primitiveType, int baseVertex, int
minVertexIndex, int numVertices, int startIndex, int primitiveCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives in the index buffer.

`baseVertex` int

Used to offset the vertex range indexed from the vertex buffer.

`minVertexIndex` int

This is unused and remains here only for XNA API compatibility.

`numVertices` int

This is unused and remains here only for XNA API compatibility.

`startIndex` int

The index within the index buffer to start drawing from.

`primitiveCount` int

The number of primitives to render from the index buffer.

Remarks

Note that minVertexIndex and numVertices are unused in MonoGame and will be ignored.

DrawInstancedPrimitives(PrimitiveType, int, int, int, int)

Draw instanced geometry from the bound vertex buffers and index buffer.

```
public void DrawInstancedPrimitives(PrimitiveType primitiveType, int baseVertex, int
startIndex, int primitiveCount, int instanceCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives in the index buffer.

`baseVertex` int

Used to offset the vertex range indexed from the vertex buffer.

`startIndex` int

The index within the index buffer to start drawing from.

`primitiveCount` int

The number of primitives in a single instance.

`instanceCount` int

The number of instances to render.

Remarks

Draw geometry with data from multiple bound vertex streams at different frequencies.

DrawInstancedPrimitives(PrimitiveType, int, int, int, int, int)

Draw instanced geometry from the bound vertex buffers and index buffer.

```
public void DrawInstancedPrimitives(PrimitiveType primitiveType, int baseVertex, int
startIndex, int primitiveCount, int baseInstance, int instanceCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives in the index buffer.

`baseVertex` int

Used to offset the vertex range indexed from the vertex buffer.

`startIndex` int

The index within the index buffer to start drawing from.

`primitiveCount` int

The number of primitives in a single instance.

`baseInstance` int

Used to offset the instance range indexed from the instance buffer.

`instanceCount` int

The number of instances to render.

Remarks

Draw geometry with data from multiple bound vertex streams at different frequencies.

DrawInstancedPrimitives(PrimitiveType, int, int, int, int, int, int)

Draw instanced geometry from the bound vertex buffers and index buffer.

```
[Obsolete("Use DrawInstancedPrimitives(PrimitiveType primitiveType, int baseVertex,
int startIndex, int primitiveCount, int instanceCount) instead. In future versions
this method can be removed.")]
public void DrawInstancedPrimitives(PrimitiveType primitiveType, int baseVertex,
int minVertexIndex, int numVertices, int startIndex, int primitiveCount,
int instanceCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives in the index buffer.

`baseVertex` int

Used to offset the vertex range indexed from the vertex buffer.

`minVertexIndex` int

This is unused and remains here only for XNA API compatibility.

`numVertices` int

This is unused and remains here only for XNA API compatibility.

`startIndex` int

The index within the index buffer to start drawing from.

`primitiveCount` int

The number of primitives in a single instance.

`instanceCount` int

The number of instances to render.

Remarks

Note that minVertexIndex and numVertices are unused in MonoGame and will be ignored.

DrawPrimitives(PrimitiveType, int, int)

Draw primitives of the specified type from the currently bound vertexbuffers without indexing.

```
public void DrawPrimitives(PrimitiveType primitiveType, int vertexStart,
int primitiveCount)
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw.

`vertexStart` int

Index of the vertex to start at.

`primitiveCount` int

The number of primitives to draw.

DrawUserIndexedPrimitives<T>(PrimitiveType, T[], int, int, short[], int, int)

Draw primitives of the specified type by indexing into the given array of vertices with 16-bit indices.

```
public void DrawUserIndexedPrimitives<T>(PrimitiveType primitiveType, T[]
vertexData, int vertexOffset, int numVertices, short[] indexData, int indexOffset,
int primitiveCount) where T : struct, IVertexType
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex to draw.

`numVertices` int

The number of vertices to draw.

`indexData` short[]

The index data.

`indexOffset` int

The index in the array of indices of the first index to use

`primitiveCount` int

The number of primitives to draw.

Type Parameters

`T`

The type of the vertices.

Remarks

The VertexDeclaration will be found by getting VertexDeclaration from an instance of `T` and cached for subsequent calls.

DrawUserIndexedPrimitives<T>(PrimitiveType, T[], int, int, short[], int, int, VertexDeclaration)

Draw primitives of the specified type by indexing into the given array of vertices with 16-bit indices.

```
public void DrawUserIndexedPrimitives<T>(PrimitiveType primitiveType, T[]
vertexData, int vertexOffset, int numVertices, short[] indexData, int indexOffset,
int primitiveCount, VertexDeclaration vertexDeclaration) where T : struct
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex to draw.

`numVertices` int

The number of vertices to draw.

`indexData` short[]

The index data.

`indexOffset` int

The index in the array of indices of the first index to use

`primitiveCount` int

The number of primitives to draw.

`vertexDeclaration` VertexDeclaration

The layout of the vertices.

Type Parameters

`T`

The type of the vertices.

Remarks

All indices in the vertex buffer are interpreted relative to the specified `vertexOffset`. For example a value of zero in the array of indices points to the vertex at index `vertexOffset` in the array of vertices.

DrawUserIndexedPrimitives<T>(PrimitiveType, T[], int, int, int[], int, int)

Draw primitives of the specified type by indexing into the given array of vertices with 32-bit indices.

```
public void DrawUserIndexedPrimitives<T>(PrimitiveType primitiveType, T[]
vertexData, int vertexOffset, int numVertices, int[] indexData, int indexOffset, int
primitiveCount) where T : struct, IVertexType
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex to draw.

`numVertices` int

The number of vertices to draw.

`indexData` int[]

The index data.

`indexOffset` int

The index in the array of indices of the first index to use

`primitiveCount` int

The number of primitives to draw.

Type Parameters

`T`

The type of the vertices.

Remarks

The VertexDeclaration will be found by getting VertexDeclaration from an instance of `T` and cached for subsequent calls.

DrawUserIndexedPrimitives<T>(PrimitiveType, T[], int, int, int[], int, int, VertexDeclaration)

Draw primitives of the specified type by indexing into the given array of vertices with 32-bit indices.

```
public void DrawUserIndexedPrimitives<T>(PrimitiveType primitiveType, T[]
vertexData, int vertexOffset, int numVertices, int[] indexData, int indexOffset, int
primitiveCount, VertexDeclaration vertexDeclaration) where T : struct
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex to draw.

`numVertices` int

The number of vertices to draw.

`indexData` int[]

The index data.

`indexOffset` int

The index in the array of indices of the first index to use

`primitiveCount` int

The number of primitives to draw.

`vertexDeclaration` VertexDeclaration

The layout of the vertices.

Type Parameters

`T`

The type of the vertices.

Remarks

All indices in the vertex buffer are interpreted relative to the specified `vertexOffset`. For example value of zero in the array of indices points to the vertex at index `vertexOffset` in the array of vertices.

DrawUserPrimitives<T>(PrimitiveType, T[], int, int)

Draw primitives of the specified type from the data in an array of vertices without indexing.

```
public void DrawUserPrimitives<T>(PrimitiveType primitiveType, T[] vertexData, int
vertexOffset, int primitiveCount) where T : struct, IVertexType
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex that should be rendered.

`primitiveCount` int

The number of primitives to draw.

Type Parameters

`T`

The type of the vertices.

Remarks

The VertexDeclaration will be found by getting VertexDeclaration from an instance of `T` and cached for subsequent calls.

DrawUserPrimitives<T>(PrimitiveType, T[], int, int, VertexDeclaration)

Draw primitives of the specified type from the data in the given array of vertices without indexing.

```
public void DrawUserPrimitives<T>(PrimitiveType primitiveType, T[] vertexData,
int vertexOffset, int primitiveCount, VertexDeclaration vertexDeclaration) where T
: struct
```

Parameters

`primitiveType` PrimitiveType

The type of primitives to draw with the vertices.

`vertexData` T[]

An array of vertices to draw.

`vertexOffset` int

The index in the array of the first vertex that should be rendered.

`primitiveCount` int

The number of primitives to draw.

`vertexDeclaration` VertexDeclaration

The layout of the vertices.

Type Parameters

`T`

The type of the vertices.

~GraphicsDevice()

```
protected ~GraphicsDevice()
```

GetBackBufferData<T>(Rectangle?, T[], int, int)

Gets the Pixel data of what is currently drawn on screen. The format is whatever the current format of the backbuffer is.

```
public void GetBackBufferData<T>(Rectangle? rect, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`rect` Rectangle?

The section of the back buffer to copy. null indicates the data will be copied from the entire back buffer.

`data` T[]

Array of data.

`startIndex` int

The first element to use.

`elementCount` int

The number of elements to use.

Type Parameters

`T`

A byte[] of size (ViewPort.Width * ViewPort.Height * 4)

Exceptions

ArgumentNullException

`data` is null

ArgumentException

GetBackBufferData<T>(T[])

Gets the Pixel data of what is currently drawn on screen. The format is whatever the current format of the backbuffer is.

```
public void GetBackBufferData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

Array of data.

Type Parameters

`T`

A byte[] of size (ViewPort.Width * ViewPort.Height * 4)

GetBackBufferData<T>(T[], int, int)

Gets the Pixel data of what is currently drawn on screen. The format is whatever the current format of the backbuffer is.

```
public void GetBackBufferData<T>(T[] data, int startIndex, int elementCount) where T
: struct
```

Parameters

`data` T[]

Array of data.

`startIndex` int

The first element to use.

`elementCount` int

The number of elements to use.

Type Parameters

`T`

A byte[] of size (ViewPort.Width * ViewPort.Height * 4)

GetRenderTargets()

Gets render target surfaces.

```
public RenderTargetBinding[] GetRenderTargets()
```

Returns

RenderTargetBinding[]

An array of bound render targets.

GetRenderTargets(RenderTargetBinding[])

Gets render target surfaces.

```
public void GetRenderTargets(RenderTargetBinding[] outTargets)
```

Parameters

`outTargets` RenderTargetBinding[]

When this method returns, contains an array of that description of bound render targets. This parameter is treated as uninitialized.

Present()

Presents the display with the contents of the next buffer in the sequence of back buffers owned by the GraphicsDevice.

```
public void Present()
```

Exceptions

InvalidOperationException

A render target is active.

Reset()

Resets the presentation parameters for the current GraphicsDevice.

```
public void Reset()
```

Reset(PresentationParameters)

Resets the current GraphicsDevice with the specified PresentationParameters.

```
public void Reset(PresentationParameters presentationParameters)
```

Parameters

`presentationParameters` PresentationParameters

Presentation parameters to set.

Exceptions

ArgumentNullException

`presentationParameters` is null

SetRenderTarget(RenderTarget2D)

Sets a new render target for this GraphicsDevice.

```
public void SetRenderTarget(RenderTarget2D renderTarget)
```

Parameters

`renderTarget` RenderTarget2D

A new render target for the device, or null to set the device render target to the back buffer of the device.

SetRenderTarget(RenderTargetCube, CubeMapFace)

Sets a new render target for this GraphicsDevice.

```
public void SetRenderTarget(RenderTargetCube renderTarget, CubeMapFace cubeMapFace)
```

Parameters

`renderTarget` RenderTargetCube

A new render target for the device, or null to set the device render target to the back buffer of the device.

`cubeMapFace` CubeMapFace

The cube map face type.

SetRenderTargets(params RenderTargetBinding[])

Sets an array of render targets.

```
public void SetRenderTargets(params RenderTargetBinding[] renderTargets)
```

Parameters

`renderTargets` RenderTargetBinding[]

An array of render targets.

SetVertexBuffer(VertexBuffer)

Sets or binds a vertex buffer to a device.

```
public void SetVertexBuffer(VertexBuffer vertexBuffer)
```

Parameters

`vertexBuffer` VertexBuffer

A vertex buffer.

SetVertexBuffer(VertexBuffer, int)

Sets or binds a vertex buffer to a device.

```
public void SetVertexBuffer(VertexBuffer vertexBuffer, int vertexOffset)
```

Parameters

`vertexBuffer` VertexBuffer

A vertex buffer.

`vertexOffset` int

The offset (in bytes) from the beginning of the buffer.

Exceptions

ArgumentOutOfRangeException

`vertexOffset` is less than 0 OR is greater than or equal to `vertexBuffer`.VertexCount.

SetVertexBuffers(params VertexBufferBinding[])

Sets the vertex buffers.

```
public void SetVertexBuffers(params VertexBufferBinding[] vertexBuffers)
```

Parameters

`vertexBuffers` VertexBufferBinding[]

An array of vertex buffers.

Exceptions

ArgumentOutOfRangeException

Length of `vertexBuffers` is more than max allowed number of vertex buffers.

Events

DeviceLost

Occurs when a GraphicsDevice is about to be lost (for example, immediately before a reset).

```
public event EventHandler<EventArgs> DeviceLost
```

Event Type

EventHandler<EventArgs>

DeviceReset

Occurs after a GraphicsDevice is reset, allowing an application to recreate all resources.

```
public event EventHandler<EventArgs> DeviceReset
```

Event Type

EventHandler<EventArgs>

DeviceResetting

Occurs when a GraphicsDevice is resetting, allowing the application to cancel the default handling of the reset.

```
public event EventHandler<EventArgs> DeviceResetting
```

Event Type

EventHandler<EventArgs>

Disposing

Occurs when Dispose() is called or when this object is finalized and collected by the garbage collector.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

ResourceCreated

Occurs when a resource is created.

```
public event EventHandler<ResourceCreatedEventArgs> ResourceCreated
```

Event Type

EventHandler<ResourceCreatedEventArgs>

ResourceDestroyed

Occurs when a resource is destroyed.

```
public event EventHandler<ResourceDestroyedEventArgs> ResourceDestroyed
```

Event Type

EventHandler<ResourceDestroyedEventArgs>

Enum GraphicsDeviceStatus

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes the status of the GraphicsDevice.

```
public enum GraphicsDeviceStatus
```

Fields

`Lost = 1`

The device has been lost.

`Normal = 0`

The device is normal.

`NotReset = 2`

The device has not been reset.

Struct GraphicsMetrics

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

A snapshot of rendering statistics from Metrics to be used for runtime debugging and profiling.

```
public struct GraphicsMetrics
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

ClearCount

Number of times Clear was called.

```
public long ClearCount { get; }
```

Property Value

long

DrawCount

Number of times Draw was called.

```
public long DrawCount { get; }
```

Property Value

long

PixelShaderCount

Number of times the pixel shader was changed on the GPU.

```
public long PixelShaderCount { get; }
```

Property Value

long

PrimitiveCount

Number of rendered primitives.

```
public long PrimitiveCount { get; }
```

Property Value

long

SpriteCount

Number of sprites and text characters rendered via SpriteBatch.

```
public long SpriteCount { get; }
```

Property Value

long

TargetCount

Number of times a target was changed on the GPU.

```
public long TargetCount { get; }
```

Property Value

long

TextureCount

Number of times a texture was changed on the GPU.

```
public long TextureCount { get; }
```

Property Value

long

VertexShaderCount

Number of times the vertex shader was changed on the GPU.

```
public long VertexShaderCount { get; }
```

Property Value

long

Operators

operator +(GraphicsMetrics, GraphicsMetrics)

Returns the combination of two sets of metrics.

```
public static GraphicsMetrics operator +(GraphicsMetrics value1,
GraphicsMetrics value2)
```

Parameters

`value1` GraphicsMetrics

Source GraphicsMetrics on the left of the add sign.

`value2` GraphicsMetrics

Source GraphicsMetrics on the right of the add sign.

Returns

GraphicsMetrics

Combination of two sets of metrics.

operator -(GraphicsMetrics, GraphicsMetrics)

Returns the difference between two sets of metrics.

```
public static GraphicsMetrics operator -(GraphicsMetrics value1,
GraphicsMetrics value2)
```

Parameters

`value1` GraphicsMetrics

Source GraphicsMetrics on the left of the sub sign.

`value2` GraphicsMetrics

Source GraphicsMetrics on the right of the sub sign.

Returns

GraphicsMetrics

Difference between two sets of metrics.

Enum GraphicsProfile

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines a set of graphic capabilities.

```
public enum GraphicsProfile
```

Fields

`HiDef = 1`

Use the largest available set of graphic features and capabilities to target devices, that have more enhanced graphic capabilities.

`Reach = 0`

Use a limited set of graphic features and capabilities, allowing the game to support the widest variety of devices.

Class GraphicsResource

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Queries and prepares resources.

```
public abstract class GraphicsResource : IDisposable
```

Inheritance

object ← GraphicsResource

Implements

IDisposable

Derived

BlendState, DepthStencilState, Effect, IndexBuffer, OcclusionQuery, RasterizerState, SamplerState, SpriteBatch, Texture, VertexBuffer, VertexDeclaration

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

GraphicsDevice

Gets the GraphicsDevice associated with this GraphicsResource.

```
public GraphicsDevice GraphicsDevice { get; }
```

Property Value

GraphicsDevice

IsDisposed

Gets a value that indicates whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Name

Gets the name of the resource.

```
public string Name { get; set; }
```

Property Value

string

Tag

Gets the resource tags for this resource.

```
public object Tag { get; set; }
```

Property Value

object

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

The method that derived classes should override to implement disposing of managed and native resources.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if managed objects should be disposed.

Remarks

Native resources should always be released regardless of the value of the disposing parameter.

~GraphicsResource()

```
protected ~GraphicsResource()
```

GraphicsDeviceResetting()

Called before the device is reset. Allows graphics resources to invalidate their state so they can be recreated after the device reset. Warning: This may be called after a call to Dispose() up until the resource is garbage collected.

```
protected virtual void GraphicsDeviceResetting()
```

ToString()

Gets a string representation of the current instance.

```
public override string ToString()
```

Returns

string

Events

Disposing

Occurs when Dispose() is called or when this object is finalized and collected by the garbage collector.

```
public event EventHandler<EventArgs> Disposing
```

Event Type

EventHandler<EventArgs>

Interface IEffectBones

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Interface for Effects that support bone transforms.

```
public interface IEffectBones
```

Methods

SetBoneTransforms(Matrix[])

Sets an array of skinning bone transform matrices.

```
void SetBoneTransforms(Matrix[] boneTransforms)
```

Parameters

`boneTransforms` Matrix[]

Interface IEffectFog

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The common effect fog rendering parameters.

```
public interface IEffectFog
```

Properties

FogColor

The floating point fog color.

```
Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Used to toggle the rendering of fog.

```
bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

The world space distance from the camera at which fogging is fully applied.

```
float FogEnd { get; set; }
```

Property Value

float

Remarks

FogEnd should be greater than FogStart. If FogEnd and FogStart are the same value everything is fully fogged.

FogStart

The world space distance from the camera at which fogging begins.

```
float FogStart { get; set; }
```

Property Value

float

Remarks

FogStart should be less than FogEnd. If FogEnd and FogStart are the same value everything is fully fogged.

Interface IEffectLights

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The common effect light rendering parameters.

```
public interface IEffectLights
```

Properties

AmbientLightColor

The floating point ambient light color.

```
Vector3 AmbientLightColor { get; set; }
```

Property Value

Vector3

DirectionalLight0

Returns the first directional light.

```
DirectionalLight DirectionalLight0 { get; }
```

Property Value

DirectionalLight

DirectionalLight1

Returns the second directional light.

```
DirectionalLight DirectionalLight1 { get; }
```

Property Value

DirectionalLight

DirectionalLight2

Returns the third directional light.

```
DirectionalLight DirectionalLight2 { get; }
```

Property Value

DirectionalLight

LightingEnabled

Toggles the rendering of lighting.

```
bool LightingEnabled { get; set; }
```

Property Value

bool

Methods

EnableDefaultLighting()

Initializes the lights to the standard key/fill/back lighting rig.

```
void EnableDefaultLighting()
```

Interface IEffectMatrices

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Gets or sets transformation matrix parameters for the current effect.

```
public interface IEffectMatrices
```

Properties

Projection

Gets or sets the projection matrix in the current effect.

```
Matrix Projection { get; set; }
```

Property Value

Matrix

View

Gets or sets the view matrix in the current effect.

```
Matrix View { get; set; }
```

Property Value

Matrix

World

Gets or sets the world matrix in the current effect.

```
Matrix World { get; set; }
```

Property Value

Matrix

Interface IGraphicsDeviceService

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provider of a GraphicsDevice.

```
public interface IGraphicsDeviceService
```

Properties

GraphicsDevice

The provided GraphicsDevice.

```
GraphicsDevice GraphicsDevice { get; }
```

Property Value

GraphicsDevice

Events

DeviceCreated

Raised when a new GraphicsDevice has been created.

```
event EventHandler<EventArgs> DeviceCreated
```

Event Type

EventHandler<EventArgs>

DeviceDisposing

Raised when the GraphicsDevice is disposed.

```
event EventHandler<EventArgs> DeviceDisposing
```

Event Type

EventHandler<EventArgs>

DeviceReset

Raised when the GraphicsDevice has reset.

```
event EventHandler<EventArgs> DeviceReset
```

Event Type

EventHandler<EventArgs>

See Also
Reset()

DeviceResetting

Raised before the GraphicsDevice is resetting.

```
event EventHandler<EventArgs> DeviceResetting
```

Event Type

EventHandler<EventArgs>

Interface IVertexType

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Vertex type interface which is implemented by a custom vertex type structure.

```
public interface IVertexType
```

Properties

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
VertexDeclaration VertexDeclaration { get; }
```

Property Value

VertexDeclaration

Class IndexBuffer

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes the rendering order of the vertices in a vertex buffer.

```
public class IndexBuffer : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← IndexBuffer

Implements

IDisposable

Derived

DynamicIndexBuffer

Inherited Members

GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

IndexBuffer(GraphicsDevice, IndexElementSize, int, BufferUsage)

Creates a new instance of **IndexBuffer** with the specified parameters.

```
public IndexBuffer(GraphicsDevice graphicsDevice, IndexElementSize indexElementSize,
int indexCount, BufferUsage bufferUsage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexElementSize` IndexElementSize

The size, in bits, of an index element.

`indexCount` int

Number of indices in the buffer.

`bufferUsage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentOutOfRangeException

One of the following conditions is true

- The `indexCount` parameter is invalid. Index buffers can only be created for indices which are sixteen or thirty-two bits in length
- The `indexElementSize` parameter is invalid. The resource size must be greater than zero.
- The `indexElementSize` parameter is invalid. The total size of the index buffer must be an even multiple of the index element size (either 16 or 32 bits).

IndexBuffer(GraphicsDevice, IndexElementSize, int, BufferUsage, bool)

Creates a new instance of **IndexBuffer** with the specified parameters.

```
protected IndexBuffer(GraphicsDevice graphicsDevice, IndexElementSize
indexElementSize, int indexCount, BufferUsage usage, bool dynamic)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexElementSize` IndexElementSize

The size, in bits, of an index element.

`indexCount` int

Number of indices in the buffer.

`usage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

`dynamic` bool

true if this is a dynamic index buffer; otherwise, **false**.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentOutOfRangeException

One of the following conditions is true

- The `indexCount` parameter is invalid. Index buffers can only be created for indices which are sixteen or thirty-two bits in length
- The `indexElementSize` parameter is invalid. The resource size must be greater than zero.
- The `indexElementSize` parameter is invalid. The total size of the index buffer must be an even multiple of the index element size (either 16 or 32 bits).

InvalidOperationException

The resource could not be created

IndexBuffer(GraphicsDevice, Type, int, BufferUsage)

Creates a new instance of **IndexBuffer** with the specified parameters.

```
public IndexBuffer(GraphicsDevice graphicsDevice, Type indexType, int indexCount,
BufferUsage usage)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexType` Type

Type to use for index values.

`indexCount` int

Number of indices in the buffer.

`usage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

Exceptions

ArgumentOutOfRangeException

The `indexCount` parameter is invalid. It must be greater than zero, and the index must be sixteen or thirty-two bits in length. Index buffers can only be created for indices which are sixteen or thirty-two bits in length.

InvalidOperationException

The resource could not be created

IndexBuffer(GraphicsDevice, Type, int, BufferUsage, bool)

Creates a new instance of **IndexBuffer** with the specified parameters.

```
protected IndexBuffer(GraphicsDevice graphicsDevice, Type indexType, int indexCount,
BufferUsage usage, bool dynamic)
```

Parameters

`graphicsDevice` GraphicsDevice

The associated graphics device of the index buffer.

`indexType` Type

Type to use for index values.

`indexCount` int

Number of indices in the buffer.

`usage` BufferUsage

A set of options identifying the behaviors of this index buffer resource.

`dynamic` bool

true if this is a dynamic index buffer; otherwise, **false**.

Exceptions

ArgumentOutOfRangeException

The `indexCount` parameter is invalid. Index buffers can only be created for indices which are sixteen or thirty-two bits in length

InvalidOperationException

The resource could not be created

Properties

BufferUsage

Gets the usage hint for optimizing memory placement of this **IndexBuffer**.

```
public BufferUsage BufferUsage { get; }
```

Property Value

BufferUsage

IndexCount

Gets the total number of elements in this **IndexBuffer**.

```
public int IndexCount { get; }
```

Property Value

int

IndexElementSize

Gets a value indicating the size of each element in this **IndexBuffer**

```
public IndexElementSize IndexElementSize { get; }
```

Property Value

IndexElementSize

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GetData<T>(int, T[], int, int)

Copies the index buffer into an array.

```
public void GetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`offsetInBytes` int

The number of bytes into the index buffer where copying will start.

`data` T[]

The array to receive index buffer data.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

NotSupportedException

This **IndexBuffer** was created with a usage type of WriteOnly.

GetData<T>(T[])

Copies the index buffer into an array.

```
public void GetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array to receive index buffer data.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

NotSupportedException

This **IndexBuffer** was created with a usage type of WriteOnly.

GetData<T>(T[], int, int)

Copies the index buffer into an array.

```
public void GetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array to receive index buffer data.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

NotSupportedException

This **IndexBuffer** was created with a usage type of WriteOnly.

GraphicsDeviceResetting()

The GraphicsDevice is resetting, so GPU resources must be recreated.

```
protected override void GraphicsDeviceResetting()
```

SetDataInternal<T>(int, T[], int, int, SetDataOptions)

Copies array data to the index buffer.

```
protected void SetDataInternal<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, SetDataOptions options) where T : struct
```

Parameters

`offsetInBytes` int

Number of bytes into the index buffer where copying will start.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

`options` SetDataOptions

Specifies whether existing data in the buffer will be kept after this operation.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

SetData<T>(int, T[], int, int)

```
public void SetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`offsetInBytes` int

`data` T[]

`startIndex` int

`elementCount` int

Type Parameters

`T`

SetData<T>(T[])

Copies array data to the index buffer.

```
public void SetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

SetData<T>(T[], int, int)

Copies array data to the index buffer.

```
public void SetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of elements in the array.

Exceptions

ArgumentNullException

`data` is null.

InvalidOperationException

`data` is not the correct size for the amount of data requested.

Enum IndexElementSize

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines size for index in IndexBuffer and DynamicIndexBuffer.

```
public enum IndexElementSize
```

Fields

`SixteenBits = 0`

16-bit short/ushort value been used.

`ThirtyTwoBits = 1`

32-bit int/uint value been used.

Class Model

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

A basic 3D model with per mesh parent bones.

```
public sealed class Model
```

Inheritance

object ← Model

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Model(GraphicsDevice, List<ModelBone>, List<ModelMesh>)

Constructs a model.

```
public Model(GraphicsDevice graphicsDevice, List<ModelBone> bones,
List<ModelMesh> meshes)
```

Parameters

`graphicsDevice` GraphicsDevice

A valid reference to GraphicsDevice.

`bones` List<ModelBone>

The collection of bones.

`meshes` List<ModelMesh>

The collection of meshes.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

ArgumentNullException

`bones` is null.

ArgumentNullException

`meshes` is null.

Properties

Bones

A collection of ModelBone objects which describe how each mesh in the mesh collection for this model relates to its parent mesh.

```
public ModelBoneCollection Bones { get; }
```

Property Value

ModelBoneCollection

Meshes

A collection of ModelMesh objects which compose the model. Each ModelMesh in a model may be moved independently and may be composed of multiple materials identified as ModelMeshPart objects.

```
public ModelMeshCollection Meshes { get; }
```

Property Value

ModelMeshCollection

Root

Root bone for this model.

```
public ModelBone Root { get; set; }
```

Property Value

ModelBone

Tag

Custom attached object. Skinning data is example of attached object for model.

```
public object Tag { get; set; }
```

Property Value

object

Methods

CopyAbsoluteBoneTransformsTo(Matrix[])

Copies bone transforms relative to all parent bones of the each bone from this model to a given array.

```
public void CopyAbsoluteBoneTransformsTo(Matrix[] destinationBoneTransforms)
```

Parameters

`destinationBoneTransforms` Matrix[]

The array receiving the transformed bones.

CopyBoneTransformsFrom(Matrix[])

Copies bone transforms relative to Root bone from a given array to this model.

```
public void CopyBoneTransformsFrom(Matrix[] sourceBoneTransforms)
```

Parameters

`sourceBoneTransforms` Matrix[]

The array of prepared bone transform data.

Exceptions

ArgumentNullException

`sourceBoneTransforms` is null.

ArgumentOutOfRangeException

`sourceBoneTransforms` is invalid.

CopyBoneTransformsTo(Matrix[])

Copies bone transforms relative to Root bone from this model to a given array.

```
public void CopyBoneTransformsTo(Matrix[] destinationBoneTransforms)
```

Parameters

`destinationBoneTransforms` Matrix[]

The array receiving the transformed bones.

Exceptions

ArgumentNullException

`destinationBoneTransforms` is null.

ArgumentOutOfRangeException

`destinationBoneTransforms` is invalid.

Draw(Matrix, Matrix, Matrix)

Draws the model meshes.

```
public void Draw(Matrix world, Matrix view, Matrix projection)
```

Parameters

`world` Matrix

The world transform.

`view` Matrix

The view transform.

`projection` Matrix

The projection transform.

Class ModelBone

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents bone data for a model.

```
public sealed class ModelBone
```

Inheritance

object ← ModelBone

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModelBone()

Creates a new collection of ModelBone to denote the child bones in this model.

```
public ModelBone()
```

Properties

Children

Gets a collection of bones that are children of this bone.

```
public ModelBoneCollection Children { get; }
```

Property Value

ModelBoneCollection

Index

Gets the index of this bone in the Model.Bones collection.

```
public int Index { get; set; }
```

Property Value

int

Meshes

List of the meshes for this bone.

```
public List<ModelMesh> Meshes { get; }
```

Property Value

List<ModelMesh>

ModelTransform

Transform of this node from the root of the model not from the parent

```
public Matrix ModelTransform { get; set; }
```

Property Value

Matrix

Name

Gets the name of this bone.

```
public string Name { get; set; }
```

Property Value

string

Parent

Gets the parent of this bone.

```
public ModelBone Parent { get; set; }
```

Property Value

ModelBone

Transform

Gets or sets the matrix used to transform this bone relative to its parent bone.

```
public Matrix Transform { get; set; }
```

Property Value

Matrix

Methods

AddChild(ModelBone)

Adds a new child bone to this bone.

```
public void AddChild(ModelBone modelBone)
```

Parameters

`modelBone` ModelBone

ModelBone to be added.

AddMesh(ModelMesh)

Add a ModelMesh to the mesh collection.

```
public void AddMesh(ModelMesh mesh)
```

Parameters

`mesh` ModelMesh

ModelMesh to be added

Class ModelBoneCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a set of bones associated with a model.

```
public class ModelBoneCollection : ReadOnlyCollection<ModelBone>, IList<ModelBone>,
ICollection<ModelBone>, IReadOnlyList<ModelBone>, IReadOnlyCollection<ModelBone>,
IEnumerable<ModelBone>, IList, ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelBone> ← ModelBoneCollection

Implements

IList<ModelBone>, ICollection<ModelBone>, IReadOnlyList<ModelBone>, IReadOnlyCollection<ModelBone>, IEnumerable<ModelBone>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelBone>.Contains(ModelBone) , ReadOnlyCollection<ModelBone>.CopyTo(ModelBone[], int) , ReadOnlyCollection<ModelBone>.IndexOf(ModelBone) , ReadOnlyCollection<ModelBone>.Count , ReadOnlyCollection<ModelBone>.Empty , ReadOnlyCollection<ModelBone>.this[int] , ReadOnlyCollection<ModelBone>.Items , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModelBoneCollection(IList<ModelBone>)

Initializes a new instance of the ModelBoneCollection class that is a read-only wrapper around the specified list.

```
public ModelBoneCollection(IList<ModelBone> list)
```

Parameters

`list` IList<ModelBone>

The list to wrap.

Properties

this[string]

Retrieves a ModelBone from the collection, given the name of the bone.

```
public ModelBone this[string boneName] { get; }
```

Parameters

`boneName` string

The name of the bone to retrieve.

Property Value

ModelBone

Methods

GetEnumerator()

Returns a ModelBoneCollection.Enumerator that can iterate through a collection.

```
public ModelBoneCollection.Enumerator GetEnumerator()
```

Returns

ModelBoneCollection.Enumerator

TryGetValue(string, out ModelBone)

Finds a bone with a given name if it exists in the collection.

```
public bool TryGetValue(string boneName, out ModelBone value)
```

Parameters

`boneName` string

The name of the bone to find.

`value` ModelBone

The bone named boneName, if found.

Returns

bool

true if the bone was found

Struct ModelBoneCollection.Enumerator

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provides the ability to iterate through the bones in an ModelBoneCollection.

```
public struct ModelBoneCollection.Enumerator : IEnumerator<ModelBone>,
IEnumerator, IDisposable
```

Implements

IEnumerator<ModelBone>, IEnumerator, IDisposable

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Current

Gets the current element in the ModelBoneCollection.

```
public ModelBone Current { get; }
```

Property Value

ModelBone

Methods

Dispose()

Immediately releases the unmanaged resources used by this object.

```
public void Dispose()
```

MoveNext()

Advances the enumerator to the next element of the ModelBoneCollection.

```
public bool MoveNext()
```

Returns

bool

Reset()

Sets the enumerator to its initial position, which is before the first element in the collection.

```
public void Reset()
```

Exceptions

InvalidOperationException

The collection was modified after the enumerator was created.

NotSupportedException

The enumerator does not support being reset.

Class ModelEffectCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of effects associated with a model.

```
public sealed class ModelEffectCollection : ReadOnlyCollection<Effect>,
IList<Effect>, ICollection<Effect>, IReadOnlyList<Effect>,
IReadOnlyCollection<Effect>, IEnumerable<Effect>, IList, ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<Effect> ← ModelEffectCollection

Implements

IList<Effect>, ICollection<Effect>, IReadOnlyList<Effect>, IReadOnlyCollection<Effect>, IEnumerable<Effect>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<Effect>.Contains(Effect) , ReadOnlyCollection<Effect>.CopyTo(Effect[], int) , ReadOnlyCollection<Effect>.IndexOf(Effect) , ReadOnlyCollection<Effect>.Count , ReadOnlyCollection<Effect>.Empty , ReadOnlyCollection<Effect>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetEnumerator()

Returns a ModelEffectCollection.Enumerator that can iterate through a collection.

```
public ModelEffectCollection.Enumerator GetEnumerator()
```

Returns

ModelEffectCollection.Enumerator

Struct ModelEffectCollection.Enumerator

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Enumerator to iterate through the ModelEffectCollection

```
public struct ModelEffectCollection.Enumerator : IEnumerator<Effect>,
IDisposable, IEnumerator
```

Implements

IEnumerator<Effect>, IDisposable, IEnumerator

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Current

Gets the element in the collection at the current position of the enumerator.

```
public Effect Current { get; }
```

Property Value

Effect

The element in the collection at the current position of the enumerator.

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

MoveNext()

Advances the enumerator to the next element of the collection.

```
public bool MoveNext()
```

Returns

bool

true if the enumerator was successfully advanced to the next element; false if the enumerator has passed the end of the collection.

Exceptions

InvalidOperationException

The collection was modified after the enumerator was created.

Class ModelMesh

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a mesh that is part of a Model.

```
public sealed class ModelMesh
```

Inheritance

object ← ModelMesh

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModelMesh(GraphicsDevice, List<ModelMeshPart>)

Creates a new instance of ModelMesh.

```
public ModelMesh(GraphicsDevice graphicsDevice, List<ModelMeshPart> parts)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphicss device.

`parts` List<ModelMeshPart>

Parts of this mesh.

Properties

BoundingSphere

Gets the BoundingSphere that encompasses this mesh.

```
public BoundingSphere BoundingSphere { get; set; }
```

Property Value

BoundingSphere

Effects

Gets a collection of effects associated with this mesh.

```
public ModelEffectCollection Effects { get; }
```

Property Value

ModelEffectCollection

MeshParts

Gets the collection of ModelMeshPart objects that make up this mesh. Each part of a mesh is composed of a set of primitives that share the same material.

```
public ModelMeshPartCollection MeshParts { get; set; }
```

Property Value

ModelMeshPartCollection

Name

Gets the name of this mesh.

```
public string Name { get; set; }
```

Property Value

string

ParentBone

Gets the parent bone for this mesh.

The parent bone of a mesh contains a transformation matrix that describes how the mesh is located relative to any parent meshes in a model.

```
public ModelBone ParentBone { get; set; }
```

Property Value

ModelBone

Tag

Gets or sets an object identifying this mesh.

```
public object Tag { get; set; }
```

Property Value

object

Methods

Draw()

Draws all of the ModelMeshPart objects in this mesh, using their current Effect settings.

```
public void Draw()
```

Class ModelMeshCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of ModelMesh objects.

```
public sealed class ModelMeshCollection : ReadOnlyCollection<ModelMesh>,
IList<ModelMesh>, ICollection<ModelMesh>, IReadOnlyList<ModelMesh>,
IReadOnlyCollection<ModelMesh>, IEnumerable<ModelMesh>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelMesh> ← ModelMeshCollection

Implements

IList<ModelMesh>, ICollection<ModelMesh>, IReadOnlyList<ModelMesh>, IReadOnlyCollection<ModelMesh>, IEnumerable<ModelMesh>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelMesh>.Contains(ModelMesh) , ReadOnlyCollection<ModelMesh>.CopyTo(ModelMesh[], int) , ReadOnlyCollection<ModelMesh>.IndexOf(ModelMesh) , ReadOnlyCollection<ModelMesh>.Count , ReadOnlyCollection<ModelMesh>.Empty , ReadOnlyCollection<ModelMesh>.this[int] , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

this[string]

Retrieves a ModelMesh from the collection, given the name of the mesh.

```
public ModelMesh this[string meshName] { get; }
```

Parameters

`meshName` string

The name of the mesh to retrieve.

Property Value

ModelMesh

Methods

GetEnumerator()

Returns a ModelMeshCollection.Enumerator that can iterate through a collection.

```
public ModelMeshCollection.Enumerator GetEnumerator()
```

Returns

ModelMeshCollection.Enumerator

TryGetValue(string, out ModelMesh)

Finds a mesh with a given name if it exists in the collection.

```
public bool TryGetValue(string meshName, out ModelMesh value)
```

Parameters

`meshName` string

The name of the mesh to find.

`value` ModelMesh

The mesh named meshName, if found.

Returns

bool

true if a mesh was found

Struct ModelMeshCollection.Enumerator

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provides the ability to iterate through the bones in an ModelMeshCollection.

```
public struct ModelMeshCollection.Enumerator : IEnumerator<ModelMesh>,
IEnumerator, IDisposable
```

Implements

IEnumerator<ModelMesh>, IEnumerator, IDisposable

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Current

Gets the current element in the ModelMeshCollection.

```
public ModelMesh Current { get; }
```

Property Value

ModelMesh

Methods

Dispose()

Immediately releases the unmanaged resources used by this object.

```
public void Dispose()
```

MoveNext()

Advances the enumerator to the next element of the ModelMeshCollection.

```
public bool MoveNext()
```

Returns

bool

Reset()

Sets the enumerator to its initial position, which is before the first element in the collection.

```
public void Reset()
```

Exceptions

InvalidOperationException

The collection was modified after the enumerator was created.

NotSupportedException

The enumerator does not support being reset.

Class ModelMeshPart

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a batch of geometry information to submit to the graphics device during rendering. Each **ModelMeshPart** is a subdivision of a ModelMesh object. The ModelMesh class is split into multiple **ModelMeshPart** objects, typically based on material information.

```
public sealed class ModelMeshPart
```

Inheritance

object ← ModelMeshPart

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

It is not necessary to use this class directly. In advanced rendering scenarios, it is possible to draw using **ModelMeshPart** properties in combination with the vertex and index buffers on ModelMesh. However, in most cases, Draw() will be sufficient.

Constructors

ModelMeshPart()

Using this constructor is strongly discouraged. Adding meshes to models at runtime is not supported and may lead to NullReferenceExceptions if parent is not set.

```
[Obsolete("This constructor is deprecated and will be made internal in a
future release.")]
public ModelMeshPart()
```

Properties

Effect

Gets or sets the material Effect for this mesh part.

```
public Effect Effect { get; set; }
```

Property Value

Effect

IndexBuffer

Gets the index buffer for this mesh part.

```
public IndexBuffer IndexBuffer { get; set; }
```

Property Value

IndexBuffer

NumVertices

Gets the number of vertices used during a draw call.

```
public int NumVertices { get; set; }
```

Property Value

int

PrimitiveCount

Gets the number of primitives to render.

```
public int PrimitiveCount { get; set; }
```

Property Value

int

StartIndex

Gets the location in the index array at which to start reading vertices.

```
public int StartIndex { get; set; }
```

Property Value

int

Tag

Gets or sets an object identifying this model mesh part.

```
public object Tag { get; set; }
```

Property Value

object

VertexBuffer

Gets the vertex buffer for this mesh part.

```
public VertexBuffer VertexBuffer { get; set; }
```

Property Value

VertexBuffer

VertexOffset

Gets the offset (in vertices) from the top of vertex buffer.

```
public int VertexOffset { get; set; }
```

Property Value

int

Class ModelMeshPartCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of ModelMeshPart objects.

```
public sealed class ModelMeshPartCollection : ReadOnlyCollection<ModelMeshPart>,
IList<ModelMeshPart>, ICollection<ModelMeshPart>, IReadOnlyList<ModelMeshPart>,
IReadOnlyCollection<ModelMeshPart>, IEnumerable<ModelMeshPart>, IList,
ICollection, IEnumerable
```

Inheritance

object ← ReadOnlyCollection<ModelMeshPart> ← ModelMeshPartCollection

Implements

IList<ModelMeshPart>, ICollection<ModelMeshPart>, IReadOnlyList<ModelMeshPart>, IReadOnlyCollection<ModelMeshPart>, IEnumerable<ModelMeshPart>, IList, ICollection, IEnumerable

Inherited Members

ReadOnlyCollection<ModelMeshPart>.Contains(ModelMeshPart) ,
ReadOnlyCollection<ModelMeshPart>.CopyTo(ModelMeshPart[], int) ,
ReadOnlyCollection<ModelMeshPart>.GetEnumerator() ,
ReadOnlyCollection<ModelMeshPart>.IndexOf(ModelMeshPart) ,
ReadOnlyCollection<ModelMeshPart>.Count , ReadOnlyCollection<ModelMeshPart>.Empty ,
ReadOnlyCollection<ModelMeshPart>.this[int] , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModelMeshPartCollection(IList<ModelMeshPart>)

Initializes a new instance of the ModelMeshPartCollection class that is a read-only wrapper around the specified list.

```
public ModelMeshPartCollection(IList<ModelMeshPart> list)
```

Parameters

`list` IList<ModelMeshPart>

The list to wrap.

Class NoSuitableGraphicsDeviceException

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Thrown when no available graphics device fits the given device preferences.

```
[DataContract]
public sealed class NoSuitableGraphicsDeviceException : Exception, ISerializable
```

Inheritance

object ← Exception ← NoSuitableGraphicsDeviceException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

NoSuitableGraphicsDeviceException()

Initializes a new instance of the NoSuitableGraphicsDeviceException class.

```
public NoSuitableGraphicsDeviceException()
```

NoSuitableGraphicsDeviceException(string)

Initializes a new instance of the NoSuitableGraphicsDeviceException class with a specified error message.

```
public NoSuitableGraphicsDeviceException(string message)
```

Parameters

`message` string

The message that describes the error.

NoSuitableGraphicsDeviceException(string, Exception)

Initializes a new instance of the NoSuitableGraphicsDeviceException with a specified error message and a reference to the inner exception that is the cause of this exception.

```
public NoSuitableGraphicsDeviceException(string message, Exception inner)
```

Parameters

`message` string

The message that describes the error.

`inner` Exception

The exception that is the cause of the current exception, or a null reference if no inner exception is specified.

Class OcclusionQuery

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Used to perform an occlusion query against the latest drawn objects.

```
public class OcclusionQuery : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← OcclusionQuery

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

OcclusionQuery(GraphicsDevice)

Initializes a new instance of the OcclusionQuery class.

```
public OcclusionQuery(GraphicsDevice graphicsDevice)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

Exceptions

ArgumentNullException

`graphicsDevice` is null.

NotSupportedException

The current graphics profile does not support occlusion queries.

Properties

IsComplete

Gets a value indicating whether the occlusion query has completed.

```
public bool IsComplete { get; }
```

Property Value

bool

true if the occlusion query has completed; otherwise, false.

PixelCount

Gets the number of visible pixels.

```
public int PixelCount { get; }
```

Property Value

int

The number of visible pixels.

Exceptions

InvalidOperationException

The occlusion query has not yet completed. Check IsComplete before reading the result!

Methods

Begin()

Begins the occlusion query.

```
public void Begin()
```

Exceptions

InvalidOperationException

Begin() is called again before calling End().

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

End()

Ends the occlusion query.

```
public void End()
```

Exceptions

InvalidOperationException

End() is called before calling Begin().

Enum PresentInterval

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines how Present() updates the game window.

```
public enum PresentInterval
```

Fields

`Default = 0`

Equivalent to One.

`Immediate = 3`

The driver updates the window client area immediately. Present operations might be affected immediately. There is no limit for framerate.

`One = 1`

The driver waits for the vertical retrace period, before updating window client area. Present operations are not affected more frequently than the screen refresh rate.

`Two = 2`

The driver waits for the vertical retrace period, before updating window client area. Present operations are not affected more frequently than every second screen refresh.

Class PresentationParameters

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains graphics presentation parameters.

```
public class PresentationParameters
```

Inheritance

object ← PresentationParameters

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PresentationParameters()

Create a PresentationParameters instance with default values for all properties.

```
public PresentationParameters()
```

Fields

DefaultPresentRate

Default presentation rate

```
public const int DefaultPresentRate = 60
```

Field Value

int

Properties

BackBufferFormat

Get or set the format of the back buffer.

```
public SurfaceFormat BackBufferFormat { get; set; }
```

Property Value

SurfaceFormat

BackBufferHeight

Get or set the height of the back buffer.

```
public int BackBufferHeight { get; set; }
```

Property Value

int

BackBufferWidth

Get or set the width of the back buffer.

```
public int BackBufferWidth { get; set; }
```

Property Value

int

Bounds

Get the bounds of the back buffer.

```
public Rectangle Bounds { get; }
```

Property Value

Rectangle

DepthStencilFormat

Get or set the depth stencil format for the back buffer.

```
public DepthFormat DepthStencilFormat { get; set; }
```

Property Value

DepthFormat

DeviceWindowHandle

Get or set the handle of the window that will present the back buffer.

```
public nint DeviceWindowHandle { get; set; }
```

Property Value

nint

DisplayOrientation

Get or set the display orientation.

```
public DisplayOrientation DisplayOrientation { get; set; }
```

Property Value

DisplayOrientation

HardwareModeSwitch

If

`true`

the GraphicsDevice will do a mode switch when going to full screen mode. If

`false`

it will instead do a soft full screen by maximizing the window and making it borderless.

```
public bool HardwareModeSwitch { get; set; }
```

Property Value

bool

IsFullScreen

Get or set a value indicating if we are in full screen mode.

```
public bool IsFullScreen { get; set; }
```

Property Value

bool

MultiSampleCount

Get or set the multisample count for the back buffer.

```
public int MultiSampleCount { get; set; }
```

Property Value

int

PresentationInterval

Get or set the presentation interval.

```
public PresentInterval PresentationInterval { get; set; }
```

Property Value

PresentInterval

RenderTargetUsage

Get or set the RenderTargetUsage for the back buffer. Determines if the back buffer is cleared when it is set as the render target by the GraphicsDevice. GraphicsDevice target.

```
public RenderTargetUsage RenderTargetUsage { get; set; }
```

Property Value

RenderTargetUsage

Methods

Clear()

Reset all properties to their default values.

```
public void Clear()
```

Clone()

Create a copy of this PresentationParameters instance.

```
public PresentationParameters Clone()
```

Returns

PresentationParameters

Enum PrimitiveType

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines how vertex data is ordered.

```
public enum PrimitiveType
```

Fields

`LineList = 2`

Renders the vertices as a list of isolated straight line segments; the count may be any positive integer.

`LineStrip = 3`

Renders the vertices as a single polyline; the count may be any positive integer.

`PointList = 4`

Renders the vertices as individual points; the count may be any positive integer.

`TriangleList = 0`

Renders the specified vertices as a sequence of isolated triangles. Each group of three vertices defines a separate triangle. Back-face culling is affected by the current winding-order render state.

`TriangleStrip = 1`

Renders the vertices as a triangle strip. The back-face culling flag is flipped automatically on even-numbered triangles.

Class RasterizerState

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains rasterizer state, which determines how to convert vector data (shapes) into raster data (pixels).

```
public class RasterizerState : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← RasterizerState

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

RasterizerState()

Creates a new instance of **RasterizerState**.

```
public RasterizerState()
```

Fields

CullClockwise

A built-in state object with settings for culling primitives with clockwise winding order.

```
public static readonly RasterizerState CullClockwise
```

Field Value

RasterizerState

Remarks

This build-in state object has the following settings

```
CullMode = CullMode.CullClockwiseFace
```

CullCounterClockwise

A built-in state object with settings for culling primitives with counter-clockwise winding order.

```
public static readonly RasterizerState CullCounterClockwise
```

Field Value

RasterizerState

Remarks

This build-in state object has the following settings

```
CullMode = CullMode.CullCounterClockwiseFace
```

CullNone

A built-in state object with settings for not culling any primitives.

```
public static readonly RasterizerState CullNone
```

Field Value

RasterizerState

Remarks

This build-in state object has the following settings

```
CullMode = CullMode.None
```

Properties

CullMode

Gets or Sets the conditions for culling or removing triangles. The default value is CullCounterClockwise Face

```
public CullMode CullMode { get; set; }
```

Property Value

CullMode

DepthBias

Gets or Sets the depth bias for polygons, which is the amount of bias to apply to the depth of a primitive to alleviate depth testing problems for primitives of similar depth. The default value is `0.0f`.

```
public float DepthBias { get; set; }
```

Property Value

float

Remarks

A polygon with a larger z-bias value appears in front of a polygon with a smaller value. For example, a a polygon with a value of `1.0f` appears drawn in front of a polygon with a value of `0.0f`

DepthClipEnable

Gets or Sets a value that indicates whether depth clipping is enabled.

```
public bool DepthClipEnable { get; set; }
```

Property Value

bool

FillMode

Gets or Sets the fille mode, which defines how a triangle is filled during rendering. The default is Solid.

```
public FillMode FillMode { get; set; }
```

Property Value

FillMode

MultiSampleAntiAlias

Gets or Sets a value that indicates whether multisample antialiasing is enabled.

```
public bool MultiSampleAntiAlias { get; set; }
```

Property Value

bool

Remarks

When multisample antialiasing is enabled, full-scene antialiasing is performed by calculating sampling locations at different sample positions for each multiple sample. When disabled, each multiple sample is written with the same sample value (sampled at a pixel center) which allows non-antialiased rendering to a mutlisample buffer. This property has no effect when rendering to a buffer that does not support multisampling.

ScissorTestEnable

Gets or Sets a value that indicates whether scissor testing is enabled.

```
public bool ScissorTestEnable { get; set; }
```

Property Value

bool

Remarks

Scissor testing can improve drawing performance by only drawing triangles (or parts of triangles) that are contained within a GraphicsDevice.ScissorRectangle

SlopeScaleDepthBias

Gets or Sets a bias value that takes into account the slope of a polygon. This bias value is applied to coplanar primitives to reduce aliasing and other rendering artifacts caused by z-fighting.

```
public float SlopeScaleDepthBias { get; set; }
```

Property Value

float

Remarks

An application can help ensure that coplanar polygons are rendered property by adding a bias to the z-values that the system uses when rendering sets of coplanar polygons. The following formula shows how to calculate the bias to be applied to coplanar primitives

```
bias = (m x SlopeScaleDepthBias) + DepthBias
```

Where *m* is the maximum depth slope of the triangle being rendered, defined as

```
m = max( abs(delta z / delta x), abs(delta z / delta y) )
```

Methods

Dispose(bool)

The method that derived classes should override to implement disposing of managed and native resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if managed objects should be disposed.

Remarks

Native resources should always be released regardless of the value of the disposing parameter.

Class RenderTarget2D

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a 2D texture resource that will be written to at the end of a render pass.

```
public class RenderTarget2D : Texture2D, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture ← Texture2D ← RenderTarget2D

Implements

IDisposable

Inherited Members

Texture2D.Bounds , Texture2D.Width , Texture2D.Height ,
Texture2D.SetData<T>(int, int, Rectangle?, T[], int, int) ,
Texture2D.SetData<T>(int, Rectangle?, T[], int, int) , Texture2D.SetData<T>(T[], int, int) ,
Texture2D.SetData<T>(T[]) , Texture2D.GetData<T>(int, int, Rectangle?, T[], int, int) ,
Texture2D.GetData<T>(int, Rectangle?, T[], int, int) , Texture2D.GetData<T>(T[], int, int) ,
Texture2D.GetData<T>(T[]) , Texture2D.FromFile(GraphicsDevice, string, Action<byte[]>) ,
Texture2D.FromFile(GraphicsDevice, string) ,
Texture2D.FromStream(GraphicsDevice, Stream, Action<byte[]>) ,
Texture2D.FromStream(GraphicsDevice, Stream) , Texture2D.SaveAsJpeg(Stream, int, int) ,
Texture2D.SaveAsPng(Stream, int, int) , Texture2D.Reload(Stream) , Texture.Format ,
Texture.LevelCount , GraphicsResource.Dispose() , GraphicsResource.Disposing ,
GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name ,
GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

After a render pass the render target contains the color information of a rendered image.

Render targets represent a linear area of display memory and usually resides in the display memory of the display card. Because of this, objects must be recreated when the device is reset.

Constructors

RenderTarget2D(GraphicsDevice, int, int)

Creates a new **RenderTarget2D** instance with the specified parameters.

```
public RenderTarget2D(GraphicsDevice graphicsDevice, int width, int height)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to associate with this render target resource.

`width` int

Width, in pixels, of the render target.

`height` int

Height, in pixels, of the render target.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` parameters less than or equal to zero.

RenderTarget2D(GraphicsDevice, int, int, bool, SurfaceFormat, DepthFormat)

Creates a new **RenderTarget2D** instance with the specified parameters.

```
public RenderTarget2D(GraphicsDevice graphicsDevice, int width, int height, bool
mipMap, SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to associate with this render target resource.

`width` int

Width, in pixels, of the render target.

`height` int

Height, in pixels, of the render target.

`mipMap` bool

true if mipmapping is enabled; otherwise, **false**.

`preferredFormat` SurfaceFormat

The preferred surface format of the render target.

`preferredDepthFormat` DepthFormat

The preferred depth format of the render target.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` parameters less than or equal to zero.

RenderTarget2D(GraphicsDevice, int, int, bool, SurfaceFormat, DepthFormat, int, RenderTargetUsage)

Creates a new **RenderTarget2D** instance with the specified parameters.

```
public RenderTarget2D(GraphicsDevice graphicsDevice, int width, int height, bool
mipMap, SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat, int
preferredMultiSampleCount, RenderTargetUsage usage)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to associate with this render target resource.

`width` int

Width, in pixels, of the render target.

`height` int

Height, in pixels, of the render target.

`mipMap` bool

true if mipmapping is enabled; otherwise, **false**.

`preferredFormat` SurfaceFormat

The preferred surface format of the render target.

`preferredDepthFormat` DepthFormat

The preferred depth format of the render target.

`preferredMultiSampleCount` int

The preferred number of samples per pixel when multisampling.

`usage` RenderTargetUsage

The behavior to use when binding the render target to the graphics device.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` parameters less than or equal to zero.

RenderTarget2D(GraphicsDevice, int, int, bool, SurfaceFormat, DepthFormat, int, RenderTargetUsage, SurfaceType)

Allows child class to specify the surface type, eg: a swap chain.

```
protected RenderTarget2D(GraphicsDevice graphicsDevice, int width, int
height, bool mipMap, SurfaceFormat format, DepthFormat depthFormat, int
preferredMultiSampleCount, RenderTargetUsage usage, Texture2D.SurfaceType
surfaceType)
```

Parameters

`graphicsDevice` GraphicsDevice

`width` int

`height` int

`mipMap` bool

`format` SurfaceFormat

`depthFormat` DepthFormat

`preferredMultiSampleCount` int

`usage` RenderTargetUsage

`surfaceType` Texture2D.SurfaceType

RenderTarget2D(GraphicsDevice, int, int, bool, SurfaceFormat, DepthFormat, int, RenderTargetUsage, bool)

Creates a new **RenderTarget2D** instance with the specified parameters.

```
public RenderTarget2D(GraphicsDevice graphicsDevice, int width, int height, bool
mipMap, SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat, int
preferredMultiSampleCount, RenderTargetUsage usage, bool shared)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to associate with this render target resource.

`width` int

Width, in pixels, of the render target.

`height` int

Height, in pixels, of the render target.

`mipMap` bool

true if mipmapping is enabled; otherwise, **false**.

`preferredFormat` SurfaceFormat

The preferred surface format of the render target.

`preferredDepthFormat` DepthFormat

The preferred depth format of the render target.

`preferredMultiSampleCount` int

The preferred number of samples per pixel when multisampling.

`usage` RenderTargetUsage

The behavior to use when binding the render target to the graphics device.

`shared` bool

Whether this render target resource should be a shared resource accessible on another device. This property is only valid for DirectX targets.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` parameters less than or equal to zero.

RenderTarget2D(GraphicsDevice, int, int, bool, SurfaceFormat, DepthFormat, int, RenderTargetUsage, bool, int)

Creates a new **RenderTarget2D** instance with the specified parameters.

```
public RenderTarget2D(GraphicsDevice graphicsDevice, int width, int height, bool
mipMap, SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat, int
preferredMultiSampleCount, RenderTargetUsage usage, bool shared, int arraySize)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to associate with this render target resource.

`width` int

Width, in pixels, of the render target.

`height` int

Height, in pixels, of the render target.

`mipMap` bool

true if mipmapping is enabled; otherwise, **false**.

`preferredFormat` SurfaceFormat

The preferred surface format of the render target.

`preferredDepthFormat` DepthFormat

The preferred depth format of the render target.

`preferredMultiSampleCount` int

The preferred number of samples per pixel when multisampling.

`usage` RenderTargetUsage

The behavior to use when binding the render target to the graphics device.

`shared` bool

Whether this render target resource should be a shared resource accessible on another device. This property is only valid for DirectX targets.

`arraySize` int

The size of the texture array.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` parameters less than or equal to zero.

ArgumentException

`arraySize` is greater than 0 and the graphics device does not support texture arrays.

Properties

DepthStencilFormat

Gets the depth format used by this **RenderTarget2D**

```
public DepthFormat DepthStencilFormat { get; }
```

Property Value

DepthFormat

IsContentLost

Gets a value that indicates whether the contents of this **RenderTarget2D** has been lost due to a lost device event.

```
[Obsolete("This is provided for XNA compatibility only and will always
return false")]
public bool IsContentLost { get; }
```

Property Value

bool

Remarks

This property will always return **false**. It is included for XNA compatibility.

MultiSampleCount

Gets the number of samples taken per pixel

```
public int MultiSampleCount { get; }
```

Property Value

int

RenderTargetUsage

Gets or Sets the usage type used by this **RenderTarget2D**

```
public RenderTargetUsage RenderTargetUsage { get; }
```

Property Value

RenderTargetUsage

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GraphicsDeviceResetting()

Called before the device is reset. Allows graphics resources to invalidate their state so they can be recreated after the device reset. Warning: This may be called after a call to Dispose() up until the resource is garbage collected.

```
protected override void GraphicsDeviceResetting()
```

QuerySelectedFormat(GraphicsDevice, SurfaceFormat)

```
protected static SurfaceFormat QuerySelectedFormat(GraphicsDevice graphicsDevice,
SurfaceFormat preferredFormat)
```

Parameters

`graphicsDevice` GraphicsDevice

`preferredFormat` SurfaceFormat

Returns

SurfaceFormat

Events

ContentLost

Occurs when a graphics device lost event is triggered.

```
[Obsolete("This is provided for XNA compatibility is never called by MonoGame")]
public event EventHandler<EventArgs> ContentLost
```

Event Type

EventHandler<EventArgs>

Remarks

This event is never called. It is included for XNA compatibility.

Struct RenderTargetBinding

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents the binding of a RenderTarget2D used in an array of bindings when setting multiple render targets on the graphics device.

```
public struct RenderTargetBinding
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

RenderTargetBinding(RenderTarget2D)

Creates a new **RenderTargetBinding** for the specified render target.

```
public RenderTargetBinding(RenderTarget2D renderTarget)
```

Parameters

`renderTarget` RenderTarget2D

The render target to create the binding for.

Exceptions

ArgumentNullException

The `renderTarget` parameter is null.

RenderTargetBinding(RenderTargetCube, CubeMapFace)

Creates a new **RenderTargetBinding** with the specified parameters.

```
public RenderTargetBinding(RenderTargetCube renderTarget, CubeMapFace cubeMapFace)
```

Parameters

`renderTarget` RenderTargetCube

A cube map render target.

`cubeMapFace` CubeMapFace

The cube map of the render target.

Exceptions

ArgumentNullException

The `renderTarget` parameter is null.

ArgumentOutOfRangeException

The `cubeMapFace` parameter is less than PositiveX or greater than NegativeZ.

Properties

ArraySlice

```
public int ArraySlice { get; }
```

Property Value

int

RenderTarget

Gets the the render target texture

```
public Texture RenderTarget { get; }
```

Property Value

Texture

Operators

implicit operator RenderTargetBinding(RenderTarget2D)

Implicitly converts a RenderTarget2D instance to a **RenderTargetBinding** instance.

```
public static implicit operator RenderTargetBinding(RenderTarget2D renderTarget)
```

Parameters

`renderTarget` RenderTarget2D

The render target to convert.

Returns

RenderTargetBinding

A new **RenderTargetBinding** instance bound to the specified render target.

Class RenderTargetCube

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a texture cube that can be used as a render target.

```
public class RenderTargetCube : TextureCube, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture ← TextureCube ← RenderTargetCube

Implements

IDisposable

Inherited Members

TextureCube.Size , TextureCube.GetData<T>(CubeMapFace, T[]) , TextureCube.GetData<T>(CubeMapFace, T[], int, int) , TextureCube.GetData<T>(CubeMapFace, int, Rectangle?, T[], int, int) , TextureCube.SetData<T>(CubeMapFace, T[]) , TextureCube.SetData<T>(CubeMapFace, T[], int, int) , TextureCube.SetData<T>(CubeMapFace, int, Rectangle?, T[], int, int) , Texture.Format , Texture.LevelCount , Texture.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

RenderTargetCube(GraphicsDevice, int, bool, SurfaceFormat, DepthFormat)

Initializes a new instance of the RenderTargetCube class.

```
public RenderTargetCube(GraphicsDevice graphicsDevice, int size, bool mipMap,
SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

`size` int

The width and height of a texture cube face in pixels.

`mipMap` bool

true to generate a full mipmap chain; otherwise false.

`preferredFormat` SurfaceFormat

The preferred format of the surface.

`preferredDepthFormat` DepthFormat

The preferred format of the depth-stencil buffer.

RenderTargetCube(GraphicsDevice, int, bool, SurfaceFormat, DepthFormat, int, RenderTargetUsage)

Initializes a new instance of the RenderTargetCube class.

```
public RenderTargetCube(GraphicsDevice graphicsDevice, int size, bool mipMap,
SurfaceFormat preferredFormat, DepthFormat preferredDepthFormat, int
preferredMultiSampleCount, RenderTargetUsage usage)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

`size` int

The width and height of a texture cube face in pixels.

`mipMap` bool

true to generate a full mipmap chain; otherwise false.

`preferredFormat` SurfaceFormat

The preferred format of the surface.

`preferredDepthFormat` DepthFormat

The preferred format of the depth-stencil buffer.

`preferredMultiSampleCount` int

The preferred number of multisample locations.

`usage` RenderTargetUsage

The usage mode of the render target.

Properties

DepthStencilFormat

Gets the depth-stencil buffer format of this render target.

```
public DepthFormat DepthStencilFormat { get; }
```

Property Value

DepthFormat

The format of the depth-stencil buffer.

IsContentLost

Gets a value that indicates whether the contents of this **RenderTargetCube** has been lost due to a lost device event.

```
[Obsolete("This is provided for XNA compatibility only and will always
return false")]
public bool IsContentLost { get; }
```

Property Value

bool

Remarks

This property will always return **false**. It is included for XNA compatibility.

MultiSampleCount

Gets the number of multisample locations.

```
public int MultiSampleCount { get; }
```

Property Value

int

The number of multisample locations.

RenderTargetUsage

Gets the usage mode of this render target.

```
public RenderTargetUsage RenderTargetUsage { get; }
```

Property Value

RenderTargetUsage

The usage mode of the render target.

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

QuerySelectedFormat(GraphicsDevice, SurfaceFormat)

```
protected static SurfaceFormat QuerySelectedFormat(GraphicsDevice graphicsDevice,
SurfaceFormat preferredFormat)
```

Parameters

`graphicsDevice` GraphicsDevice

`preferredFormat` SurfaceFormat

Returns

SurfaceFormat

Events

ContentLost

Occurs when a graphics device lost event is triggered.

```
[Obsolete("This is provided for XNA compatibility is never called by MonoGame")]
public event EventHandler<EventArgs> ContentLost
```

Event Type

EventHandler<EventArgs>

Remarks

This event is never called. It is included for XNA compatibility.

Enum RenderTargetUsage

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines if the previous content in a render target is preserved when it set on the graphics device.

```
public enum RenderTargetUsage
```

Fields

`DiscardContents = 0`

The render target content will not be preserved.

`PlatformContents = 2`

The render target content might be preserved if the platform can do so without a penalty in performance or memory usage.

`PreserveContents = 1`

The render target content will be preserved even if it is slow or requires extra memory.

Class ResourceCreatedEventArgs

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provides data for the ResourceCreated event. This class cannot be inherited.

```
public sealed class ResourceCreatedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← ResourceCreatedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Resource

The newly created resource object.

```
public object Resource { get; }
```

Property Value

object

Class ResourceDestroyedEventArgs

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Provides data for the ResourceDestroyed event. This class cannot be inherited.

```
public sealed class ResourceDestroyedEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← ResourceDestroyedEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Name

The name of the destroyed resource.

```
public string Name { get; }
```

Property Value

string

Tag

The resource manager tag of the destroyed resource.

```
public object Tag { get; }
```

Property Value

object

Class SamplerState

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Contains sampler state, which determines how to sample texture data.

```
public class SamplerState : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← SamplerState

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

SamplerState()

Creates a new instance of the SamplerState class with default values equivalent to LinearWrap.

```
public SamplerState()
```

Fields

AnisotropicClamp

Contains default state for anisotropic filtering and texture coordinate clamping.

```
public static readonly SamplerState AnisotropicClamp
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Anisotropic,
AddressU = TextureAddressMode.Clamp,
AddressV = TextureAddressMode.Clamp,
AddressW = TextureAddressMode.Clamp,
```

AnisotropicWrap

Contains default state for anisotropic filtering and texture coordinate wrapping.

```
public static readonly SamplerState AnisotropicWrap
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Anisotropic,
AddressU = TextureAddressMode.Wrap,
AddressV = TextureAddressMode.Wrap,
AddressW = TextureAddressMode.Wrap,
```

LinearClamp

Contains default state for linear filtering and texture coordinate clamping.

```
public static readonly SamplerState LinearClamp
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Linear,
AddressU = TextureAddressMode.Clamp,
AddressV = TextureAddressMode.Clamp,
AddressW = TextureAddressMode.Clamp,
```

LinearWrap

Contains default state for linear filtering and texture coordinate wrapping.

```
public static readonly SamplerState LinearWrap
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Linear,
AddressU = TextureAddressMode.Wrap,
AddressV = TextureAddressMode.Wrap,
AddressW = TextureAddressMode.Wrap,
```

PointClamp

Contains default state for point filtering and texture coordinate clamping.

```
public static readonly SamplerState PointClamp
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Point,
AddressU = TextureAddressMode.Clamp,
AddressV = TextureAddressMode.Clamp,
AddressW = TextureAddressMode.Clamp,
```

PointWrap

Contains default state for point filtering and texture coordinate wrapping.

```
public static readonly SamplerState PointWrap
```

Field Value

SamplerState

Remarks

This built-in state object has the following settings:

```
Filter = TextureFilter.Point,
AddressU = TextureAddressMode.Wrap,
AddressV = TextureAddressMode.Wrap,
AddressW = TextureAddressMode.Wrap,
```

Properties

AddressU

Gets or sets the texture-address mode for the u-coordinate.

```
public TextureAddressMode AddressU { get; set; }
```

Property Value

TextureAddressMode

AddressV

Gets or sets the texture-address mode for the v-coordinate.

```
public TextureAddressMode AddressV { get; set; }
```

Property Value

TextureAddressMode

AddressW

Gets or sets the texture-address mode for the w-coordinate.

```
public TextureAddressMode AddressW { get; set; }
```

Property Value

TextureAddressMode

BorderColor

Gets or sets the color to use for texels outside of range, when Border is used.

```
public Color BorderColor { get; set; }
```

Property Value

Color

ComparisonFunction

When using comparison sampling, also set FilterMode to Comparison.

```
public CompareFunction ComparisonFunction { get; set; }
```

Property Value

CompareFunction

Filter

Gets or sets the type of filtering during sampling.

```
public TextureFilter Filter { get; set; }
```

Property Value

TextureFilter

FilterMode

Gets or sets filtering mode for texture samplers.

```
public TextureFilterMode FilterMode { get; set; }
```

Property Value

TextureFilterMode

MaxAnisotropy

Gets or sets the maximum anisotropy. The default value is 4.

```
public int MaxAnisotropy { get; set; }
```

Property Value

int

Remarks

Use anisotropic filtering to reduce blur and aliasing effects when texturing a surface that will be viewed at an extreme viewing angle.

MaxMipLevel

Gets or sets the level of detail (LOD) index of the largest map to use.

```
public int MaxMipLevel { get; set; }
```

Property Value

int

The maximum LOD, which ranges from 0 to n-1, where n is the index of the largest map.

MipMapLevelOfDetailBias

Gets or sets the mipmap LOD bias. The default value is 0.

A negative value indicates a larger mipmap level; a positive value indicates a smaller mipmap level.

```
public float MipMapLevelOfDetailBias { get; set; }
```

Property Value

float

Remarks

Mipmap LOD bias offsets the mipmap level from which a texture is sampled (the result is computed using trilinear texturing between the nearest two levels).

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

Class SamplerStateCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of SamplerState objects,

```
public sealed class SamplerStateCollection
```

Inheritance

object ← SamplerStateCollection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

this[int]

Gets or sets the SamplerState at the specified index in the collection.

```
public SamplerState this[int index] { get; set; }
```

Parameters

`index` int

Property Value

SamplerState

Enum SetDataOptions

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines how vertex or index buffer data will be flushed during a SetData operation.

```
public enum SetDataOptions
```

Fields

`Discard = 1`

The SetData will discard the entire buffer. A pointer to a new memory area is returned and rendering from the previous area do not stall.

`NoOverwrite = 2`

The SetData operation will not overwrite existing data. This allows the driver to return immediately from a SetData operation and continue rendering.

`None = 0`

The SetData can overwrite the portions of existing data.

Class ShaderCompilerException

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

This exception occurs when a shader compiled at runtime has errors. It should provide enough information to help diagnose the bug.

```
[DataContract]
public sealed class ShaderCompilerException : Exception, ISerializable
```

Inheritance

object ← Exception ← ShaderCompilerException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.ReferenceEquals(object, object)

Constructors

ShaderCompilerException(string, string, ShaderStage, string, string)

Constructor.

```
public ShaderCompilerException(string sourceFile, string entrypoint, ShaderStage
stage, string errors, string sourceCode)
```

Parameters

`sourceFile` string

The relative shader source file.

`entrypoint` string

The shader entrypoint function.

`stage` ShaderStage

The shader stage.

`errors` string

The error logging from the shader compiler.

`sourceCode` string

The source code that was being compiled.

Properties

Entrypoint

The shader entrypoint function.

```
public string Entrypoint { get; }
```

Property Value

string

Errors

The error logging from the shader compiler.

```
public string Errors { get; }
```

Property Value

string

SourceCode

The source code being compiled.

```
public string SourceCode { get; }
```

Property Value

string

SourceFile

The relative shader source file.

```
public string SourceFile { get; }
```

Property Value

string

Stage

The shader stage.

```
public ShaderStage Stage { get; }
```

Property Value

ShaderStage

Methods

ToString()

Creates and returns a string representation of the current exception.

```
public override string ToString()
```

Returns

string

A string representation of the current exception.

Enum ShaderStage

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

```
public enum ShaderStage
```

Fields

`Count = 2`

`Pixel = 1`

`Vertex = 0`

Class SkinnedEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Built-in effect for rendering skinned character models.

```
public class SkinnedEffect : Effect, IDisposable, IEffectMatrices, IEffectLights,
IEffectFog, IEffectBones
```

Inheritance

object ← GraphicsResource ← Effect ← SkinnedEffect

Implements

IDisposable, IEffectMatrices, IEffectLights, IEffectFog, IEffectBones

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

SkinnedEffect(GraphicsDevice)

Creates a new SkinnedEffect with default parameter settings.

```
public SkinnedEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

SkinnedEffect(SkinnedEffect)

Creates a new SkinnedEffect by cloning parameter settings from an existing instance.

```
protected SkinnedEffect(SkinnedEffect cloneSource)
```

Parameters

`cloneSource` SkinnedEffect

Fields

MaxBones

The maximum number of bones.

```
public const int MaxBones = 72
```

Field Value

int

Properties

Alpha

Gets or sets the material alpha.

```
public float Alpha { get; set; }
```

Property Value

float

AmbientLightColor

Gets or sets the ambient light color (range 0 to 1).

```
public Vector3 AmbientLightColor { get; set; }
```

Property Value

Vector3

DiffuseColor

Gets or sets the material diffuse color (range 0 to 1).

```
public Vector3 DiffuseColor { get; set; }
```

Property Value

Vector3

DirectionalLight0

Gets the first directional light.

```
public DirectionalLight DirectionalLight0 { get; }
```

Property Value

DirectionalLight

DirectionalLight1

Gets the second directional light.

```
public DirectionalLight DirectionalLight1 { get; }
```

Property Value

DirectionalLight

DirectionalLight2

Gets the third directional light.

```
public DirectionalLight DirectionalLight2 { get; }
```

Property Value

DirectionalLight

EmissiveColor

Gets or sets the material emissive color (range 0 to 1).

```
public Vector3 EmissiveColor { get; set; }
```

Property Value

Vector3

FogColor

Gets or sets the fog color.

```
public Vector3 FogColor { get; set; }
```

Property Value

Vector3

FogEnabled

Gets or sets the fog enable flag.

```
public bool FogEnabled { get; set; }
```

Property Value

bool

FogEnd

Gets or sets the fog end distance.

```
public float FogEnd { get; set; }
```

Property Value

float

FogStart

Gets or sets the fog start distance.

```
public float FogStart { get; set; }
```

Property Value

float

PreferPerPixelLighting

Gets or sets the per-pixel lighting prefer flag.

```
public bool PreferPerPixelLighting { get; set; }
```

Property Value

bool

Projection

Gets or sets the projection matrix.

```
public Matrix Projection { get; set; }
```

Property Value

Matrix

SpecularColor

Gets or sets the material specular color (range 0 to 1).

```
public Vector3 SpecularColor { get; set; }
```

Property Value

Vector3

SpecularPower

Gets or sets the material specular power.

```
public float SpecularPower { get; set; }
```

Property Value

float

Texture

Gets or sets the current texture.

```
public Texture2D Texture { get; set; }
```

Property Value

Texture2D

View

Gets or sets the view matrix.

```
public Matrix View { get; set; }
```

Property Value

Matrix

WeightsPerVertex

Gets or sets the number of skinning weights to evaluate for each vertex (1, 2, or 4).

```
public int WeightsPerVertex { get; set; }
```

Property Value

int

World

Gets or sets the world matrix.

```
public Matrix World { get; set; }
```

Property Value

Matrix

Methods

Clone()

Creates a clone of the current SkinnedEffect instance.

```
public override Effect Clone()
```

Returns

Effect

EnableDefaultLighting()

Sets up the standard key/fill/back lighting rig.

```
public void EnableDefaultLighting()
```

GetBoneTransforms(int)

Gets a copy of the current skinning bone transform matrices.

```
public Matrix[] GetBoneTransforms(int count)
```

Parameters

`count` int

Returns

Matrix[]

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

SetBoneTransforms(Matrix[])

Sets an array of skinning bone transform matrices.

```
public void SetBoneTransforms(Matrix[] boneTransforms)
```

Parameters

`boneTransforms` Matrix[]

Class SpriteBatch

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Helper class for drawing text strings and sprites in one or more optimized batches.

```
public class SpriteBatch : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← SpriteBatch

Implements

IDisposable

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

SpriteBatch(GraphicsDevice)

Constructs a SpriteBatch.

```
public SpriteBatch(GraphicsDevice graphicsDevice)
```

Parameters

`graphicsDevice` GraphicsDevice

The GraphicsDevice, which will be used for sprite rendering.

Exceptions

ArgumentNullException

Thrown when `graphicsDevice` is null.

SpriteBatch(GraphicsDevice, int)

Constructs a SpriteBatch.

```
public SpriteBatch(GraphicsDevice graphicsDevice, int capacity)
```

Parameters

`graphicsDevice` GraphicsDevice

The GraphicsDevice, which will be used for sprite rendering.

`capacity` int

The initial capacity of the internal array holding batch items (the value will be rounded to the next multiple of 64).

Exceptions

ArgumentNullException

Thrown when `graphicsDevice` is null.

Methods

Begin(SpriteSortMode, BlendState, SamplerState, DepthStencilState, RasterizerState, Effect, Matrix?)

Begins a new sprite and text batch with the specified render state.

```
public void Begin(SpriteSortMode sortMode = SpriteSortMode.Deferred, BlendState
blendState = null, SamplerState samplerState = null, DepthStencilState
depthStencilState = null, RasterizerState rasterizerState = null, Effect effect =
null, Matrix? transformMatrix = null)
```

Parameters

`sortMode` SpriteSortMode

The drawing order for sprite and text drawing. Deferred by default.

`blendState` BlendState

State of the blending. Uses AlphaBlend if null.

`samplerState` SamplerState

State of the sampler. Uses LinearClamp if null.

`depthStencilState` DepthStencilState

State of the depth-stencil buffer. Uses None if null.

`rasterizerState` RasterizerState

State of the rasterization. Uses CullCounterClockwise if null.

`effect` Effect

A custom Effect to override the default sprite effect. Uses default sprite effect if null.

`transformMatrix` Matrix?

An optional matrix used to transform the sprite geometry. Uses Identity if null.

Remarks

This method uses optional parameters.

Exceptions

InvalidOperationException

Thrown if Begin(SpriteSortMode, BlendState, SamplerState, DepthStencilState, RasterizerState, Effect, Matrix?) is called next time without previous End().

Dispose(bool)

Immediately releases the unmanaged resources used by this object.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

`true` to release both managed and unmanaged resources; `false` to release only unmanaged resources.

Draw(Texture2D, Rectangle, Color)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Rectangle destinationRectangle, Color color)
```

Parameters

`texture` Texture2D

A texture.

`destinationRectangle` Rectangle

The drawing bounds on screen.

`color` Color

A color mask.

Draw(Texture2D, Rectangle, Rectangle?, Color)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Rectangle destinationRectangle, Rectangle?
sourceRectangle, Color color)
```

Parameters

`texture` Texture2D

A texture.

`destinationRectangle` Rectangle

The drawing bounds on screen.

`sourceRectangle` Rectangle?

An optional region on the texture which will be rendered. If null - draws full texture.

`color` Color

A color mask.

Draw(Texture2D, Rectangle, Rectangle?, Color, float, Vector2, SpriteEffects, float)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Rectangle destinationRectangle, Rectangle?
sourceRectangle, Color color, float rotation, Vector2 origin, SpriteEffects effects,
float layerDepth)
```

Parameters

`texture` Texture2D

A texture.

`destinationRectangle` Rectangle

The drawing bounds on screen.

`sourceRectangle` Rectangle?

An optional region on the texture which will be rendered. If null - draws full texture.

`color` Color

A color mask.

`rotation` float

A rotation of this sprite.

`origin` Vector2

Center of the rotation. 0,0 by default.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this sprite.

Draw(Texture2D, Vector2, Color)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Vector2 position, Color color)
```

Parameters

`texture` Texture2D

A texture.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

Draw(Texture2D, Vector2, Rectangle?, Color)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Vector2 position, Rectangle? sourceRectangle,
Color color)
```

Parameters

`texture` Texture2D

A texture.

`position` Vector2

The drawing location on screen.

`sourceRectangle` Rectangle?

An optional region on the texture which will be rendered. If null - draws full texture.

`color` Color

A color mask.

Draw(Texture2D, Vector2, Rectangle?, Color, float, Vector2, Vector2, SpriteEffects, float)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Vector2 position, Rectangle? sourceRectangle,
Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`texture` Texture2D

A texture.

`position` Vector2

The drawing location on screen.

`sourceRectangle` Rectangle?

An optional region on the texture which will be rendered. If null - draws full texture.

`color` Color

A color mask.

`rotation` float

A rotation of this sprite.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` Vector2

A scaling of this sprite.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this sprite.

Draw(Texture2D, Vector2, Rectangle?, Color, float, Vector2, float, SpriteEffects, float)

Submit a sprite for drawing in the current batch.

```
public void Draw(Texture2D texture, Vector2 position, Rectangle? sourceRectangle,
Color color, float rotation, Vector2 origin, float scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`texture` Texture2D

A texture.

`position` Vector2

The drawing location on screen.

`sourceRectangle` Rectangle?

An optional region on the texture which will be rendered. If null - draws full texture.

`color` Color

A color mask.

`rotation` float

A rotation of this sprite.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` float

A scaling of this sprite.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this sprite.

DrawString(SpriteFont, string, Vector2, Color)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, string text, Vector2 position,
Color color)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` string

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

DrawString(SpriteFont, string, Vector2, Color, float, Vector2, Vector2, SpriteEffects, float)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, string text, Vector2 position,
Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` string

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` Vector2

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

DrawString(SpriteFont, string, Vector2, Color, float, Vector2, Vector2, SpriteEffects, float, bool)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, string text, Vector2 position, Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects, float layerDepth, bool rtl)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` string

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` Vector2

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

`rtl` bool

Text is Right to Left.

DrawString(SpriteFont, string, Vector2, Color, float, Vector2, float, SpriteEffects, float)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, string text, Vector2 position,
Color color, float rotation, Vector2 origin, float scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` string

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` float

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

DrawString(SpriteFont, StringBuilder, Vector2, Color)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, StringBuilder text, Vector2 position,
Color color)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` StringBuilder

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

DrawString(SpriteFont, StringBuilder, Vector2, Color, float, Vector2, Vector2, SpriteEffects, float)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, StringBuilder text, Vector2 position,
Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` StringBuilder

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` Vector2

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

DrawString(SpriteFont, StringBuilder, Vector2, Color, float, Vector2, Vector2, SpriteEffects, float, bool)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, StringBuilder text, Vector2 position,
Color color, float rotation, Vector2 origin, Vector2 scale, SpriteEffects effects,
float layerDepth, bool rtl)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` StringBuilder

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` Vector2

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

`rtl` bool

Text is Right to Left.

DrawString(SpriteFont, StringBuilder, Vector2, Color, float, Vector2, float, SpriteEffects, float)

Submit a text string of sprites for drawing in the current batch.

```
public void DrawString(SpriteFont spriteFont, StringBuilder text, Vector2 position,
Color color, float rotation, Vector2 origin, float scale, SpriteEffects effects,
float layerDepth)
```

Parameters

`spriteFont` SpriteFont

A font.

`text` StringBuilder

The text which will be drawn.

`position` Vector2

The drawing location on screen.

`color` Color

A color mask.

`rotation` float

A rotation of this string.

`origin` Vector2

Center of the rotation. 0,0 by default.

`scale` float

A scaling of this string.

`effects` SpriteEffects

Modificators for drawing. Can be combined.

`layerDepth` float

A depth of the layer of this string.

End()

Flushes all batched text and sprites to the screen.

```
public void End()
```

Remarks

This command should be called after Begin(SpriteSortMode, BlendState, SamplerState, DepthStencilState, RasterizerState, Effect, Matrix?) and drawing commands.

Class SpriteEffect

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

The default effect used by SpriteBatch.

```
public class SpriteEffect : Effect, IDisposable
```

Inheritance

object ← GraphicsResource ← Effect ← SpriteEffect

Implements

IDisposable

Inherited Members

Effect.Parameters , Effect.Techniques , Effect.CurrentTechnique , Effect.Dispose(bool) , Effect.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

SpriteEffect(GraphicsDevice)

Creates a new SpriteEffect.

```
public SpriteEffect(GraphicsDevice device)
```

Parameters

`device` GraphicsDevice

SpriteEffect(SpriteEffect)

Creates a new SpriteEffect by cloning parameter settings from an existing instance.

```
protected SpriteEffect(SpriteEffect cloneSource)
```

Parameters

`cloneSource` SpriteEffect

Properties

TransformMatrix

An optional matrix used to transform the sprite geometry. Uses Identity if null.

```
public Matrix? TransformMatrix { get; set; }
```

Property Value

Matrix?

Methods

Clone()

Creates a clone of the current SpriteEffect instance.

```
public override Effect Clone()
```

Returns

Effect

OnApply()

Lazily computes derived parameter values immediately before applying the effect.

```
protected override void OnApply()
```

Enum SpriteEffects

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines sprite visual options for mirroring.

```
[Flags]
public enum SpriteEffects
```

Fields

`FlipHorizontally = 1`

Render the sprite reversed along the X axis.

`FlipVertically = 2`

Render the sprite reversed along the Y axis.

`None = 0`

No options specified.

Class SpriteFont

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a font texture.

```
public sealed class SpriteFont
```

Inheritance

object ← SpriteFont

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SpriteFont(Texture2D, List<Rectangle>, List<Rectangle>, List<char>, int, float, List<Vector3>, char?)

Initializes a new instance of the SpriteFont class.

```
public SpriteFont(Texture2D texture, List<Rectangle> glyphBounds, List<Rectangle>
cropping, List<char> characters, int lineSpacing, float spacing, List<Vector3>
kerning, char? defaultCharacter)
```

Parameters

`texture` Texture2D

The font texture.

`glyphBounds` List<Rectangle>

The rectangles in the font texture containing letters.

`cropping` List<Rectangle>

The cropping rectangles, which are applied to the corresponding glyphBounds to calculate the bounds of the actual character.

`characters` List<char>

The characters.

`lineSpacing` int

The line spacing (the distance from baseline to baseline) of the font.

`spacing` float

The spacing (tracking) between characters in the font.

`kerning` List<Vector3>

The letters kernings (X - left side bearing, Y - width and Z - right side bearing).

`defaultCharacter` char?

The character that will be substituted when a given character is not included in the font.

Properties

Characters

Gets a collection of the characters in the font.

```
public ReadOnlyCollection<char> Characters { get; }
```

Property Value

ReadOnlyCollection<char>

DefaultCharacter

Gets or sets the character that will be substituted when a given character is not included in the font.

```
public char? DefaultCharacter { get; set; }
```

Property Value

char?

Glyphs

All the glyphs in this SpriteFont.

```
public SpriteFont.Glyph[] Glyphs { get; }
```

Property Value

Glyph[]

LineSpacing

Gets or sets the line spacing (the distance from baseline to baseline) of the font.

```
public int LineSpacing { get; set; }
```

Property Value

int

Spacing

Gets or sets the spacing (tracking) between characters in the font.

```
public float Spacing { get; set; }
```

Property Value

float

Texture

Gets the texture that this SpriteFont draws from.

```
public Texture2D Texture { get; }
```

Property Value

Texture2D

Remarks

Can be used to implement custom rendering of a SpriteFont

Methods

GetGlyphs()

Returns a copy of the dictionary containing the glyphs in this SpriteFont.

```
public Dictionary<char, SpriteFont.Glyph> GetGlyphs()
```

Returns

Dictionary<char, SpriteFont.Glyph>

A new Dictionary containing all of the glyphs in this SpriteFont

Remarks

Can be used to calculate character bounds when implementing custom SpriteFont rendering.

MeasureString(string)

Returns the size of a string when rendered in this font.

```
public Vector2 MeasureString(string text)
```

Parameters

`text` string

The text to measure.

Returns

Vector2

The size, in pixels, of 'text' when rendered in this font.

MeasureString(StringBuilder)

Returns the size of the contents of a StringBuilder when rendered in this font.

```
public Vector2 MeasureString(StringBuilder text)
```

Parameters

`text` StringBuilder

The text to measure.

Returns

Vector2

The size, in pixels, of 'text' when rendered in this font.

Struct SpriteFont.Glyph

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Struct that defines the spacing, Kerning, and bounds of a character.

```
public struct SpriteFont.Glyph
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

Provides the data necessary to implement custom SpriteFont rendering.

Fields

BoundsInTexture

Rectangle in the font texture where this letter exists.

```
public Rectangle BoundsInTexture
```

Field Value

Rectangle

Character

The char associated with this glyph.

```
public char Character
```

Field Value

char

Cropping

Cropping applied to the BoundsInTexture to calculate the bounds of the actual character.

```
public Rectangle Cropping
```

Field Value

Rectangle

Empty

Returns an empty glyph.

```
public static readonly SpriteFont.Glyph Empty
```

Field Value

SpriteFont.Glyph

LeftSideBearing

The amount of space between the left side of the character and its first pixel in the X dimension.

```
public float LeftSideBearing
```

Field Value

float

RightSideBearing

The amount of space between the right side of the character and its last pixel in the X dimension.

```
public float RightSideBearing
```

Field Value

float

Width

Width of the character before kerning is applied.

```
public float Width
```

Field Value

float

WidthIncludingBearings

Width of the character before kerning is applied.

```
public float WidthIncludingBearings
```

Field Value

float

Methods

ToString()

Returns a string representation of this SpriteFont.Glyph.

```
public override string ToString()
```

Returns

string

Enum SpriteSortMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines sprite sort rendering options.

```
public enum SpriteSortMode
```

Fields

`BackToFront = 3`

Same as Deferred, except sprites are sorted by depth in back-to-front order prior to drawing. An unstable sort is used, which means sprites with equal depth may not have their order preserved.

`Deferred = 0`

All sprites are drawing when End() invokes, in order of draw call sequence. Depth is ignored.

`FrontToBack = 4`

Same as Deferred, except sprites are sorted by depth in front-to-back order prior to drawing. An unstable sort is used, which means sprites with equal depth may not have their order preserved.

`Immediate = 1`

Each sprite is drawing at individual draw call, instead of End(). Depth is ignored.

`Texture = 2`

Same as Deferred, except sprites are sorted by texture prior to drawing. Depth is ignored.

Enum StencilOperation

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines stencil buffer operations.

```
public enum StencilOperation
```

Fields

`Decrement = 4`

Decrements the stencil buffer entry, wrapping to the maximum value if the new value is less than 0.

`DecrementSaturation = 6`

Decrements the stencil buffer entry, clamping to 0.

`Increment = 3`

Increments the stencil buffer entry, wrapping to 0 if the new value exceeds the maximum value.

`IncrementSaturation = 5`

Increments the stencil buffer entry, clamping to the maximum value.

`Invert = 7`

Inverts the bits in the stencil buffer entry.

`Keep = 0`

Does not update the stencil buffer entry.

`Replace = 2`

Replaces the stencil buffer entry with a reference value.

`Zero = 1`

Sets the stencil buffer entry to 0.

Enum SurfaceFormat

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines types of surface formats.

```
public enum SurfaceFormat
```

Fields

`Alpha8 = 12`

Unsigned A 8-bit format for store 8 bits to alpha channel.

`Astc4X4Rgba = 96`

Adaptive scalable texture compression ; 4x4 matrix using rgba channel interpretation

`Bgr32 = 20`

For compatibility with WPF D3DImage.

`Bgr32SRgb = 31`

Unsigned 32-bit sRGB pixel format that supports 8 bits per channel. 8 bits are unused.

`Bgr565 = 1`

Unsigned 16-bit BGR pixel format for store 5 bits for blue, 6 bits for green, and 5 bits for red.

`Bgra32 = 21`

For compatibility with WPF D3DImage.

`Bgra32SRgb = 32`

Unsigned 32-bit sRGB pixel format that supports 8 bits per channel.

`Bgra4444 = 3`

Unsigned 16-bit BGRA pixel format for store 4 bits per channel.

```
Bgra5551 = 2
```

Unsigned 16-bit BGRA pixel format where 5 bits reserved for each color and last bit is reserved for alpha.

```
Color = 0
```

Unsigned 32-bit ARGB pixel format for store 8 bits per channel.

```
ColorSRgb = 30
```

Unsigned 32-bit RGBA sRGB pixel format that supports 8 bits per channel.

```
Dxt1 = 4
```

DXT1. Texture format with compression. Surface dimensions must be a multiple 4.

```
Dxt1SRgb = 33
```

DXT1. sRGB texture format with compression. Surface dimensions must be a multiple of 4.

```
Dxt1a = 70
```

DXT1 version where 1-bit alpha is used.

```
Dxt3 = 5
```

DXT3. Texture format with compression. Surface dimensions must be a multiple 4.

```
Dxt3SRgb = 34
```

DXT3. sRGB texture format with compression. Surface dimensions must be a multiple of 4.

```
Dxt5 = 6
```

DXT5. Texture format with compression. Surface dimensions must be a multiple 4.

```
Dxt5SRgb = 35
```

DXT5. sRGB texture format with compression. Surface dimensions must be a multiple of 4.

```
HalfSingle = 16
```

Float 16-bit R format for store 16 bits to red channel.

```
HalfVector2 = 17
```

Float 32-bit RG format for store 16 bits per channel.

`HalfVector4 = 18`

Float 64-bit ARGB format for store 16 bits per channel.

`HdrBlendable = 19`

Float pixel format for high dynamic range data.

`NormalizedByte2 = 7`

Signed 16-bit bump-map format for store 8 bits for `u` and `v` data.

`NormalizedByte4 = 8`

Signed 32-bit bump-map format for store 8 bits per channel.

`Rg32 = 10`

Unsigned 32-bit RG pixel format using 16 bits per channel.

`Rgb8A1Etc2 = 92`

Etc2 RGB8A1 (Android/iOS withh OpenglES 3.0)

`Rgb8Etc2 = 90`

Etc2 RGB8 (Android/iOS withh OpenglES 3.0)

`RgbEtc1 = 60`

Ericcson Texture Compression (Android)

`RgbPvrtc2Bpp = 50`

PowerVR texture compression format (iOS and Android).

`RgbPvrtc4Bpp = 51`

PowerVR texture compression format (iOS and Android).

`Rgba1010102 = 9`

Unsigned 32-bit RGBA pixel format for store 10 bits for each color and 2 bits for alpha.

`Rgba64 = 11`

Unsigned 64-bit RGBA pixel format using 16 bits per channel.

`Rgba8Etc2 = 94`

Etc2 RGBA8 EAC (Android/iOS withh OpenglES 3.0)

`RgbaAtcExplicitAlpha = 80`

ATC/ATITC compression (Android)

`RgbaAtcInterpolatedAlpha = 81`

ATC/ATITC compression (Android)

`RgbaPvrtc2Bpp = 52`

PowerVR texture compression format (iOS and Android).

`RgbaPvrtc4Bpp = 53`

PowerVR texture compression format (iOS and Android).

`SRgb8A8Etc2 = 95`

Etc2 SRGB8A8 EAC (Android/iOS withh OpenglES 3.0)

`Single = 13`

IEEE 32-bit R float format for store 32 bits to red channel.

`Srgb8A1Etc2 = 93`

Etc2 SRGB8A1 (Android/iOS withh OpenglES 3.0)

`Srgb8Etc2 = 91`

Etc2 SRGB8 (Android/iOS withh OpenglES 3.0)

`Vector2 = 14`

IEEE 64-bit RG float format for store 32 bits per channel.

`Vector4 = 15`

IEEE 128-bit RGBA float format for store 32 bits per channel.

Class TargetBlendState

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the blend state for a single render target.

```
public class TargetBlendState
```

Inheritance

object ← TargetBlendState

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

AlphaBlendFunction

Gets or Sets the blend function for the alpha component.

```
public BlendFunction AlphaBlendFunction { get; set; }
```

Property Value

BlendFunction

Remarks

This property specifies the blending operation that will be used to combine the alpha components of the source and destination pixels. The blend function affects how the final alpha value is calculated.

Exceptions

InvalidOperationException

The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

AlphaDestinationBlend

Gets or Sets the blend factor for the alpha component of the destination pixel.

```
public Blend AlphaDestinationBlend { get; set; }
```

Property Value

Blend

Remarks

This property specifies the blend factor that will be used for the destination alpha component when combining the source and destination pixels. The blend factor affects how the final alpha value is calculated.

Exceptions

InvalidOperationException

The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

AlphaSourceBlend

Gets or Sets the blend factor for the alpha component of the source pixel.

```
public Blend AlphaSourceBlend { get; set; }
```

Property Value

Blend

Remarks

This property specifies the blend factor that will be used for the source alpha component when combining the source and destination pixels. The blend factor affects how the final alpha value is calculated.

Exceptions

InvalidOperationException

> The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

ColorBlendFunction

Gets or Sets the blend function for the color components (red, green, and blue).

```
public BlendFunction ColorBlendFunction { get; set; }
```

Property Value

BlendFunction

Remarks

This property specifies the blending operation that will be used to combine the color components (red, green, and blue) of the source and destination pixels. The blend function affects how the final color values are calculated.

Exceptions

InvalidOperationException

> The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

ColorDestinationBlend

Gets or Sets the blend factor for the color components (red, green, and blue) of the destination pixel.

```
public Blend ColorDestinationBlend { get; set; }
```

Property Value

Blend

Remarks

This property specifies the blend factor that will be used for the destination color components when combining the source and destination pixels. The blend factor affects how the final color values are calculated.

Exceptions

InvalidOperationException

> The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

ColorSourceBlend

Gets or Sets the blend factor for the color components (red, green, and blue) of the source pixel.

```
public Blend ColorSourceBlend { get; set; }
```

Property Value

Blend

Remarks

This property specifies the blend factor that will be used for the source color components when combining the source and destination pixels. The blend factor affects how the final color values are calculated.

Exceptions

InvalidOperationException

The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

ColorWriteChannels

Gets or Sets a value that determines which color channels to enable for writing to the render target.

```
public ColorWriteChannels ColorWriteChannels { get; set; }
```

Property Value

ColorWriteChannels

Remarks

This property controls which color channels (red, green, blue, and alpha) are enabled for writing to the render target during the blending operation. By default, all color channels are enabled for writing.

Exceptions

InvalidOperationException

The exception is thrown if you attempt to change the property after the BlendState has been bound to the graphics device.

Class Texture

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a texture resource

```
public abstract class Texture : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture

Implements

IDisposable

Derived

Texture2D, Texture3D, TextureCube

Inherited Members

GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

Format

Gets the surface format used by this **Texture**.

```
public SurfaceFormat Format { get; }
```

Property Value

SurfaceFormat

LevelCount

Gets the number of mipmap levels in this **Texture**.

```
public int LevelCount { get; }
```

Property Value

int

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GraphicsDeviceResetting()

Called before the device is reset. Allows graphics resources to invalidate their state so they can be recreated after the device reset. Warning: This may be called after a call to Dispose() up until the resource is garbage collected.

```
protected override void GraphicsDeviceResetting()
```

Class Texture2D

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a 2D grid of texels.

```
public class Texture2D : Texture, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture ← Texture2D

Implements

IDisposable

Derived

RenderTarget2D

Inherited Members

Texture.Format , Texture.LevelCount , Texture.GraphicsDeviceResetting() , Texture.Dispose(bool) , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

A texel represents the smallest unit of a texture that can be read from or written to by the GPU. A texel is composed of 1 to 4 components. Specifically, a texel may be any one of the available texture formats represented in the SurfaceFormat enumeration.

Constructors

Texture2D(GraphicsDevice, int, int)

Creates an uninitialized **Texture2D** resource with the specified parameters.

```
public Texture2D(GraphicsDevice graphicsDevice, int width, int height)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device used to display the texture.

`width` int

The width, in pixels, of the texture.

`height` int

The height, in pixels, of the texture.

Remarks

To initialize the texture with data after creating, you can use one of the SetData method.

To initialize a **Texture2D** from an existing file use the ContentManager.Load method if loading a pipeline preprocessed **Texture2D** from an .xnb file or Texture2D.FromFile to load directly from a file.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` less than or equal to zero.

Texture2D(GraphicsDevice, int, int, bool, SurfaceFormat)

Creates an uninitialized **Texture2D** resource with the specified parameters.

```
public Texture2D(GraphicsDevice graphicsDevice, int width, int height, bool mipmap,
SurfaceFormat format)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device used to display the texture.

`width` int

The width, in pixels, of the texture.

`height` int

The height, in pixels, of the texture.

`mipmap` bool

true if mimapping is enabled for the texture; otherwise, **false**.

`format` SurfaceFormat

The surface format of the texture.

Remarks

To initialize the texture with data after creating, you can use one of the SetData method.

To initialize a **Texture2D** from an existing file use the ContentManager.Load method if loading a pipeline preprocessed **Texture2D** from an .xnb file or Texture2D.FromFile to load directly from a file.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` less than or equal to zero.

Texture2D(GraphicsDevice, int, int, bool, SurfaceFormat, SurfaceType, bool, int)

Creates an uninitialized **Texture2D** resource with the specified parameters.

```
protected Texture2D(GraphicsDevice graphicsDevice, int width, int height,
bool mipmap, SurfaceFormat format, Texture2D.SurfaceType type, bool shared,
int arraySize)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device used to display the texture.

`width` int

The width, in pixels, of the texture.

`height` int

The height, in pixels, of the texture.

`mipmap` bool

true if mimapping is enabled for the texture; otherwise, **false**.

`format` SurfaceFormat

The surface format of the texture.

`type` Texture2D.SurfaceType

The surface type of the texture

`shared` bool

Whether this render target resource should be a shared resource accessible on another device. This property is only valid for DirectX targets.

`arraySize` int

The size of the texture array.

Remarks

To initialize the texture with data after creating, you can use one of the SetData method.

To initialize a **Texture2D** from an existing file use the ContentManager.Load method if loading a pipeline preprocessed **Texture2D** from an .xnb file or Texture2D.FromFile to load directly from a file.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` less than or equal to zero.

ArgumentException

The `arraySize` parameter is greater than 1 and the graphics device does not support texture arrays.

Texture2D(GraphicsDevice, int, int, bool, SurfaceFormat, int)

Creates an uninitialized **Texture2D** resource with the specified parameters.

```
public Texture2D(GraphicsDevice graphicsDevice, int width, int height, bool mipmap,
SurfaceFormat format, int arraySize)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device used to display the texture.

`width` int

The width, in pixels, of the texture.

`height` int

The height, in pixels, of the texture.

`mipmap` bool

true if mimapping is enabled for the texture; otherwise, **false**.

`format` SurfaceFormat

The surface format of the texture.

`arraySize` int

The size of the texture array.

Remarks

To initialize the texture with data after creating, you can use one of the SetData method.

To initialize a **Texture2D** from an existing file use the ContentManager.Load method if loading a pipeline preprocessed **Texture2D** from an .xnb file or Texture2D.FromFile to load directly from a file.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width` and/or `height` less than or equal to zero.

ArgumentException

The `arraySize` parameter is greater than 1 and the graphics device does not support texture arrays.

Properties

Bounds

Gets the dimensions of the texture

```
public Rectangle Bounds { get; }
```

Property Value

Rectangle

Height

Gets the height of the texture in pixels.

```
public int Height { get; }
```

Property Value

int

Width

Gets the width of the texture in pixels.

```
public int Width { get; }
```

Property Value

int

Methods

FromFile(GraphicsDevice, string)

Creates a Texture2D from a file, supported formats bmp, gif, jpg, png, tif and dds (only for simple textures). May work with other formats, but will not work with tga files. This internally calls FromStream(GraphicsDevice, Stream, Action<byte[]>).

```
public static Texture2D FromFile(GraphicsDevice graphicsDevice, string path)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to use to create the texture.

`path` string

The path to the image file.

Returns

Texture2D

The Texture2D created from the given file.

Remarks

Note that different image decoders may generate slight differences between platforms, but perceptually the images should be identical. This call does not premultiply the image alpha, but areas of zero alpha will result in black color data.

FromFile(GraphicsDevice, string, Action<byte[]>)

Creates a Texture2D from a file, supported formats bmp, gif, jpg, png, tif and dds (only for simple textures). May work with other formats, but will not work with tga files. This internally calls FromFile(GraphicsDevice, string, Action<byte[]>).

```
public static Texture2D FromFile(GraphicsDevice graphicsDevice, string path,
Action<byte[]> colorProcessor)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to use to create the texture.

`path` string

The path to the image file.

`colorProcessor` Action<byte[]>

Function that is applied to the data in RGBA format before the texture is sent to video memory. Could be null(no processing then).

Returns

Texture2D

The Texture2D created from the given file.

Remarks

Note that different image decoders may generate slight differences between platforms, but perceptually the images should be identical.

FromStream(GraphicsDevice, Stream)

Creates a Texture2D from a stream, supported formats bmp, gif, jpg, png, tif and dds (only for simple textures). May work with other formats, but will not work with tga files.

```
public static Texture2D FromStream(GraphicsDevice graphicsDevice, Stream stream)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to use to create the texture.

`stream` Stream

The stream from which to read the image data.

Returns

Texture2D

The Texture2D created from the image stream.

Remarks

Note that different image decoders may generate slight differences between platforms, but perceptually the images should be identical. This call does not premultiply the image alpha, but areas of zero alpha will result in black color data.

FromStream(GraphicsDevice, Stream, Action<byte[]>)

Creates a Texture2D from a stream, supported formats bmp, gif, jpg, png, tif and dds (only for simple textures). May work with other formats, but will not work with tga files.

```
public static Texture2D FromStream(GraphicsDevice graphicsDevice, Stream stream,
Action<byte[]> colorProcessor)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device to use to create the texture.

`stream` Stream

The stream from which to read the image data.

`colorProcessor` Action<byte[]>

Function that is applied to the data in RGBA format before the texture is sent to video memory. Could be null(no processing then).

Returns

Texture2D

The Texture2D created from the image stream.

Remarks

Note that different image decoders may generate slight differences between platforms, but perceptually the images should be identical.

GetData<T>(int, int, Rectangle?, T[], int, int)

Copies texture data into an array

```
public void GetData<T>(int level, int arraySlice, Rectangle? rect, T[] data, int
startIndex, int elementCount) where T : struct
```

Parameters

`level` int

The mipmap level to copy from.

`arraySlice` int

Index inside the texture array

`rect` Rectangle?

The section of the texture where the data will be copied from. null indicates the data will be copied over the entire texture.

`data` T[]

The array to receive texture data. If `rect` is null, the number of elements in the array must be equal to the size of the texture, which is Width x Height; otherwise, the number of elements in the array should be equal to the size of the rectangle.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `arraySlice` parameter is greater than zero and the texture arrays are not supported on the graphics device.
- The `arraySlice` parameter is less than zero or is greater than or equal to the internal array buffer of this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

GetData<T>(int, Rectangle?, T[], int, int)

Copies texture data into an array

```
public void GetData<T>(int level, Rectangle? rect, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`level` int

The mipmap level to copy from.

`rect` Rectangle?

The section of the texture where the data will be copied from. null indicates the data will be copied over the entire texture.

`data` T[]

The array to receive texture data. If `rect` is null, the number of elements in the array must be equal to the size of the texture, which is Width x Height; otherwise, the number of elements in the array should be equal to the size of the rectangle.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

GetData<T>(T[])

Copies texture data into an array

```
public void GetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array to receive texture data.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

GetData<T>(T[], int, int)

Copies texture data into an array

```
public void GetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array to receive texture data.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

Reload(Stream)

Reloads the texture

```
public void Reload(Stream textureStream)
```

Parameters

`textureStream` Stream

Remarks

This method allows games that use Texture2D.FromStream to reload their textures after the GL context is lost.

SaveAsJpeg(Stream, int, int)

Converts the texture to a JPG image

```
public void SaveAsJpeg(Stream stream, int width, int height)
```

Parameters

`stream` Stream

Destination for the image

`width` int

The width, in pixels, of the image.

`height` int

The height, in pixels, of the image.

SaveAsPng(Stream, int, int)

Converts the texture to a PNG image

```
public void SaveAsPng(Stream stream, int width, int height)
```

Parameters

`stream` Stream

Destination for the image

`width` int

The width, in pixels, of the image.

`height` int

The height, in pixels, of the image.

SetData<T>(int, int, Rectangle?, T[], int, int)

Copies an array of data to the texture.

```
public void SetData<T>(int level, int arraySlice, Rectangle? rect, T[] data, int
startIndex, int elementCount) where T : struct
```

Parameters

`level` int

The mipmap level where the data will be placed.

`arraySlice` int

Index inside the texture array

`rect` Rectangle?

The section of the texture where the data will be placed. null indicates the data will be copied over the entire texture.

`data` T[]

The array of data to copy. If `rect` is null, the number of elements in the array must be equal to the size of the texture, which is Width x Height; otherwise, the number of elements in the array should be equal to the size of the rectangle.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `arraySlice` parameter is greater than zero and the texture arrays are not supported on the graphics device.
- The `arraySlice` parameter is less than zero or is greater than or equal to the internal array buffer of this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

SetData<T>(int, Rectangle?, T[], int, int)

Copies an array of data to the texture.

```
public void SetData<T>(int level, Rectangle? rect, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`level` int

The mipmap level where the data will be placed.

`rect` Rectangle?

The section of the texture where the data will be placed. null indicates the data will be copied over the entire texture.

`data` T[]

The array of data to copy. If `rect` is null, the number of elements in the array must be equal to the size of the texture, which is Width x Height; otherwise, the number of elements in the array should be equal to the size of the rectangle.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

SetData<T>(T[])

Copies an array of data to the texture.

```
public void SetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

SetData<T>(T[], int, int)

Copies an array of data to the texture.

```
public void SetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small. length of the data array.

ArgumentNullException

The `data` parameter is null.

Enum Texture2D.SurfaceType

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines the values for the surface type of a **Texture2D**

```
protected enum Texture2D.SurfaceType
```

Fields

`RenderTarget = 1`

Describes that the surface type is a render target.

`SwapChainRenderTarget = 2`

Describes that the surface type is a swap chain render target.

`Texture = 0`

Describes that the surface type is a texture.

Class Texture3D

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a 3D volume of texels.

```
public class Texture3D : Texture, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture ← Texture3D

Implements

IDisposable

Inherited Members

Texture.Format , Texture.LevelCount , Texture.GraphicsDeviceResetting() , Texture.Dispose(bool) , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

A texel represents the smallest unit of a texture that can be read from or written to by the GPU. A texel is composed of 1 to 4 components. Specifically, a texel may be any one of the available texture formats represented in the SurfaceFormat enumeration.

A **Texture3D** resource (also known as a volume texture) contains a 3D volume of texels. Since it is a texture resource, it may contain mipmap levels.

When a **Texture3D** mipmap slice is bound as a render target output (by creating a RenderTargetCube), the **Texture3D** behaves identically to an array of Texture2D objects with *n* array slices, where *n* is the depth (third dimension) of the **Texture3D**.

Constructors

Texture3D(GraphicsDevice, int, int, int, bool, SurfaceFormat)

Creates an uninitialized **Texture3D** resource with the specified parameters.

```
public Texture3D(GraphicsDevice graphicsDevice, int width, int height, int depth,
bool mipMap, SurfaceFormat format)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device used to display the texture.

`width` int

The width, in pixels, of the texture.

`height` int

The height, in pixels, of the texture.

`depth` int

The depth, in pixels, of the texture.

`mipMap` bool

true if mimapping is enabled for the texture; otherwise, **false**.

`format` SurfaceFormat

The surface format of the texture.

Remarks

To initialize the texture with data after creating, you can use one of the SetData methods.

To initialize a **Texture3D** from an existing file use the ContentManager.Load method if loading a pipeline preprocessed **Texture3D** from an .xnb file.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `width`, `height`, and/or `depth` parameters are less than or equal to zero.

Texture3D(GraphicsDevice, int, int, int, bool, SurfaceFormat, bool)

```
protected Texture3D(GraphicsDevice graphicsDevice, int width, int height, int depth,
bool mipMap, SurfaceFormat format, bool renderTarget)
```

Parameters

`graphicsDevice` GraphicsDevice

`width` int

`height` int

`depth` int

`mipMap` bool

`format` SurfaceFormat

`renderTarget` bool

Properties

Depth

Gets the depth, in pixels, of this texture resource.

```
public int Depth { get; }
```

Property Value

int

Height

Gets the height, in pixels, of this texture resource.

```
public int Height { get; }
```

Property Value

int

Width

Gets the width, in pixels, of this texture resource.

```
public int Width { get; }
```

Property Value

int

Methods

GetData<T>(int, int, int, int, int, int, int, T[], int, int)

Copies texture data into an array.

```
public void GetData<T>(int level, int left, int top, int right, int bottom, int
front, int back, T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`level` int

The mipmap level where the data will be placed.

`left` int

Position of the left side of the box on the x-axis.

`top` int

Position of the top of the box on the y-axis.

`right` int

Position of the right side of the box on the x-axis.

`bottom` int

Position of the bottom of the box on the y-axis.

`front` int

Position of the front of the box on the z-axis.

`back` int

Position of the back of the box on the z-axis.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.
- The `left`, `top`, `back`, and/or `right` parameters are less than zero.
- The `right` parameter is greater than the width of the texture.
- The `bottom` parameter is greater than the height of the texture.
- The `top` parameter is greater than the depth of the texture.
- The `elementCount` parameter is the incorrect size.

ArgumentNullException

The `data` parameter is null.

GetData<T>(T[])

Copies texture data into an array.

```
public void GetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

GetData<T>(T[], int, int)

Copies texture data into an array.

```
public void GetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.
- The `elementCount` parameter is the incorrect size.

ArgumentNullException

The `data` parameter is null.

SetData<T>(int, int, int, int, int, int, int, T[], int, int)

Copies an array of data to the texture.

```
public void SetData<T>(int level, int left, int top, int right, int bottom, int
front, int back, T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`level` int

The mipmap level where the data will be placed.

`left` int

Position of the left side of the box on the x-axis.

`top` int

Position of the top of the box on the y-axis.

`right` int

Position of the right side of the box on the x-axis.

`bottom` int

Position of the bottom of the box on the y-axis.

`front` int

Position of the front of the box on the z-axis.

`back` int

Position of the back of the box on the z-axis.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.
- The `left`, `top`, `back`, and/or `right` parameters are less than zero.
- The `right` parameter is greater than the width of the texture.
- The `bottom` parameter is greater than the height of the texture.
- The `top` parameter is greater than the depth of the texture.
- The `elementCount` parameter is the incorrect size.

ArgumentNullException

The `data` parameter is null.

SetData<T>(T[])

Copies an array of data to the texture.

```
public void SetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

SetData<T>(T[], int, int)

Copies an array of data to the texture.

```
public void SetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

Enum TextureAddressMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines modes for addressing texels using texture coordinates that are outside of the range of 0.0 to 1.0.

```
public enum TextureAddressMode
```

Fields

`Border = 3`

Texels outside range will be set to the border color.

`Clamp = 1`

Texels outside range will be set to color of 0.0 or 1.0 texel.

`Mirror = 2`

Same as Wrap but tiles will also flipped at every integer junction.

`Wrap = 0`

Texels outside range will form the tile at every integer junction.

Class TextureCollection

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a collection of Texture objects. This class cannot be inherited.

```
public sealed class TextureCollection
```

Inheritance

object ← TextureCollection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

this[int]

Gets or sets the Texture at the specified sampler number.

```
public Texture this[int index] { get; set; }
```

Parameters

`index` int

Property Value

Texture

Class TextureCube

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a set of six 2D textures, one for each face of a cube.

```
public class TextureCube : Texture, IDisposable
```

Inheritance

object ← GraphicsResource ← Texture ← TextureCube

Implements

IDisposable

Derived

RenderTargetCube

Inherited Members

Texture.Format , Texture.LevelCount , Texture.GraphicsDeviceResetting() , Texture.Dispose(bool) , GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

A texel represents the smallest unit of a texture that can be read from or written to by the GPU. A texel is composed of 1 to 4 components. Specifically, a texel may be any one of the available texture formats represented in the SurfaceFormat enumeration.

A cube texture is a collection of six textures, one for each face of the cube. All faces must be present in the cube texture. Also, a cube map surface must be the same pixel size in all three dimensions (x, y, and z).

Constructors

TextureCube(GraphicsDevice, int, bool, SurfaceFormat)

Creates an uninitialized **TextureCube** resource of the specified dimensions

```
public TextureCube(GraphicsDevice graphicsDevice, int size, bool mipMap,
SurfaceFormat format)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device that will display the cube texture.

`size` int

The width and height, in pixels, of the cube texture.

`mipMap` bool

true if mimapping is enabled for this cube texture; otherwise, **false**.

`format` SurfaceFormat

The surface format used by this cube texture.

Exceptions

ArgumentNullException

The `graphicsDevice` parameter is null.

ArgumentOutOfRangeException

The `size` parameter is less than or equal to zero.

Properties

Size

Gets the width and height of the cube map face in pixels.

```
public int Size { get; }
```

Property Value

int

The width and height of a cube map face in pixels.

Methods

GetData<T>(CubeMapFace, int, Rectangle?, T[], int, int)

Copies the texture cube data into an array.

```
public void GetData<T>(CubeMapFace cubeMapFace, int level, Rectangle? rect, T[]
data, int startIndex, int elementCount) where T : struct
```

Parameters

`cubeMapFace` CubeMapFace

The cube map face.

`level` int

The mipmap level to copy from.

`rect` Rectangle?

The section of the texture where the data will be placed. null indicates the data will be copied over the entire texture.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

GetData<T>(CubeMapFace, T[])

Copies the texture cube data into an array.

```
public void GetData<T>(CubeMapFace cubeMapFace, T[] data) where T : struct
```

Parameters

`cubeMapFace` CubeMapFace

The cube map face.

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

GetData<T>(CubeMapFace, T[], int, int)

Copies the texture cube data into an array.

```
public void GetData<T>(CubeMapFace cubeMapFace, T[] data, int startIndex, int
elementCount) where T : struct
```

Parameters

`cubeMapFace` CubeMapFace

The cube map face.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

SetData<T>(CubeMapFace, int, Rectangle?, T[], int, int)

Copies an array of data to the texture cube.

```
public void SetData<T>(CubeMapFace face, int level, Rectangle? rect, T[] data, int
startIndex, int elementCount) where T : struct
```

Parameters

`face` CubeMapFace

The Cubemap face.

`level` int

The mipmap level where the data will be placed.

`rect` Rectangle?

The section of the texture where the data will be placed. null indicates the data will be copied over the entire texture.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `level` parameter is larger than the number of levels in this texture.
- The `rect` is outside the bounds of the texture.
- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

SetData<T>(CubeMapFace, T[])

Copies an array of data to the texture cube.

```
public void SetData<T>(CubeMapFace face, T[] data) where T : struct
```

Parameters

`face` CubeMapFace

The Cubemap face.

`data` T[]

The array of data to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

SetData<T>(CubeMapFace, T[], int, int)

Copies an array of data to the texture cube.

```
public void SetData<T>(CubeMapFace face, T[] data, int startIndex, int elementCount)
where T : struct
```

Parameters

`face` CubeMapFace

The Cubemap face.

`data` T[]

The array of data to copy.

`startIndex` int

The index of the element in the array at which to start copying.

`elementCount` int

The number of elements to copy.

Type Parameters

`T`

The type of the elements in the array.

Exceptions

ArgumentException

One of the following conditions is true:

- The `T` type size is invalid for the format of this texture.
- The `startIndex` parameter is less than zero or is greater than or equal to the length of the data array.
- The `data` array parameter is too small.

ArgumentNullException

The `data` parameter is null.

Enum TextureFilter

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines filtering types for texture sampler.

```
public enum TextureFilter
```

Fields

`Anisotropic = 2`

Use anisotropic filtering.

`Linear = 0`

Use linear filtering.

`LinearMipPoint = 3`

Use linear filtering to shrink or expand, and point filtering between mipmap levels (mip).

`MinLinearMagPointMipLinear = 5`

Use linear filtering to shrink, point filtering to expand, and linear filtering between mipmap levels.

`MinLinearMagPointMipPoint = 6`

Use linear filtering to shrink, point filtering to expand, and point filtering between mipmap levels.

`MinPointMagLinearMipLinear = 7`

Use point filtering to shrink, linear filtering to expand, and linear filtering between mipmap levels.

`MinPointMagLinearMipPoint = 8`

Use point filtering to shrink, linear filtering to expand, and point filtering between mipmap levels.

`Point = 1`

Use point filtering.

```
PointMipLinear = 4
```

Use point filtering to shrink (minify) or expand (magnify), and linear filtering between mipmap levels.

Enum TextureFilterMode

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents the texture filtering mode used for texture sampling.

```
public enum TextureFilterMode
```

Fields

`Comparison = 1`

The comparison texture filtering mode, which performs a comparison operation between the sampled texture value and a reference value during texture sampling.

`Default = 0`

The default texture filtering mode, which performs standard texture sampling without comparison.

Class VertexBuffer

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Represents a list of 3D vertices to be streamed to the graphics device.

```
public class VertexBuffer : GraphicsResource, IDisposable
```

Inheritance

object ← GraphicsResource ← VertexBuffer

Implements

IDisposable

Derived

DynamicVertexBuffer

Inherited Members

GraphicsResource.Dispose() , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

VertexBuffer(GraphicsDevice, VertexDeclaration, int, BufferUsage)

Creates a new instance of VertexBuffer

```
public VertexBuffer(GraphicsDevice graphicsDevice, VertexDeclaration
vertexDeclaration, int vertexCount, BufferUsage bufferUsage)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

`vertexDeclaration` VertexDeclaration

The vertex declaration, which describes per-vertex data.

`vertexCount` int

The number of vertices.

`bufferUsage` BufferUsage

Behavior options.

Exceptions

ArgumentNullException

`graphicsDevice` is null

VertexBuffer(GraphicsDevice, VertexDeclaration, int, BufferUsage, bool)

Creates a new instance of VertexBuffer

```
protected VertexBuffer(GraphicsDevice graphicsDevice, VertexDeclaration
vertexDeclaration, int vertexCount, BufferUsage bufferUsage, bool dynamic)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

`vertexDeclaration` VertexDeclaration

The vertex declaration, which describes per-vertex data.

`vertexCount` int

The number of vertices.

`bufferUsage` BufferUsage

Behavior options.

`dynamic` bool

Whether this buffer is dynmic.

Exceptions

ArgumentNullException

`graphicsDevice` is null

VertexBuffer(GraphicsDevice, Type, int, BufferUsage)

Creates a new instance of VertexBuffer

```
public VertexBuffer(GraphicsDevice graphicsDevice, Type type, int vertexCount,
BufferUsage bufferUsage)
```

Parameters

`graphicsDevice` GraphicsDevice

The graphics device.

`type` Type

The data type. Must be a value type which implements the IVertexType interface.

`vertexCount` int

The number of vertices.

`bufferUsage` BufferUsage

Behavior options.

Exceptions

ArgumentNullException

`graphicsDevice` is null

Properties

BufferUsage

A usage hint for optimizing memory placement of this buffer.

```
public BufferUsage BufferUsage { get; }
```

Property Value

BufferUsage

VertexCount

Gets the number of vertices.

```
public int VertexCount { get; }
```

Property Value

int

VertexDeclaration

Defines per-vertex data in a buffer.

```
public VertexDeclaration VertexDeclaration { get; }
```

Property Value

VertexDeclaration

Methods

Dispose(bool)

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

GetData<T>(int, T[], int, int, int)

Get the vertex data froom this VertexBuffer.

```
public void GetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, int vertexStride = 0) where T : struct
```

Parameters

`offsetInBytes` int

The offset to the first element in the vertex buffer in bytes.

`data` T[]

An array of T's to be filled.

`startIndex` int

The index to start filling the data array.

`elementCount` int

The number of T's to get.

`vertexStride` int

The size of how a vertex buffer element should be interpreted.

Type Parameters

`T`

The struct you want to fill.

Remarks

Note that this pulls data from VRAM into main memory and because of that is a very expensive operation. It is often a better idea to keep a copy of the data in main memory.

GetData<T>(T[])

Get the vertex data froom this VertexBuffer.

```
public void GetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

An array of T's to be filled.

Type Parameters

`T`

The struct you want to fill.

Remarks

Note that this pulls data from VRAM into main memory and because of that is a very expensive operation. It is often a better idea to keep a copy of the data in main memory.

GetData<T>(T[], int, int)

Get the vertex data froom this VertexBuffer.

```
public void GetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

An array of T's to be filled.

`startIndex` int

The index to start filling the data array.

`elementCount` int

The number of T's to get.

Type Parameters

`T`

The struct you want to fill.

Remarks

Note that this pulls data from VRAM into main memory and because of that is a very expensive operation. It is often a better idea to keep a copy of the data in main memory.

GraphicsDeviceResetting()

The GraphicsDevice is resetting, so GPU resources must be recreated.

```
protected override void GraphicsDeviceResetting()
```

SetDataInternal<T>(int, T[], int, int, int, SetDataOptions)

```
protected void SetDataInternal<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, int vertexStride, SetDataOptions options) where T : struct
```

Parameters

`offsetInBytes` int

`data` T[]

`startIndex` int

`elementCount` int

`vertexStride` int

`options` SetDataOptions

Type Parameters

`T`

SetData<T>(int, T[], int, int, int)

Sets the vertex buffer data, specifying the index at which to start copying from the source data array, the number of elements to copy from the source data array, and how far apart elements from the source data array should be when they are copied into the vertex buffer.

```
public void SetData<T>(int offsetInBytes, T[] data, int startIndex, int
elementCount, int vertexStride) where T : struct
```

Parameters

`offsetInBytes` int

Offset in bytes from the beginning of the vertex buffer to the start of the copied data.

`data` T[]

Data array to be passed to the shader.

`startIndex` int

Index at which to start copying from `data`. Must be within the `data` array bounds.

`elementCount` int

Number of elements to copy from `data`. The combination of `startIndex` and `elementCount` must be within the `data` array bounds.

`vertexStride` int

Specifies how far apart, in bytes, elements from `data` should be when they are copied into the vertex buffer. In almost all cases this should be `sizeof(T)`, to create a tightly-packed vertex buffer. If you specify `sizeof(T)`, elements from `data` will be copied into the vertex buffer with no padding between each element. If you specify a value greater than `sizeof(T)`, elements from `data` will be copied into the vertex buffer with padding between each element. If you specify `0` for this parameter, it will be treated as if you had specified `sizeof(T)`. With the exception of `0`, you must specify a value greater than or equal to `sizeof(T)`.

Type Parameters

`T`

Type of elements in the data array.

Remarks

If `T` is VertexPositionTexture, but you want to set only the position component of the vertex data, you would call this method as follows as the position is the first piece of data in the buffer:

```
Vector3[] positions = new Vector3[numVertices];
vertexBuffer.SetData(0, positions, 0, numVertices,
vertexBuffer.VertexDeclaration.VertexStride);
```

Continuing from the previous example, if you want to set only the texture coordinate component of the vertex data, you would call this method as follows (note that `offsetInBytes` is 12, the size of a Vector3, representing the position):

```
Vector2[] texCoords = new Vector2[numVertices];
vertexBuffer.SetData(12, texCoords, 0, numVertices,
vertexBuffer.VertexDeclaration.VertexStride);
```

SetData<T>(T[])

Sets the vertex buffer data. This is the same as calling SetData<T>(int, T[], int, int, int) with `offsetInBytes` and `startIndex` equal to `0`, `elementCount` equal to `data.Length`, and `vertexStride` equal to `sizeof(T)`.

```
public void SetData<T>(T[] data) where T : struct
```

Parameters

`data` T[]

Data array to be passed to the shader.

Type Parameters

`T`

Type of elements in the data array.

SetData<T>(T[], int, int)

Sets the vertex buffer data, specifying the index at which to start copying from the source data array, and the number of elements to copy from the source data array. This is the same as calling SetData<T>(int, T[], int, int, int) with `offsetInBytes` equal to `0`, and `vertexStride` equal to `sizeof(T)`.

```
public void SetData<T>(T[] data, int startIndex, int elementCount) where T : struct
```

Parameters

`data` T[]

Data array to be passed to the shader.

`startIndex` int

Index at which to start copying from `data`. Must be within the `data` array bounds.

`elementCount` int

Number of elements to copy from `data`. The combination of `startIndex` and `elementCount` must be within the `data` array bounds.

Type Parameters

`T`

Type of elements in the data array.

Struct VertexBufferBinding

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines how a vertex buffer is bound to the graphics device for rendering.

```
public struct VertexBufferBinding
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexBufferBinding(VertexBuffer)

Creates an instance of VertexBufferBinding.

```
public VertexBufferBinding(VertexBuffer vertexBuffer)
```

Parameters

`vertexBuffer` VertexBuffer

The vertex buffer to bind.

VertexBufferBinding(VertexBuffer, int)

Creates an instance of VertexBufferBinding.

```
public VertexBufferBinding(VertexBuffer vertexBuffer, int vertexOffset)
```

Parameters

`vertexBuffer` VertexBuffer

The vertex buffer to bind.

`vertexOffset` int

The index of the first vertex in the vertex buffer to use.

VertexBufferBinding(VertexBuffer, int, int)

Creates an instance of VertexBufferBinding.

```
public VertexBufferBinding(VertexBuffer vertexBuffer, int vertexOffset,
int instanceFrequency)
```

Parameters

`vertexBuffer` VertexBuffer

The vertex buffer to bind.

`vertexOffset` int

The index of the first vertex in the vertex buffer to use.

`instanceFrequency` int

The number of instances to draw using the same per-instance data before advancing in the buffer by one element. This value must be 0 for an element that contains per-vertex data and greater than 0 for per-instance data.

Exceptions

ArgumentNullException

`vertexBuffer` is null.

ArgumentOutOfRangeException

`vertexOffset` or `instanceFrequency` is invalid.

Properties

InstanceFrequency

Gets the number of instances to draw using the same per-instance data before advancing in the buffer by one element.

```
public int InstanceFrequency { get; }
```

Property Value

int

The number of instances to draw using the same per-instance data before advancing in the buffer by one element. This value must be 0 for an element that contains per-vertex data and greater than 0 for per-instance data.

VertexBuffer

Gets the vertex buffer.

```
public VertexBuffer VertexBuffer { get; }
```

Property Value

VertexBuffer

The vertex buffer.

VertexOffset

Gets the index of the first vertex in the vertex buffer to use.

```
public int VertexOffset { get; }
```

Property Value

int

The index of the first vertex in the vertex buffer to use.

Class VertexDeclaration

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines per-vertex data of a vertex buffer.

```
public class VertexDeclaration : GraphicsResource, IDisposable,
IEquatable<VertexDeclaration>
```

Inheritance

object ← GraphicsResource ← VertexDeclaration

Implements

IDisposable, IEquatable<VertexDeclaration>

Inherited Members

GraphicsResource.GraphicsDeviceResetting() , GraphicsResource.Dispose() , GraphicsResource.Dispose(bool) , GraphicsResource.Disposing , GraphicsResource.GraphicsDevice , GraphicsResource.IsDisposed , GraphicsResource.Name , GraphicsResource.Tag , GraphicsResource.ToString() , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Remarks

VertexDeclaration implements IEquatable<T> and can be used as a key in a dictionary. Two vertex declarations are considered equal if the vertices are structurally equivalent, i.e. the vertex elements and the vertex stride are identical. (The properties Name and Tag are ignored in GetHashCode() and Equals(VertexDeclaration)!)

Constructors

VertexDeclaration(params VertexElement[])

Initializes a new instance of the VertexDeclaration class.

```
public VertexDeclaration(params VertexElement[] elements)
```

Parameters

`elements` VertexElement[]

The vertex elements.

Exceptions

ArgumentNullException

`elements` is null or empty.

VertexDeclaration(int, params VertexElement[])

Initializes a new instance of the VertexDeclaration class.

```
public VertexDeclaration(int vertexStride, params VertexElement[] elements)
```

Parameters

`vertexStride` int

The size of a vertex (including padding) in bytes.

`elements` VertexElement[]

The vertex elements.

Exceptions

ArgumentNullException

`elements` is null or empty.

Properties

VertexStride

Gets the size of a vertex (including padding) in bytes.

```
public int VertexStride { get; }
```

Property Value

int

The size of a vertex (including padding) in bytes.

Methods

Equals(VertexDeclaration)

Determines whether the specified VertexDeclaration is equal to this instance.

```
public bool Equals(VertexDeclaration other)
```

Parameters

`other` VertexDeclaration

The object to compare with the current object.

Returns

bool

true if the specified VertexDeclaration is equal to this instance; otherwise, false.

Equals(object)

Determines whether the specified object is equal to this instance.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to this instance; otherwise, false.

GetHashCode()

Returns a hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance, suitable for use in hashing algorithms and data structures like a hash table.

GetVertexElements()

Gets a copy of the vertex elements.

```
public VertexElement[] GetVertexElements()
```

Returns

VertexElement[]

A copy of the vertex elements.

Operators

operator ==(VertexDeclaration, VertexDeclaration)

Compares two VertexElement instances to determine whether they are the same.

```
public static bool operator ==(VertexDeclaration left, VertexDeclaration right)
```

Parameters

`left` VertexDeclaration

The first instance.

`right` VertexDeclaration

The second instance.

Returns

bool

true if the `left` and `right` are the same; otherwise, false.

operator !=(VertexDeclaration, VertexDeclaration)

Compares two VertexElement instances to determine whether they are different.

```
public static bool operator !=(VertexDeclaration left, VertexDeclaration right)
```

Parameters

`left` VertexDeclaration

The first instance.

`right` VertexDeclaration

The second instance.

Returns

bool

true if the `left` and `right` are the different; otherwise, false.

Struct VertexElement

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines a single element in a vertex.

```
public struct VertexElement : IEquatable<VertexElement>
```

Implements

IEquatable<VertexElement>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexElement(int, VertexElementFormat, VertexElementUsage, int)

Initializes a new instance of the VertexElement struct.

```
public VertexElement(int offset, VertexElementFormat elementFormat,
VertexElementUsage elementUsage, int usageIndex)
```

Parameters

`offset` int

The offset in bytes from the beginning of the stream to the vertex element.

`elementFormat` VertexElementFormat

The element format.

`elementUsage` VertexElementUsage

The HLSL semantic of the element in the vertex shader input-signature.

`usageIndex` int

The semantic index, which is required if the semantic is used for more than one vertex element.

Properties

Offset

Gets or sets the offset in bytes from the beginning of the stream to the vertex element.

```
public int Offset { get; set; }
```

Property Value

int

The offset in bytes.

UsageIndex

Gets or sets the semantic index.

```
public int UsageIndex { get; set; }
```

Property Value

int

The semantic index, which is required if the semantic is used for more than one vertex element.

Remarks

Usage indices in a vertex declaration usually start with 0. When multiple vertex buffers are bound to the input assembler stage (see SetVertexBuffers(params VertexBufferBinding[])), MonoGame internally adjusts the usage indices based on the order in which the vertex buffers are bound.

VertexElementFormat

Gets or sets the data format.

```
public VertexElementFormat VertexElementFormat { get; set; }
```

Property Value

VertexElementFormat

The data format.

VertexElementUsage

Gets or sets the HLSL semantic of the element in the vertex shader input.

```
public VertexElementUsage VertexElementUsage { get; set; }
```

Property Value

VertexElementUsage

The HLSL semantic of the element in the vertex shader input.

Methods

Equals(VertexElement)

Determines whether the specified VertexElement is equal to this instance.

```
public bool Equals(VertexElement other)
```

Parameters

`other` VertexElement

The object to compare with the current object.

Returns

bool

true if the specified VertexElement is equal to this instance; otherwise, false.

Equals(object)

Determines whether the specified object is equal to this instance.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to this instance; otherwise, false.

GetHashCode()

Returns a hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance, suitable for use in hashing algorithms and data structures like a hash table.

ToString()

Returns a string that represents this instance.

```
public override string ToString()
```

Returns

string

A string that represents this instance.

Operators

operator ==(VertexElement, VertexElement)

Compares two VertexElement instances to determine whether they are the same.

```
public static bool operator ==(VertexElement left, VertexElement right)
```

Parameters

`left` VertexElement

The first instance.

`right` VertexElement

The second instance.

Returns

bool

true if the `left` and `right` are the same; otherwise, false.

operator !=(VertexElement, VertexElement)

Compares two VertexElement instances to determine whether they are different.

```
public static bool operator !=(VertexElement left, VertexElement right)
```

Parameters

`left` VertexElement

The first instance.

`right` VertexElement

The second instance.

Returns

bool

true if the `left` and `right` are the different; otherwise, false.

Enum VertexElementFormat

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines vertex element formats.

```
public enum VertexElementFormat
```

Fields

`Byte4 = 5`

Four component unsigned byte.

`Color = 4`

Four component, packed unsigned byte, mapped to 0 to 1 range.

`HalfVector2 = 10`

Two component 16-bit floating point number.

`HalfVector4 = 11`

Four component 16-bit floating point number.

`NormalizedShort2 = 8`

Normalized, two component signed 16-bit integer.

`NormalizedShort4 = 9`

Normalized, four component signed 16-bit integer.

`Short2 = 6`

Two component signed 16-bit integer.

`Short4 = 7`

Four component signed 16-bit integer.

`Single = 0`

Single 32-bit floating point number.

`Vector2 = 1`

Two component 32-bit floating point number.

`Vector3 = 2`

Three component 32-bit floating point number.

`Vector4 = 3`

Four component 32-bit floating point number.

Enum VertexElementUsage

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Defines usage for vertex elements.

```
public enum VertexElementUsage
```

Fields

`Binormal = 4`

Binormal data.

`BlendIndices = 6`

Blending indices data.

`BlendWeight = 7`

Blending weight data.

`Color = 1`

Color data.

`Depth = 8`

Depth data.

`Fog = 9`

Fog data.

`Normal = 3`

Normal data.

`PointSize = 10`

Point size data. Usable for drawing point sprites.

`Position = 0`

Position data.

`Sample = 11`

Sampler data for specifies the displacement value to look up.

`Tangent = 5`

Tangent data.

`TessellateFactor = 12`

Single, positive float value, specifies a tessellation factor used in the tessellation unit to control the rate of tessellation.

`TextureCoordinate = 2`

Texture coordinate data or can be used for user-defined data.

Struct VertexPosition

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position.

```
[DataContract]
public struct VertexPosition : IVertexType
```

Implements

IVertexType

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPosition(Vector3)

Creates an instance of VertexPosition.

```
public VertexPosition(Vector3 position)
```

Parameters

`position` Vector3

Position of the vertex.

Fields

Position

The XYZ vertex position.

```
[DataMember]
public Vector3 Position
```

Field Value

Vector3

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPosition, VertexPosition)

Returns a value that indicates whether two VertexPosition are equal

```
public static bool operator ==(VertexPosition left, VertexPosition right)
```

Parameters

`left` VertexPosition

The vertex on the left of the equality operator.

`right` VertexPosition

The vertex on the right of the equality operator.

Returns

bool

true if the vertices are the same; false otherwise.

operator !=(VertexPosition, VertexPosition)

Returns a value that indicates whether two VertexPosition are different

```
public static bool operator !=(VertexPosition left, VertexPosition right)
```

Parameters

`left` VertexPosition

The vertex on the left of the equality operator.

`right` VertexPosition

The vertex on the right of the equality operator.

Returns

bool

true if the vertices are different; false otherwise.

Struct VertexPositionColor

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position and color.

```
[DataContract]
public struct VertexPositionColor : IVertexType
```

Implements

IVertexType

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionColor(Vector3, Color)

Creates an instance of VertexPositionColor.

```
public VertexPositionColor(Vector3 position, Color color)
```

Parameters

`position` Vector3

Position of the vertex.

`color` Color

Color of the vertex.

Fields

Color

The vertex color.

```
[DataMember]
public Color Color
```

Field Value

Color

Position

The XYZ vertex position.

```
[DataMember]
public Vector3 Position
```

Field Value

Vector3

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionColor, VertexPositionColor)

Returns a value that indicates whether two VertexPositionColor are equal

```
public static bool operator ==(VertexPositionColor left, VertexPositionColor right)
```

Parameters

`left` VertexPositionColor

The object on the left of the equality operator.

`right` VertexPositionColor

The object on the right of the equality operator.

Returns

bool

true if the objects are the same; false otherwise.

operator !=(VertexPositionColor, VertexPositionColor)

Returns a value that indicates whether two VertexPositionColor are different

```
public static bool operator !=(VertexPositionColor left, VertexPositionColor right)
```

Parameters

`left` VertexPositionColor

The object on the left of the inequality operator.

`right` VertexPositionColor

The object on the right of the inequality operator.

Returns

bool

true if the objects are different; false otherwise.

Struct VertexPositionColorNormal

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position, color and normal data.

```
public struct VertexPositionColorNormal : IVertexType
```

Implements
IVertexType

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionColorNormal(Vector3, Color, Vector3)

Creates an instance of VertexPositionColorNormal.

```
public VertexPositionColorNormal(Vector3 position, Color color, Vector3 normal)
```

Parameters

`position` Vector3

Position of the vertex.

`color` Color

Color of the vertex.

`normal` Vector3

The vertex normal.

Fields

Color

The vertex color.

```
public Color Color
```

Field Value

Color

Normal

The XYZ surface normal.

```
public Vector3 Normal
```

Field Value

Vector3

Position

The XYZ vertex position.

```
public Vector3 Position
```

Field Value

Vector3

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionColorNormal, VertexPositionColorNormal)

Returns a value that indicates whether two VertexPositionColorNormal are equal

```
public static bool operator ==(VertexPositionColorNormal left,
VertexPositionColorNormal right)
```

Parameters

`left` VertexPositionColorNormal

The object on the left of the equality operator.

`right` VertexPositionColorNormal

The object on the right of the equality operator.

Returns

bool

true if the objects are the same; false otherwise.

operator !=(VertexPositionColorNormal, VertexPositionColorNormal)

Returns a value that indicates whether two VertexPositionColorNormal are different

```
public static bool operator !=(VertexPositionColorNormal left,
VertexPositionColorNormal right)
```

Parameters

`left` VertexPositionColorNormal

The object on the left of the inequality operator.

`right` VertexPositionColorNormal

The object on the right of the inequality operator.

Returns

bool

true if the objects are different; false otherwise.

Struct VertexPositionColorNormalTexture

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position, color, normal data, and one set of texture coordinates.

```
public struct VertexPositionColorNormalTexture : IVertexType
```

Implements
IVertexType

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionColorNormalTexture(Vector3, Color, Vector3, Vector2)

Creates an instance of VertexPositionColorTexture.

```
public VertexPositionColorNormalTexture(Vector3 position, Color color, Vector3
normal, Vector2 textureCoordinate)
```

Parameters

`position` Vector3

Position of the vertex.

`color` Color

Color of the vertex.

`normal` Vector3

The vertex normal.

`textureCoordinate` Vector2

Texture coordinate of the vertex.

Fields

Color

The vertex color.

```
public Color Color
```

Field Value

Color

Normal

The XYZ surface normal.

```
public Vector3 Normal
```

Field Value

Vector3

Position

The XYZ vertex position.

```
public Vector3 Position
```

Field Value

Vector3

TextureCoordinate

UV texture coordinates.

```
public Vector2 TextureCoordinate
```

Field Value

Vector2

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionColorNormalTexture, VertexPositionColorNormalTexture)

Returns a value that indicates whether two VertexPositionColorNormalTexture are equal

```
public static bool operator ==(VertexPositionColorNormalTexture left,
VertexPositionColorNormalTexture right)
```

Parameters

`left` VertexPositionColorNormalTexture

The object on the left of the equality operator.

`right` VertexPositionColorNormalTexture

The object on the right of the equality operator.

Returns

bool

true if the objects are the same; false otherwise.

operator !=(VertexPositionColorNormalTexture, VertexPositionColorNormalTexture)

Returns a value that indicates whether two VertexPositionColorNormalTexture are different

```
public static bool operator !=(VertexPositionColorNormalTexture left,
VertexPositionColorNormalTexture right)
```

Parameters

`left` VertexPositionColorNormalTexture

The object on the left of the inequality operator.

`right` VertexPositionColorNormalTexture

The object on the right of the inequality operator.

Returns

bool

true if the objects are different; false otherwise.

Struct VertexPositionColorTexture

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position, color, and one set of texture coordinates.

```
public struct VertexPositionColorTexture : IVertexType
```

Implements
IVertexType

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionColorTexture(Vector3, Color, Vector2)

Creates an instance of VertexPositionColorTexture.

```
public VertexPositionColorTexture(Vector3 position, Color color,
Vector2 textureCoordinate)
```

Parameters

`position` Vector3

Position of the vertex.

`color` Color

Color of the vertex.

`textureCoordinate` Vector2

Texture coordinate of the vertex.

Fields

Color

The vertex color.

```
public Color Color
```

Field Value

Color

Position

The XYZ vertex position.

```
public Vector3 Position
```

Field Value

Vector3

TextureCoordinate

UV texture coordinates.

```
public Vector2 TextureCoordinate
```

Field Value

Vector2

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionColorTexture, VertexPositionColorTexture)

Returns a value that indicates whether two VertexPositionColorTexture are equal

```
public static bool operator ==(VertexPositionColorTexture left,
VertexPositionColorTexture right)
```

Parameters

`left` VertexPositionColorTexture

The object on the left of the equality operator.

`right` VertexPositionColorTexture

The object on the right of the equality operator.

Returns

bool

true if the objects are the same; false otherwise.

operator !=(VertexPositionColorTexture, VertexPositionColorTexture)

Returns a value that indicates whether two VertexPositionColorTexture are different

```
public static bool operator !=(VertexPositionColorTexture left,
VertexPositionColorTexture right)
```

Parameters

`left` VertexPositionColorTexture

The object on the left of the inequality operator.

`right` VertexPositionColorTexture

The object on the right of the inequality operator.

Returns

bool

true if the objects are different; false otherwise.

Struct VertexPositionNormalTexture

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position, normal data, and one set of texture coordinates.

```
public struct VertexPositionNormalTexture : IVertexType
```

Implements
IVertexType

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionNormalTexture(Vector3, Vector3, Vector2)

Creates an instance of VertexPositionTexture.

```
public VertexPositionNormalTexture(Vector3 position, Vector3 normal,
Vector2 textureCoordinate)
```

Parameters

`position` Vector3

Position of the vertex.

`normal` Vector3

The vertex normal.

`textureCoordinate` Vector2

Texture coordinate of the vertex.

Fields

Normal

The XYZ surface normal.

```
public Vector3 Normal
```

Field Value

Vector3

Position

The XYZ vertex position.

```
public Vector3 Position
```

Field Value

Vector3

TextureCoordinate

UV texture coordinates.

```
public Vector2 TextureCoordinate
```

Field Value

Vector2

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionNormalTexture, VertexPositionNormalTexture)

Returns a value that indicates whether two VertexPositionNormalTexture are equal

```
public static bool operator ==(VertexPositionNormalTexture left,
VertexPositionNormalTexture right)
```

Parameters

`left` VertexPositionNormalTexture

The object on the left of the equality operator.

`right` VertexPositionNormalTexture

The object on the right of the equality operator.

Returns

bool

true if the objects are the same; false otherwise.

operator !=(VertexPositionNormalTexture, VertexPositionNormalTexture)

Returns a value that indicates whether two VertexPositionNormalTexture are different

```
public static bool operator !=(VertexPositionNormalTexture left,
VertexPositionNormalTexture right)
```

Parameters

`left` VertexPositionNormalTexture

The object on the left of the inequality operator.

`right` VertexPositionNormalTexture

The object on the right of the inequality operator.

Returns

bool

true if the objects are different; false otherwise.

Struct VertexPositionTexture

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes a custom vertex format structure that contains position and one set of texture coordinates.

```
public struct VertexPositionTexture : IVertexType
```

Implements
IVertexType

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

VertexPositionTexture(Vector3, Vector2)

Creates an instance of VertexPositionTexture.

```
public VertexPositionTexture(Vector3 position, Vector2 textureCoordinate)
```

Parameters

`position` Vector3

Position of the vertex.

`textureCoordinate` Vector2

Texture coordinate of the vertex.

Fields

Position

The XYZ vertex position.

```
public Vector3 Position
```

Field Value

Vector3

TextureCoordinate

UV texture coordinates.

```
public Vector2 TextureCoordinate
```

Field Value

Vector2

VertexDeclaration

Vertex declaration, which defines per-vertex data.

```
public static readonly VertexDeclaration VertexDeclaration
```

Field Value

VertexDeclaration

Methods

Equals(object)

Compares an object with the vertex.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare.

Returns

bool

true if the object is equal to the current vertex; false otherwise.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Retrieves a string representation of this object.

```
public override string ToString()
```

Returns

string

String representation of this object.

Operators

operator ==(VertexPositionTexture, VertexPositionTexture)

Returns a value that indicates whether two VertexPositionTexture are equal

```
public static bool operator ==(VertexPositionTexture left,
VertexPositionTexture right)
```

Parameters

`left` VertexPositionTexture

The vertex on the left of the equality operator.

`right` VertexPositionTexture

The vertex on the right of the equality operator.

Returns

bool

true if the vertices are the same; false otherwise.

operator !=(VertexPositionTexture, VertexPositionTexture)

Returns a value that indicates whether two VertexPositionTexture are different

```
public static bool operator !=(VertexPositionTexture left,
VertexPositionTexture right)
```

Parameters

`left` VertexPositionTexture

The vertex on the left of the equality operator.

`right` VertexPositionTexture

The vertex on the right of the equality operator.

Returns

bool

true if the vertices are different; false otherwise.

Struct Viewport

Namespace: Microsoft.Xna.Framework.Graphics

Assembly: MonoGame.Framework.dll

Describes the view bounds for render-target surface.

```
[DataContract]
public struct Viewport
```

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Viewport(Rectangle)

Creates a new instance of Viewport struct.

```
public Viewport(Rectangle bounds)
```

Parameters

`bounds` Rectangle

A Rectangle that defines the location and size of the Viewport in a render target.

Viewport(int, int, int, int)

Constructs a viewport from the given values. The MinDepth will be 0.0 and MaxDepth will be 1.0.

```
public Viewport(int x, int y, int width, int height)
```

Parameters

`x` int

The x coordinate of the upper-left corner of the view bounds in pixels.

`y` int

The y coordinate of the upper-left corner of the view bounds in pixels.

`width` int

The width of the view bounds in pixels.

`height` int

The height of the view bounds in pixels.

Viewport(int, int, int, int, float, float)

Constructs a viewport from the given values.

```
public Viewport(int x, int y, int width, int height, float minDepth, float maxDepth)
```

Parameters

`x` int

The x coordinate of the upper-left corner of the view bounds in pixels.

`y` int

The y coordinate of the upper-left corner of the view bounds in pixels.

`width` int

The width of the view bounds in pixels.

`height` int

The height of the view bounds in pixels.

`minDepth` float

The lower limit of depth.

`maxDepth` float

The upper limit of depth.

Properties

AspectRatio

Gets the aspect ratio of this Viewport, which is width / height.

```
public float AspectRatio { get; }
```

Property Value

float

Bounds

Gets or sets a boundary of this Viewport.

```
public Rectangle Bounds { get; set; }
```

Property Value

Rectangle

Height

The height of the bounds in pixels.

```
[DataMember]
public int Height { get; set; }
```

Property Value

int

MaxDepth

The upper limit of depth of this viewport.

```
[DataMember]
public float MaxDepth { get; set; }
```

Property Value

float

MinDepth

The lower limit of depth of this viewport.

```
[DataMember]
public float MinDepth { get; set; }
```

Property Value

float

TitleSafeArea

Returns the subset of the viewport that is guaranteed to be visible on a lower quality display.

```
public Rectangle TitleSafeArea { get; }
```

Property Value

Rectangle

Width

The width of the bounds in pixels.

```
[DataMember]
public int Width { get; set; }
```

Property Value

int

X

The x coordinate of the beginning of this viewport.

```
[DataMember]
public int X { get; set; }
```

Property Value

int

Y

The y coordinate of the beginning of this viewport.

```
[DataMember]
public int Y { get; set; }
```

Property Value

int

Methods

Project(Vector3, Matrix, Matrix, Matrix)

Projects a Vector3 from model space into screen space. The source point is transformed from model space to world space by the world matrix, then from world space to view space by the view matrix, and finally from view space to screen space by the projection matrix.

```
public Vector3 Project(Vector3 source, Matrix projection, Matrix view, Matrix world)
```

Parameters

`source` Vector3

The Vector3 to project.

`projection` Matrix

The projection Matrix.

`view` Matrix

The view Matrix.

`world` Matrix

The world Matrix.

Returns

Vector3

ToString()

Returns a string representation of this Viewport in the format: {X:[X] Y:[Y] Width:[Width] Height:[Height] MinDepth:[MinDepth] MaxDepth:[MaxDepth]}

```
public override string ToString()
```

Returns

string

A string representation of this Viewport.

Unproject(Vector3, Matrix, Matrix, Matrix)

Unprojects a Vector3 from screen space into model space. The source point is transformed from screen space to view space by the inverse of the projection matrix, then from view space to world space by the inverse of the view matrix, and finally from world space to model space by the inverse of the world matrix. Note source.Z must be less than or equal to MaxDepth.

```
public Vector3 Unproject(Vector3 source, Matrix projection, Matrix view,
Matrix world)
```

Parameters

`source` Vector3

The Vector3 to unproject.

`projection` Matrix

The projection Matrix.

`view` Matrix

The view Matrix.

`world` Matrix

The world Matrix.

Returns

Vector3

Namespace Microsoft.Xna.Framework.Graphics.PackedVector

Structs

Alpha8

Packed vector type containing a single 8 bit normalized W values that is ranging from 0 to 1.

Bgr565

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x and z components use 5 bits, and the y component uses 6 bits.

Bgra4444

Packed vector type containing unsigned normalized values, ranging from 0 to 1, using 4 bits each for x, y, z, and w.

Bgra5551

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x , y and z components use 5 bits, and the w component uses 1 bit.

Byte4

Packed vector type containing four 8-bit unsigned integer values, ranging from 0 to 255.

HalfSingle

Packed vector type containing a single 16-bit floating point

HalfVector2

Packed vector type containing two 16-bit floating-point values.

HalfVector4

Packed vector type containing four 16-bit floating-point values.

NormalizedByte2

Packed vector type containing two 8-bit signed normalized values, ranging from −1 to 1.

NormalizedByte4

Packed vector type containing four 8-bit signed normalized values, ranging from −1 to 1.

NormalizedShort2

Packed vector type containing two 16-bit signed normalized values, ranging from −1 to 1.

NormalizedShort4

Packed vector type containing four 16-bit signed normalized values, ranging from −1 to 1.

Rg32

Packed vector type containing two 16-bit unsigned normalized values ranging from 0 to 1.

Rgba1010102

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x, y and z components use 10 bits, and the w component uses 2 bits.

Rgba64

Packed vector type containing four 16-bit unsigned normalized values ranging from 0 to 1.

Short2

Packed vector type containing two 16-bit signed integer values.

Short4

Packed vector type containing four 16-bit signed integer values.

Interfaces

IPackedVector

Interface that converts packed vector types to and from Vector4 values, allowing multiple encodings to be manipulated in a generic way.

IPackedVector<TPacked>

Converts packed vector types to and from Vector4 values.

Struct Alpha8

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing a single 8 bit normalized W values that is ranging from 0 to 1.

```
public struct Alpha8 : IPackedVector<byte>, IEquatable<Alpha8>, IPackedVector
```

Implements

IPackedVector<byte>, IEquatable<Alpha8>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Alpha8(float)

Initializes a new instance of this structure.

```
public Alpha8(float alpha)
```

Parameters

`alpha` float

The initial value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public byte PackedValue { get; set; }
```

Property Value

byte

Methods

Equals(Alpha8)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Alpha8 other)
```

Parameters

`other` Alpha8

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToAlpha()

Expands the packed representation to a System.Single

```
public float ToAlpha()
```

Returns

float

The expanded value.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Alpha8, Alpha8)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Alpha8 lhs, Alpha8 rhs)
```

Parameters

`lhs` Alpha8

The value on the left of the equality operator.

`rhs` Alpha8

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Alpha8, Alpha8)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Alpha8 lhs, Alpha8 rhs)
```

Parameters

`lhs` Alpha8

The value on the left of the inequality operator.

`rhs` Alpha8

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Bgr565

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x and z components use 5 bits, and the y component uses 6 bits.

```
public struct Bgr565 : IPackedVector<ushort>, IEquatable<Bgr565>, IPackedVector
```

Implements

IPackedVector<ushort>, IEquatable<Bgr565>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Bgr565(Vector3)

Initializes a new instance of this structure.

```
public Bgr565(Vector3 vector)
```

Parameters

`vector` Vector3

A Vector3 value who's components contain the initial values for this structure.

Bgr565(float, float, float)

Initializes a new instance of this structure.

```
public Bgr565(float x, float y, float z)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ushort PackedValue { get; set; }
```

Property Value

ushort

Methods

Equals(Bgr565)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Bgr565 other)
```

Parameters

`other` Bgr565

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector3()

Expands the packed representation of this structure to a Vector3.

```
public Vector3 ToVector3()
```

Returns

Vector3

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Bgr565, Bgr565)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Bgr565 lhs, Bgr565 rhs)
```

Parameters

`lhs` Bgr565

The value on the left of the equality operator.

`rhs` Bgr565

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Bgr565, Bgr565)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Bgr565 lhs, Bgr565 rhs)
```

Parameters

`lhs` Bgr565

The value on the left of the inequality operator.

`rhs` Bgr565

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Bgra4444

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing unsigned normalized values, ranging from 0 to 1, using 4 bits each for x, y, z, and w.

```
public struct Bgra4444 : IPackedVector<ushort>, IPackedVector, IEquatable<Bgra4444>
```

Implements
IPackedVector<ushort>, IPackedVector, IEquatable<Bgra4444>

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Bgra4444(Vector4)

Initializes a new instance of this structure.

```
public Bgra4444(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Bgra4444(float, float, float, float)

Initializes a new instance of this structure.

```
public Bgra4444(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ushort PackedValue { get; set; }
```

Property Value

ushort

Methods

Equals(Bgra4444)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Bgra4444 other)
```

Parameters

`other` Bgra4444

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Bgra4444, Bgra4444)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Bgra4444 lhs, Bgra4444 rhs)
```

Parameters

`lhs` Bgra4444

The value on the left of the equality operator.

`rhs` Bgra4444

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Bgra4444, Bgra4444)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Bgra4444 lhs, Bgra4444 rhs)
```

Parameters

`lhs` Bgra4444

The value on the left of the inequality operator.

`rhs` Bgra4444

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Bgra5551

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x , y and z components use 5 bits, and the w component uses 1 bit.

```
public struct Bgra5551 : IPackedVector<ushort>, IEquatable<Bgra5551>, IPackedVector
```

Implements

IPackedVector<ushort>, IEquatable<Bgra5551>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Bgra5551(Vector4)

Initializes a new instance of this structure.

```
public Bgra5551(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Bgra5551(float, float, float, float)

Initializes a new instance of this structure.

```
public Bgra5551(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ushort PackedValue { get; set; }
```

Property Value

ushort

Methods

Equals(Bgra5551)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Bgra5551 other)
```

Parameters

`other` Bgra5551

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Bgra5551, Bgra5551)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Bgra5551 lhs, Bgra5551 rhs)
```

Parameters

`lhs` Bgra5551

The value on the left of the equality operator.

`rhs` Bgra5551

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Bgra5551, Bgra5551)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Bgra5551 lhs, Bgra5551 rhs)
```

Parameters

`lhs` Bgra5551

The value on the left of the inequality operator.

`rhs` Bgra5551

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Byte4

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 8-bit unsigned integer values, ranging from 0 to 255.

```
[TypeConverter(typeof(Byte4TypeConverter))]
public struct Byte4 : IPackedVector<uint>, IEquatable<Byte4>, IPackedVector
```

Implements

IPackedVector<uint>, IEquatable<Byte4>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Byte4()

Initializes a new instance of this structure.

```
public Byte4()
```

Byte4(Vector4)

Initializes a new instance of this structure.

```
public Byte4(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Byte4(float, float, float, float)

Initializes a new instance of this structure.

```
public Byte4(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(Byte4)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Byte4 other)
```

Parameters

`other` Byte4

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Byte4, Byte4)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Byte4 a, Byte4 b)
```

Parameters

a Byte4

The value on the left of the equality operator.

b Byte4

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Byte4, Byte4)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Byte4 a, Byte4 b)
```

Parameters

a Byte4

The value on the left of the inequality operator.

b Byte4

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct HalfSingle

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing a single 16-bit floating point

```
public struct HalfSingle : IPackedVector<ushort>, IEquatable<HalfSingle>,
IPackedVector
```

Implements

IPackedVector<ushort>, IEquatable<HalfSingle>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

HalfSingle(float)

Initializes a new instance of this structure.

```
public HalfSingle(float single)
```

Parameters

`single` float

TheThe initial value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ushort PackedValue { get; set; }
```

Property Value

ushort

Methods

Equals(HalfSingle)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(HalfSingle other)
```

Parameters

`other` HalfSingle

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToSingle()

Expands the packed representation to a System.Single

```
public float ToSingle()
```

Returns

float

The expanded value.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(HalfSingle, HalfSingle)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(HalfSingle lhs, HalfSingle rhs)
```

Parameters

`lhs` HalfSingle

The value on the left of the equality operator.

`rhs` HalfSingle

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(HalfSingle, HalfSingle)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(HalfSingle lhs, HalfSingle rhs)
```

Parameters

`lhs` HalfSingle

The value on the left of the inequality operator.

`rhs` HalfSingle

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct HalfVector2

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing two 16-bit floating-point values.

```
public struct HalfVector2 : IPackedVector<uint>, IPackedVector,
IEquatable<HalfVector2>
```

Implements

IPackedVector<uint>, IPackedVector, IEquatable<HalfVector2>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

HalfVector2(Vector2)

Initializes a new instance of this structure.

```
public HalfVector2(Vector2 vector)
```

Parameters

`vector` Vector2

A Vector2 value who's components contain the initial values for this structure.

HalfVector2(float, float)

Initializes a new instance of this structure.

```
public HalfVector2(float x, float y)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(HalfVector2)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(HalfVector2 other)
```

Parameters

`other` HalfVector2

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector2()

Expands the packed representation to a Vector2.

```
public Vector2 ToVector2()
```

Returns

Vector2

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(HalfVector2, HalfVector2)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(HalfVector2 a, HalfVector2 b)
```

Parameters

a HalfVector2

The value on the left of the equality operator.

b HalfVector2

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(HalfVector2, HalfVector2)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(HalfVector2 a, HalfVector2 b)
```

Parameters

a HalfVector2

The value on the left of the inequality operator.

b HalfVector2

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct HalfVector4

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 16-bit floating-point values.

```
public struct HalfVector4 : IPackedVector<ulong>, IPackedVector,
IEquatable<HalfVector4>
```

Implements

IPackedVector<ulong>, IPackedVector, IEquatable<HalfVector4>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

HalfVector4(Vector4)

Initializes a new instance of this structure.

```
public HalfVector4(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

HalfVector4(float, float, float, float)

Initializes a new instance of this structure.

```
public HalfVector4(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ulong PackedValue { get; set; }
```

Property Value

ulong

Methods

Equals(HalfVector4)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(HalfVector4 other)
```

Parameters

`other` HalfVector4

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(HalfVector4, HalfVector4)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(HalfVector4 a, HalfVector4 b)
```

Parameters

a HalfVector4

The value on the left of the equality operator.

b HalfVector4

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(HalfVector4, HalfVector4)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(HalfVector4 a, HalfVector4 b)
```

Parameters

a HalfVector4

The value on the left of the inequality operator.

b HalfVector4

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Interface IPackedVector

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Interface that converts packed vector types to and from Vector4 values, allowing multiple encodings to be manipulated in a generic way.

```
public interface IPackedVector
```

Methods

PackFromVector4(Vector4)

Sets the packed representation from a Vector4.

```
void PackFromVector4(Vector4 vector)
```

Parameters

`vector` Vector4

The Vector4 to create the packed representation from.

ToVector4()

Expands the packed representation into a Vector4.

```
Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Interface IPackedVector<TPacked>

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Converts packed vector types to and from Vector4 values.

```
public interface IPackedVector<TPacked> : IPackedVector
```

Type Parameters

`TPacked`

Converts packed vector types to and from values.

Inherited Members
IPackedVector.PackFromVector4(Vector4) , IPackedVector.ToVector4()

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
TPacked PackedValue { get; set; }
```

Property Value

TPacked

Struct NormalizedByte2

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing two 8-bit signed normalized values, ranging from −1 to 1.

```
public struct NormalizedByte2 : IPackedVector<ushort>,
IPackedVector, IEquatable<NormalizedByte2>
```

Implements

IPackedVector<ushort>, IPackedVector, IEquatable<NormalizedByte2>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

NormalizedByte2(Vector2)

Initializes a new instance of this structure.

```
public NormalizedByte2(Vector2 vector)
```

Parameters

`vector` Vector2

A Vector2 value who's components contain the initial values for this structure.

NormalizedByte2(float, float)

Initializes a new instance of this structure.

```
public NormalizedByte2(float x, float y)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ushort PackedValue { get; set; }
```

Property Value

ushort

Methods

Equals(NormalizedByte2)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(NormalizedByte2 other)
```

Parameters

`other` NormalizedByte2

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector2()

Expands the packed representation to a Vector2.

```
public Vector2 ToVector2()
```

Returns

Vector2

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(NormalizedByte2, NormalizedByte2)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(NormalizedByte2 a, NormalizedByte2 b)
```

Parameters

a NormalizedByte2

The value on the left of the equality operator.

b NormalizedByte2

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(NormalizedByte2, NormalizedByte2)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(NormalizedByte2 a, NormalizedByte2 b)
```

Parameters

a NormalizedByte2

The value on the left of the inequality operator.

b NormalizedByte2

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct NormalizedByte4

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 8-bit signed normalized values, ranging from −1 to 1.

```
public struct NormalizedByte4 : IPackedVector<uint>, IPackedVector,
IEquatable<NormalizedByte4>
```

Implements

IPackedVector<uint>, IPackedVector, IEquatable<NormalizedByte4>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

NormalizedByte4(Vector4)

Initializes a new instance of this structure.

```
public NormalizedByte4(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

NormalizedByte4(float, float, float, float)

Initializes a new instance of this structure.

```
public NormalizedByte4(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(NormalizedByte4)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(NormalizedByte4 other)
```

Parameters

`other` NormalizedByte4

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(NormalizedByte4, NormalizedByte4)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(NormalizedByte4 a, NormalizedByte4 b)
```

Parameters

a NormalizedByte4

The value on the left of the equality operator.

b NormalizedByte4

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(NormalizedByte4, NormalizedByte4)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(NormalizedByte4 a, NormalizedByte4 b)
```

Parameters

a NormalizedByte4

The value on the left of the inequality operator.

b NormalizedByte4

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct NormalizedShort2

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing two 16-bit signed normalized values, ranging from −1 to 1.

```
public struct NormalizedShort2 : IPackedVector<uint>, IPackedVector,
IEquatable<NormalizedShort2>
```

Implements

IPackedVector<uint>, IPackedVector, IEquatable<NormalizedShort2>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

NormalizedShort2(Vector2)

Initializes a new instance of this structure.

```
public NormalizedShort2(Vector2 vector)
```

Parameters

`vector` Vector2

A Vector2 value who's components contain the initial values for this structure.

NormalizedShort2(float, float)

Initializes a new instance of this structure.

```
public NormalizedShort2(float x, float y)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(NormalizedShort2)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(NormalizedShort2 other)
```

Parameters

`other` NormalizedShort2

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector2()

Expands the packed representation to a Vector2.

```
public Vector2 ToVector2()
```

Returns

Vector2

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(NormalizedShort2, NormalizedShort2)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(NormalizedShort2 a, NormalizedShort2 b)
```

Parameters

a NormalizedShort2

The value on the left of the equality operator.

b NormalizedShort2

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(NormalizedShort2, NormalizedShort2)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(NormalizedShort2 a, NormalizedShort2 b)
```

Parameters

a NormalizedShort2

The value on the left of the inequality operator.

b NormalizedShort2

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct NormalizedShort4

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 16-bit signed normalized values, ranging from −1 to 1.

```
public struct NormalizedShort4 : IPackedVector<ulong>,
IPackedVector, IEquatable<NormalizedShort4>
```

Implements

IPackedVector<ulong>, IPackedVector, IEquatable<NormalizedShort4>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

NormalizedShort4(Vector4)

Initializes a new instance of this structure.

```
public NormalizedShort4(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

NormalizedShort4(float, float, float, float)

Initializes a new instance of this structure.

```
public NormalizedShort4(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ulong PackedValue { get; set; }
```

Property Value

ulong

Methods

Equals(NormalizedShort4)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(NormalizedShort4 other)
```

Parameters

`other` NormalizedShort4

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(NormalizedShort4, NormalizedShort4)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(NormalizedShort4 a, NormalizedShort4 b)
```

Parameters

a NormalizedShort4

The value on the left of the equality operator.

b NormalizedShort4

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(NormalizedShort4, NormalizedShort4)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(NormalizedShort4 a, NormalizedShort4 b)
```

Parameters

a NormalizedShort4

The value on the left of the inequality operator.

b NormalizedShort4

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Rg32

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing two 16-bit unsigned normalized values ranging from 0 to 1.

```
public struct Rg32 : IPackedVector<uint>, IEquatable<Rg32>, IPackedVector
```

Implements

IPackedVector<uint>, IEquatable<Rg32>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Rg32(Vector2)

Initializes a new instance of this structure.

```
public Rg32(Vector2 vector)
```

Parameters

`vector` Vector2

A Vector2 value who's components contain the initial values for this structure.

Rg32(float, float)

Initializes a new instance of this structure.

```
public Rg32(float x, float y)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(Rg32)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Rg32 other)
```

Parameters

`other` Rg32

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector2()

Expands the packed representation to a Vector2.

```
public Vector2 ToVector2()
```

Returns

Vector2

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Rg32, Rg32)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Rg32 lhs, Rg32 rhs)
```

Parameters

`lhs` Rg32

The value on the left of the equality operator.

`rhs` Rg32

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Rg32, Rg32)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Rg32 lhs, Rg32 rhs)
```

Parameters

`lhs` Rg32

The value on the left of the inequality operator.

`rhs` Rg32

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Rgba1010102

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing unsigned normalized values ranging from 0 to 1. The x, y and z components use 10 bits, and the w component uses 2 bits.

```
public struct Rgba1010102 : IPackedVector<uint>, IEquatable<Rgba1010102>,
IPackedVector
```

Implements

IPackedVector<uint>, IEquatable<Rgba1010102>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Rgba1010102(Vector4)

Initializes a new instance of this structure.

```
public Rgba1010102(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Rgba1010102(float, float, float, float)

Initializes a new instance of this structure.

```
public Rgba1010102(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(Rgba1010102)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Rgba1010102 other)
```

Parameters

`other` Rgba1010102

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Rgba1010102, Rgba1010102)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Rgba1010102 lhs, Rgba1010102 rhs)
```

Parameters

`lhs` Rgba1010102

The value on the left of the equality operator.

`rhs` Rgba1010102

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Rgba1010102, Rgba1010102)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Rgba1010102 lhs, Rgba1010102 rhs)
```

Parameters

`lhs` Rgba1010102

The value on the left of the inequality operator.

`rhs` Rgba1010102

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Rgba64

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 16-bit unsigned normalized values ranging from 0 to 1.

```
public struct Rgba64 : IPackedVector<ulong>, IEquatable<Rgba64>, IPackedVector
```

Implements

IPackedVector<ulong>, IEquatable<Rgba64>, IPackedVector

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Rgba64(Vector4)

Initializes a new instance of this structure.

```
public Rgba64(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Rgba64(float, float, float, float)

Initializes a new instance of this structure.

```
public Rgba64(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ulong PackedValue { get; set; }
```

Property Value

ulong

Methods

Equals(Rgba64)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Rgba64 other)
```

Parameters

`other` Rgba64

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Rgba64, Rgba64)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Rgba64 lhs, Rgba64 rhs)
```

Parameters

`lhs` Rgba64

The value on the left of the equality operator.

`rhs` Rgba64

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Rgba64, Rgba64)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Rgba64 lhs, Rgba64 rhs)
```

Parameters

`lhs` Rgba64

The value on the left of the inequality operator.

`rhs` Rgba64

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Short2

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing two 16-bit signed integer values.

```
public struct Short2 : IPackedVector<uint>, IPackedVector, IEquatable<Short2>
```

Implements

IPackedVector<uint>, IPackedVector, IEquatable<Short2>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Short2(Vector2)

Initializes a new instance of this structure.

```
public Short2(Vector2 vector)
```

Parameters

`vector` Vector2

A Vector2 value who's components contain the initial values for this structure.

Short2(float, float)

Initializes a new instance of this structure.

```
public Short2(float x, float y)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public uint PackedValue { get; set; }
```

Property Value

uint

Methods

Equals(Short2)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Short2 other)
```

Parameters

`other` Short2

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector2()

Expands the packed representation to a Vector2.

```
public Vector2 ToVector2()
```

Returns

Vector2

The expanded value.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Short2, Short2)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Short2 a, Short2 b)
```

Parameters

a Short2

The value on the left of the equality operator.

b Short2

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Short2, Short2)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Short2 a, Short2 b)
```

Parameters

a Short2

The value on the left of the inequality operator.

b Short2

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Struct Short4

Namespace: Microsoft.Xna.Framework.Graphics.PackedVector

Assembly: MonoGame.Framework.dll

Packed vector type containing four 16-bit signed integer values.

```
public struct Short4 : IPackedVector<ulong>, IPackedVector, IEquatable<Short4>
```

Implements

IPackedVector<ulong>, IPackedVector, IEquatable<Short4>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Short4(Vector4)

Initializes a new instance of this structure.

```
public Short4(Vector4 vector)
```

Parameters

`vector` Vector4

A Vector4 value who's components contain the initial values for this structure.

Short4(float, float, float, float)

Initializes a new instance of this structure.

```
public Short4(float x, float y, float z, float w)
```

Parameters

x float

The initial x-component value for this structure.

y float

The initial y-component value for this structure.

z float

The initial z-component value for this structure.

w float

The initial 2-component value for this structure.

Properties

PackedValue

Directly gets or sets the packed representation of the value.

```
public ulong PackedValue { get; set; }
```

Property Value

ulong

Methods

Equals(Short4)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Short4 other)
```

Parameters

`other` Short4

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

string

The fully qualified type name.

ToVector4()

Expands the packed representation into a Vector4.

```
public Vector4 ToVector4()
```

Returns

Vector4

The expanded Vector4.

Operators

operator ==(Short4, Short4)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(Short4 a, Short4 b)
```

Parameters

a Short4

The value on the left of the equality operator.

b Short4

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(Short4, Short4)

Returns a value that indicates whether the two value are not equal.

```
public static bool operator !=(Short4 a, Short4 b)
```

Parameters

a Short4

The value on the left of the inequality operator.

b Short4

The value on the right of the inequality operator.

Returns

bool

true if the two value are not equal; otherwise, false.

Namespace Microsoft.Xna.Framework.Input

Classes

GamePad

Supports querying the game controllers and setting the vibration motors.

Joystick

Allows interaction with joysticks. Unlike GamePad the number of Buttons/Axes/DPads is not limited.

Keyboard

Allows getting keystrokes from keyboard.

KeyboardInput

Provides access to the keyboard input user interface.

MessageBox

Provides access to the message box user interface.

Mouse

Allows reading position and button click information from mouse.

MouseCursor

Describes a mouse cursor.

Structs

GamePadButtons

A struct that represents the current button states for the controller.

GamePadCapabilities

A stuct that represents the controller capabilities.

GamePadDPad

A struct that represents which directions on the directional pad of a controller are being pressed.

GamePadState

Represents specific information about the state of the controller, including the current state of buttons and sticks.

This is implemented as a partial struct to allow for individual platforms to offer additional data without separate state queries to GamePad.

GamePadThumbSticks

A struct that represents the current stick (thumbstick) states for the controller.

GamePadTriggers

A struct that contains information on the left and the right trigger buttons.

JoystickCapabilities

Describes joystick capabilities.

JoystickHat

Describes joystick hat state.

JoystickState

Describes current joystick state.

KeyboardState

Holds the state of keystrokes by a keyboard.

MouseState

Represents a mouse state with cursor position and button press information.

Enums

ButtonState

Defines a button state for buttons of mouse, gamepad or joystick.

Buttons

Defines the buttons on gamepad.

GamePadDeadZone

Specifies a type of dead zone processing to apply to Xbox 360 Controller analog sticks when calling GetState.

GamePadType

Defines a type of gamepad.

KeyState

Identifies the state of a keyboard key.

Keys

Defines the keys on a keyboard.

Enum ButtonState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Defines a button state for buttons of mouse, gamepad or joystick.

```
public enum ButtonState
```

Fields

`Pressed = 1`

The button is pressed.

`Released = 0`

The button is released.

Enum Buttons

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Defines the buttons on gamepad.

```
[Flags]
public enum Buttons
```

Fields

`A = 4096`

A button.

`B = 8192`

B button.

`Back = 32`

BACK button.

`BigButton = 2048`

Big button.

`DPadDown = 2`

Directional pad down.

`DPadLeft = 4`

Directional pad left.

`DPadRight = 8`

Directional pad right.

`DPadUp = 1`

Directional pad up.

`LeftShoulder = 256`

Left bumper (shoulder) button.

`LeftStick = 64`

Left stick button (pressing the left stick).

`LeftThumbstickDown = 536870912`

Left stick is towards down.

`LeftThumbstickLeft = 2097152`

Left stick is towards the left.

`LeftThumbstickRight = 1073741824`

Left stick is towards the right.

`LeftThumbstickUp = 268435456`

Left stick is towards up.

`LeftTrigger = 8388608`

Left trigger.

`None = 0`

No button set.

`RightShoulder = 512`

Right bumper (shoulder) button.

`RightStick = 128`

Right stick button (pressing the right stick).

`RightThumbstickDown = 33554432`

Right stick is towards down.

`RightThumbstickLeft = 134217728`

Right stick is towards the left.

`RightThumbstickRight = 67108864`

Right stick is towards the right.

`RightThumbstickUp = 16777216`

Right stick is towards up.

`RightTrigger = 4194304`

Right trigger.

`Start = 16`

START button.

`X = 16384`

X button.

`Y = 32768`

Y button.

Class GamePad

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Supports querying the game controllers and setting the vibration motors.

```
public static class GamePad
```

Inheritance

object ← GamePad

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

MaximumGamePadCount

The maximum number of game pads supported on this system.

```
public static int MaximumGamePadCount { get; }
```

Property Value

int

Methods

GetCapabilities(PlayerIndex)

Returns the capabilities of the connected controller.

```
public static GamePadCapabilities GetCapabilities(PlayerIndex playerIndex)
```

Parameters

`playerIndex` PlayerIndex

Player index for the controller you want to query.

Returns

GamePadCapabilities

The capabilities of the controller.

GetCapabilities(int)

Returns the capabilities of the connected controller.

```
public static GamePadCapabilities GetCapabilities(int index)
```

Parameters

`index` int

Index for the controller you want to query.

Returns

GamePadCapabilities

The capabilities of the controller.

GetState(PlayerIndex)

Gets the current state of a game pad controller with an independent axes dead zone.

```
public static GamePadState GetState(PlayerIndex playerIndex)
```

Parameters

`playerIndex` PlayerIndex

Player index for the controller you want to query.

Returns

GamePadState

The state of the controller.

GetState(PlayerIndex, GamePadDeadZone)

Gets the current state of a game pad controller, using a specified dead zone on analog stick positions.

```
public static GamePadState GetState(PlayerIndex playerIndex,
GamePadDeadZone deadZoneMode)
```

Parameters

`playerIndex` PlayerIndex

Player index for the controller you want to query.

`deadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use.

Returns

GamePadState

The state of the controller.

GetState(PlayerIndex, GamePadDeadZone, GamePadDeadZone)

Gets the current state of a game pad controller, using a specified dead zone on analog stick positions.

```
public static GamePadState GetState(PlayerIndex playerIndex, GamePadDeadZone
leftDeadZoneMode, GamePadDeadZone rightDeadZoneMode)
```

Parameters

`playerIndex` PlayerIndex

Player index for the controller you want to query.

`leftDeadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use for the left stick.

`rightDeadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use for the right stick.

Returns

GamePadState

The state of the controller.

GetState(int)

Gets the current state of a game pad controller with an independent axes dead zone.

```
public static GamePadState GetState(int index)
```

Parameters

`index` int

Index for the controller you want to query.

Returns

GamePadState

The state of the controller.

GetState(int, GamePadDeadZone)

Gets the current state of a game pad controller, using a specified dead zone on analog stick positions.

```
public static GamePadState GetState(int index, GamePadDeadZone deadZoneMode)
```

Parameters

`index` int

Index for the controller you want to query.

`deadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use.

Returns

GamePadState

The state of the controller.

GetState(int, GamePadDeadZone, GamePadDeadZone)

Gets the current state of a game pad controller, using a specified dead zone on analog stick positions.

```
public static GamePadState GetState(int index, GamePadDeadZone leftDeadZoneMode,
GamePadDeadZone rightDeadZoneMode)
```

Parameters

`index` int

Index for the controller you want to query.

`leftDeadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use for the left stick.

`rightDeadZoneMode` GamePadDeadZone

Enumerated value that specifies what dead zone type to use for the right stick.

Returns

GamePadState

The state of the controller.

SetVibration(PlayerIndex, float, float)

Sets the vibration motor speeds on the controller device if supported.

```
public static bool SetVibration(PlayerIndex playerIndex, float leftMotor,
float rightMotor)
```

Parameters

`playerIndex` PlayerIndex

Player index that identifies the controller to set.

`leftMotor` float

The speed of the left motor, between 0.0 and 1.0. This motor is a low-frequency motor.

`rightMotor` float

The speed of the right motor, between 0.0 and 1.0. This motor is a high-frequency motor.

Returns

bool

Returns true if the vibration motors were set.

SetVibration(PlayerIndex, float, float, float, float)

Sets the vibration motor speeds on the controller device if supported.

```
public static bool SetVibration(PlayerIndex playerIndex, float leftMotor, float
rightMotor, float leftTrigger, float rightTrigger)
```

Parameters

`playerIndex` PlayerIndex

Player index that identifies the controller to set.

`leftMotor` float

The speed of the left motor, between 0.0 and 1.0. This motor is a low-frequency motor.

`rightMotor` float

The speed of the right motor, between 0.0 and 1.0. This motor is a high-frequency motor.

`leftTrigger` float

(Xbox One controller only) The speed of the left trigger motor, between 0.0 and 1.0. This motor is a high-frequency motor.

`rightTrigger` float

(Xbox One controller only) The speed of the right trigger motor, between 0.0 and 1.0. This motor is a high-frequency motor.

Returns

bool

Returns true if the vibration motors were set.

SetVibration(int, float, float)

Sets the vibration motor speeds on the controller device if supported.

```
public static bool SetVibration(int index, float leftMotor, float rightMotor)
```

Parameters

`index` int

Index for the controller you want to query.

`leftMotor` float

The speed of the left motor, between 0.0 and 1.0. This motor is a low-frequency motor.

`rightMotor` float

The speed of the right motor, between 0.0 and 1.0. This motor is a high-frequency motor.

Returns

bool

Returns true if the vibration motors were set.

SetVibration(int, float, float, float, float)

Sets the vibration motor speeds on the controller device if supported.

```
public static bool SetVibration(int index, float leftMotor, float rightMotor, float
leftTrigger, float rightTrigger)
```

Parameters

`index` int

Index for the controller you want to query.

`leftMotor` float

The speed of the left motor, between 0.0 and 1.0. This motor is a low-frequency motor.

`rightMotor` float

The speed of the right motor, between 0.0 and 1.0. This motor is a high-frequency motor.

`leftTrigger` float

(Xbox One controller only) The speed of the left trigger motor, between 0.0 and 1.0. This motor is a high-frequency motor.

`rightTrigger` float

(Xbox One controller only) The speed of the right trigger motor, between 0.0 and 1.0. This motor is a high-frequency motor.

Returns

bool

Returns true if the vibration motors were set.

Struct GamePadButtons

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

A struct that represents the current button states for the controller.

```
public struct GamePadButtons
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GamePadButtons(Buttons)

Initializes a new instance of the GamePadButtons structure, setting the specified buttons to pressed in.

```
public GamePadButtons(Buttons buttons)
```

Parameters

`buttons` Buttons

Buttons to be set as pressed in.

Properties

A

Gets a value indicating if the button A is pressed.

```
public ButtonState A { get; }
```

Property Value

ButtonState

Pressed if the button A is pressed; otherwise, Released.

B

Gets a value indicating if the button B is pressed.

```
public ButtonState B { get; }
```

Property Value

ButtonState

Pressed if the button B is pressed; otherwise, Released.

Back

Gets a value indicating if the button Back is pressed.

```
public ButtonState Back { get; }
```

Property Value

ButtonState

Pressed if the button Back is pressed; otherwise, Released.

BigButton

Gets a value indicating if the guide button is pressed.

```
public ButtonState BigButton { get; }
```

Property Value

ButtonState

Pressed if the guide button is pressed; otherwise, Released.

LeftShoulder

Gets a value indicating if the left shoulder button is pressed.

```
public ButtonState LeftShoulder { get; }
```

Property Value

ButtonState

Pressed if the left shoulder button is pressed; otherwise, Released.

LeftStick

Gets a value indicating if the left stick button is pressed.

```
public ButtonState LeftStick { get; }
```

Property Value

ButtonState

Pressed if the left stick button is pressed; otherwise, Released.

RightShoulder

Gets a value indicating if the right shoulder button is pressed.

```
public ButtonState RightShoulder { get; }
```

Property Value

ButtonState

Pressed if the right shoulder button is pressed; otherwise, Released.

RightStick

Gets a value indicating if the right stick button is pressed.

```
public ButtonState RightStick { get; }
```

Property Value

ButtonState

Pressed if the right stick button is pressed; otherwise, Released.

Start

Gets a value indicating if the button Start is pressed.

```
public ButtonState Start { get; }
```

Property Value

ButtonState

Pressed if the button Start is pressed; otherwise, Released.

X

Gets a value indicating if the button X is pressed.

```
public ButtonState X { get; }
```

Property Value

ButtonState

Pressed if the button X is pressed; otherwise, Released.

Y

Gets a value indicating if the button Y is pressed.

```
public ButtonState Y { get; }
```

Property Value

ButtonState

Pressed if the button Y is pressed; otherwise, Released.

Methods

Equals(object)

Returns a value indicating whether this instance is equal to a specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

An object to compare to this instance.

Returns

bool

true if `obj` is a GamePadButtons and has the same value as this instance; otherwise, false.

GetHashCode()

Serves as a hash function for a GamePadButtons object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current GamePadButtons.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadButtons.

Operators

operator ==(GamePadButtons, GamePadButtons)

Determines whether two specified instances of GamePadButtons are equal.

```
public static bool operator ==(GamePadButtons left, GamePadButtons right)
```

Parameters

`left` GamePadButtons

The first object to compare.

`right` GamePadButtons

The second object to compare.

Returns

bool

true if `left` and `right` are equal; otherwise, false.

operator !=(GamePadButtons, GamePadButtons)

Determines whether two specified instances of GamePadButtons are not equal.

```
public static bool operator !=(GamePadButtons left, GamePadButtons right)
```

Parameters

`left` GamePadButtons

The first object to compare.

`right` GamePadButtons

The second object to compare.

Returns

bool

true if `left` and `right` are not equal; otherwise, false.

Struct GamePadCapabilities

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

A stuct that represents the controller capabilities.

```
public struct GamePadCapabilities
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

DisplayName

Gets the gamepad display name.

This property is not available in XNA.

```
public readonly string DisplayName { get; }
```

Property Value

string

String representing the display name of the gamepad.

GamePadType

Gets the type of the controller.

```
public readonly GamePadType GamePadType { get; }
```

Property Value

GamePadType

A GamePadType representing the controller type..

HasAButton

Gets a value indicating whether the controller has the button A.

```
public readonly bool HasAButton { get; }
```

Property Value

bool

`true` if it has the button A; otherwise, `false`.

HasBButton

Gets a value indicating whether the controller has the button B.

```
public readonly bool HasBButton { get; }
```

Property Value

bool

`true` if it has the button B; otherwise, `false`.

HasBackButton

Gets a value indicating whether the controller has the button Back.

```
public readonly bool HasBackButton { get; }
```

Property Value

bool

`true` if it has the button Back; otherwise, `false`.

HasBigButton

Gets a value indicating whether the controller has the guide button.

```
public readonly bool HasBigButton { get; }
```

Property Value

bool

`true` if it has the guide button; otherwise, `false`.

HasDPadDownButton

Gets a value indicating whether the controller has the directional pad down button.

```
public readonly bool HasDPadDownButton { get; }
```

Property Value

bool

`true` if it has the directional pad down button; otherwise, `false`.

HasDPadLeftButton

Gets a value indicating whether the controller has the directional pad left button.

```
public readonly bool HasDPadLeftButton { get; }
```

Property Value

bool

`true` if it has the directional pad left button; otherwise, `false`.

HasDPadRightButton

Gets a value indicating whether the controller has the directional pad right button.

```
public readonly bool HasDPadRightButton { get; }
```

Property Value

bool

`true` if it has the directional pad right button; otherwise, `false`.

HasDPadUpButton

Gets a value indicating whether the controller has the directional pad up button.

```
public readonly bool HasDPadUpButton { get; }
```

Property Value

bool

`true` if it has the directional pad up button; otherwise, `false`.

HasLeftShoulderButton

Gets a value indicating whether the controller has the left shoulder button.

```
public readonly bool HasLeftShoulderButton { get; }
```

Property Value

bool

`true` if it has the left shoulder button; otherwise, `false`.

HasLeftStickButton

Gets a value indicating whether the controller has the left stick button.

```
public readonly bool HasLeftStickButton { get; }
```

Property Value

bool

`true` if it has the left stick button; otherwise, `false`.

HasLeftTrigger

Gets a value indicating whether the controller has the left trigger button.

```
public readonly bool HasLeftTrigger { get; }
```

Property Value

bool

`true` if it has the left trigger button; otherwise, `false`.

HasLeftVibrationMotor

Gets a value indicating whether the controller has the left vibration motor.

```
public readonly bool HasLeftVibrationMotor { get; }
```

Property Value

bool

`true` if it has the left vibration motor; otherwise, `false`.

HasLeftXThumbStick

Gets a value indicating whether the controller has X axis for the left stick (thumbstick) button.

```
public readonly bool HasLeftXThumbStick { get; }
```

Property Value

bool

`true` if it has X axis for the left stick (thumbstick) button; otherwise, `false`.

HasLeftYThumbStick

Gets a value indicating whether the controller has Y axis for the left stick (thumbstick) button.

```
public readonly bool HasLeftYThumbStick { get; }
```

Property Value

bool

`true` if it has Y axis for the left stick (thumbstick) button; otherwise, `false`.

HasRightShoulderButton

Gets a value indicating whether the controller has the right shoulder button.

```
public readonly bool HasRightShoulderButton { get; }
```

Property Value

bool

`true` if it has the right shoulder button; otherwise, `false`.

HasRightStickButton

Gets a value indicating whether the controller has the right stick button.

```
public readonly bool HasRightStickButton { get; }
```

Property Value

bool

`true` if it has the right stick button; otherwise, `false`.

HasRightTrigger

Gets a value indicating whether the controller has the right trigger button.

```
public readonly bool HasRightTrigger { get; }
```

Property Value

bool

`true` if it has the right trigger button; otherwise, `false`.

HasRightVibrationMotor

Gets a value indicating whether the controller has the right vibration motor.

```
public readonly bool HasRightVibrationMotor { get; }
```

Property Value

bool

`true` if it has the right vibration motor; otherwise, `false`.

HasRightXThumbStick

Gets a value indicating whether the controller has X axis for the right stick (thumbstick) button.

```
public readonly bool HasRightXThumbStick { get; }
```

Property Value

bool

`true` if it has X axis for the right stick (thumbstick) button; otherwise, `false`.

HasRightYThumbStick

Gets a value indicating whether the controller has Y axis for the right stick (thumbstick) button.

```
public readonly bool HasRightYThumbStick { get; }
```

Property Value

bool

`true` if it has Y axis for the right stick (thumbstick) button; otherwise, `false`.

HasStartButton

Gets a value indicating whether the controller has the button Start.

```
public readonly bool HasStartButton { get; }
```

Property Value

bool

`true` if it has the button Start; otherwise, `false`.

HasVoiceSupport

Gets a value indicating whether the controller has a microphone.

```
public readonly bool HasVoiceSupport { get; }
```

Property Value

bool

`true` if it has a microphone; otherwise, `false`.

HasXButton

Gets a value indicating whether the controller has the button X.

```
public readonly bool HasXButton { get; }
```

Property Value

bool

`true` if it has the button X; otherwise, `false`.

HasYButton

Gets a value indicating whether the controller has the button Y.

```
public readonly bool HasYButton { get; }
```

Property Value

bool

`true` if it has the button Y; otherwise, `false`.

Identifier

Gets the unique identifier of the gamepad.

This property is not available in XNA.

```
public readonly string Identifier { get; }
```

Property Value

string

String representing the unique identifier of the gamepad.

IsConnected

Gets a value indicating if the controller is connected.

```
public readonly bool IsConnected { get; }
```

Property Value

bool

`true` if it is connected; otherwise, `false`.

Methods

Equals(object)

Determines whether the specified object is equal to the current GamePadCapabilities.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current GamePadCapabilities.

Returns

bool

`true` if the specified object is equal to the current GamePadCapabilities; otherwise, `false`.

GetHashCode()

Serves as a hash function for a GamePadCapabilities object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current GamePadCapabilities.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadCapabilities.

Operators

operator ==(GamePadCapabilities, GamePadCapabilities)

Determines whether a specified instance of GamePadCapabilities is equal to another specified GamePadCapabilities.

```
public static bool operator ==(GamePadCapabilities left, GamePadCapabilities right)
```

Parameters

`left` GamePadCapabilities

The first GamePadCapabilities to compare.

`right` GamePadCapabilities

The second GamePadCapabilities to compare.

Returns

bool

`true` if `left` and `right` are equal; otherwise, `false`.

operator !=(GamePadCapabilities, GamePadCapabilities)

Determines whether a specified instance of GamePadCapabilities is not equal to another specified GamePadCapabilities.

```
public static bool operator !=(GamePadCapabilities left, GamePadCapabilities right)
```

Parameters

`left` GamePadCapabilities

The first GamePadCapabilities to compare.

`right` GamePadCapabilities

The second GamePadCapabilities to compare.

Returns

bool

`true` if `left` and `right` are not equal; otherwise, `false`.

Struct GamePadDPad

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

A struct that represents which directions on the directional pad of a controller are being pressed.

```
public struct GamePadDPad
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GamePadDPad(ButtonState, ButtonState, ButtonState, ButtonState)

Initializes a new instance of the GamePadDPad struct.

```
public GamePadDPad(ButtonState upValue, ButtonState downValue, ButtonState
leftValue, ButtonState rightValue)
```

Parameters

`upValue` ButtonState

Current state of directional pad up.

`downValue` ButtonState

Current state of directional pad down.

`leftValue` ButtonState

Current state of directional pad left.

`rightValue` ButtonState

Current state of directional pad right.

Properties

Down

Gets a value indicating whether down is pressed on the directional pad.

```
public readonly ButtonState Down { get; }
```

Property Value

ButtonState

Pressed if the down button is pressed; otherwise, Released.

Left

Gets a value indicating whether left is pressed on the directional pad.

```
public readonly ButtonState Left { get; }
```

Property Value

ButtonState

Pressed if the left button is pressed; otherwise, Released.

Right

Gets a value indicating whether right is pressed on the directional pad.

```
public readonly ButtonState Right { get; }
```

Property Value

ButtonState

Pressed if the right button is pressed; otherwise, Released.

Up

Gets a value indicating whether up is pressed on the directional pad.

```
public readonly ButtonState Up { get; }
```

Property Value

ButtonState

Pressed if the up button is pressed; otherwise, Released.

Methods

Equals(object)

Returns a value indicating whether this instance is equal to a specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

An object to compare to this instance.

Returns

bool

true if `obj` is a GamePadDPad and has the same value as this instance; otherwise, false.

GetHashCode()

Serves as a hash function for a GamePadDPad object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current GamePadDPad in a format of 0000 where each number represents a boolean value of each respecting object property: Left, Up, Right, Down.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadDPad.

Operators

operator ==(GamePadDPad, GamePadDPad)

Determines whether two specified instances of GamePadDPad are equal.

```
public static bool operator ==(GamePadDPad left, GamePadDPad right)
```

Parameters

`left` GamePadDPad

The first object to compare.

`right` GamePadDPad

The second object to compare.

Returns

bool

true if `left` and `right` are equal; otherwise, false.

operator !=(GamePadDPad, GamePadDPad)

Determines whether two specified instances of GamePadDPad are not equal.

```
public static bool operator !=(GamePadDPad left, GamePadDPad right)
```

Parameters

`left` GamePadDPad

The first object to compare.

`right` GamePadDPad

The second object to compare.

Returns

bool

true if `left` and `right` are not equal; otherwise, false.

Enum GamePadDeadZone

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Specifies a type of dead zone processing to apply to Xbox 360 Controller analog sticks when calling GetState.

```
public enum GamePadDeadZone
```

Fields

`Circular = 2`

The combined X and Y position of each stick is compared to the dead zone. This provides better control than IndependentAxes when the stick is used as a two-dimensional control surface, such as when controlling a character's view in a first-person game.

`IndependentAxes = 1`

The X and Y positions of each stick are compared against the dead zone independently. This setting is the default when calling GetState.

`None = 0`

The values of each stick are not processed and are returned by GetState as "raw" values. This is best if you intend to implement your own dead zone processing.

Struct GamePadState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Represents specific information about the state of the controller, including the current state of buttons and sticks.

This is implemented as a partial struct to allow for individual platforms to offer additional data without separate state queries to GamePad.

```
public struct GamePadState
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GamePadState(GamePadThumbSticks, GamePadTriggers, GamePadButtons, GamePadDPad)

Initializes a new instance of the GamePadState struct using the specified GamePadThumbSticks, GamePadTriggers, GamePadButtons, and GamePadDPad.

```
public GamePadState(GamePadThumbSticks thumbSticks, GamePadTriggers triggers,
GamePadButtons buttons, GamePadDPad dPad)
```

Parameters

`thumbSticks` GamePadThumbSticks

Initial thumbstick state.

`triggers` GamePadTriggers

Initial trigger state.

`buttons` GamePadButtons

Initial button state.

`dPad` GamePadDPad

Initial directional pad state.

GamePadState(Vector2, Vector2, float, float, Buttons)

Initializes a new instance of the GamePadState struct using the specified stick, trigger, and button values.

```
public GamePadState(Vector2 leftThumbStick, Vector2 rightThumbStick, float
leftTrigger, float rightTrigger, Buttons button)
```

Parameters

`leftThumbStick` Vector2

Left stick value. Each axis is clamped between −1.0 and 1.0.

`rightThumbStick` Vector2

Right stick value. Each axis is clamped between −1.0 and 1.0.

`leftTrigger` float

Left trigger value. This value is clamped between 0.0 and 1.0.

`rightTrigger` float

Right trigger value. This value is clamped between 0.0 and 1.0.

`button` Buttons

Button(s) to initialize as pressed.

GamePadState(Vector2, Vector2, float, float, Buttons[])

Initializes a new instance of the GamePadState struct using the specified stick, trigger, and button values.

```
public GamePadState(Vector2 leftThumbStick, Vector2 rightThumbStick, float
leftTrigger, float rightTrigger, Buttons[] buttons)
```

Parameters

`leftThumbStick` Vector2

Left stick value. Each axis is clamped between −1.0 and 1.0.

`rightThumbStick` Vector2

Right stick value. Each axis is clamped between −1.0 and 1.0.

`leftTrigger` float

Left trigger value. This value is clamped between 0.0 and 1.0.

`rightTrigger` float

Right trigger value. This value is clamped between 0.0 and 1.0.

`buttons` Buttons[]

Array of Buttons to initialize as pressed.

Fields

Default

The default initialized gamepad state.

```
public static readonly GamePadState Default
```

Field Value

GamePadState

Properties

Buttons

Gets a structure that identifies what buttons on the controller are pressed.

```
public readonly GamePadButtons Buttons { get; }
```

Property Value

GamePadButtons

The buttons structure.

DPad

Gets a structure that identifies what directions of the directional pad on the controller are pressed.

```
public readonly GamePadDPad DPad { get; }
```

Property Value

GamePadDPad

The directional pad structure.

IsConnected

Gets a value indicating if the controller is connected.

```
public readonly bool IsConnected { get; }
```

Property Value

bool

`true` if it is connected; otherwise, `false`.

PacketNumber

Gets the packet number associated with this state.

```
public readonly int PacketNumber { get; }
```

Property Value

int

The packet number.

ThumbSticks

Gets a structure that indicates the position of the controller sticks (thumbsticks).

```
public readonly GamePadThumbSticks ThumbSticks { get; }
```

Property Value

GamePadThumbSticks

The thumbsticks position.

Triggers

Gets a structure that identifies the position of triggers on the controller.

```
public readonly GamePadTriggers Triggers { get; }
```

Property Value

GamePadTriggers

Positions of the triggers.

Methods

Equals(object)

Determines whether the specified object is equal to the current GamePadState.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current GamePadState.

Returns

bool

`true` if the specified object is equal to the current GamePadState; otherwise, `false`.

GetHashCode()

Serves as a hash function for a GamePadState object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

IsButtonDown(Buttons)

Determines whether specified input device buttons are pressed in this GamePadState.

```
public bool IsButtonDown(Buttons button)
```

Parameters

`button` Buttons

Buttons to query. Specify a single button, or combine multiple buttons using a bitwise OR operation.

Returns

bool

`true`, if button was pressed, `false` otherwise.

IsButtonUp(Buttons)

Determines whether specified input device buttons are released (not pressed) in this GamePadState.

```
public bool IsButtonUp(Buttons button)
```

Parameters

`button` Buttons

Buttons to query. Specify a single button, or combine multiple buttons using a bitwise OR operation.

Returns

bool

`true`, if button was released (not pressed), `false` otherwise.

ToString()

Returns a string that represents the current GamePadState.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadState.

Operators

operator ==(GamePadState, GamePadState)

Determines whether a specified instance of GamePadState is equal to another specified GamePadState.

```
public static bool operator ==(GamePadState left, GamePadState right)
```

Parameters

`left` GamePadState

The first GamePadState to compare.

`right` GamePadState

The second GamePadState to compare.

Returns

bool

`true` if `left` and `right` are equal; otherwise, `false`.

operator !=(GamePadState, GamePadState)

Determines whether a specified instance of GamePadState is not equal to another specified GamePadState.

```
public static bool operator !=(GamePadState left, GamePadState right)
```

Parameters

`left` GamePadState

The first GamePadState to compare.

`right` GamePadState

The second GamePadState to compare.

Returns

bool

`true` if `left` and `right` are not equal; otherwise, `false`.

Struct GamePadThumbSticks

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

A struct that represents the current stick (thumbstick) states for the controller.

```
public struct GamePadThumbSticks
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GamePadThumbSticks(Vector2, Vector2)

Initializes a new instance of the GamePadThumbSticks struct, setting positions of the sticks.

```
public GamePadThumbSticks(Vector2 leftPosition, Vector2 rightPosition)
```

Parameters

`leftPosition` Vector2

Position of the left controller stick.

`rightPosition` Vector2

Position of the right controller stick.

Properties

Left

Gets a value indicating the position of the left stick (thumbstick).

```
public Vector2 Left { get; }
```

Property Value

Vector2

A Vector2 indicating the current position of the left stick (thumbstick).

Right

Gets a value indicating the position of the right stick (thumbstick).

```
public Vector2 Right { get; }
```

Property Value

Vector2

A Vector2 indicating the current position of the right stick (thumbstick).

Methods

Equals(object)

Returns a value indicating whether this instance is equal to a specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

An object to compare to this instance.

Returns

bool

true if `obj` is a GamePadThumbSticks and has the same value as this instance; otherwise, false.

GetHashCode()

Serves as a hash function for a GamePadThumbSticks object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current GamePadThumbSticks.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadThumbSticks.

Operators

operator ==(GamePadThumbSticks, GamePadThumbSticks)

Determines whether two specified instances of GamePadThumbSticks are equal.

```
public static bool operator ==(GamePadThumbSticks left, GamePadThumbSticks right)
```

Parameters

`left` GamePadThumbSticks

The first object to compare.

`right` GamePadThumbSticks

The second object to compare.

Returns

bool

true if `left` and `right` are equal; otherwise, false.

operator !=(GamePadThumbSticks, GamePadThumbSticks)

Determines whether two specified instances of GamePadThumbSticks are not equal.

```
public static bool operator !=(GamePadThumbSticks left, GamePadThumbSticks right)
```

Parameters

`left` GamePadThumbSticks

The first object to compare.

`right` GamePadThumbSticks

The second object to compare.

Returns

bool

true if `left` and `right` are not equal; otherwise, false.

Struct GamePadTriggers

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

A struct that contains information on the left and the right trigger buttons.

```
public struct GamePadTriggers
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GamePadTriggers(float, float)

Initializes a new instance of the GamePadTriggers struct.

```
public GamePadTriggers(float leftTrigger, float rightTrigger)
```

Parameters

`leftTrigger` float

The position of the left trigger, the value will get clamped between 0.0f and 1.0f.

`rightTrigger` float

The position of the right trigger, the value will get clamped between 0.0f and 1.0f.

Properties

Left

Gets the position of the left trigger.

```
public readonly float Left { get; }
```

Property Value

float

A value from 0.0f to 1.0f representing left trigger.

Right

Gets the position of the right trigger.

```
public readonly float Right { get; }
```

Property Value

float

A value from 0.0f to 1.0f representing right trigger.

Methods

Equals(object)

Returns a value indicating whether this instance is equal to a specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

An object to compare to this instance.

Returns

bool

true if `obj` is a GamePadTriggers and has the same value as this instance; otherwise, false.

GetHashCode()

Serves as a hash function for a GamePadTriggers object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current GamePadTriggers.

```
public override string ToString()
```

Returns

string

A string that represents the current GamePadTriggers.

Operators

operator ==(GamePadTriggers, GamePadTriggers)

Determines whether two specified instances of GamePadTriggers are equal.

```
public static bool operator ==(GamePadTriggers left, GamePadTriggers right)
```

Parameters

`left` GamePadTriggers

The first object to compare.

`right` GamePadTriggers

The second object to compare.

Returns

bool

true if `left` and `right` are equal; otherwise, false.

operator !=(GamePadTriggers, GamePadTriggers)

Determines whether two specified instances of GamePadTriggers are not equal.

```
public static bool operator !=(GamePadTriggers left, GamePadTriggers right)
```

Parameters

`left` GamePadTriggers

The first object to compare.

`right` GamePadTriggers

The second object to compare.

Returns

bool

true if `left` and `right` are not equal; otherwise, false.

Enum GamePadType

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Defines a type of gamepad.

```
public enum GamePadType
```

Fields

`AlternateGuitar = 7`

GamePad is an alternate guitar.

`ArcadeStick = 3`

GamePad is an arcade stick.

`BigButtonPad = 768`

GamePad is a big button pad.

`DancePad = 5`

GamePad is a dance pad.

`DrumKit = 8`

GamePad is a drum kit.

`FlightStick = 4`

GamePad is a flight stick.

`GamePad = 1`

GamePad is the Xbox controller.

`Guitar = 6`

GamePad is a guitar.

`Unknown = 0`

Unknown.

`Wheel = 2`

GamePad is a wheel.

Class Joystick

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Allows interaction with joysticks. Unlike GamePad the number of Buttons/Axes/DPads is not limited.

```
public static class Joystick
```

Inheritance

object ← Joystick

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

IsSupported

Gets a value indicating whether the current platform supports reading raw joystick data.

```
public static bool IsSupported { get; }
```

Property Value

bool

`true` if the current platform supports reading raw joystick data; otherwise, `false`.

LastConnectedIndex

Gets a value indicating the last joystick index connected to the system. If this value is less than 0, no joysticks are connected.

The order joysticks are connected and disconnected determines their index. As such, this value may be larger than 0 even if only one joystick is connected.

```
public static int LastConnectedIndex { get; }
```

Property Value

int

Methods

GetCapabilities(int)

Gets the capabilities of the joystick.

```
public static JoystickCapabilities GetCapabilities(int index)
```

Parameters

`index` int

Index of the joystick you want to access.

Returns

JoystickCapabilities

The capabilities of the joystick.

GetState(ref JoystickState, int)

Gets the current state of the joystick by updating an existing JoystickState.

```
public static void GetState(ref JoystickState joystickState, int index)
```

Parameters

`joystickState` JoystickState

The JoystickState to update.

`index` int

Index of the joystick you want to access.

GetState(int)

Gets the current state of the joystick.

```
public static JoystickState GetState(int index)
```

Parameters

`index` int

Index of the joystick you want to access.

Returns

JoystickState

The state of the joystick.

Struct JoystickCapabilities

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Describes joystick capabilities.

```
public struct JoystickCapabilities
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

AxisCount

Gets the axis count.

```
public readonly int AxisCount { get; }
```

Property Value

int

The number of axes that the joystick possesses.

ButtonCount

Gets the button count.

```
public readonly int ButtonCount { get; }
```

Property Value

int

The number of buttons that the joystick possesses.

DisplayName

Gets the joystick's display name.

```
public readonly string DisplayName { get; }
```

Property Value

string

String representing the display name of the joystick.

HatCount

Gets the hat count.

```
public readonly int HatCount { get; }
```

Property Value

int

The number of hats/dpads that the joystick possesses.

Identifier

Gets the unique identifier of the joystick.

```
public readonly string Identifier { get; }
```

Property Value

string

String representing the unique identifier of the joystick.

IsConnected

Gets a value indicating whether the joystick is connected.

```
public readonly bool IsConnected { get; }
```

Property Value

bool

`true` if the joystick is connected; otherwise, `false`.

IsGamepad

Gets a value indicating if the joystick is a gamepad.

```
public readonly bool IsGamepad { get; }
```

Property Value

bool

`true` if the joystick is a gamepad; otherwise, `false`.

Methods

Equals(object)

Determines whether the specified object is equal to the current JoystickCapabilities.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current JoystickCapabilities.

Returns

bool

`true` if the specified object is equal to the current JoystickCapabilities; otherwise, `false`.

GetHashCode()

Serves as a hash function for a JoystickCapabilities object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current JoystickCapabilities.

```
public override string ToString()
```

Returns

string

A string that represents the current JoystickCapabilities.

Operators

operator ==(JoystickCapabilities, JoystickCapabilities)

Determines whether a specified instance of JoystickCapabilities is equal to another specified JoystickCapabilities.

```
public static bool operator ==(JoystickCapabilities left, JoystickCapabilities
right)
```

Parameters

`left` JoystickCapabilities

The first JoystickCapabilities to compare.

`right` JoystickCapabilities

The second JoystickCapabilities to compare.

Returns

bool

`true` if `left` and `right` are equal; otherwise, `false`.

operator !=(JoystickCapabilities, JoystickCapabilities)

Determines whether a specified instance of JoystickCapabilities is not equal to another specified JoystickCapabilities.

```
public static bool operator !=(JoystickCapabilities left, JoystickCapabilities
right)
```

Parameters

`left` JoystickCapabilities

The first JoystickCapabilities to compare.

`right` JoystickCapabilities

The second JoystickCapabilities to compare.

Returns

bool

`true` if `left` and `right` are not equal; otherwise, `false`.

Struct JoystickHat

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Describes joystick hat state.

```
public struct JoystickHat
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Down

Gets if joysticks hat "down" is pressed.

```
public readonly ButtonState Down { get; }
```

Property Value

ButtonState

Pressed if the button is pressed otherwise, Released.

Left

Gets if joysticks hat "left" is pressed.

```
public readonly ButtonState Left { get; }
```

Property Value

ButtonState

Pressed if the button is pressed otherwise, Released.

Right

Gets if joysticks hat "right" is pressed.

```
public readonly ButtonState Right { get; }
```

Property Value

ButtonState

Pressed if the button is pressed otherwise, Released.

Up

Gets if joysticks hat "up" is pressed.

```
public readonly ButtonState Up { get; }
```

Property Value

ButtonState

Pressed if the button is pressed otherwise, Released.

Methods

Equals(object)

Determines whether the specified object is equal to the current JoystickHat.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current JoystickHat.

Returns

bool

`true` if the specified object is equal to the current JoystickHat; otherwise, `false`.

GetHashCode()

Serves as a hash function for a JoystickHat object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current JoystickHat in a format of 0000 where each number represents a boolean value of each respecting object property: Left, Up, Right, Down.

```
public override string ToString()
```

Returns

string

A string that represents the current JoystickHat.

Operators

operator ==(JoystickHat, JoystickHat)

Determines whether a specified instance of JoystickHat is equal to another specified JoystickHat.

```
public static bool operator ==(JoystickHat left, JoystickHat right)
```

Parameters

`left` JoystickHat

The first JoystickHat to compare.

`right` JoystickHat

The second JoystickHat to compare.

Returns

bool

`true` if `left` and `right` are equal; otherwise, `false`.

operator !=(JoystickHat, JoystickHat)

Determines whether a specified instance of JoystickHat is not equal to another specified JoystickHat.

```
public static bool operator !=(JoystickHat left, JoystickHat right)
```

Parameters

`left` JoystickHat

The first JoystickHat to compare.

`right` JoystickHat

The second JoystickHat to compare.

Returns

bool

`true` if `left` and `right` are not equal; otherwise, `false`.

Struct JoystickState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Describes current joystick state.

```
public struct JoystickState
```

Inherited Members
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Axes

Gets the joystick axis values.

```
public readonly int[] Axes { get; }
```

Property Value

int[]

An array list of ints that indicate axis values.

Buttons

Gets the joystick button values.

```
public readonly ButtonState[] Buttons { get; }
```

Property Value

ButtonState[]

An array list of ButtonState that indicate button values.

Hats

Gets the joystick hat values.

```
public readonly JoystickHat[] Hats { get; }
```

Property Value

JoystickHat[]

An array list of JoystickHat that indicate hat values.

IsConnected

Gets a value indicating whether the joystick is connected.

```
public readonly bool IsConnected { get; }
```

Property Value

bool

`true` if the joystick is connected; otherwise, `false`.

Methods

Equals(object)

Determines whether the specified object is equal to the current JoystickState.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current JoystickState.

Returns

bool

`true` if the specified object is equal to the current JoystickState; otherwise, `false`.

GetHashCode()

Serves as a hash function for a JoystickState object.

```
public override int GetHashCode()
```

Returns

int

A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.

ToString()

Returns a string that represents the current JoystickState.

```
public override string ToString()
```

Returns

string

A string that represents the current JoystickState.

Operators

operator ==(JoystickState, JoystickState)

Determines whether a specified instance of JoystickState is equal to another specified JoystickState.

```
public static bool operator ==(JoystickState left, JoystickState right)
```

Parameters

`left` JoystickState

The first JoystickState to compare.

`right` JoystickState

The second JoystickState to compare.

Returns

bool

`true` if `left` and `right` are equal; otherwise, `false`.

operator !=(JoystickState, JoystickState)

Determines whether a specified instance of JoystickState is not equal to another specified JoystickState.

```
public static bool operator !=(JoystickState left, JoystickState right)
```

Parameters

`left` JoystickState

The first JoystickState to compare.

`right` JoystickState

The second JoystickState to compare.

Returns

bool

`true` if `left` and `right` are not equal; otherwise, `false`.

Enum KeyState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Identifies the state of a keyboard key.

```
public enum KeyState
```

Fields

`Down = 1`

Key is pressed.

`Up = 0`

Key is released.

Class Keyboard

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Allows getting keystrokes from keyboard.

```
public static class Keyboard
```

Inheritance

object ← Keyboard

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetState()

Returns the current keyboard state.

```
public static KeyboardState GetState()
```

Returns

KeyboardState

Current keyboard state.

GetState(PlayerIndex)

Returns the current keyboard state for a given player.

```
[Obsolete("Use GetState() instead. In future versions this method can be removed.")]
public static KeyboardState GetState(PlayerIndex playerIndex)
```

Parameters

`playerIndex` PlayerIndex

Player index of the keyboard.

Returns

KeyboardState

Current keyboard state.

Class KeyboardInput

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Provides access to the keyboard input user interface.

```
public static class KeyboardInput
```

Inheritance

object ← KeyboardInput

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

IsVisible

Determines if a keyboard user interface screen is active.

```
public static bool IsVisible { get; }
```

Property Value

bool

Methods

Cancel(string)

Hides the keyboard input interface and returns the parameter as the result of Show(string, string, string, bool)

```
public static void Cancel(string result)
```

Parameters

`result` string

Result to return

Examples

```
var nameTask = KeyboardInput.Show("Name", "What's your name?", "Player");
KeyboardInput.Cancel("John Doe");
var name = await nameTask;
```

Exceptions

Exception

Thrown when the keyboard input is not visible

Show(string, string, string, bool)

Displays the keyboard input interface asynchronously.

```
public static Task<string> Show(string title, string description, string defaultText
= "", bool usePasswordMode = false)
```

Parameters

`title` string

Title of the dialog box.

`description` string

Description of the dialog box.

`defaultText` string

Default text displayed in the input area.

`usePasswordMode` bool

If password mode is enabled, the characters entered are not displayed.

Returns

Task<string>

Text entered by the player. Null if back was used.

Examples

```
var name = await KeyboardInput.Show("Name", "What's your name?", "Player");
```

Exceptions

Exception

Thrown when the message box is already visible

Struct KeyboardState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Holds the state of keystrokes by a keyboard.

```
public struct KeyboardState
```

Inherited Members
ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

KeyboardState(params Keys[])

Initializes a new instance of the KeyboardState class.

```
public KeyboardState(params Keys[] keys)
```

Parameters

`keys` Keys[]

List of keys to be flagged as pressed on initialization.

KeyboardState(Keys[], bool, bool)

Initializes a new instance of the KeyboardState class.

```
public KeyboardState(Keys[] keys, bool capsLock = false, bool numLock = false)
```

Parameters

`keys` Keys[]

List of keys to be flagged as pressed on initialization.

`capsLock` bool

Caps Lock state.

`numLock` bool

Num Lock state.

Properties

CapsLock

Gets the current state of the Caps Lock key.

```
public bool CapsLock { get; }
```

Property Value

bool

this[Keys]

Returns the state of a specified key.

```
public KeyState this[Keys key] { get; }
```

Parameters

`key` Keys

The key to query.

Property Value

KeyState

The state of the key.

NumLock

Gets the current state of the Num Lock key.

```
public bool NumLock { get; }
```

Property Value

bool

Methods

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The KeyboardState to compare.

Returns

bool

true if the provided KeyboardState instance is same with current; false otherwise.

GetHashCode()

Gets the hash code for KeyboardState instance.

```
public override int GetHashCode()
```

Returns

int

Hash code of the object.

GetPressedKeyCount()

Returns the number of pressed keys in this KeyboardState.

```
public int GetPressedKeyCount()
```

Returns

int

An integer representing the number of keys currently pressed in this KeyboardState.

GetPressedKeys()

Returns an array of values holding keys that are currently being pressed.

```
public Keys[] GetPressedKeys()
```

Returns

Keys[]

The keys that are currently being pressed.

GetPressedKeys(Keys[])

Fills an array of values holding keys that are currently being pressed.

```
public void GetPressedKeys(Keys[] keys)
```

Parameters

`keys` Keys[]

The keys array to fill. This array is not cleared, and it must be equal to or larger than the number of keys pressed.

IsKeyDown(Keys)

Gets whether given key is currently being pressed.

```
public bool IsKeyDown(Keys key)
```

Parameters

`key` Keys

The key to query.

Returns

bool

true if the key is pressed; false otherwise.

IsKeyUp(Keys)

Gets whether given key is currently being not pressed.

```
public bool IsKeyUp(Keys key)
```

Parameters

`key` Keys

The key to query.

Returns

bool

true if the key is not pressed; false otherwise.

Operators

operator ==(KeyboardState, KeyboardState)

Compares whether two KeyboardState instances are equal.

```
public static bool operator ==(KeyboardState a, KeyboardState b)
```

Parameters

a KeyboardState

KeyboardState instance to the left of the equality operator.

b KeyboardState

KeyboardState instance to the right of the equality operator.

Returns

bool

true if the instances are equal; false otherwise.

operator !=(KeyboardState, KeyboardState)

Compares whether two KeyboardState instances are not equal.

```
public static bool operator !=(KeyboardState a, KeyboardState b)
```

Parameters

a KeyboardState

KeyboardState instance to the left of the inequality operator.

b KeyboardState

KeyboardState instance to the right of the inequality operator.

Returns

bool

true if the instances are different; false otherwise.

Enum Keys

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Defines the keys on a keyboard.

```
public enum Keys
```

Fields

`A = 65`

A key.

`Add = 107`

Add key.

`Apps = 93`

Applications key.

`Attn = 246`

Attn key.

`B = 66`

B key.

`Back = 8`

BACKSPACE key.

`BrowserBack = 166`

Browser Back key.

`BrowserFavorites = 171`

Browser Favorites key.

`BrowserForward = 167`

Browser Forward key.

`BrowserHome = 172`

Browser Start and Home key.

`BrowserRefresh = 168`

Browser Refresh key.

`BrowserSearch = 170`

Browser Search key.

`BrowserStop = 169`

Browser Stop key.

`C = 67`

C key.

`CapsLock = 20`

CAPS LOCK key.

`ChatPadGreen = 202`

Green ChatPad key.

`ChatPadOrange = 203`

Orange ChatPad key.

`Crsel = 247`

CrSel key.

`D = 68`

D key.

`D0 = 48`

Used for miscellaneous characters; it can vary by keyboard.

`D1 = 49`

Used for miscellaneous characters; it can vary by keyboard.

`D2 = 50`

Used for miscellaneous characters; it can vary by keyboard.

`D3 = 51`

Used for miscellaneous characters; it can vary by keyboard.

`D4 = 52`

Used for miscellaneous characters; it can vary by keyboard.

`D5 = 53`

Used for miscellaneous characters; it can vary by keyboard.

`D6 = 54`

Used for miscellaneous characters; it can vary by keyboard.

`D7 = 55`

Used for miscellaneous characters; it can vary by keyboard.

`D8 = 56`

Used for miscellaneous characters; it can vary by keyboard.

`D9 = 57`

Used for miscellaneous characters; it can vary by keyboard.

`Decimal = 110`

Decimal key.

`Delete = 46`

DEL key.

`Divide = 111`

Divide key.

`Down = 40`

DOWN ARROW key.

`E = 69`

E key.

`End = 35`

END key.

`Enter = 13`

ENTER key.

`EraseEof = 249`

Erase EOF key.

`Escape = 27`

ESC key.

`Execute = 43`

EXECUTE key.

`Exsel = 248`

ExSel key.

`F = 70`

F key.

`F1 = 112`

F1 key.

`F10 = 121`

F10 key.

`F11 = 122`

F11 key.

`F12 = 123`

F12 key.

`F13 = 124`

F13 key.

`F14 = 125`

F14 key.

`F15 = 126`

F15 key.

`F16 = 127`

F16 key.

`F17 = 128`

F17 key.

`F18 = 129`

F18 key.

`F19 = 130`

F19 key.

`F2 = 113`

F2 key.

`F20 = 131`

F20 key.

`F21 = 132`

F21 key.

`F22 = 133`

F22 key.

`F23 = 134`

F23 key.

`F24 = 135`

F24 key.

`F3 = 114`

F3 key.

`F4 = 115`

F4 key.

`F5 = 116`

F5 key.

`F6 = 117`

F6 key.

`F7 = 118`

F7 key.

`F8 = 119`

F8 key.

`F9 = 120`

F9 key.

`G = 71`

G key.

`H = 72`

H key.

`Help = 47`

HELP key.

`Home = 36`

HOME key.

`I = 73`

I key.

`ImeConvert = 28`

IME Convert key.

`ImeNoConvert = 29`

IME NoConvert key.

`Insert = 45`

INS key.

`J = 74`

J key.

`K = 75`

K key.

`Kana = 21`

Kana key on Japanese keyboards.

`Kanji = 25`

Kanji key on Japanese keyboards.

`L = 76`

L key.

`LaunchApplication1 = 182`

Start Application 1 key.

`LaunchApplication2 = 183`

Start Application 2 key.

`LaunchMail = 180`

Start Mail key.

`Left = 37`

LEFT ARROW key.

`LeftAlt = 164`

Left ALT key.

`LeftControl = 162`

Left CONTROL key.

`LeftShift = 160`

Left SHIFT key.

`LeftWindows = 91`

Left Windows key.

`M = 77`

M key.

`MediaNextTrack = 176`

Next Track key.

`MediaPlayPause = 179`

Play/Pause Media key.

`MediaPreviousTrack = 177`

Previous Track key.

`MediaStop = 178`

Stop Media key.

`Multiply = 106`

Multiply key.

`N = 78`

N key.

`None = 0`

Reserved.

`NumLock = 144`

NUM LOCK key.

`NumPad0 = 96`

Numeric keypad 0 key.

`NumPad1 = 97`

Numeric keypad 1 key.

`NumPad2 = 98`

Numeric keypad 2 key.

`NumPad3 = 99`

Numeric keypad 3 key.

`NumPad4 = 100`

Numeric keypad 4 key.

`NumPad5 = 101`

Numeric keypad 5 key.

`NumPad6 = 102`

Numeric keypad 6 key.

`NumPad7 = 103`

Numeric keypad 7 key.

`NumPad8 = 104`

Numeric keypad 8 key.

`NumPad9 = 105`

Numeric keypad 9 key.

`O = 79`

O key.

`Oem8 = 223`

Used for miscellaneous characters; it can vary by keyboard.

`OemAuto = 243`

OEM Auto key.

`OemBackslash = 226`

The OEM angle bracket or backslash key on the RT 102 key keyboard.

`OemClear = 254`

CLEAR key.

`OemCloseBrackets = 221`

The OEM close bracket key on a US standard keyboard.

`OemComma = 188`

For any country/region, the ',' key.

`OemCopy = 242`

OEM Copy key.

`OemEnlW = 244`

OEM Enlarge Window key.

`OemMinus = 189`

For any country/region, the '-' key.

`OemOpenBrackets = 219`

The OEM open bracket key on a US standard keyboard.

`OemPeriod = 190`

For any country/region, the '.' key.

`OemPipe = 220`

The OEM pipe key on a US standard keyboard.

`OemPlus = 187`

For any country/region, the '+' key.

`OemQuestion = 191`

The OEM question mark key on a US standard keyboard.

`OemQuotes = 222`

The OEM singled/double quote key on a US standard keyboard.

`OemSemicolon = 186`

The OEM Semicolon key on a US standard keyboard.

`OemTilde = 192`

The OEM tilde key on a US standard keyboard.

`P = 80`

P key.

`Pa1 = 253`

PA1 key.

`PageDown = 34`

PAGE DOWN key.

`PageUp = 33`

PAGE UP key.

`Pause = 19`

PAUSE key.

`Play = 250`

Play key.

`Print = 42`

PRINT key.

`PrintScreen = 44`

PRINT SCREEN key.

`ProcessKey = 229`

IME PROCESS key.

`Q = 81`

Q key.

`R = 82`

R key.

`Right = 39`

RIGHT ARROW key.

`RightAlt = 165`

Right ALT key.

`RightControl = 163`

Right CONTROL key.

`RightShift = 161`

Right SHIFT key.

`RightWindows = 92`

Right Windows key.

`S = 83`

S key.

`Scroll = 145`

SCROLL LOCK key.

`Select = 41`

SELECT key.

`SelectMedia = 181`

Select Media key.

`Separator = 108`

Separator key.

`Sleep = 95`

Computer Sleep key.

`Space = 32`

SPACEBAR key.

`Subtract = 109`

Subtract key.

`T = 84`

T key.

`Tab = 9`

TAB key.

`U = 85`

U key.

`Up = 38`

UP ARROW key.

`V = 86`

V key.

`VolumeDown = 174`

Volume Down key.

`VolumeMute = 173`

Volume Mute key.

`VolumeUp = 175`

Volume Up key.

`W = 87`

W key.

`X = 88`

X key.

`Y = 89`

Y key.

`Z = 90`

Z key.

`Zoom = 251`

Zoom key.

Class MessageBox

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Provides access to the message box user interface.

```
public static class MessageBox
```

Inheritance

object ← MessageBox

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

IsVisible

Determines if a message box user interface screen is active.

```
public static bool IsVisible { get; }
```

Property Value

bool

Methods

Cancel(int?)

Hides the message box interface and returns the parameter as the result of Show(string, string, IEnumerable<string>)

```
public static void Cancel(int? result)
```

Parameters

`result` int?

Result to return

Examples

```
var colorTask = MessageBox.Show("Color", "What's your favorite color?", new[] {
"Red", "Green", "Blue" });
MessageBox.Cancel(0);
var color = await colorTask;
```

Exceptions

Exception

Thrown when the message box is not visible

Show(string, string, IEnumerable<string>)

Displays the message box interface asynchronously.

```
public static Task<int?> Show(string title, string description,
IEnumerable<string> buttons)
```

Parameters

`title` string

Title of the message box.

`description` string

Description of the message box.

`buttons` IEnumerable<string>

Captions of the message box buttons. Up to three supported.

Returns

Task<int?>

Index of button selected by the player. Null if back was used.

Examples

```
var color = await MessageBox.Show("Color", "What's your favorite color?", new[] {
"Red", "Green", "Blue" });
```

Exceptions

Exception

Thrown when the message box is already visible

Class Mouse

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Allows reading position and button click information from mouse.

```
public static class Mouse
```

Inheritance

object ← Mouse

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

WindowHandle

Gets or sets the window handle for current mouse processing.

```
public static nint WindowHandle { get; set; }
```

Property Value

nint

Methods

GetState()

Gets mouse state information that includes position and button presses for the primary window

```
public static MouseState GetState()
```

Returns

MouseState

Current state of the mouse.

GetState(GameWindow)

This API is an extension to XNA. Gets mouse state information that includes position and button presses for the provided window

```
public static MouseState GetState(GameWindow window)
```

Parameters

`window` GameWindow

Returns

MouseState

Current state of the mouse.

SetCursor(MouseCursor)

Sets the cursor image to the specified MouseCursor.

```
public static void SetCursor(MouseCursor cursor)
```

Parameters

`cursor` MouseCursor

Mouse cursor to use for the cursor image.

SetPosition(int, int)

Sets mouse cursor's relative position to game-window.

```
public static void SetPosition(int x, int y)
```

Parameters

x int

Relative horizontal position of the cursor.

y int

Relative vertical position of the cursor.

Class MouseCursor

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Describes a mouse cursor.

```
public class MouseCursor : IDisposable
```

Inheritance

object ← MouseCursor

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Arrow

Gets the default arrow cursor.

```
public static MouseCursor Arrow { get; }
```

Property Value

MouseCursor

Crosshair

Gets the crosshair ("+") cursor.

```
public static MouseCursor Crosshair { get; }
```

Property Value

MouseCursor

Hand

Gets the hand cursor, usually used for web links.

```
public static MouseCursor Hand { get; }
```

Property Value

MouseCursor

Handle

Gets a handle for this cursor.

```
public nint Handle { get; }
```

Property Value

nint

IBeam

Gets the cursor that appears when the mouse is over text editing regions.

```
public static MouseCursor IBeam { get; }
```

Property Value

MouseCursor

No

Gets the cursor that points that something is invalid, usually a cross.

```
public static MouseCursor No { get; }
```

Property Value

MouseCursor

SizeAll

Gets the size all cursor which points in all directions.

```
public static MouseCursor SizeAll { get; }
```

Property Value

MouseCursor

SizeNESW

Gets the northeast/southwest ("/") cursor.

```
public static MouseCursor SizeNESW { get; }
```

Property Value

MouseCursor

SizeNS

Gets the vertical north/south ("|") cursor.

```
public static MouseCursor SizeNS { get; }
```

Property Value

MouseCursor

SizeNWSE

Gets the northwest/southeast ("") cursor.

```
public static MouseCursor SizeNWSE { get; }
```

Property Value

MouseCursor

SizeWE

Gets the horizontal west/east ("-") cursor.

```
public static MouseCursor SizeWE { get; }
```

Property Value

MouseCursor

Wait

Gets the waiting cursor that appears while the application/system is busy.

```
public static MouseCursor Wait { get; }
```

Property Value

MouseCursor

WaitArrow

Gets the cross between Arrow and Wait cursors.

```
public static MouseCursor WaitArrow { get; }
```

Property Value

MouseCursor

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

FromTexture2D(Texture2D, int, int)

Creates a mouse cursor from the specified texture.

```
public static MouseCursor FromTexture2D(Texture2D texture, int originx, int originy)
```

Parameters

`texture` Texture2D

Texture to use as the cursor image.

`originx` int

X cordinate of the image that will be used for mouse position.

`originy` int

Y cordinate of the image that will be used for mouse position.

Returns

MouseCursor

Struct MouseState

Namespace: Microsoft.Xna.Framework.Input

Assembly: MonoGame.Framework.dll

Represents a mouse state with cursor position and button press information.

```
public struct MouseState
```

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

MouseState(int, int, int, ButtonState, ButtonState, ButtonState, ButtonState, ButtonState)

Initializes a new instance of the MouseState.

```
public MouseState(int x, int y, int scrollWheel, ButtonState leftButton, ButtonState
middleButton, ButtonState rightButton, ButtonState xButton1, ButtonState xButton2)
```

Parameters

`x` int

Horizontal position of the mouse in relation to the window.

`y` int

Vertical position of the mouse in relation to the window.

`scrollWheel` int

Mouse scroll wheel's value.

`leftButton` ButtonState

Left mouse button's state.

`middleButton` ButtonState

Middle mouse button's state.

`rightButton` ButtonState

Right mouse button's state.

`xButton1` ButtonState

XBUTTON1's state.

`xButton2` ButtonState

XBUTTON2's state.

Remarks

Normally GetState() should be used to get mouse current state. The constructor is provided for simulating mouse input.

MouseState(int, int, int, ButtonState, ButtonState, ButtonState, ButtonState, ButtonState, int)

Initializes a new instance of the MouseState.

```
public MouseState(int x, int y, int scrollWheel, ButtonState leftButton, ButtonState
middleButton, ButtonState rightButton, ButtonState xButton1, ButtonState xButton2,
int horizontalScrollWheel)
```

Parameters

`x` int

Horizontal position of the mouse in relation to the window.

`y` int

Vertical position of the mouse in relation to the window.

`scrollWheel` int

Mouse scroll wheel's value.

`leftButton` ButtonState

Left mouse button's state.

`middleButton` ButtonState

Middle mouse button's state.

`rightButton` ButtonState

Right mouse button's state.

`xButton1` ButtonState

XBUTTON1's state.

`xButton2` ButtonState

XBUTTON2's state.

`horizontalScrollWheel` int

Mouse horizontal scroll wheel's value.

Remarks

Normally GetState() should be used to get mouse current state. The constructor is provided for simulating mouse input.

Properties

HorizontalScrollWheelValue

Returns the cumulative horizontal scroll wheel value since the game start

```
public int HorizontalScrollWheelValue { get; }
```

Property Value

int

Remarks

The current mouse wheel value is incremented when the wheel is scrolled up and decremented on scroll down. This is a real value showing the current position since the game started, to capture the delta, store the value and comnpare to the previous frame.

LeftButton

Gets state of the left mouse button.

```
public ButtonState LeftButton { get; }
```

Property Value

ButtonState

MiddleButton

Gets state of the middle mouse button.

```
public ButtonState MiddleButton { get; }
```

Property Value

ButtonState

Position

Gets cursor position.

```
public Point Position { get; }
```

Property Value

Point

RightButton

Gets state of the right mouse button.

```
public ButtonState RightButton { get; }
```

Property Value

ButtonState

ScrollWheelValue

Returns cumulative scroll wheel value since the game start.

```
public int ScrollWheelValue { get; }
```

Property Value

int

Remarks

The current mouse wheel value is incremented when the wheel is scrolled up and decremented on scroll down. This is a real value showing the current position since the game started, to capture the delta, store the value and comnpare to the previous frame.

X

Gets horizontal position of the cursor in relation to the window.

```
public int X { get; }
```

Property Value

int

XButton1

Gets state of the XButton1.

```
public ButtonState XButton1 { get; }
```

Property Value

ButtonState

XButton2

Gets state of the XButton2.

```
public ButtonState XButton2 { get; }
```

Property Value

ButtonState

Y

Gets vertical position of the cursor in relation to the window.

```
public int Y { get; }
```

Property Value

int

Methods

Equals(object)

Compares whether current instance is equal to specified object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The MouseState to compare.

Returns

bool

GetHashCode()

Gets the hash code for MouseState instance.

```
public override int GetHashCode()
```

Returns

int

Hash code of the object.

ToString()

Returns a string describing the mouse state.

```
public override string ToString()
```

Returns

string

Operators

operator ==(MouseState, MouseState)

Compares whether two MouseState instances are equal.

```
public static bool operator ==(MouseState left, MouseState right)
```

Parameters

`left` MouseState

MouseState instance on the left of the equal sign.

`right` MouseState

MouseState instance on the right of the equal sign.

Returns

bool

true if the instances are equal; false otherwise.

operator !=(MouseState, MouseState)

Compares whether two MouseState instances are not equal.

```
public static bool operator !=(MouseState left, MouseState right)
```

Parameters

`left` MouseState

MouseState instance on the left of the equal sign.

`right` MouseState

MouseState instance on the right of the equal sign.

Returns

bool

true if the objects are not equal; false otherwise.

Namespace Microsoft.Xna.Framework.Input.Touch

Classes

TouchPanel

Allows retrieval of information from Touch Panel device.

TouchPanelState

Represents specific information about the state of the touch panel, including the current state of touch locations and gestures.

Structs

GestureSample

Represents data from a multi-touch gesture over a span of time.

TouchCollection

Provides state information for a touch screen enabled device.

TouchCollection.Enumerator

Provides the ability to iterate through the TouchLocations in an TouchCollection.

TouchLocation

Provides methods and properties for interacting with a touch location on a touch screen device.

TouchPanelCapabilities

Allows retrieval of capabilities information from touch panel device.

Enums

GestureType

Enumeration of values that represent different gestures that can be processed by ReadGesture().

TouchLocationState

Holds the possible state information for a touch location..

Struct GestureSample

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Represents data from a multi-touch gesture over a span of time.

```
public struct GestureSample
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

GestureSample(GestureType, TimeSpan, Vector2, Vector2, Vector2, Vector2)

Initializes a new GestureSample.

```
public GestureSample(GestureType gestureType, TimeSpan timestamp, Vector2 position,
Vector2 position2, Vector2 delta, Vector2 delta2)
```

Parameters

`gestureType` GestureType

GestureType

`timestamp` TimeSpan

`position` Vector2

`position2` Vector2

`delta` Vector2

`delta2` Vector2

Properties

Delta

Gets the delta information for the first touch-point in the gesture sample.

```
public Vector2 Delta { get; }
```

Property Value

Vector2

Delta2

Gets the delta information for the second touch-point in the gesture sample.

```
public Vector2 Delta2 { get; }
```

Property Value

Vector2

GestureType

Gets the type of the gesture.

```
public GestureType GestureType { get; }
```

Property Value

GestureType

Position

Gets the position of the first touch-point in the gesture sample.

```
public Vector2 Position { get; }
```

Property Value

Vector2

Position2

Gets the position of the second touch-point in the gesture sample.

```
public Vector2 Position2 { get; }
```

Property Value

Vector2

Timestamp

Gets the starting time for this multi-touch gesture sample.

```
public TimeSpan Timestamp { get; }
```

Property Value

TimeSpan

Enum GestureType

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Enumeration of values that represent different gestures that can be processed by ReadGesture().

```
[Flags]
public enum GestureType
```

Fields

`DoubleTap = 256`

The user tapped the device twice which is always preceded by a Tap gesture.

`DragComplete = 2`

States completion of a drag gesture(VerticalDrag, HorizontalDrag, or FreeDrag).

`Flick = 4`

States that a touch was combined with a quick swipe.

`FreeDrag = 8`

The user touched a point and then performed a free-form drag.

`Hold = 16`

The user touched a single point for approximately one second.

`HorizontalDrag = 32`

The user touched the screen and performed either left to right or right to left drag gesture.

`None = 0`

No gestures.

`Pinch = 64`

The user either converged or diverged two touch-points on the screen which is like a two-finger drag.

`PinchComplete = 128`

An in-progress pinch operation was completed.

`Tap = 1`

The user touched a single point.

`VerticalDrag = 512`

The user touched the screen and performed either top to bottom or bottom to top drag gesture.

Struct TouchCollection

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Provides state information for a touch screen enabled device.

```
public struct TouchCollection : IList<TouchLocation>, ICollection<TouchLocation>,
IEnumerable<TouchLocation>, IEnumerable
```

Implements

IList<TouchLocation>, ICollection<TouchLocation>, IEnumerable<TouchLocation>, IEnumerable

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

TouchCollection(TouchLocation[])

Initializes a new instance of the TouchCollection with a pre-determined set of touch locations.

```
public TouchCollection(TouchLocation[] touches)
```

Parameters

`touches` TouchLocation[]

Array of TouchLocation items to initialize with.

Properties

Count

Returns the number of TouchLocation items that exist in the collection.

```
public int Count { get; }
```

Property Value

int

IsConnected

States if a touch screen is available.

```
public bool IsConnected { get; }
```

Property Value

bool

IsReadOnly

States if touch collection is read only.

```
public bool IsReadOnly { get; }
```

Property Value

bool

this[int]

Gets or sets the item at the specified index of the collection.

```
public TouchLocation this[int index] { get; set; }
```

Parameters

`index` int

Position of the item.

Property Value

TouchLocation

TouchLocation

Methods

Add(TouchLocation)

Adds a TouchLocation to the collection.

```
public void Add(TouchLocation item)
```

Parameters

`item` TouchLocation

The TouchLocation item to be added.

Clear()

Clears all the items in collection.

```
public void Clear()
```

Contains(TouchLocation)

Returns true if specified TouchLocation item exists in the collection, false otherwise./>

```
public bool Contains(TouchLocation item)
```

Parameters

`item` TouchLocation

The TouchLocation item to query for.

Returns

bool

Returns true if queried item is found, false otherwise.

CopyTo(TouchLocation[], int)

Copies the TouchLocationcollection to specified array starting from the given index.

```
public void CopyTo(TouchLocation[] array, int arrayIndex)
```

Parameters

`array` TouchLocation[]

The array to copy TouchLocation items.

`arrayIndex` int

The starting index of the copy operation.

FindById(int, out TouchLocation)

Returns TouchLocation specified by ID.

```
public bool FindById(int id, out TouchLocation touchLocation)
```

Parameters

`id` int

`touchLocation` TouchLocation

Returns

bool

GetEnumerator()

Returns an enumerator for the TouchCollection.

```
public TouchCollection.Enumerator GetEnumerator()
```

Returns

TouchCollection.Enumerator

Enumerable list of TouchLocation objects.

IndexOf(TouchLocation)

Returns the index of the first occurrence of specified TouchLocation item in the collection.

```
public int IndexOf(TouchLocation item)
```

Parameters

`item` TouchLocation

TouchLocation to query.

Returns

int

Insert(int, TouchLocation)

Inserts a TouchLocation item into the indicated position.

```
public void Insert(int index, TouchLocation item)
```

Parameters

`index` int

The position to insert into.

`item` TouchLocation

The TouchLocation item to insert.

Remove(TouchLocation)

Removes the specified TouchLocation item from the collection.

```
public bool Remove(TouchLocation item)
```

Parameters

`item` TouchLocation

The TouchLocation item to remove.

Returns

bool

RemoveAt(int)

Removes the TouchLocation item at specified index.

```
public void RemoveAt(int index)
```

Parameters

`index` int

Index of the item that will be removed from collection.

Struct TouchCollection.Enumerator

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Provides the ability to iterate through the TouchLocations in an TouchCollection.

```
public struct TouchCollection.Enumerator : IEnumerator<TouchLocation>,
IEnumerator, IDisposable
```

Implements

IEnumerator<TouchLocation>, IEnumerator, IDisposable

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Current

Gets the current element in the TouchCollection.

```
public TouchLocation Current { get; }
```

Property Value

TouchLocation

Methods

Dispose()

Immediately releases the unmanaged resources used by this object.

```
public void Dispose()
```

MoveNext()

Advances the enumerator to the next element of the TouchCollection.

```
public bool MoveNext()
```

Returns

bool

Reset()

Sets the enumerator to its initial position, which is before the first element in the collection.

```
public void Reset()
```

Exceptions

InvalidOperationException

The collection was modified after the enumerator was created.

NotSupportedException

The enumerator does not support being reset.

Struct TouchLocation

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Provides methods and properties for interacting with a touch location on a touch screen device.

```
public struct TouchLocation : IEquatable<TouchLocation>
```

Implements

IEquatable<TouchLocation>

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

TouchLocation(int, TouchLocationState, Vector2)

Initializes a new instance of the TouchLocation structure.

```
public TouchLocation(int id, TouchLocationState state, Vector2 position)
```

Parameters

`id` int

ID of the touch location.

`state` TouchLocationState

Current state of the touch location.

`position` Vector2

Position of the touch location.

TouchLocation(int, TouchLocationState, Vector2, TouchLocationState, Vector2)

Initializes a new instance of the TouchLocation structure.

```
public TouchLocation(int id, TouchLocationState state, Vector2 position,
TouchLocationState previousState, Vector2 previousPosition)
```

Parameters

`id` int

ID of the touch location.

`state` TouchLocationState

Current state of the touch location.

`position` Vector2

Position of the touch location.

`previousState` TouchLocationState

Previous state of this touch location.

`previousPosition` Vector2

Previous position of this touch location.

Properties

Id

Gets the ID of the touch location.

```
public int Id { get; }
```

Property Value

int

Position

Gets the position of the touch location.

```
public Vector2 Position { get; }
```

Property Value

Vector2

Pressure

Gets the pressure of the touch location.

```
public float Pressure { get; }
```

Property Value

float

Remarks

Only used in Android devices

State

Gets the current state of the touch location.

```
public TouchLocationState State { get; }
```

Property Value

TouchLocationState

Methods

Equals(TouchLocation)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(TouchLocation other)
```

Parameters

`other` TouchLocation

An object to compare with this object.

Returns

bool

true if the current object is equal to the `other` parameter; otherwise, false.

Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current instance.

Returns

bool

true if `obj` and this instance are the same type and represent the same value; otherwise, false.

GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

A 32-bit signed integer that is the hash code for this instance.

IsHighFrequencyEvent()

Returns true if the touch panel is configured to process high frequence touch events.

```
public bool IsHighFrequencyEvent()
```

Returns

bool

ToString()

Gets a string representation of the TouchLocation.

```
public override string ToString()
```

Returns

string

TryGetPreviousLocation(out TouchLocation)

Attempts to get the previous location of this touch location object.

```
public bool TryGetPreviousLocation(out TouchLocation aPreviousLocation)
```

Parameters

`aPreviousLocation` TouchLocation

Previous location data, as a TouchLocation.

Returns

bool

Operators

operator ==(TouchLocation, TouchLocation)

Returns a value that indicates whether the two values are equal.

```
public static bool operator ==(TouchLocation value1, TouchLocation value2)
```

Parameters

`value1` TouchLocation

The value on the left of the equality operator.

`value2` TouchLocation

The value on the right of the equality operator.

Returns

bool

true if the two values are equal; otherwise, false.

operator !=(TouchLocation, TouchLocation)

Returns a value that indicates whether the two values are not equal.

```
public static bool operator !=(TouchLocation value1, TouchLocation value2)
```

Parameters

`value1` TouchLocation

The value on the left of the inequality operator.

`value2` TouchLocation

The value on the right of the inequality operator.

Returns

bool

true if the two values are not equal; otherwise, false.

Enum TouchLocationState

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Holds the possible state information for a touch location..

```
public enum TouchLocationState
```

Fields

`Invalid = 0`

This touch location position is invalid.

`Moved = 1`

This touch location position was updated or pressed at the same position.

`Pressed = 2`

This touch location position is new.

`Released = 3`

This touch location position was released.

Class TouchPanel

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Allows retrieval of information from Touch Panel device.

```
public static class TouchPanel
```

Inheritance

object ← TouchPanel

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

DisplayHeight

Gets or sets the display height of the touch panel.

```
public static int DisplayHeight { get; set; }
```

Property Value

int

DisplayOrientation

Gets or sets the display orientation of the touch panel.

```
public static DisplayOrientation DisplayOrientation { get; set; }
```

Property Value

DisplayOrientation

DisplayWidth

Gets or sets the display width of the touch panel.

```
public static int DisplayWidth { get; set; }
```

Property Value

int

EnableHighFrequencyTouch

Gets or sets if high-frequency events are sent to any listeners (if the underlying OS supports it). By default, it's false and hence no additional CPU usage is incurred. This is currently supported on iOS (including support for Apple Pencil) and Android (tablets/phones).

```
public static bool EnableHighFrequencyTouch { get; set; }
```

Property Value

bool

EnableMouseGestures

Gets or sets a value indicating whether to enable mouse gestures.

```
public static bool EnableMouseGestures { get; set; }
```

Property Value

bool

EnableMouseTouchPoint

Gets or sets a value indicating whether to enable the mouse touch point.

```
public static bool EnableMouseTouchPoint { get; set; }
```

Property Value

bool

EnabledGestures

Gets or sets enabled gestures.

```
public static GestureType EnabledGestures { get; set; }
```

Property Value

GestureType

IsGestureAvailable

Returns true if a touch gesture is available.

```
public static bool IsGestureAvailable { get; }
```

Property Value

bool

WindowHandle

The window handle of the touch panel. Purely for Xna compatibility.

```
public static nint WindowHandle { get; set; }
```

Property Value

nint

Methods

GetCapabilities()

Gets the touch panel capabilities for an available device.

```
public static TouchPanelCapabilities GetCapabilities()
```

Returns

TouchPanelCapabilities

GetState()

Gets the current state of the touch panel.

```
public static TouchCollection GetState()
```

Returns

TouchCollection

TouchCollection

GetState(GameWindow)

Gets the current state of the touch panel.

```
public static TouchPanelState GetState(GameWindow window)
```

Parameters

`window` GameWindow

Game windows to get state from

Returns

TouchPanelState

ReadGesture()

Returns the next available gesture on touch panel device.

```
public static GestureSample ReadGesture()
```

Returns

GestureSample

GestureSample

Struct TouchPanelCapabilities

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Allows retrieval of capabilities information from touch panel device.

```
public struct TouchPanelCapabilities
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

HasPressure

Returns true if a touch device supports pressure.

```
public bool HasPressure { get; }
```

Property Value

bool

IsConnected

Returns true if a device is available for use.

```
public bool IsConnected { get; }
```

Property Value

bool

MaximumTouchCount

Returns the maximum number of touch locations tracked by the touch panel device.

```
public int MaximumTouchCount { get; }
```

Property Value

int

Class TouchPanelState

Namespace: Microsoft.Xna.Framework.Input.Touch

Assembly: MonoGame.Framework.dll

Represents specific information about the state of the touch panel, including the current state of touch locations and gestures.

```
public class TouchPanelState
```

Inheritance

object ← TouchPanelState

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

DisplayHeight

Gets or sets the display height of the touch panel.

```
public int DisplayHeight { get; set; }
```

Property Value

int

DisplayOrientation

Gets or sets the display orientation of the touch panel.

```
public DisplayOrientation DisplayOrientation { get; set; }
```

Property Value

DisplayOrientation

DisplayWidth

Gets or sets the display width of the touch panel.

```
public int DisplayWidth { get; set; }
```

Property Value

int

EnableHighFrequencyTouch

Gets or sets if high-frequency touch event processing is enabled. See EnableHighFrequencyTouch for more details.

```
public bool EnableHighFrequencyTouch { get; set; }
```

Property Value

bool

EnableMouseGestures

Gets or sets whether mouse gestures are enabled.

```
public bool EnableMouseGestures { get; set; }
```

Property Value

bool

EnableMouseTouchPoint

Gets or sets whether mouse touch points are enabled.

```
public bool EnableMouseTouchPoint { get; set; }
```

Property Value

bool

EnabledGestures

Gets or sets enabled gestures.

```
public GestureType EnabledGestures { get; set; }
```

Property Value

GestureType

IsGestureAvailable

Returns true if a touch gesture is available.

```
public bool IsGestureAvailable { get; }
```

Property Value

bool

WindowHandle

The window handle of the touch panel. Purely for Xna compatibility.

```
public nint WindowHandle { get; set; }
```

Property Value

nint

Methods

GetCapabilities()

Returns capabilities of touch panel device.

```
public TouchPanelCapabilities GetCapabilities()
```

Returns

TouchPanelCapabilities

TouchPanelCapabilities

GetState()

Returns the current touch panel state

```
public TouchCollection GetState()
```

Returns

TouchCollection

ReadGesture()

Returns the next available gesture on touch panel device.

```
public GestureSample ReadGesture()
```

Returns

GestureSample

GestureSample

Events

OnTouchEvent

Raised when a new touch event is processed. Clients can use this to obtain per-frame touch events beyond the scope of gestures/touch panel.

```
public event EventHandler<TouchLocation> OnTouchEvent
```

Event Type

EventHandler<TouchLocation>

Namespace Microsoft.Xna.Framework.Media

Classes

Album

Provides access to an album in the media library

AlbumCollection

Represents a collection of albums in the device media library.

Artist

Provides access to artist information in the media library.

Genre

Provides access to genre information in the media library.

MediaLibrary

Provides access to songs, playlists, and pictures in the device's media library.

MediaPlayer

Provides methods and properties to play, pause, resume, and stop songs. MediaPlayer also exposes shuffle, repeat, volume, play position, and visualization capabilities.

MediaQueue

Provides methods and properties to access and control the queue of playing songs.

MediaSource

Provides methods and properties to access the source or sources from which the media will be read.

Playlist

Provides access to a playlist in the media library.

PlaylistCollection

A collection of playlists in the media library.

Song

Provides access to a song in the song library.

SongCollection

A collection of songs in the song library.

Video

Represents a video.

VideoPlayer

Provides methods and properties to play back, pause, resume, and stop Video. VideoPlayer also exposes repeat, volume, and play position information.

Enums

MediaSourceType

Type of the media source.

MediaState

Media playback state

VideoSoundtrackType

Type of sounds in a video

Class Album

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to an album in the media library

```
public sealed class Album : IDisposable
```

Inheritance

object ← Album

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

The **Album** class provides information about an album, including the album's Name, Artist, and Songs.

You can obtain an **Album** object through the this[int] indexer and the Song.Album property.

Properties

Artist

Gets the Artist of the Album.

```
public Artist Artist { get; }
```

Property Value

Artist

Artist of this Album.

Duration

Gets the duration of the Album.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

Genre

Gets the Genre of the Album.

```
public Genre Genre { get; }
```

Property Value

Genre

HasArt

Gets a value indicating whether the Album has associated album art.

```
public bool HasArt { get; }
```

Property Value

bool

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Name

Gets the name of the Album.

```
public string Name { get; }
```

Property Value

string

Songs

Gets a SongCollection that contains the songs on the Album.

```
public SongCollection Songs { get; }
```

Property Value

SongCollection

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetAlbumArt()

Returns the stream that contains the album art image data.

```
public Stream GetAlbumArt()
```

Returns

Stream

GetHashCode()

Gets the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

GetThumbnail()

Returns the stream that contains the album thumbnail image data.

```
public Stream GetThumbnail()
```

Returns

Stream

ToString()

Returns a String representation of this Album.

```
public override string ToString()
```

Returns

string

Class AlbumCollection

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Represents a collection of albums in the device media library.

```
public sealed class AlbumCollection : IDisposable
```

Inheritance

object ← AlbumCollection

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The AlbumCollection class provides access to albums in the device's media library

Use the MediaLibrary.Albums property to obtain a collection of all albums in the media library, the Artist.Albums property to obtain a collection of albums associated with a particular artist, and the Genre.Albums property to obtain a collection of albums associated with a particular genre.

Constructors

AlbumCollection(List<Album>)

Initializes a new instance of the AlbumCollection class, using a specified collection of Album instances.

```
public AlbumCollection(List<Album> albums)
```

Parameters

`albums` List<Album>

The Album collection to initialize this AlbumCollection with.

Fields

Empty

Returns an AlbumCollection with no contents.

```
public static readonly AlbumCollection Empty
```

Field Value

AlbumCollection

Properties

Count

Gets the number of Album objects in the AlbumCollection.

```
public int Count { get; }
```

Property Value

int

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

this[int]

Gets the Album at the specified index in the AlbumCollection.

```
public Album this[int index] { get; }
```

Parameters

`index` int

Index of the Album to get.

Property Value

Album

A new Album representing the album at the specified index in this AlbumCollection

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Class Artist

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to artist information in the media library.

```
public sealed class Artist : IDisposable
```

Inheritance

object ← Artist

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

The Artist class provides information about an artist, including the artist's Name, Albums, and Songs

You can obtain an **Artist** through the Album.Artist and Song.Artist properties.

Constructors

Artist(string)

Creates a new instance of Artist class.

```
public Artist(string artist)
```

Parameters

`artist` string

Name of the artist.

Properties

Albums

Gets the AlbumCollection for the Artist.

```
public AlbumCollection Albums { get; }
```

Property Value

AlbumCollection

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Name

Gets the name of the Artist.

```
public string Name { get; }
```

Property Value

string

Songs

Gets the SongCollection for the Artist.

```
public SongCollection Songs { get; }
```

Property Value

SongCollection

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetHashCode()

Gets the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

ToString()

Returns a String representation of the Artist.

```
public override string ToString()
```

Returns

string

Class Genre

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to genre information in the media library.

```
public sealed class Genre : IDisposable
```

Inheritance

object ← Genre

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Remarks

The Genre class provides information about a genre, including the genre's Name, and the Albums and Songs in that genre that are on the device.

You can obtain a Genre object through the Album.Genre and Song.Genre properties.

Constructors

Genre(string)

Creates a new instance of Genre class.

```
public Genre(string genre)
```

Parameters

`genre` string

Name of the genre.

Properties

Albums

Gets the AlbumCollection for the Genre.

```
public AlbumCollection Albums { get; }
```

Property Value

AlbumCollection

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

Name

Gets the name of the Genre.

```
public string Name { get; }
```

Property Value

string

Songs

Gets the SongCollection for the Genre.

```
public SongCollection Songs { get; }
```

Property Value

SongCollection

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetHashCode()

Gets the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

ToString()

Returns a String representation of the Genre.

```
public override string ToString()
```

Returns

string

Class MediaLibrary

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to songs, playlists, and pictures in the device's media library.

```
public class MediaLibrary : IDisposable
```

Inheritance

object ← MediaLibrary

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

MediaLibrary provides the following properties that return media collections: Albums, Songs. Each property returns a collection object that can be enumerated and indexed. The collection object represents all media of that type in the device's media library.

Constructors

MediaLibrary()

Creates a new instance of MediaLibrary.

```
public MediaLibrary()
```

MediaLibrary(MediaSource)

Creates a new instance of MediaLibrary from a supplie MediaSource.

```
public MediaLibrary(MediaSource mediaSource)
```

Parameters

`mediaSource` MediaSource

Remarks

Current implenetation will always throw NotSupportedException

Properties

Albums

Gets the AlbumCollection that contains all albums in the media library.

```
public AlbumCollection Albums { get; }
```

Property Value

AlbumCollection

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

MediaSource

Gets the MediaSource with which this media library was constructed.

```
public MediaSource MediaSource { get; }
```

Property Value

MediaSource

Songs

Gets the SongCollection that contains all songs in the media library.

```
public SongCollection Songs { get; }
```

Property Value

SongCollection

Methods

Dispose()

Immediately releases the unmanaged resources used by this object.

```
public void Dispose()
```

Load(Action<int>)

Load the contents of MediaLibrary. This blocking call might take up to a few minutes depending on the platform and the size of the user's music library.

```
public void Load(Action<int> progressCallback = null)
```

Parameters

`progressCallback` Action<int>

Callback that reports back the progress of the music library loading in percents (0-100).

Class MediaPlayer

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides methods and properties to play, pause, resume, and stop songs. MediaPlayer also exposes shuffle, repeat, volume, play position, and visualization capabilities.

```
public static class MediaPlayer
```

Inheritance

object ← MediaPlayer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

GameHasControl

Determines whether the game has control of the background music.

```
public static bool GameHasControl { get; }
```

Property Value

bool

Remarks

If a gamer is currently playing their own music as the background to your game, calls to Play(Song), Stop(), Pause(), Resume(), MoveNext(), and MovePrevious() might have no effect, depending on the platform.

IsMuted

Gets or set the muted setting for the media player.

```
public static bool IsMuted { get; set; }
```

Property Value

bool

IsRepeating

Gets or sets the repeat setting for the media player.

```
public static bool IsRepeating { get; set; }
```

Property Value

bool

Remarks

When set to true, the playback queue will begin playing again after all songs in the queue have been played.

IsShuffled

Gets or sets the shuffle setting for the media player.

```
public static bool IsShuffled { get; set; }
```

Property Value

bool

Remarks

When set to true, songs in the playback queue are played in random order rather than from first to last.

IsVisualizationEnabled

Gets or sets the visualization enabled setting for the media player.

```
public static bool IsVisualizationEnabled { get; }
```

Property Value

bool

Remarks

Always returns false

PlayPosition

Gets the play position within the currently playing song.

```
public static TimeSpan PlayPosition { get; }
```

Property Value

TimeSpan

Queue

Gets the media playback queue, MediaQueue.

```
public static MediaQueue Queue { get; }
```

Property Value

MediaQueue

State

Gets the media playback state, MediaState

```
public static MediaState State { get; }
```

Property Value

MediaState

Volume

Gets or sets the media player volume

```
public static float Volume { get; set; }
```

Property Value

float

Media player volume, from 0.0f (silence) to 1.0f (full volume relative to the current device volume)

Remarks

Volume adjustment is based on a decibel, not multiplicative, scale. Setting Volume to 0.0 subtracts 96 dB from the volume. Setting Volume to 1.0 subtracts 0 dB from the volume. Values in between 0.0f and 1.0f subtract dB from the volume proportionally.

Methods

MoveNext()

Stops currently playing song, moves to the next song in the queue of playing songs and plays it.

```
public static void MoveNext()
```

Remarks

If the current song is the last song in the queue, MoveNext() will stay on current song

MovePrevious()

Stops currently playing song, moves to the previous song in the queue of playing songs and plays it.

```
public static void MovePrevious()
```

Remarks

If the current song is the first song in the queue, MovePrevious() will stay on current song

Pause()

Pauses the currently playing song.

```
public static void Pause()
```

Play(Song)

Play clears the current playback queue, and then queues up the specified song for playback. Playback starts immediately at the beginning of the song.

```
public static void Play(Song song)
```

Parameters

`song` Song

Play(Song, TimeSpan?)

Play clears the current playback queue, and then queues up the specified song for playback. Playback starts immediately at the given position of the song.

```
public static void Play(Song song, TimeSpan? startPosition)
```

Parameters

`song` Song

`startPosition` TimeSpan?

Play(SongCollection, int)

Play clears the current playback queue, and then queues up the specified song collection for playback. Playback starts immediately at the beginning of the song, specified by song collection index.

```
public static void Play(SongCollection collection, int index = 0)
```

Parameters

`collection` SongCollection

`index` int

Resume()

Resumes a paused song.

```
public static void Resume()
```

Stop()

Stops playing a song.

```
public static void Stop()
```

Events

ActiveSongChanged

Raised when the active song changes due to active playback or due to explicit calls to the MoveNext() or MovePrevious() methods.

```
public static event EventHandler<EventArgs> ActiveSongChanged
```

Event Type

EventHandler<EventArgs>

MediaStateChanged

Raised when the media player play state changes.

```
public static event EventHandler<EventArgs> MediaStateChanged
```

Event Type

EventHandler<EventArgs>

Class MediaQueue

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides methods and properties to access and control the queue of playing songs.

```
public sealed class MediaQueue
```

Inheritance

object ← MediaQueue

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

MediaQueue is a read-only queue of songs. With MediaQueue, you can control which song is playing in the queue, but you cannot add or remove songs from the queue. Either MediaPlayer.Play(Song) or the songs already queued up when the game starts determine the songs that are in the queue of playing songs.

Constructors

MediaQueue()

Creates a new instance of MediaQueue.

```
public MediaQueue()
```

Properties

ActiveSong

Gets the current Song in the queue of playing songs.

```
public Song ActiveSong { get; }
```

Property Value

Song

ActiveSongIndex

Gets or sets the index of the current (active) song in the queue of playing songs.

```
public int ActiveSongIndex { get; set; }
```

Property Value

int

Remarks

Changing the active song index does not alter the current media state (playing, paused, or stopped).

this[int]

Gets the Song at the specified index in the MediaQueue

```
public Song this[int index] { get; }
```

Parameters

`index` int

Property Value

Song

Class MediaSource

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides methods and properties to access the source or sources from which the media will be read.

```
public sealed class MediaSource
```

Inheritance

object ← MediaSource

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

MediaSource provides access to the source or sources from which the media will be read. A source can be either the local device, or a device connected through Windows Media Connect.

Properties

MediaSourceType

Gets the MediaSourceType of this media source.

```
public MediaSourceType MediaSourceType { get; }
```

Property Value

MediaSourceType

Name

Gets the name of this media source.

```
public string Name { get; }
```

Property Value

string

Methods

GetAvailableMediaSources()

Gets the available media sources with which a media library can be constructed.

```
public static IList<MediaSource> GetAvailableMediaSources()
```

Returns

IList<MediaSource>

This method will always return a single media source: the local device.

Enum MediaSourceType

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Type of the media source.

```
public enum MediaSourceType
```

Fields

`LocalDevice = 0`

A local device.

`WindowsMediaConnect = 4`

A Windows Media Connect device.

Remarks

Indicates the type of the current media source. The type can be either a local device, or a device connected through Windows Media Connect.

Enum MediaState

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Media playback state

```
public enum MediaState
```

Fields

`Paused = 2`

Media playback is paused.

`Playing = 1`

Media is currently playing and not paused.

`Stopped = 0`

Media playback is stopped.

Class Playlist

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to a playlist in the media library.

```
public sealed class Playlist : IDisposable
```

Inheritance

object ← Playlist

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

Obtain Playlist objects through the PlaylistCollection.this indexer.

Properties

Duration

Gets the duration of the Playlist.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

Name

Gets the name of the Playlist.

```
public string Name { get; }
```

Property Value

string

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Class PlaylistCollection

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

A collection of playlists in the media library.

```
public sealed class PlaylistCollection : ICollection<Playlist>,
IEnumerable<Playlist>, IEnumerable, IDisposable
```

Inheritance

object ← PlaylistCollection

Implements

ICollection<Playlist>, IEnumerable<Playlist>, IEnumerable, IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Count

Gets the number of Playlist objects in the PlaylistCollection.

```
public int Count { get; }
```

Property Value

int

IsReadOnly

Gets whether this collection is read-only,

```
public bool IsReadOnly { get; }
```

Property Value

bool

this[int]

Gets the Playlist at the specified index in the PlaylistCollection.

```
public Playlist this[int index] { get; }
```

Parameters

`index` int

Property Value

Playlist

Methods

Add(Playlist)

Adds a Playlist to this PlaylistCollection.

```
public void Add(Playlist item)
```

Parameters

`item` Playlist

Clear()

Removes all items from this PlaylistCollection.

```
public void Clear()
```

Clone()

Creates a new object that is a copy of the current instance.

```
public PlaylistCollection Clone()
```

Returns

PlaylistCollection

A new object that is a copy of this instance.

Contains(Playlist)

Determines whether the collection contains specified Playlist

```
public bool Contains(Playlist item)
```

Parameters

`item` Playlist

Returns

bool

CopyTo(Playlist[], int)

Copies the elements of the collection to an Array, starting at a particular Array index.

```
public void CopyTo(Playlist[] array, int arrayIndex)
```

Parameters

`array` Playlist[]

The one-dimensional Array that is the destination of the elements copied from collection. The Array must have zero-based indexing.

`arrayIndex` int

The zero-based index in array at which copying begins.

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<Playlist> GetEnumerator()
```

Returns

IEnumerator<Playlist>

An enumerator that can be used to iterate through the collection.

IndexOf(Playlist)

Searches for the specified Playlist and returns the zero-based index of the first occurence within the entire PlaylistCollection.

```
public int IndexOf(Playlist item)
```

Parameters

`item` Playlist

The Playlist to locate

Returns

int

The zero-based index of the first occurence of `item` within the entire PlaylistCollection, if found. otherwise, -1.

Remove(Playlist)

Removes the first occurrence of a Playlist from the PlaylistCollection.

```
public bool Remove(Playlist item)
```

Parameters

`item` Playlist

The object to remove from the PlaylistCollection.

Returns

bool

true if item was successfully removed from the PlaylistCollection; otherwise, false. This method also returns false if item is not found in the original PlaylistCollection.

Class Song

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides access to a song in the song library.

```
public sealed class Song : IEquatable<Song>, IDisposable
```

Inheritance

object ← Song

Implements

IEquatable<Song>, IDisposable

Inherited Members

object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Album

Gets the Album on which the Song appears.

```
public Album Album { get; }
```

Property Value

Album

Artist

Gets the Artist of the Song.

```
public Artist Artist { get; }
```

Property Value

Artist

Duration

Gets the duration of the Song.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

Genre

Gets the Genre of the Song.

```
public Genre Genre { get; }
```

Property Value

Genre

IsDisposed

Gets a value indicating whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsProtected

Gets a value that indicates whether the song is DRM protected content.

```
public bool IsProtected { get; }
```

Property Value

bool

IsRated

Gets a value that indicates whether the song has been rated by the user.

```
public bool IsRated { get; }
```

Property Value

bool

Name

Gets the name of the Song.

```
public string Name { get; }
```

Property Value

string

PlayCount

Gets the song play count.

```
public int PlayCount { get; }
```

Property Value

int

Position

Gets the position of the Song.

```
public TimeSpan Position { get; }
```

Property Value

TimeSpan

Rating

Gets the user's rating for the Song.

```
public int Rating { get; }
```

Property Value

int

User's rating for this Song, or 0 if the song is unrated. Ratings range from 1 (dislike the most) to 10 (like the most).

TrackNumber

Gets the track number of the song on the song's Album.

```
public int TrackNumber { get; }
```

Property Value

int

Track number of this Song on the song's Album.

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Equals(Song)

Determines whether two instances of Song are equal.

```
public bool Equals(Song song)
```

Parameters

`song` Song

The Song to compare with the current object

Returns

bool

true if the specified Song is equal to the current object; otherwise, false

Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

`obj` object

The object to compare with the current object.

Returns

bool

true if the specified object is equal to the current object; otherwise, false.

~Song()

```
protected ~Song()
```

FromUri(string, Uri)

Returns a song that can be played via MediaPlayer.

```
public static Song FromUri(string name, Uri uri)
```

Parameters

`name` string

The name for the song. See Song.Name.

`uri` Uri

The path to the song file.

Returns

Song

GetHashCode()

Gets the hash code for this instance.

```
public override int GetHashCode()
```

Returns

int

Operators

operator ==(Song, Song)

Determines whether the specified Song instances are equal.

```
public static bool operator ==(Song song1, Song song2)
```

Parameters

`song1` Song

`song2` Song

Returns

bool

operator !=(Song, Song)

Determines whether the specified Song instances are not equal.

```
public static bool operator !=(Song song1, Song song2)
```

Parameters

`song1` Song

song2 Song

Returns

bool

Class SongCollection

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

A collection of songs in the song library.

```
public class SongCollection : ICollection<Song>, IEnumerable<Song>,
IEnumerable, IDisposable
```

Inheritance

object ← SongCollection

Implements

ICollection<Song>, IEnumerable<Song>, IEnumerable, IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Remarks

The SongCollection class provides access to songs in the device's song library.

Use the Songs property to obtain the following collections: All songs in the media library; Songs on a particular album; Songs associated with a particular artist; Songs associated with a particular genre;

Fields

Empty

Returns a SongCollection with no contents.

```
public static readonly SongCollection Empty
```

Field Value

SongCollection

Properties

Count

Gets the number of Song objects in the SongCollection.

```
public int Count { get; }
```

Property Value

int

IsReadOnly

Gets whether this collection is read-only,

```
public bool IsReadOnly { get; }
```

Property Value

bool

this[int]

Gets the Song at the specified index in the SongCollection.

```
public Song this[int index] { get; }
```

Parameters

`index` int

Property Value

Song

Methods

Add(Song)

Adds a Song to this SongCollection.

```
public void Add(Song item)
```

Parameters

`item` Song

Clear()

Removes all items from this SongCollection.

```
public void Clear()
```

Clone()

Creates a new object that is a copy of the current instance.

```
public SongCollection Clone()
```

Returns

SongCollection

A new object that is a copy of this instance.

Contains(Song)

Determines whether a Song is in the SongCollection

```
public bool Contains(Song item)
```

Parameters

`item` Song

Returns

bool

true if `item` is found in the SongCollection; otherwise, false.

CopyTo(Song[], int)

Copies the elements of the collection to an Array, starting at a particular Array index.

```
public void CopyTo(Song[] array, int arrayIndex)
```

Parameters

`array` Song[]

The one-dimensional Array that is the destination of the elements copied from collection. The Array must have zero-based indexing.

`arrayIndex` int

The zero-based index in array at which copying begins.

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<Song> GetEnumerator()
```

Returns

IEnumerator<Song>

An enumerator that can be used to iterate through the collection.

IndexOf(Song)

Searches for the specified Song and returns the zero-based index of the first occurence within the entire SongCollection.

```
public int IndexOf(Song item)
```

Parameters

`item` Song

The Song to locate

Returns

int

The zero-based index of the first occurence of `item` within the entire SongCollection, if found. otherwise, -1.

Remove(Song)

Removes the first occurrence of a Song from the SongCollection.

```
public bool Remove(Song item)
```

Parameters

`item` Song

The object to remove from the SongCollection.

Returns

bool

true if item was successfully removed from the SongCollection; otherwise, false. This method also returns false if item is not found in the original SongCollection.

Class Video

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Represents a video.

```
public sealed class Video : IDisposable
```

Inheritance

object ← Video

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Duration

Gets the duration of the Video.

```
public TimeSpan Duration { get; }
```

Property Value

TimeSpan

FileName

I actually think this is a file PATH...

```
public string FileName { get; }
```

Property Value

string

FramesPerSecond

Gets the frame rate of this video.

```
public float FramesPerSecond { get; }
```

Property Value

float

Height

Gets the height of this video, in pixels.

```
public int Height { get; }
```

Property Value

int

VideoSoundtrackType

Gets the VideoSoundtrackType for this video.

```
public VideoSoundtrackType VideoSoundtrackType { get; }
```

Property Value

VideoSoundtrackType

Width

Gets the width of this video, in pixels.

```
public int Width { get; }
```

Property Value

int

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

~Video()

```
protected ~Video()
```

Class VideoPlayer

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Provides methods and properties to play back, pause, resume, and stop Video. VideoPlayer also exposes repeat, volume, and play position information.

```
public sealed class VideoPlayer : IDisposable
```

Inheritance

object ← VideoPlayer

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

VideoPlayer()

Creates a new instance of VideoPlayer class.

```
public VideoPlayer()
```

Properties

IsDisposed

Gets a value that indicates whether the object is disposed.

```
public bool IsDisposed { get; }
```

Property Value

bool

IsLooped

Gets a value that indicates whether the player is playing video in a loop.

```
public bool IsLooped { get; set; }
```

Property Value

bool

IsMuted

Gets or sets the muted setting for the video player.

```
public bool IsMuted { get; set; }
```

Property Value

bool

PlayPosition

Gets the play position within the currently playing video.

```
public TimeSpan PlayPosition { get; }
```

Property Value

TimeSpan

State

Gets the media playback state, MediaState.

```
public MediaState State { get; }
```

Property Value

MediaState

Video

Gets the Video that is currently playing.

```
public Video Video { get; }
```

Property Value

Video

Volume

Video player volume, from 0.0f (silence) to 1.0f (full volume relative to the current device volume).

```
public float Volume { get; set; }
```

Property Value

float

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

GetTexture()

Retrieves a Texture2D containing the current frame of video being played.

```
public Texture2D GetTexture()
```

Returns

Texture2D

The current frame of video.

Exceptions

InvalidOperationException

Thrown if no video is set on the player

InvalidOperationException

Thrown if the platform was unable to get a texture in a reasonable amount of time. Often the platform specific media code is running in a different thread or process. Note: This may be a change from XNA behaviour

Pause()

Pauses the currently playing video.

```
public void Pause()
```

Play(Video)

Plays a Video.

```
public void Play(Video video)
```

Parameters

`video` Video

Video to play.

Resume()

Resumes a paused video.

```
public void Resume()
```

Stop()

Stops playing a video.

```
public void Stop()
```

Enum VideoSoundtrackType

Namespace: Microsoft.Xna.Framework.Media

Assembly: MonoGame.Framework.dll

Type of sounds in a video

```
public enum VideoSoundtrackType
```

Fields

`Dialog = 1`

This video contains only dialog.

`Music = 0`

This video contains only music.

`MusicAndDialog = 2`

This video contains music and dialog.

Namespace MonoGame.Effect

Classes

CommandLineParser

CommandLineParser.NameAttribute

CommandLineParser.ProfileNameAttribute

CommandLineParser.RequiredAttribute

Options

PassInfo

Preprocessor

SamplerStateInfo

ShaderCompilerException

ShaderInfo

ShaderProfile

ShaderResult

TechniqueInfo

Interfaces

IEffectCompilerOutput

Enums

TextureFilterType

Class CommandLineParser

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class CommandLineParser
```

Inheritance

object ← CommandLineParser

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CommandLineParser(object)

```
public CommandLineParser(object optionsObject)
```

Parameters

`optionsObject` object

Properties

Title

```
public string Title { get; set; }
```

Property Value

string

Methods

ParseCommandLine(string[])

```
public bool ParseCommandLine(string[] args)
```

Parameters

`args` string[]

Returns

bool

Class CommandLineParser.NameAttribute

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[AttributeUsage(AttributeTargets.Field)]
public class CommandLineParser.NameAttribute : Attribute
```

Inheritance

object ← Attribute ← CommandLineParser.NameAttribute

Derived

CommandLineParser.ProfileNameAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,

Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NameAttribute(string)

```
public NameAttribute(string name)
```

Parameters

`name` string

NameAttribute(string, string)

```
public NameAttribute(string name, string description)
```

Parameters

`name` string

`description` string

Properties

Description

```
public string Description { get; protected set; }
```

Property Value

string

Name

```
public string Name { get; }
```

Property Value

string

Class CommandLineParser.ProfileNameAttribute

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[AttributeUsage(AttributeTargets.Field)]
public sealed class CommandLineParser.ProfileNameAttribute :
CommandLineParser.NameAttribute
```

Inheritance

object ← Attribute ← CommandLineParser.NameAttribute ← CommandLineParser.ProfileNameAttribute

Inherited Members

CommandLineParser.NameAttribute.Name , CommandLineParser.NameAttribute.Description , Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,

Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ProfileNameAttribute()

```
public ProfileNameAttribute()
```

Class CommandLineParser.RequiredAttribute

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[AttributeUsage(AttributeTargets.Field)]
public sealed class CommandLineParser.RequiredAttribute : Attribute
```

Inheritance

object ← Attribute ← CommandLineParser.RequiredAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,
object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object) ,
object.ToString()

Interface IEffectCompilerOutput

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public interface IEffectCompilerOutput
```

Methods

WriteError(string, int, int, string)

```
void WriteError(string file, int line, int column, string message)
```

Parameters

`file` string

`line` int

`column` int

`message` string

WriteWarning(string, int, int, string)

```
void WriteWarning(string file, int line, int column, string message)
```

Parameters

`file` string

`line` int

`column` int

`message` string

Class Options

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class Options
```

Inheritance

object ← Options

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

Debug

```
[CommandLineParser.Name("Debug", "\t\t - Include extra debug information in the
compiled effect.")]
public bool Debug
```

Field Value

bool

Defines

```
[CommandLineParser.Name("Defines", "\t - Semicolon-delimited define assignments")]
public string Defines
```

Field Value

string

OutputFile

```
[CommandLineParser.Name("OutputFile", "\t - The output file path.  Use a .h
extension to generate a C header file.")]
public string OutputFile
```

Field Value

string

Profile

```
[CommandLineParser.ProfileName]
public ShaderProfile Profile
```

Field Value

ShaderProfile

SourceFile

```
public string SourceFile
```

Field Value

string

Class PassInfo

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class PassInfo
```

Inheritance

object ← PassInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

blendState

```
public BlendState blendState
```

Field Value

BlendState

depthStencilState

```
public DepthStencilState depthStencilState
```

Field Value

DepthStencilState

name

```
public string name
```

Field Value

string

psFunction

```
public string psFunction
```

Field Value

string

psModel

```
public string psModel
```

Field Value

string

rasterizerState

```
public RasterizerState rasterizerState
```

Field Value

RasterizerState

vsFunction

```
public string vsFunction
```

Field Value

string

vsModel

```
public string vsModel
```

Field Value

string

Properties

AlphaBlendEnable

```
public bool AlphaBlendEnable { set; }
```

Property Value

bool

BlendOp

```
public BlendFunction BlendOp { set; }
```

Property Value

BlendFunction

ColorWriteEnable

```
public ColorWriteChannels ColorWriteEnable { set; }
```

Property Value

ColorWriteChannels

CullMode

```
public CullMode CullMode { set; }
```

Property Value

CullMode

DepthBias

```
public float DepthBias { set; }
```

Property Value

float

DepthBufferFunction

```
public CompareFunction DepthBufferFunction { set; }
```

Property Value

CompareFunction

DestBlend

```
public Blend DestBlend { set; }
```

Property Value

Blend

FillMode

```
public FillMode FillMode { set; }
```

Property Value

FillMode

MultiSampleAntiAlias

```
public bool MultiSampleAntiAlias { set; }
```

Property Value

bool

ScissorTestEnable

```
public bool ScissorTestEnable { set; }
```

Property Value

bool

SlopeScaleDepthBias

```
public float SlopeScaleDepthBias { set; }
```

Property Value

float

SrcBlend

```
public Blend SrcBlend { set; }
```

Property Value

Blend

StencilEnable

```
public bool StencilEnable { set; }
```

Property Value

bool

StencilFail

```
public StencilOperation StencilFail { set; }
```

Property Value

StencilOperation

StencilFunc

```
public CompareFunction StencilFunc { set; }
```

Property Value

CompareFunction

StencilMask

```
public int StencilMask { set; }
```

Property Value

int

StencilPass

```
public StencilOperation StencilPass { set; }
```

Property Value

StencilOperation

StencilRef

```
public int StencilRef { set; }
```

Property Value

int

StencilWriteMask

```
public int StencilWriteMask { set; }
```

Property Value

int

StencilZFail

```
public StencilOperation StencilZFail { set; }
```

Property Value

StencilOperation

ZEnable

```
public bool ZEnable { set; }
```

Property Value

bool

ZWriteEnable

```
public bool ZWriteEnable { set; }
```

Property Value

bool

Class Preprocessor

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public static class Preprocessor
```

Inheritance

object ← Preprocessor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Preprocess(string, string, IDictionary<string, string>, List<string>, IEffectCompilerOutput)

```
public static string Preprocess(string effectCode, string filePath,
IDictionary<string, string> defines, List<string> dependencies,
IEffectCompilerOutput output)
```

Parameters

`effectCode` string

`filePath` string

`defines` IDictionary<string, string>

`dependencies` List<string>

`output` IEffectCompilerOutput

Returns

string

Class SamplerStateInfo

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class SamplerStateInfo
```

Inheritance

object ← SamplerStateInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SamplerStateInfo()

```
public SamplerStateInfo()
```

Properties

AddressU

```
public TextureAddressMode AddressU { set; }
```

Property Value

TextureAddressMode

AddressV

```
public TextureAddressMode AddressV { set; }
```

Property Value

TextureAddressMode

AddressW

```
public TextureAddressMode AddressW { set; }
```

Property Value

TextureAddressMode

BorderColor

```
public Color BorderColor { set; }
```

Property Value

Color

Filter

```
public TextureFilterType Filter { set; }
```

Property Value

TextureFilterType

MagFilter

```
public TextureFilterType MagFilter { set; }
```

Property Value

TextureFilterType

MaxAnisotropy

```
public int MaxAnisotropy { set; }
```

Property Value

int

MaxMipLevel

```
public int MaxMipLevel { set; }
```

Property Value

int

MinFilter

```
public TextureFilterType MinFilter { set; }
```

Property Value

TextureFilterType

MipFilter

```
public TextureFilterType MipFilter { set; }
```

Property Value

TextureFilterType

MipMapLevelOfDetailBias

```
public float MipMapLevelOfDetailBias { set; }
```

Property Value

float

Name

```
public string Name { get; set; }
```

Property Value

string

State

```
public SamplerState State { get; }
```

Property Value

SamplerState

TextureName

```
public string TextureName { get; set; }
```

Property Value

string

Class ShaderCompilerException

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class ShaderCompilerException : Exception, ISerializable
```

Inheritance

object ← Exception ← ShaderCompilerException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

ShaderCompilerException()

```
public ShaderCompilerException()
```

Class ShaderInfo

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class ShaderInfo
```

Inheritance

object ← ShaderInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

SamplerStates

```
public Dictionary<string, SamplerStateInfo> SamplerStates
```

Field Value

Dictionary<string, SamplerStateInfo>

Techniques

```
public List<TechniqueInfo> Techniques
```

Field Value

List<TechniqueInfo>

Class ShaderProfile

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
[TypeConverter(typeof(ShaderProfile.StringConverter))]
public abstract class ShaderProfile
```

Inheritance

object ← ShaderProfile

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ShaderProfile(string, byte)

```
protected ShaderProfile(string name, byte formatId)
```

Parameters

`name` string

`formatId` byte

Fields

DirectX_11

```
public static readonly ShaderProfile DirectX_11
```

Field Value

ShaderProfile

DirectX_12

```
public static readonly ShaderProfile DirectX_12
```

Field Value

ShaderProfile

OpenGL

```
public static readonly ShaderProfile OpenGL
```

Field Value

ShaderProfile

Vulkan

```
public static readonly ShaderProfile Vulkan
```

Field Value

ShaderProfile

Properties

All

Returns all the loaded shader profiles.

```
public static IEnumerable<ShaderProfile> All { get; }
```

Property Value

IEnumerable<ShaderProfile>

FormatId

Returns the format identifier used in the MGFX file format.

```
public byte FormatId { get; }
```

Property Value

byte

Name

Returns the name of the shader profile.

```
public string Name { get; }
```

Property Value

string

Methods

FromName(string)

Returns the profile by name or null if no match is found.

```
public static ShaderProfile FromName(string name)
```

Parameters

`name` string

Returns

ShaderProfile

GetProfileForPlatform(TargetPlatform)

```
public static ShaderProfile GetProfileForPlatform(TargetPlatform platform)
```

Parameters

`platform` TargetPlatform

Returns

ShaderProfile

ParseShaderModel(string, Regex, out int, out int)

```
protected static void ParseShaderModel(string text, Regex regex, out int major, out
int minor)
```

Parameters

`text` string

`regex` Regex

`major` int

`minor` int

Class ShaderResult

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class ShaderResult
```

Inheritance

object ← ShaderResult

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

AdditionalOutputFiles

```
public List<string> AdditionalOutputFiles { get; }
```

Property Value

List<string>

Debug

```
public bool Debug { get; }
```

Property Value

bool

Dependencies

```
public List<string> Dependencies { get; }
```

Property Value

List<string>

FileContent

```
public string FileContent { get; }
```

Property Value

string

FilePath

```
public string FilePath { get; }
```

Property Value

string

OutputFilePath

```
public string OutputFilePath { get; }
```

Property Value

string

Profile

```
public ShaderProfile Profile { get; }
```

Property Value

ShaderProfile

RelativeFilePath

```
public string RelativeFilePath { get; set; }
```

Property Value

string

ShaderInfo

```
public ShaderInfo ShaderInfo { get; }
```

Property Value

ShaderInfo

Methods

FromFile(string, Options, IEffectCompilerOutput)

```
public static ShaderResult FromFile(string path, Options options,
IEffectCompilerOutput output)
```

Parameters

`path` string

`options` Options

`output` IEffectCompilerOutput

Returns

ShaderResult

FromString(string, string, Options, IEffectCompilerOutput)

```
public static ShaderResult FromString(string effectSource, string filePath, Options
options, IEffectCompilerOutput output)
```

Parameters

`effectSource` string

`filePath` string

`options` Options

`output` IEffectCompilerOutput

Returns

ShaderResult

Class TechniqueInfo

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class TechniqueInfo
```

Inheritance

object ← TechniqueInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

Passes

```
public List<PassInfo> Passes
```

Field Value

List<PassInfo>

length

```
public int length
```

Field Value

int

name

```
public string name
```

Field Value

string

startPos

```
public int startPos
```

Field Value

int

Enum TextureFilterType

Namespace: MonoGame.Effect

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public enum TextureFilterType
```

Fields

`Anisotropic = 3`

`Linear = 2`

`None = 0`

`Point = 1`

Namespace MonoGame.Framework.Content

Interfaces

IContentProvider

A content provider used for loading of remote content by TitleContainer.

Interface IContentProvider

Namespace: MonoGame.Framework.Content

Assembly: MonoGame.Framework.dll

A content provider used for loading of remote content by TitleContainer.

```
public interface IContentProvider
```

Methods

FetchContent(string)

Sends a request for the content to be acquired from the remote content server.

```
Task<bool> FetchContent(string relativePath)
```

Parameters

`relativePath` string

Relative path to the content.

Returns

Task<bool>

`true` if no unexpected error has occured, `false` otherwise.

OpenReadStream(string)

Opens the file stream that was acquired by FetchContent(string) if its available.

```
Stream OpenReadStream(string relativePath)
```

Parameters

`relativePath` string

Relative path to the content.

Returns

Stream

A valid Stream if the content was found, null otherwise.

Namespace MonoGame.Framework.Content.Pipeline.Builder

Classes

ContentBuilder

This class is the entry point for the content builder system.

ContentBuilderParams

A list of arguments used by ContentBuilder.

Use Parse(params string[]) to acquire the arguments from the passed cli args.

ContentCollection

Collection of rules on how the content should be handled.

The way this system works is by adding each call you make with include/exclude to a list of rules, then during the GetContentInfo(string, out List<ContentInfo>) it will check if the filepath is valid for each one and return the last match it finds.

ContentInfo

Describes how the ContentBuilder should handle the content.

ContentRule

A rule by which ContentCollection includes or excludes the files.

FileHelper

Common file and path helpers.

PathHelper

Methods provided to normalize and manipulate paths.

PipelineBuildEvent

Class to provide methods and properties for handling pipeline build events.

PipelineBuildEvent.Pair

Class that represents a key / value pair.

PipelineBuildLogger

Provides methods for reporting informational messages or warnings from content importers and processors. Do not use this class to report errors. Instead, report errors by throwing a PipelineException or InvalidContentException.

PipelineImporterContext

PipelineManager

Class to provide methods and properties for handling content pipelines.

PipelineProcessorContext

Provides access to custom processor parameters, methods for converting member data, and triggering nested builds.

RegexRule

A regex based rule by which ContentCollection includes or excludes the files.

TypeExtensions

Provides extension methods for common types.

WildcardRule

A wildcard based rule by which ContentCollection includes or excludes the files.

XmlColor

Helper for serializing color types with the XmlSerializer.

Interfaces

IContentCache

An interface for storing information about the compiled content.

IContentCollection

An interface for a collection of data on how the content should be handled.

IContentFileCache

An interface for storing information about the compiled content.

Enums

ContentBuilderMode

Describes the mode in which the content builder will run in.

Class ContentBuilder

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

This class is the entry point for the content builder system.

```
public abstract class ContentBuilder
```

Inheritance

object ← ContentBuilder

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ContentCache

Gets or sets the content cahcing system to be used by the ContentBuilder.

```
public virtual IContentCache ContentCache { get; init; }
```

Property Value

IContentCache

FailedToBuild

Returns the number of content items that failed to build.

```
public uint FailedToBuild { get; }
```

Property Value

uint

Logger

Gets or sets the logger to be used by the ContentBuilder.

```
public ContentBuildLogger Logger { get; set; }
```

Property Value

ContentBuildLogger

ContentBuildLogger by default.

Parameters

Parameters to be used by the ContentBuilder or any of its subsystems.

Can be passed from CLI args, see Run(string[]).

```
public ContentBuilderParams Parameters { get; set; }
```

Property Value

ContentBuilderParams

SucceededToBuild

Returns the number of content items that built successfully.

```
public uint SucceededToBuild { get; }
```

Property Value

uint

Methods

BuildAndLoadContent(string, ContentInfo, string?, ContentProcessorContext?)

Initiates a build of the specified asset and then loads the result into memory.

```
public object? BuildAndLoadContent(string relativeSrcPath, ContentInfo contentInfo,
string? relativeDstPath = null, ContentProcessorContext? parentContext = null)
```

Parameters

`relativeSrcPath` string

A relative path to the source asset.

`contentInfo` ContentInfo

The desired ContentInfo to be used for the content building.

`relativeDstPath` string

The desired relative output path.

`parentContext` ContentProcessorContext

Only set when the method is being called by the ContentProcessorContext to build one of its children.

Returns

object

The built object that the Process(object, ContentProcessorContext) returned.

BuildAndWriteContent(string, ContentInfo, string?, ContentProcessorContext?)

Initiates a build of the specified asset and then writes down the result to disk..

```
public void BuildAndWriteContent(string relativeSrcPath, ContentInfo contentInfo,
string? relativeDstPath = null, ContentProcessorContext? parentContext = null)
```

Parameters

`relativeSrcPath` string

A relative path to the source asset.

`contentInfo` ContentInfo

The desired ContentInfo to be used for the content building.

`relativeDstPath` string

The desired relative output path.

`parentContext` ContentProcessorContext

Only set when the method is being called by the ContentProcessorContext to build one of its children.

GetContentCollection()

Called to build system to gather information about how to handle the content. It gets called only once during initialization.

```
public abstract IContentCollection GetContentCollection()
```

Returns

IContentCollection

An IContentCollection that contains information about the content handling.

Run(ContentBuilderParams)

Runs the ContentBuilder with the specified parameters.

```
public bool Run(ContentBuilderParams parameters)
```

Parameters

`parameters` ContentBuilderParams

A ContentBuilderParams describing both the platform paramteres for the content compilation as well as the configuration of the ContentBuilder itself.

Returns

bool

Run(string[])

A helper method to run the ContentBuilder with the passed ContentBuilderParams from the entry point args.

```
public void Run(string[] args)
```

Parameters

`args` string[]

An array of string to be deserialized into ContentBuilderParams.

Enum ContentBuilderMode

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Describes the mode in which the content builder will run in.

```
public enum ContentBuilderMode
```

Fields

`Builder = 1`

Indicates that the content builder should build the passed content files.

`None = 0`

Indicates that no builder mode has been specified, so the builder will show the help menu.

`Server = 2`

Indicates that the content builder should run in server mode and wait for content load requests.

Class ContentBuilderParams

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

A list of arguments used by ContentBuilder.

Use Parse(params string[]) to acquire the arguments from the passed cli args.

```
public class ContentBuilderParams
```

Inheritance

object ← ContentBuilderParams

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

CompressContent

Gets or sets if ContentBuilder should compress each built content file.

```
public bool CompressContent { get; set; }
```

Property Value

bool

`false` by default.

GraphicsProfile

Gets or sets the desired graphics profile for ContentBuilder to build the content for.

```
public GraphicsProfile GraphicsProfile { get; set; }
```

Property Value

GraphicsProfile

HiDef by default.

IntermediateDirectory

Gets or sets the location for the intermediate files for content build relative to the WorkingDirectory.

```
public string IntermediateDirectory { get; set; }
```

Property Value

string

`obj/Content` by default.

LogLevel

Gets or sets the logging level of information that ContentBuilder will display to console.

```
public LogLevel LogLevel { get; set; }
```

Property Value

LogLevel

Info by default.

Mode

Set the mode in which the content builder is run in. See ContentBuilderMode for available modes.

```
public ContentBuilderMode Mode { get; set; }
```

Property Value

ContentBuilderMode

None by default.

OutputDirectory

Gets or sets the location for the content output relative to the WorkingDirectory.

```
public string OutputDirectory { get; set; }
```

Property Value

string

`bin/Content` by default.

Platform

Gets or sets the desired platform for ContentBuilder to build the content for.

```
public TargetPlatform Platform { get; set; }
```

Property Value

TargetPlatform

DesktopGL by default.

Rebuild

Should the ContentBuilder rebuild all the assets and ignore the content cache.

```
public bool Rebuild { get; set; }
```

Property Value

bool

`false` by default.

RootedIntermediateDirectory

Gets the rooted location of IntermediateDirectory.

```
public string RootedIntermediateDirectory { get; }
```

Property Value

string

RootedOutputDirectory

Gets the rooted location of OutputDirectory.

```
public string RootedOutputDirectory { get; }
```

Property Value

string

RootedSourceDirectory

Gets the rooted location of SourceDirectory.

```
public string RootedSourceDirectory { get; }
```

Property Value

string

Servers

A list of servers to start up when the Mode is set to Server.

```
public List<ContentServer> Servers { get; set; }
```

Property Value

List<ContentServer>

A collection of ContentServer classes found by scaning all referenced assemblies.

SkipClean

Should the ContentBuilder skip cleaning up old content cache data after the build is finished in Builder mode.

```
public bool SkipClean { get; set; }
```

Property Value

bool

`false` by default.

SourceDirectory

Gets or sets the location of the content relative to the WorkingDirectory.

```
public string SourceDirectory { get; set; }
```

Property Value

string

`Content` by default.

WorkingDirectory

Gets or sets the working directory of the ContentBuilder.

```
public string WorkingDirectory { get; set; }
```

Property Value

string

GetCurrentDirectory() by default.

Methods

Parse(params string[])

Parses out the main entry point args into a ContentBuilderParams to be used by ContentBuilder.

```
public static ContentBuilderParams Parse(params string[] args)
```

Parameters

`args` string[]

Arguments passed to the main entry point of the app.

Returns

ContentBuilderParams

ContentBuilderParams containing the parsed arguments, or an empty ContentBuilderParams if no arguments were passed.

Class ContentCollection

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Collection of rules on how the content should be handled.

The way this system works is by adding each call you make with include/exclude to a list of rules, then during the GetContentInfo(string, out List<ContentInfo>) it will check if the filepath is valid for each one and return the last match it finds.

```
public class ContentCollection : IContentCollection
```

Inheritance

object ← ContentCollection

Implements

IContentCollection

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Exclude(string)

Marks the file at excludePath to be excluded from the content handling.

```
public void Exclude(string excludePath)
```

Parameters

`excludePath` string

Relative path to the content file.

Exclude<T>(string)

Marks the files that match the excludePattern to be excluded from the content handling.

```
public void Exclude<T>(string excludePattern) where T : ContentRule, new()
```

Parameters

`excludePattern` string

A pattern to use for exclusion of content files.

Type Parameters

`T`

A ContentRule to be used for matching the includePattern. Some built in options are:

WildcardRule - specifies that input pattern should be a wildcard.

RegexRule - specifies that input pattern should be a regex.

GetContentInfo(string, out List<ContentInfo>)

Gets information about how the ocntent should be handled for the passed relative filepath.

```
public bool GetContentInfo(string filePath, out List<ContentInfo> contentInfos)
```

Parameters

`filePath` string

Relative path to the content file.

`contentInfos` List<ContentInfo>

A list of ContentInfo describing the desired content handling.

Returns

bool

`true` if the content should be handled, `false` otherwise.

Include(string, IContentImporter?, IContentProcessor?)

Marks the file at inputPath to be built with specified settings.

```
public void Include(string inputPath, IContentImporter? contentImporter = null,
IContentProcessor? contentProcessor = null)
```

Parameters

`inputPath` string

Relative path to the content file.

`contentImporter` IContentImporter

An IContentImporter to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate importer.

`contentProcessor` IContentProcessor

An IContentProcessor to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate processor.

Include(string, string, IContentImporter?, IContentProcessor?)

Marks the file at inputPath to be built with specified settings.

```
public void Include(string inputPath, string outputPath, IContentImporter?
contentImporter = null, IContentProcessor? contentProcessor = null)
```

Parameters

`inputPath` string

Relative path to the content file.

`outputPath` string

Relative path for the output content, if its not specified, the inputPath will be used with .xnb extension.

`contentImporter` IContentImporter

An IContentImporter to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate importer.

`contentProcessor` IContentProcessor

An IContentProcessor to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate processor.

IncludeCopy(string, string?)

Marks the file at inputPath to be copied to the output directory.

```
public void IncludeCopy(string inputPath, string? outputPath)
```

Parameters

`inputPath` string

Relative path to the content file.

`outputPath` string

Relative path for the copy, if its not specified, the inputPath will be used.

IncludeCopy<T>(string, Func<string, string>?)

Marks the files that match the includePattern to be copied to the output directory.

```
public void IncludeCopy<T>(string includePattern, Func<string, string>? outputPath =
null) where T : ContentRule, new()
```

Parameters

`includePattern` string

A pattern to use for building of content files.

`outputPath` Func<string, string>

Relative path for the output content, if its not specified, the relative filePath will be used with .xnb extension.

Type Parameters

`T`

A ContentRule to be used for matching the includePattern. Some built in options are:

WildcardRule - specifies that input pattern should be a wildcard.

RegexRule - specifies that input pattern should be a regex.

Include<T>(string, IContentImporter?, IContentProcessor?)

Marks the files that match the includePattern to be built with the specified settings.

```
public void Include<T>(string includePattern, IContentImporter? contentImporter =
null, IContentProcessor? contentProcessor = null) where T : ContentRule, new()
```

Parameters

`includePattern` string

A pattern to use for building of content files.

`contentImporter` IContentImporter

An IContentImporter to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate importer.

`contentProcessor` IContentProcessor

An IContentProcessor to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate processor.

Type Parameters

`T`

A ContentRule to be used for matching the includePattern. Some built in options are:

WildcardRule - specifies that input pattern should be a wildcard.

RegexRule - specifies that input pattern should be a regex.

Include<T>(string, Func<string, string>, IContentImporter?, IContentProcessor?)

Marks the files that match the includePattern to be built with the specified settings.

```
public void Include<T>(string includePattern, Func<string, string> outputPath,
IContentImporter? contentImporter = null, IContentProcessor? contentProcessor =
null) where T : ContentRule, new()
```

Parameters

`includePattern` string

A pattern to use for building of content files.

`outputPath` Func<string, string>

A function that gives a relative output filepath for the specified relative content filepath.

`contentImporter` IContentImporter

An IContentImporter to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate importer.

`contentProcessor` IContentProcessor

An IContentProcessor to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate processor.

Type Parameters

`T`

A ContentRule to be used for matching the includePattern. Some built in options are:

WildcardRule - specifies that input pattern should be a wildcard.

RegexRule - specifies that input pattern should be a regex.

SetContentRoot(string)

Sets the content root path that will be used when generating the output paths during include methods.

```
public void SetContentRoot(string contentRoot)
```

Parameters

`contentRoot` string

Relative path that will get prefixed to the output paths.

Class ContentInfo

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Describes how the ContentBuilder should handle the content.

```
public class ContentInfo
```

Inheritance

object ← ContentInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentInfo(string, bool, IContentImporter?, IContentProcessor?, Func<string, string>?)

Describes how the ContentBuilder should handle the content.

```
public ContentInfo(string contentRoot = "", bool shouldBuild = true,
IContentImporter? importer = null, IContentProcessor? processor = null, Func<string,
string>? outputPath = null)
```

Parameters

`contentRoot` string

The desired content root that will get prefixed to the output path.

`shouldBuild` bool

Indicates if the content should be built or copied.

`importer` IContentImporter

The desired IContentImporter to be used for the content building.

`processor` IContentProcessor

The desired IContentProcessor to be used for the content building.

`outputPath` Func<string, string>

The desired output path to be setup based on the input path to the content.

Properties

ContentRoot

A relative path to be used as a prefix to the output path.

```
public string ContentRoot { get; init; }
```

Property Value

string

Importer

An IContentImporter to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate importer.

```
public IContentImporter? Importer { get; init; }
```

Property Value

IContentImporter

Processor

An IContentProcessor to be used for the building, if its not specified, the system will use reflection to try to figure out an apropriate processor.

```
public IContentProcessor? Processor { get; init; }
```

Property Value

IContentProcessor

ShouldBuild

`true` if the content should be built, `false` if the content should be copied.

```
public bool ShouldBuild { get; init; }
```

Property Value

bool

Methods

GetOutputPath(string)

Gets the desired output path for the current ShouldBuild operation.

```
public string GetOutputPath(string filePath)
```

Parameters

`filePath` string

A relative path to the content file (without extension in case of build action).

Returns

string

Desired relative path for the output content.

Class ContentRule

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

A rule by which ContentCollection includes or excludes the files.

```
public abstract class ContentRule
```

Inheritance

object ← ContentRule

Derived

RegexRule, WildcardRule

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Pattern

A pattern passed from ContentCollection for using in IsMatch(string) method.

```
public string Pattern { get; init; }
```

Property Value

string

Methods

IsMatch(string)

Used in ContentCollection to check if the passed filePath matches the current Pattern.

```
public abstract bool IsMatch(string filePath)
```

Parameters

`filePath` string

Relative path to the content file.

Returns

bool

Returns a boolean indicating if the rule has matched the file.

Class FileHelper

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Common file and path helpers.

```
public static class FileHelper
```

Inheritance

object ← FileHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

BackwardSlash

A backward slash.

```
public static readonly char BackwardSlash
```

Field Value

char

ForwardSlash

A forward slash.

```
public static readonly char ForwardSlash
```

Field Value

char

NotSeparator

The directory seperator that is not common for the current platform.

```
public static readonly char NotSeparator
```

Field Value

char

Separator

The required or default directory seperator for this platform.

```
public static readonly char Separator
```

Field Value

char

Methods

DeleteIfExists(string)

Checks deletes a file from disk without throwing exceptions.

```
public static void DeleteIfExists(string path)
```

Parameters

`path` string

NormalizeDirectorySeparators(string)

Fixes the incoming path to have the default directory seperators for the current platform.

```
public static string NormalizeDirectorySeparators(string path)
```

Parameters

`path` string

Returns

string

Interface IContentCache

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

An interface for storing information about the compiled content.

```
public interface IContentCache
```

Methods

CleanCache(ContentBuilder)

Clears out any unused content files from the cache and the disk that were not marked by Mark Used(IContentFileCache).

```
void CleanCache(ContentBuilder builder)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

CreateContentFileCache(ContentBuilder, ContentInfo)

Creates a new insteance of IContentFileCache that the current implementation of IContentCache uses.

```
IContentFileCache CreateContentFileCache(ContentBuilder builder, ContentInfo info)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

`info` ContentInfo

A ContentInfo for which to create the IContentFileCache for.

Returns

IContentFileCache

FlushCache(ContentBuilder)

Saves any pending content cache information to disk for the specified builder.

```
void FlushCache(ContentBuilder builder)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

LoadCache(ContentBuilder)

Loads the content cache for the specified builder.

```
void LoadCache(ContentBuilder builder)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

MarkUsed(IContentFileCache)

Marks the specified IContentFileCache as being used by the ContentBuilder and that is should not get cleaned up by the CleanCache(ContentBuilder)

```
void MarkUsed(IContentFileCache fileCache)
```

Parameters

`fileCache` IContentFileCache

An insteance of IContentFileCache that contains information about the built content.

ReadContentFileCache(ContentBuilder, string)

Reads (from memory or disk) cached information about the content file and returns it if its valid.

```
IContentFileCache? ReadContentFileCache(ContentBuilder builder,
string relativeDstPath)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

`relativeDstPath` string

A relative path to the content file.

Returns

IContentFileCache

An instance of ContentFileCache if cached information about the content is found, `null` otherwise.

WriteContentFileCache(ContentBuilder, string, IContentFileCache?)

Writes down (to memory or disk) information about the content file.

Do note that FlushCache(ContentBuilder) will always be called before exiting the content builder.

```
void WriteContentFileCache(ContentBuilder builder, string relativeDstPath,
IContentFileCache? fileCache)
```

Parameters

`builder` ContentBuilder

A ContentBuilder that the cache file is related to.

`relativeDstPath` string

A relative path to the content file.

`fileCache` IContentFileCache

An insteance of IContentFileCache that contains information about the built content.

Interface IContentCollection

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

An interface for a collection of data on how the content should be handled.

```
public interface IContentCollection
```

Methods

GetContentInfo(string, out List<ContentInfo>)

Gets information about how the ocntent should be handled for the passed relative filepath.

```
bool GetContentInfo(string filePath, out List<ContentInfo> contentInfos)
```

Parameters

`filePath` string

Relative path to the content file.

`contentInfos` List<ContentInfo>

A list of ContentInfo describing the desired content handling.

Returns

bool

`true` if the content should be handled, `false` otherwise.

Interface IContentFileCache

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

An interface for storing information about the compiled content.

```
public interface IContentFileCache
```

Methods

AddDependency(ContentBuilder, IContentFileCache)

Adds the specified file cache as a dependency related to the current content file.

```
void AddDependency(ContentBuilder builder, IContentFileCache fileCache)
```

Parameters

`builder` ContentBuilder

A ContentBuilder the added depedency is related to.

`fileCache` IContentFileCache

A dependent file cache.

AddDependency(ContentBuilder, string)

Adds the specified file as a dependency related to the current content file.

```
void AddDependency(ContentBuilder builder, string dependencyPath)
```

Parameters

`builder` ContentBuilder

A ContentBuilder the added depedency is related to.

`dependencyPath` string

A relative or absolute path to the dependency file.

AddOutputFile(ContentBuilder, string)

Adds the specified file as an output file related to the current content file.

```
void AddOutputFile(ContentBuilder builder, string outputPath)
```

Parameters

`builder` ContentBuilder

A ContentBuilder the output file is related to.

`outputPath` string

A relative or absolute path to the output file.

IsValid(ContentBuilder, ContentInfo)

```
bool IsValid(ContentBuilder builder, ContentInfo info)
```

Parameters

`builder` ContentBuilder

A ContentBuilder the added depedency is related to.

`info` ContentInfo

A ContentInfo for which to use to check the validity of the cached asset.

Returns

bool

`true` if the file cache is still valid, `false` otherwise.

Class PathHelper

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Methods provided to normalize and manipulate paths.

```
public static class PathHelper
```

Inheritance

object ← PathHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

DirectorySeparator

The/universal/standard/directory/seperator.

```
public const char DirectorySeparator = '/'
```

Field Value

char

Methods

GetRelativePath(string, string)

Returns a path relative to the base path.

```
public static string GetRelativePath(string basePath, string path)
```

Parameters

`basePath` string

The path to make relative to. Must end with directory seperator.

`path` string

The path to be made relative to the basePath.

Returns

string

The relative path or the original string if it is not absolute or cannot be made relative.

Normalize(string)

Returns a path string normalized to the/universal/standard.

```
public static string Normalize(string path)
```

Parameters

`path` string

Returns

string

NormalizeDirectory(string)

Returns a directory path string normalized to the/universal/standard with a trailing seperator.

```
public static string NormalizeDirectory(string path)
```

Parameters

`path` string

Returns

string

NormalizeWindows(string)

Returns a path string normalized to the\Windows\standard.

```
public static string NormalizeWindows(string path)
```

Parameters

`path` string

Returns

string

Class PipelineBuildEvent

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide methods and properties for handling pipeline build events.

```
public class PipelineBuildEvent
```

Inheritance

object ← PipelineBuildEvent

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PipelineBuildEvent()

Creates a new pipeline build event.

```
public PipelineBuildEvent()
```

Fields

Extension

Gets the extension of the file to use in this pipeline event

```
public static readonly string Extension
```

Field Value

string

Properties

BuildAsset

Gets or sets the additional (nested) assets.

```
public List<string> BuildAsset { get; set; }
```

Property Value

List<string>

The additional (nested) assets.

Remarks

Additional assets are built by using an ExternalReference<T> and calling BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>, string) or BuildAsset<TInput, TOutput>(ExternalReference<TInput>, string).

Examples: The mesh processor may build textures and effects in addition to the mesh.

BuildOutput

Gets or sets the related output files.

```
public List<string> BuildOutput { get; set; }
```

Property Value

List<string>

The related output files.

Remarks

Related output files are non-XNB files that are included in addition to the XNB files. Related output files need to be copied to the output folder by a content processor and registered by calling AddOutputFile(string).

Dependencies

Gets or sets the dependencies.

```
public List<string> Dependencies { get; set; }
```

Property Value

List<string>

The dependencies.

Remarks

Dependencies are extra files that are required in addition to the SourceFile. Dependencies are added using AddDependency(string). Changes to the dependent file causes a rebuilt of the content.

DestFile

Absolute path to the output file.

```
public string DestFile { get; set; }
```

Property Value

string

DestTime

The date/time stamp of the destination file.

```
public DateTime DestTime { get; set; }
```

Property Value

DateTime

Importer

The name of the DLL containing the importer.

```
public string Importer { get; set; }
```

Property Value

string

ImporterTime

The date/time stamp of the DLL containing the importer.

```
public DateTime ImporterTime { get; set; }
```

Property Value

DateTime

Parameters

Gets or sets the parameters of this build event.

```
public OpaqueDataDictionary Parameters { get; set; }
```

Property Value

OpaqueDataDictionary

Remarks

Parameter are stored in an OpaqueDataDictionary

ParametersXml

Gets or sets the list of parameters.

```
public List<PipelineBuildEvent.Pair> ParametersXml { get; set; }
```

Property Value

List<PipelineBuildEvent.Pair>

Processor

The name of the DLL containing the processor.

```
public string Processor { get; set; }
```

Property Value

string

ProcessorTime

The date/time stamp of the DLL containing the processor.

```
public DateTime ProcessorTime { get; set; }
```

Property Value

DateTime

SourceFile

Absolute path to the source file.

```
public string SourceFile { get; set; }
```

Property Value

string

SourceTime

The date/time stamp of the source file.

```
public DateTime SourceTime { get; set; }
```

Property Value

DateTime

Methods

Load(string)

Creates a pipeline build event from a file.

```
public static PipelineBuildEvent Load(string filePath)
```

Parameters

`filePath` string

Path of the file to process.

Returns

PipelineBuildEvent

PipelineBuildEvent instance.

NeedsRebuild(PipelineManager, PipelineBuildEvent)

Checks if the current content files need to be rebuilt.

```
public bool NeedsRebuild(PipelineManager manager, PipelineBuildEvent cachedEvent)
```

Parameters

`manager` PipelineManager

Pipeline manager.

`cachedEvent` PipelineBuildEvent

Cached build event.

Returns

bool

`true` if the content needs to be rebuilt; otherwise `false`.

Save(string)

Saves the build event to a file as serialized Xml.

```
public void Save(string filePath)
```

Parameters

`filePath` string

Full path to save the file.

Class PipelineBuildEvent.Pair

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class that represents a key / value pair.

```
public class PipelineBuildEvent.Pair
```

Inheritance

object ← PipelineBuildEvent.Pair

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Key

Name of the key..

```
public string Key { get; set; }
```

Property Value

string

Value

Value related to the key.

```
public string Value { get; set; }
```

Property Value

string

Class PipelineBuildLogger

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides methods for reporting informational messages or warnings from content importers and processors. Do not use this class to report errors. Instead, report errors by throwing a PipelineException or InvalidContentException.

```
public class PipelineBuildLogger : ContentBuildLogger
```

Inheritance

object ← ContentBuildLogger ← PipelineBuildLogger

Inherited Members

ContentBuildLogger.LoggerRootDirectory , ContentBuildLogger.LoggerLogLevel , ContentBuildLogger.IndentString , ContentBuildLogger.GetCurrentFilename(ContentIdentity) , ContentBuildLogger.Log(LogLevel, string) , ContentBuildLogger.Log(string) , ContentBuildLogger.PushFile(string) , ContentBuildLogger.PopFile() , ContentBuildLogger.Indent() , ContentBuildLogger.Unindent() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

LogImportantMessage(string, params object[])

Outputs a high-priority status message from the content system.

```
public override void LogImportantMessage(string message, params
object[] messageArgs)
```

Parameters

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

LogMessage(string, params object[])

Outputs a low priority status message from the content system.

```
public override void LogMessage(string message, params object[] messageArgs)
```

Parameters

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

LogWarning(string, ContentIdentity, string, params object[])

Outputs a warning message from the content system.

```
public override void LogWarning(string helpLink, ContentIdentity contentIdentity,
string message, params object[] messageArgs)
```

Parameters

`helpLink` string

Link to an existing online help topic containing related information.

`contentIdentity` ContentIdentity

Identity of the content item that generated the message.

`message` string

Message being reported.

`messageArgs` object[]

Arguments for the reported message.

Class PipelineImporterContext

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

```
public class PipelineImporterContext : ContentImporterContext
```

Inheritance

object ← ContentImporterContext ← PipelineImporterContext

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PipelineImporterContext(PipelineManager, PipelineBuildEvent)

```
public PipelineImporterContext(PipelineManager manager, PipelineBuildEvent
pipelineEvent)
```

Parameters

`manager` PipelineManager

`pipelineEvent` PipelineBuildEvent

Properties

IntermediateDirectory

The absolute path to the root of the build intermediate (object) directory.

```
public override string IntermediateDirectory { get; }
```

Property Value

string

Logger

Gets the logger for an importer.

```
public override ContentBuildLogger Logger { get; }
```

Property Value

ContentBuildLogger

OutputDirectory

The absolute path to the root of the build output (binaries) directory.

```
public override string OutputDirectory { get; }
```

Property Value

string

Methods

AddDependency(string)

Adds a dependency to the specified file. This causes a rebuild of the file, when modified, on subsequent incremental builds.

```
public override void AddDependency(string filename)
```

Parameters

`filename` string

Name of an asset file.

Class PipelineManager

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Class to provide methods and properties for handling content pipelines.

```
public class PipelineManager
```

Inheritance

object ← PipelineManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PipelineManager(string, string, string)

Creates a new instance of PipelineManager.

```
public PipelineManager(string projectDir, string outputDir, string intermediateDir)
```

Parameters

`projectDir` string

The directory that contains the content project.

`outputDir` string

The directory that contains the results of the project.

`intermediateDir` string

The directory that is used for temporary files created by the content build process

Properties

Assemblies

Gets the list of assemblies referenced by the content project file.

```
public List<string> Assemblies { get; }
```

Property Value

List<string>

CompressContent

Gets or sets if the content is compressed.

```
public bool CompressContent { get; set; }
```

Property Value

bool

Config

The build configuration passed thru to content processors.

```
public string Config { get; set; }
```

Property Value

string

ContentStats

Gets the content stats of the previous build.

```
public ContentStatsCollection ContentStats { get; }
```

Property Value

ContentStatsCollection

IntermediateDirectory

Gets or sets the directory that is used for temporary files created by the content build process.

```
public string IntermediateDirectory { get; }
```

Property Value

string

Logger

Gets or sets the content build logger.

```
public ContentBuildLogger Logger { get; set; }
```

Property Value

ContentBuildLogger

OutputDirectory

Gets or sets the directory that contains the results of the project.

```
public string OutputDirectory { get; }
```

Property Value

string

Platform

The current target platform for which all content is built.

```
public TargetPlatform Platform { get; set; }
```

Property Value

TargetPlatform

Profile

The current target graphics profile for which all content is built.

```
public GraphicsProfile Profile { get; set; }
```

Property Value

GraphicsProfile

ProjectDirectory

Gets or sets the directory that contains the content project.

```
public string ProjectDirectory { get; }
```

Property Value

string

Quiet

If true, messages which are non-essential to the build process will be suppressed.

```
public bool Quiet { get; set; }
```

Property Value

bool

RethrowExceptions

If true exceptions thrown from within an importer or processor are caught and then thrown from the context. Default value is true.

```
public bool RethrowExceptions { get; set; }
```

Property Value

bool

Methods

AddAssembly(string)

Adds an assembly to be used by the pipeline.

```
public void AddAssembly(string assemblyFilePath)
```

Parameters

`assemblyFilePath` string

Assembly file path.

Exceptions

ArgumentException

Thrown if the assembly file path is not absolute or null.

AssignTypeConverter<TType, TTypeConverter>()

Assigns a type converter for a type

```
public void AssignTypeConverter<TType, TTypeConverter>()
```

Type Parameters

`TType`

Type to be converted.

`TTypeConverter`

Type converter to be applied to the type.

BuildContent(string, string, string, string, OpaqueDataDictionary)

Content builder.

```
public PipelineBuildEvent BuildContent(string sourceFilepath, string outputFilepath
= null, string importerName = null, string processorName = null,
OpaqueDataDictionary processorParameters = null)
```

Parameters

`sourceFilepath` string

Source file path.

`outputFilepath` string

Output file path.

`importerName` string

Name of the importer to use.

`processorName` string

Name of the processor to use.

`processorParameters` OpaqueDataDictionary

Opaque data dictionary containing the processor parameters.

Returns

PipelineBuildEvent

PipelineBuildEvent instance.

CleanContent(string, string)

Cleans content from the intermediate folder.

```
public void CleanContent(string sourceFilepath, string outputFilepath = null)
```

Parameters

`sourceFilepath` string

Path of the source file.

`outputFilepath` string

Path of the output file.

CreateImporter(string)

Creates an instance of an importer.

```
public IContentImporter CreateImporter(string name)
```

Parameters

`name` string

Name of the importer.

Returns

IContentImporter

Content importer.

CreateProcessor(string, OpaqueDataDictionary)

Creates an instance of a processor.

```
public IContentProcessor CreateProcessor(string name, OpaqueDataDictionary
processorParameters)
```

Parameters

`name` string

Name of the processor.

`processorParameters` OpaqueDataDictionary

Opaque data dictionary containing the processor parameters.

Returns

IContentProcessor

Content processor.

FindDefaultProcessor(string)

Gets the default processor for an importer.

```
public string FindDefaultProcessor(string importer)
```

Parameters

`importer` string

Name of the importer.

Returns

string

Name of the processor or null if not found.

FindImporterByFileName(string)

Returns the importer type name based on the file name (including extension).

```
public string FindImporterByFileName(string fileNameWithExt)
```

Parameters

`fileNameWithExt` string

Then name of the file including the extension.

Returns

string

Importer type name or null if not found.

GetAssetName(string, string, string, OpaqueDataDictionary)

Gets an automatic asset name, such as "AssetName_0".

```
public string GetAssetName(string sourceFileName, string importerName, string
processorName, OpaqueDataDictionary processorParameters)
```

Parameters

`sourceFileName` string

The source file name.

`importerName` string

The name of the content importer. Can be null.

`processorName` string

The name of the content processor. Can be null.

`processorParameters` OpaqueDataDictionary

The processor parameters. Can be null.

Returns

string

The asset name.

GetImporterAssemblyTimestamp(string)

Gets the importer assembly timestamp.

```
public DateTime GetImporterAssemblyTimestamp(string name)
```

Parameters

`name` string

Assembly name.

Returns

DateTime

Timestamp.

GetImporterTypes()

Gets the importer types.

```
public Type[] GetImporterTypes()
```

Returns

Type[]

Array of types.

GetProcessorAssemblyTimestamp(string)

Gets the timestamp of the processor assembly.

```
public DateTime GetProcessorAssemblyTimestamp(string name)
```

Parameters

`name` string

Name of the processor.

Returns

DateTime

Timestamp.

GetProcessorDefaultValues(string)

Gets the default values for the content processor parameters.

```
public OpaqueDataDictionary GetProcessorDefaultValues(string processorName)
```

Parameters

`processorName` string

The name of the content processor.

Returns

OpaqueDataDictionary

A dictionary containing the default value for each parameter. Returns null if the content processor has not been created yet.

GetProcessorType(string)

Gets the processor type.

```
public Type GetProcessorType(string name)
```

Parameters

`name` string

Name of the processor.

Returns

Type

Type of the processor or null if not found.

GetProcessorTypes()

Gets the processor types.

```
public Type[] GetProcessorTypes()
```

Returns

Type[]

Array of types.

ProcessContent(PipelineBuildEvent)

Processes the content from the pipeline event.

```
public object ProcessContent(PipelineBuildEvent pipelineEvent)
```

Parameters

`pipelineEvent` PipelineBuildEvent

Pipeline event.

Returns

object

Object processed by the pipeline.

Exceptions

PipelineException

Thrown if there was an error processing the content.

RegisterContent(string, string, string, string, OpaqueDataDictionary)

Registers content to be processed by the pipeline manager.

```
public void RegisterContent(string sourceFilepath, string outputFilepath = null,
string importerName = null, string processorName = null, OpaqueDataDictionary
processorParameters = null)
```

Parameters

`sourceFilepath` string

Source file path.

`outputFilepath` string

Output file path.

`importerName` string

Name of the importer to use.

`processorName` string

Name of the processor to use.

`processorParameters` OpaqueDataDictionary

Opaque data dictionary containing the processor parameters.

ResolveImporterAndProcessor(string, ref string, ref string)

Resolves the importer and processor names.

```
public void ResolveImporterAndProcessor(string sourceFilepath, ref string
importerName, ref string processorName)
```

Parameters

`sourceFilepath` string

Path of the source file.

`importerName` string

Name of the importer.

`processorName` string

Name of the processor.

Exceptions

Exception

Thrown if a default importer or processor could not be found.

ValidateProcessorParameters(string, OpaqueDataDictionary)

Validate a list of processor parameters for a given processor type.

```
public OpaqueDataDictionary ValidateProcessorParameters(string name,
OpaqueDataDictionary processorParameters)
```

Parameters

`name` string

Name of the processor type.

`processorParameters` OpaqueDataDictionary

Opaque data dictionary containing the processor parameters.

Returns

OpaqueDataDictionary

Opaque data dictionary containing the validated parameters.

Class PipelineProcessorContext

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides access to custom processor parameters, methods for converting member data, and triggering nested builds.

```
public class PipelineProcessorContext : ContentProcessorContext
```

Inheritance

object ← ContentProcessorContext ← PipelineProcessorContext

Inherited Members

ContentProcessorContext.BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>, string) , ContentProcessorContext.BuildAsset<TInput, TOutput>(ExternalReference<TInput>, string) , ContentProcessorContext.Convert<TInput, TOutput>(TInput, string) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PipelineProcessorContext(PipelineManager, PipelineBuildEvent)

Creates a new pipeline processor context.

```
public PipelineProcessorContext(PipelineManager manager, PipelineBuildEvent
pipelineEvent)
```

Parameters

`manager` PipelineManager

Pipeline manager.

`pipelineEvent` PipelineBuildEvent

Pipeline event.

Properties

BuildConfiguration

Gets the name of the current content build configuration.

```
public override string BuildConfiguration { get; }
```

Property Value

string

IntermediateDirectory

Gets the path of the directory that will contain any intermediate files generated by the content processor.

```
public override string IntermediateDirectory { get; }
```

Property Value

string

Logger

Gets the logger interface used for status messages or warnings.

```
public override ContentBuildLogger Logger { get; }
```

Property Value

ContentBuildLogger

OutputDirectory

Gets the output path of the content processor.

```
public override string OutputDirectory { get; }
```

Property Value

string

OutputFilename

Gets the output file name of the content processor.

```
public override string OutputFilename { get; }
```

Property Value

string

Parameters

Gets the collection of parameters used by the content processor.

```
public override OpaqueDataDictionary Parameters { get; }
```

Property Value

OpaqueDataDictionary

ProjectDirectory

Gets the directory that contains the content project.

```
public override string ProjectDirectory { get; }
```

Property Value

string

SourceIdentity

Gets the ContentIdentity representing the source asset imported.

```
public override ContentIdentity SourceIdentity { get; }
```

Property Value

ContentIdentity

TargetPlatform

Gets the current content build target platform.

```
public override TargetPlatform TargetPlatform { get; }
```

Property Value

TargetPlatform

TargetProfile

Gets the current content build target profile.

```
public override GraphicsProfile TargetProfile { get; }
```

Property Value

GraphicsProfile

Methods

AddDependency(string)

Adds a dependency to the specified file. This causes a rebuild of the file, when modified, on subsequent incremental builds.

```
public override void AddDependency(string filename)
```

Parameters

`filename` string

Name of an asset file.

AddOutputFile(string)

Add a file name to the list of related output files maintained by the build item. This allows tracking build items that build multiple output files.

```
public override void AddOutputFile(string filename)
```

Parameters

`filename` string

The name of the file.

BuildAndLoadAsset<TInput, TOutput> (ExternalReference<TInput>, IContentImporter, IContentProcessor)

Initiates a nested build of the specified asset and then loads the result into memory.

```
public override TOutput BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>
sourceAsset, IContentImporter importer, IContentProcessor processor)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`importer` IContentImporter

Optional importer for this content.

`processor` IContentProcessor

Optional processor for this content.

Returns

TOutput

Copy of the final converted content.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the converted output.

Remarks

An example of usage would be a mesh processor calling BuildAndLoadAsset to build any associated textures and replace the original .tga file references with an embedded copy of the converted texture.

BuildAndLoadAsset<TInput, TOutput> (ExternalReference<TInput>, string, OpaqueDataDictionary, string)

Initiates a nested build of the specified asset and then loads the result into memory.

```
public override TOutput BuildAndLoadAsset<TInput, TOutput>(ExternalReference<TInput>
sourceAsset, string processorName, OpaqueDataDictionary processorParameters,
string importerName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional collection of named values available as input to the content processor.

`importerName` string

Optional importer for this content.

Returns

TOutput

Copy of the final converted content.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the converted output.

Remarks

An example of usage would be a mesh processor calling BuildAndLoadAsset to build any associated textures and replace the original .tga file references with an embedded copy of the converted texture.

BuildAsset<TInput, TOutput>(ExternalReference<TInput>, IContentImporter, IContentProcessor, string?)

Initiates a nested build of an additional asset.

```
public override ExternalReference<TOutput> BuildAsset<TInput, TOutput>
(ExternalReference<TInput> sourceAsset, IContentImporter importer, IContentProcessor
processor, string? assetName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`importer` IContentImporter

Optional importer for this content.

`processor` IContentProcessor

Optional processor for this content.

`assetName` string

Optional name of the final compiled content.

Returns

ExternalReference<TOutput>

Reference to the final compiled content. The build work is not required to complete before returning. Therefore, this file may not be up to date when BuildAsset returns but it will be available for loading by the game at runtime.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the output.

Remarks

An example of usage for BuildAsset is being called by a mesh processor to request that any related textures used are also built, replacing the original TGA file references with new references to the converted texture files.

BuildAsset<TInput, TOutput>(ExternalReference<TInput>, string, OpaqueDataDictionary, string, string)

Initiates a nested build of an additional asset.

```
public override ExternalReference<TOutput> BuildAsset<TInput, TOutput>
(ExternalReference<TInput> sourceAsset, string processorName, OpaqueDataDictionary
processorParameters, string importerName, string assetName)
```

Parameters

`sourceAsset` ExternalReference<TInput>

Reference to the source asset.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional collection of named values available as input to the content processor.

`importerName` string

Optional importer for this content.

`assetName` string

Optional name of the final compiled content.

Returns

ExternalReference<TOutput>

Reference to the final compiled content. The build work is not required to complete before returning. Therefore, this file may not be up to date when BuildAsset returns but it will be available for loading by the game at runtime.

Type Parameters

`TInput`

Type of the input.

`TOutput`

Type of the output.

Remarks

An example of usage for BuildAsset is being called by a mesh processor to request that any related textures used are also built, replacing the original TGA file references with new references to the converted texture files.

Convert<TInput, TOutput>(TInput, IContentProcessor)

Converts a content item object using the specified content processor.

```
public override TOutput Convert<TInput, TOutput>(TInput input,
IContentProcessor processor)
```

Parameters

`input` TInput

Source content to be converted.

`processor` IContentProcessor

Optional processor for this content.

Returns

TOutput

Reference of the final converted content.

Type Parameters

`TInput`

Type of the input content.

`TOutput`

Type of the converted output.

Convert<TInput, TOutput>(TInput, string, OpaqueDataDictionary)

Converts a content item object using the specified content processor.

```
public override TOutput Convert<TInput, TOutput>(TInput input, string processorName,
OpaqueDataDictionary processorParameters)
```

Parameters

`input` TInput

Source content to be converted.

`processorName` string

Optional processor for this content.

`processorParameters` OpaqueDataDictionary

Optional parameters for the processor.

Returns

TOutput

Reference of the final converted content.

Type Parameters

`TInput`

Type of the input content.

`TOutput`

Type of the converted output.

Class RegexRule

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

A regex based rule by which ContentCollection includes or excludes the files.

```
public class RegexRule : ContentRule
```

Inheritance

object ← ContentRule ← RegexRule

Inherited Members

ContentRule.Pattern , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

IsMatch(string)

Used in ContentCollection to check if the passed filePath matches the current Regex Pattern.

```
public override bool IsMatch(string filePath)
```

Parameters

`filePath` string

Relative path to the content file.

Returns

bool

Returns true if filePath matches the Pattern.

Class TypeExtensions

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Provides extension methods for common types.

```
public static class TypeExtensions
```

Inheritance

object ← TypeExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AddRangeUnique<T>(List<T>, List<T>)

Adds a range of items to the list if they are not already present.

```
public static void AddRangeUnique<T>(this List<T> dstList, List<T> list)
```

Parameters

`dstList` List<T>

The list to add the items to.

`list` List<T>

The list of items to add.

Type Parameters

`T`

The type of the items in the list.

AddUnique<T>(List<T>, T)

Adds an item to the list if it is not already present.

```
public static void AddUnique<T>(this List<T> list, T item)
```

Parameters

`list` List<T>

The list to add the item to.

`item` T

The item to add to the list.

Type Parameters

`T`

The type of the items in the list.

ToColor(Color)

Takes a System.Drawing.Color and converts it to a Xna.Framework.Color.

```
public static Color ToColor(this Color color)
```

Parameters

`color` Color

The color to convert.

Returns

Color

The converted color.

ToVector3(Color)

Takes a System.Drawing.Color and converts it to a Vector3.

```
public static Vector3 ToVector3(this Color color)
```

Parameters

`color` Color

The color to convert.

Returns

Vector3

The resulting Vector3.

Class WildcardRule

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

A wildcard based rule by which ContentCollection includes or excludes the files.

```
public class WildcardRule : ContentRule
```

Inheritance

object ← ContentRule ← WildcardRule

Inherited Members

ContentRule.Pattern , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

IsMatch(string)

Used in ContentCollection to check if the passed filePath matches the current Wildcard Pattern.

```
public override bool IsMatch(string filePath)
```

Parameters

`filePath` string

Relative path to the content file.

Returns

bool

Returns true if filePath matches the Wildcard Pattern.

Class XmlColor

Namespace: MonoGame.Framework.Content.Pipeline.Builder

Assembly: MonoGame.Framework.Content.Pipeline.dll

Helper for serializing color types with the XmlSerializer.

```
public class XmlColor
```

Inheritance

object ← XmlColor

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

XmlColor()

Creates a new instance of XmlColor with Empty color.

```
public XmlColor()
```

XmlColor(Color)

Creates a new instance of XmlColor with provided color

```
public XmlColor(Color c)
```

Parameters

c Color

Color to be stored

Properties

Default

Default color value as a string, used for serialization and deserialization.

```
public string Default { get; set; }
```

Property Value

string

Methods

FromColor(Color)

Returns a string representation of the supplied Color.

```
public static string FromColor(Color color)
```

Parameters

`color` Color

Color to be converted.

Returns

string

The predefined name of the color, if it's named, or a string in the format **"R, G, B, A"**, if the color is not named, with every component in the [0..255] range.

Remarks

This string value can be used in ToColor(string) to get identical color object.

ToColor(string)

Returns a new Color object from its string representation.

```
public static Color ToColor(string value)
```

Parameters

`value` string

A predefined color name, OR a string in the format **"R, G, B, A"**, where each component is in the [0..255] range.

Returns

Color

Operators

implicit operator Color(XmlColor)

Implicit XmlColor -> Color conversion

```
public static implicit operator Color(XmlColor x)
```

Parameters

`x` XmlColor

Returns

Color

implicit operator XmlColor(Color)

Implicit Color -> XmlColor conversion

```
public static implicit operator XmlColor(Color c)
```

Parameters

c Color

Returns

XmlColor

Namespace MonoGame.Framework.Content.Pipeline.Builder.Server

Classes

ContentRequestedArgs

EventArgs for the ContentRequested event that gets raised when content is being requested by an IContentProvider.

ContentServer

Describes a content server implementation to be used by the ContentBuilder.

ContentServerParameterAttribute

An ttribute used to specify that a property should be displaying as a cli argument.

NetworkContentServer

A ContentServer that is hosting a network.

Class ContentRequestedArgs

Namespace: MonoGame.Framework.Content.Pipeline.Builder.Server

Assembly: MonoGame.Framework.Content.Pipeline.dll

EventArgs for the ContentRequested event that gets raised when content is being requested by an IContentProvider.

```
public class ContentRequestedArgs : EventArgs
```

Inheritance

object ← EventArgs ← ContentRequestedArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentRequestedArgs(string)

EventArgs for the ContentRequested event that gets raised when content is being requested by an IContentProvider.

```
public ContentRequestedArgs(string contentPath)
```

Parameters

`contentPath` string

Represents a relative output path for the content.

Properties

CompilationStarted

Determines if ContentServer should wait for the content to be compiled.

```
public bool CompilationStarted { get; set; }
```

Property Value

bool

ContentPath

A relative output path for the content.

```
public string ContentPath { get; }
```

Property Value

string

FilePath

An absolute path on the system to the content file passed by ContentPath.

```
public string FilePath { get; set; }
```

Property Value

string

Class ContentServer

Namespace: MonoGame.Framework.Content.Pipeline.Builder.Server

Assembly: MonoGame.Framework.Content.Pipeline.dll

Describes a content server implementation to be used by the ContentBuilder.

```
public abstract class ContentServer
```

Inheritance

object ← ContentServer

Derived

NetworkContentServer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Logger

ContentBuildLogger to use for the server to log messages.

If run from ContentBuilder it will be set by it.

```
public ContentBuildLogger Logger { get; set; }
```

Property Value

ContentBuildLogger

Methods

NotifyContentRequestCompiled()

The content server calls this event once it has finished compiling content.

This method is called by the main thread and should be a non blocking call.

```
public abstract void NotifyContentRequestCompiled()
```

OnContentRequested(ContentRequestedArgs)

A protected method used to invoke ContentRequested event.

```
protected void OnContentRequested(ContentRequestedArgs args)
```

Parameters

`args` ContentRequestedArgs

ContentRequestedArgs to use for invoking ContentRequested event.

StartListening()

Requests the content server to start listening for the events.

This method is called by the main thread and should be a non blocking call.

```
public abstract void StartListening()
```

StopListening()

Requests the content server to stop listening for the events.

This method is called by the main thread and should be a non blocking call.

```
public abstract void StopListening()
```

Events

ContentRequested

An event that gets called when the content server recieves a content request by an IContentProvider.

```
public event EventHandler<ContentRequestedArgs>? ContentRequested
```

Event Type

EventHandler<ContentRequestedArgs>

Class ContentServerParameterAttribute

Namespace: MonoGame.Framework.Content.Pipeline.Builder.Server

Assembly: MonoGame.Framework.Content.Pipeline.dll

An ttribute used to specify that a property should be displaying as a cli argument.

```
[AttributeUsage(AttributeTargets.Property)]
public class ContentServerParameterAttribute : Attribute
```

Inheritance

object ← Attribute ← ContentServerParameterAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) ,
Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) ,
Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) ,
Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) ,
Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ContentServerParameterAttribute(string, string)

An ttribute used to specify that a property should be displaying as a cli argument.

```
public ContentServerParameterAttribute(string name, string description)
```

Parameters

`name` string

Name of the parameter to be shown in the cli

`description` string

Description for the parameter to be shown in the cli

Properties

Description

Description for the parameter to be shown in the cli.

```
public string Description { get; set; }
```

Property Value

string

Name

Name of the parameter to be shown in the cli.

```
public string Name { get; set; }
```

Property Value

string

Class NetworkContentServer

Namespace: MonoGame.Framework.Content.Pipeline.Builder.Server

Assembly: MonoGame.Framework.Content.Pipeline.dll

A ContentServer that is hosting a network.

```
public class NetworkContentServer : ContentServer
```

Inheritance

object ← ContentServer ← NetworkContentServer

Inherited Members

ContentServer.Logger , ContentServer.ContentRequested , ContentServer.OnContentRequested(ContentRequestedArgs) , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Port

Server port to use for the listening for content requests in Server mode.

```
[ContentServerParameter("port", "The port to be used for the content server mode.")]
public ushort Port { get; init; }
```

Property Value

ushort

7771 by default.

Methods

NotifyContentRequestCompiled()

The content server calls this event once it has finished compiling content.

This method is called by the main thread and should be a non blocking call.

```
public override void NotifyContentRequestCompiled()
```

StartListening()

Requests the content server to start listening for the events.

This method is called by the main thread and should be a non blocking call.

```
public override void StartListening()
```

StopListening()

Requests the content server to stop listening for the events.

This method is called by the main thread and should be a non blocking call.

```
public override void StopListening()
```

Namespace MonoGame.Framework.Devices.Sensors

Classes

Accelerometer

Provides Desktop applications access to the device's accelerometer sensor. Currently stubs, for cross-platform development.

AccelerometerFailedException

The exception that may be thrown during a call to Start() or Stop(). The Message field describes the reason for the exception and the ErrorId field contains the error code from the underlying native code implementation of the accelerometer framework.

CalibrationEventArgs

Provides data for Calibrate and events.

Compass

Provides Desktop applications access to the device's compass sensor. Currently stubs, for cross-platform development.

SensorBase<TSensorReading>

SensorFailedException

Represents errors related to sensors.

SensorReadingEventArgs<T>

Provides data for sensor events.

Structs

AccelerometerReading

Stores values from an accelerometer event

CompassReading

Provides properties for compass readings.

Interfaces

ISensorReading

Common interface used for sensor readings.

Enums

SensorState

Defines the state of the sensor.

Class Accelerometer

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Provides Desktop applications access to the device's accelerometer sensor. Currently stubs, for cross-platform development.

```
public sealed class Accelerometer : SensorBase<AccelerometerReading>, IDisposable
```

Inheritance

object ← SensorBase<AccelerometerReading> ← Accelerometer

Implements

IDisposable

Inherited Members

SensorBase<AccelerometerReading>.CurrentValue ,
SensorBase<AccelerometerReading>.IsDataValid ,
SensorBase<AccelerometerReading>.TimeBetweenUpdates ,
SensorBase<AccelerometerReading>.CurrentValueChanged ,
SensorBase<AccelerometerReading>.Dispose() , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

Accelerometer()

Creates a new instance of the Accelerometer object.

```
public Accelerometer()
```

Properties

IsSupported

Gets or sets whether the device on which the application is running supports the accelerometer sensor.

```
public static bool IsSupported { get; }
```

Property Value

bool

State

Gets the current state of the accelerometer. The value is a member of the SensorState enumeration.

```
public SensorState State { get; }
```

Property Value

SensorState

Methods

Dispose(bool)

Derived classes override this method to dispose of managed and unmanaged resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if unmanaged resources are to be disposed.

Start()

Starts data acquisition from the accelerometer.

```
public override void Start()
```

Stop()

Stops data acquisition from the accelerometer.

```
public override void Stop()
```

Class AccelerometerFailedException

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

The exception that may be thrown during a call to Start() or Stop(). The Message field describes the reason for the exception and the ErrorId field contains the error code from the underlying native code implementation of the accelerometer framework.

```
public class AccelerometerFailedException : SensorFailedException, ISerializable
```

Inheritance

object ← Exception ← SensorFailedException ← AccelerometerFailedException

Implements

ISerializable

Inherited Members

SensorFailedException.ErrorId , Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Struct AccelerometerReading

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Stores values from an accelerometer event

```
public struct AccelerometerReading : ISensorReading
```

Implements
ISensorReading

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

Acceleration

Acceleration vector

```
public readonly Vector3 Acceleration { get; }
```

Property Value

Vector3

Timestamp

Timestamp of the accelerometer reading

```
public readonly DateTimeOffset Timestamp { get; }
```

Property Value

DateTimeOffset

Class CalibrationEventArgs

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Provides data for Calibrate and events.

```
public class CalibrationEventArgs : EventArgs
```

Inheritance

object ← EventArgs ← CalibrationEventArgs

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CalibrationEventArgs()

Initializes a new instance of the CalibrationEventArgs class.

```
public CalibrationEventArgs()
```

Remarks

Obtain a CalibrationEventArgs object by implementing a handler for the Compass.Calibrate event.

Class Compass

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Provides Desktop applications access to the device's compass sensor. Currently stubs, for cross-platform development.

```
public sealed class Compass : SensorBase<CompassReading>, IDisposable
```

Inheritance

object ← SensorBase<CompassReading> ← Compass

Implements

IDisposable

Inherited Members

SensorBase<CompassReading>.CurrentValue , SensorBase<CompassReading>.IsDataValid , SensorBase<CompassReading>.TimeBetweenUpdates , SensorBase<CompassReading>.CurrentValueChanged , SensorBase<CompassReading>.Dispose() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Compass()

Creates a new instance of the Compass object.

```
public Compass()
```

Properties

IsSupported

Gets whether the device on which the application is running supports the compass sensor.

```
public static bool IsSupported { get; }
```

Property Value

bool

State

Gets the current state of the compass. The value is a member of the SensorState enumeration.

```
public SensorState State { get; }
```

Property Value

SensorState

Methods

Dispose(bool)

Derived classes override this method to dispose of managed and unmanaged resources.

```
protected override void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if unmanaged resources are to be disposed.

Start()

Starts data acquisition from the compass.

```
public override void Start()
```

Stop()

Stops data acquisition from the compass.

```
public override void Stop()
```

Struct CompassReading

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Provides properties for compass readings.

```
public struct CompassReading : ISensorReading
```

Implements
ISensorReading

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Properties

HeadingAccuracy

Heading accuracy.

```
public readonly double HeadingAccuracy { get; }
```

Property Value

double

MagneticHeading

Magnetic heading.

```
public readonly double MagneticHeading { get; }
```

Property Value

double

MagnetometerReading

Magnetometer reading.

```
public readonly Vector3 MagnetometerReading { get; }
```

Property Value

Vector3

Timestamp

Timestamp of the sensor reading.

```
public readonly DateTimeOffset Timestamp { get; }
```

Property Value

DateTimeOffset

TrueHeading

True heading.

```
public readonly double TrueHeading { get; }
```

Property Value

double

Interface ISensorReading

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Common interface used for sensor readings.

```
public interface ISensorReading
```

Properties

Timestamp

Timestamp of the sensor reading.

```
DateTimeOffset Timestamp { get; }
```

Property Value

DateTimeOffset

Class SensorBase<TSensorReading>

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

```
public abstract class SensorBase<TSensorReading> : IDisposable where TSensorReading
: ISensorReading
```

Type Parameters

`TSensorReading`

Inheritance

object ← SensorBase<TSensorReading>

Implements

IDisposable

Derived

Accelerometer, Compass

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SensorBase()

```
public SensorBase()
```

Properties

CurrentValue

```
public TSensorReading CurrentValue { get; protected set; }
```

Property Value

TSensorReading

IsDataValid

```
public bool IsDataValid { get; protected set; }
```

Property Value

bool

IsDisposed

```
protected bool IsDisposed { get; }
```

Property Value

bool

TimeBetweenUpdates

```
public TimeSpan TimeBetweenUpdates { get; set; }
```

Property Value

TimeSpan

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

Derived classes override this method to dispose of managed and unmanaged resources.

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

True if unmanaged resources are to be disposed.

~SensorBase()

```
protected ~SensorBase()
```

Start()

```
public abstract void Start()
```

Stop()

```
public abstract void Stop()
```

Events

CurrentValueChanged

```
public event EventHandler<SensorReadingEventArgs<TSensorReading>>
CurrentValueChanged
```

Event Type

EventHandler<SensorReadingEventArgs<TSensorReading>>

TimeBetweenUpdatesChanged

```
protected event EventHandler<EventArgs> TimeBetweenUpdatesChanged
```

Event Type

EventHandler<EventArgs>

Class SensorFailedException

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Represents errors related to sensors.

```
public class SensorFailedException : Exception, ISerializable
```

Inheritance

object ← Exception ← SensorFailedException

Implements

ISerializable

Derived

AccelerometerFailedException

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

ErrorId

Error Id.

```
public int ErrorId { get; protected set; }
```

Property Value

int

Class SensorReadingEventArgs<T>

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Provides data for sensor events.

```
public class SensorReadingEventArgs<T> : EventArgs where T : ISensorReading
```

Type Parameters

`T`

The type of the sensor.

Inheritance

object ← EventArgs ← SensorReadingEventArgs<T>

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

SensorReadingEventArgs(T)

Creates a new instance of the SensorReadingEventArgs class.

```
public SensorReadingEventArgs(T sensorReading)
```

Parameters

`sensorReading` T

Sensor reader.

Properties

SensorReading

Sensor reader type.

```
public T SensorReading { get; set; }
```

Property Value

T

Enum SensorState

Namespace: MonoGame.Framework.Devices.Sensors

Assembly: MonoGame.Framework.dll

Defines the state of the sensor.

```
public enum SensorState
```

Fields

`Disabled = 5`

Sensor is disabled.

`Initializing = 2`

Sensor is initializing.

`NoData = 3`

Sensor has no data.

`NoPermissions = 4`

No permissions to use the sensor.

`NotSupported = 0`

Sensor is not supported.

`Ready = 1`

Sensor is ready to be used.

Namespace MonoGame.Framework.Utilities

Classes

PlatformInfo

Utility class that returns information about the underlying platform

Enums

GraphicsBackend

Type of the underlying graphics backend.

MonoGamePlatform

Type of the underlying game platform.

Enum GraphicsBackend

Namespace: MonoGame.Framework.Utilities

Assembly: MonoGame.Framework.dll

Type of the underlying graphics backend.

```
public enum GraphicsBackend
```

Fields

`DirectX = 0`

Represents the Microsoft DirectX 11 graphics backend.

`DirectX12 = 4`

Represents the Microsoft DirectX 12 graphics backend.

`Metal = 3`

Represents the Apple Metal graphics backend.

`OpenGL = 1`

Represents the OpenGL graphics backend.

`Vulkan = 2`

Represents the Vulkan graphics backend.

Enum MonoGamePlatform

Namespace: MonoGame.Framework.Utilities

Assembly: MonoGame.Framework.dll

Type of the underlying game platform.

```
public enum MonoGamePlatform
```

Fields

`Android = 0`

Android platform.

`DesktopGL = 3`

Cross platform desktop using OpenGL.

`DesktopVK = 12`

Cross platform desktop using Vulkan.

`NintendoSwitch = 11`

Nintendo Switch platform.

`PlayStation4 = 9`

PlayStation 4 platform.

`PlayStation5 = 10`

PlayStation 5 platform.

`WebGL = 5`

WebGL platform.

`Windows = 4`

Windows platform using DirectX 11.

`WindowsDX12 = 7`

Windows platform using DirectX 12.

`XboxOne = 6`

Xbox One platform.

`XboxSeries = 8`

Xbox Series X|S platform.

`iOS = 1`

iOS platform.

`tvOS = 2`

tvOS platform.

Class PlatformInfo

Namespace: MonoGame.Framework.Utilities

Assembly: MonoGame.Framework.dll

Utility class that returns information about the underlying platform

```
public static class PlatformInfo
```

Inheritance

object ← PlatformInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

GraphicsBackend

Graphics backend

```
public static GraphicsBackend GraphicsBackend { get; }
```

Property Value

GraphicsBackend

MonoGamePlatform

Underlying game platform type

```
public static MonoGamePlatform MonoGamePlatform { get; }
```

Property Value

MonoGamePlatform