

Table of Contents

Articles

Introduction

Setup and Useful Tricks

Elements of a mod

Manifest

Blocks

Entities

Custom Code

Logic

Mapper Types

Other useful information

Reference Values

XML File Documentation

Common

Blocks

Entities

Step-by-step guides

Starting a mod

Creating a block

Creating an entity

Creating a trigger

Creating an event

API Documentation

Modding

BlockScript

Configuration

DataType

EventProperty

EventProperty.Choice

EventProperty.Choice.Option

EventProperty.Icon

EventProperty.NumberInput

EventProperty.Picker

EventProperty.TeamButton
EventProperty.Text
EventProperty.TextInput
EventProperty.Toggle
Events
Game
Game.UI
Game.UI.BlockTab
GameMaterials
GameMaterials.BlockGhostMaterial
GameMaterials.BlockGhostShaders
GameMaterials.BlockMaterial
GameMaterials.BlockShaders
GameMaterials.EntityMaterial
GameMaterials.EntityShaders
GameMaterials.MiscMaterial
GameMaterials.MiscShaders
GameMaterials.ParticleMaterial
GameMaterials.ParticleShaders
GameMaterials.Shaders
GamePrefabs
GamePrefabs.ExplosionType
GamePrefabs.ProjectileType
Message
MessageType
ModAssetBundle
ModAudioClip
ModBlockBehaviour
ModConsole
ModEntryPoint
ModEvents
ModEvents.OnEventExecute
ModIO
ModKey
ModKeys
ModMesh
ModNetworking

ModResource

ModResource.ResourceType

Mods

ModTexture

ModTriggers

ModTriggers.OnTriggerChanged

ModUtility

Modding.Blocks

Block

BlockInfo

BlockPrefabInfo

PlayerMachine

PlayerMachineInfo

Modding.Common

Player

Modding.Levels

Entity

EntityBehaviour

EntityPrefabInfo

Level

LevelSetup

LogicChain

LogicEvent

Modding.Mapper

CustomMapperTypes

CustomSelector<T, TMapper>

MCustom<T>

SelectorElements

SelectorMaterials

Modding.Modules

BlockModule

BlockModuleBehaviour<TModule>

CustomModules

Modding.Modules.Official

ParticleHelper

ParticleHelper.CollisionSettings

ParticleHelper.CustomParticles

ParticleHelper.FireParticles
ParticleHelper.ParticleDefinition
ParticleHelper.ParticleSystemsInformation
ParticleHelper.SteamParticles
ParticleHelper.WaterParticles
ShootingModule
ShootingModule.CannonSound
ShootingModule.CrossbowSounds
ShootingModule.ModdedProjectileColliderComponent
ShootingModule.Projectile
ShootingModuleBehaviour
ShootingModuleSoundHolder
SpewingModule
SpewingModuleBehaviour
SpewingModuleFireParticleTrigger
SpinningModule
SpinningModuleBehaviour
SteeringModule
SteeringModuleBehaviour
Modding.Serialization
BoxModCollider
CanBeEmptyAttribute
CapsuleModCollider
CapsuleModCollider.CapsuleWrapper
Direction
DirectionExtensions
Element
IReloadable
IValidatable
MapperTypeDefinition
MapperTypeReference
MColourSliderDefinition
MColourSliderReference
MeshReference
MeshTexturePair
MKeyDefinition
MKeyReference

ModCollider
MSliderDefinition
MSliderReference
MToggleDefinition
MToggleReference
MValueDefinition
MValueReference
ReloadableAttribute
RequireToValidateAttribute
ResourceReference
SphereModCollider
TransformValues
VanillaBlockType
VanillaEntityType
Vector3

Introduction to Modding

Besiege provides a built-in mod loader that enables the creation of custom Blocks, Level Objects (Entities) and Logic Events, among others. This document contains an introduction to the basic concepts required for creating mods.

There are [step-by-step guides](#) to creating simple mods, these reference the full documentation that starts here as appropriate. These are probably sufficient for simple mods, but when creating more complex mods, it is advisable to at least skim the full documentation for the relevant topics.

For specific information on the various APIs available to mods, see the documentation for individual features like blocks and entities, the documentation about [common APIs](#), as well as the [full API reference](#).

There are some general tips&tricks to be found at [Developing Mods](#). It is recommended to read through those tips before starting mod development, and turn on log output as well as debug mode.

Please note that the mod loader allows a lot of freedom in what mods can do. They are almost unrestricted in what they can do to the game and there a lot of options that are included because they make the system more flexible but may annoy users when used. It's the responsibility of mod authors to write well-behaved mods that are enjoyable to use. Where applicable, the documentation will contain warnings when certain options may make the mod significantly less user-friendly.

Creating a mod

Get started creating a mod using the `createmod` console command.

This will create a project directory for your mod and a basic manifest file as described below.

Mod Structure

Each mod is made up of various files collected in one directory. The file that the mod loader looks up first is called the Mod Manifest file, and it must be named `Mod.xml`. It contains basic metadata as well as information about the various components of the mod. This is a minimal Mod manifest containing only the required elements:

```
<Mod>
  <Name>Example Mod</Name>
  <Author>Your Name</Author>
  <Version>0.0.1</Version>
  <Description>Some Description</Description>
  <Debug>true</Debug>
  <MultiplayerCompatible>true</MultiplayerCompatible>
</Mod>
```

This mod does not actually do anything, but it will be loaded by the mod loader without errors. For more information on the elements of the mod manifest, see [Manifest](#).

Note: Please keep the `Version` element up-to-date when you release new versions of a mod. It is responsible for controlling some required maintenance.

A Note on IDs in Mods

Many elements of a mod (including the mod itself) need to be assigned an ID to be uniquely identified. This includes for example blocks, entities and triggers.

These IDs are generally integers, the mod ID is GUID. The mod ID is generated automatically the first time the mod is loaded.

IDs except the mod ID only need to be unique within a mod itself, not among all mods. This means the mod loader handles separating two blocks with the ID "1" as long as they are from different mods.

IDs should not be changed after a mod was published. Doing so would break compatibility with all machines&levels created using earlier version of the mod.

In many cases, the game will also identify elements by some ID internally. Do not rely on any of these being the same IDs! Not only are they just generated based on the specified IDs, they may also change across different runs of the game or even while the game is running if a new mod is loaded.

Resource Handling

The mod loader is able to automatically load textures, meshes, sounds and Unity asset bundles.

To take advantage of this feature, resources need to be placed inside the `Resources/` folder. It is possible to create additional folders inside the Resources folder to create a more organized layout, this is up to the mod author.

There are two ways to use the Resource Handling system:

1. Let the mod loader handle everything: Some resources, like the mesh and texture of blocks and entities, are specified in the corresponding XML declarations. The mod loader will automatically load and apply these resources.
2. Access resources using the ModResource API: Any resources the mod loader does not handle automatically can be accessed using the ModResource API. The loading process is still handled by the mod loader but it is up to the mod to do something with these resources. See [Resources](#) for details.

All resources to be handled by this system need to be declared in the mod manifest, inside the Resources element:

```
<Mod>
  ...
  <Resources>
    <Mesh name="some-unique-identifier" path="test.obj" />
  </Resources>
</Mod>
```

The Resources element can contain `<Mesh>`, `<Texture>`, `<AudioClip>` and `<AssetBundle>` elements. They all follow the same pattern shown above.

The `name` attribute specifies an identifier by which the resource will be referenced everywhere else. This must be unique inside your mod, but the mod loader will handle separating resources with the same name by different mods.

The `path` attribute is the path to the resource file, relative to the `Resources/` directory.

Adding content to a mod

There is a number of things a mod can add to the game:

- [Blocks](#)
- [Entities \(Level Objects\)](#)
- [Custom Code](#)
- [Triggers and Events for the logic system](#)

Elements of the Mod manifest

There is a variety of elements that can be placed in the mod manifest. For a full reference, see [Mod Manifest](#).

Introduction to Modding

Besiege provides a built-in mod loader that enables the creation of custom Blocks, Level Objects (Entities) and Logic Events, among others. This document contains an introduction to the basic concepts required for creating mods.

There are [step-by-step guides](#) to creating simple mods, these reference the full documentation that starts here as appropriate. These are probably sufficient for simple mods, but when creating more complex mods, it is advisable to at least skim the full documentation for the relevant topics.

For specific information on the various APIs available to mods, see the documentation for individual features like blocks and entities, the documentation about [common APIs](#), as well as the [full API reference](#).

There are some general tips&tricks to be found at [Developing Mods](#). It is recommended to read through those tips before starting mod development, and turn on log output as well as debug mode.

Please note that the mod loader allows a lot of freedom in what mods can do. They are almost unrestricted in what they can do to the game and there a lot of options that are included because they make the system more flexible but may annoy users when used. It's the responsibility of mod authors to write well-behaved mods that are enjoyable to use. Where applicable, the documentation will contain warnings when certain options may make the mod significantly less user-friendly.

Creating a mod

Get started creating a mod using the `createmod` console command.

This will create a project directory for your mod and a basic manifest file as described below.

Mod Structure

Each mod is made up of various files collected in one directory. The file that the mod loader looks up first is called the Mod Manifest file, and it must be named `Mod.xml`. It contains basic metadata as well as information about the various components of the mod. This is a minimal Mod manifest containing only the required elements:

```
<Mod>
  <Name>Example Mod</Name>
  <Author>Your Name</Author>
  <Version>0.0.1</Version>
  <Description>Some Description</Description>
  <Debug>true</Debug>
  <MultiplayerCompatible>true</MultiplayerCompatible>
</Mod>
```

This mod does not actually do anything, but it will be loaded by the mod loader without errors. For more information on the elements of the mod manifest, see [Manifest](#).

Note: Please keep the `Version` element up-to-date when you release new versions of a mod. It is responsible for controlling some required maintenance.

A Note on IDs in Mods

Many elements of a mod (including the mod itself) need to be assigned an ID to be uniquely identified. This includes for example blocks, entities and triggers.

These IDs are generally integers, the mod ID is GUID. The mod ID is generated automatically the first time the mod is loaded.

IDs except the mod ID only need to be unique within a mod itself, not among all mods. This means the mod loader handles separating two blocks with the ID "1" as long as they are from different mods.

IDs should not be changed after a mod was published. Doing so would break compatibility with all machines&levels created using earlier version of the mod.

In many cases, the game will also identify elements by some ID internally. Do not rely on any of these being the same IDs! Not only are they just generated based on the specified IDs, they may also change across different runs of the game or even while the game is running if a new mod is loaded.

Resource Handling

The mod loader is able to automatically load textures, meshes, sounds and Unity asset bundles.

To take advantage of this feature, resources need to be placed inside the `Resources/` folder. It is possible to create additional folders inside the Resources folder to create a more organized layout, this is up to the mod author.

There are two ways to use the Resource Handling system:

1. Let the mod loader handle everything: Some resources, like the mesh and texture of blocks and entities, are specified in the corresponding XML declarations. The mod loader will automatically load and apply these resources.
2. Access resources using the ModResource API: Any resources the mod loader does not handle automatically can be accessed using the ModResource API. The loading process is still handled by the mod loader but it is up to the mod to do something with these resources. See [Resources](#) for details.

All resources to be handled by this system need to be declared in the mod manifest, inside the Resources element:

```
<Mod>
  ...
  <Resources>
    <Mesh name="some-unique-identifier" path="test.obj" />
  </Resources>
</Mod>
```

The Resources element can contain `<Mesh>`, `<Texture>`, `<AudioClip>` and `<AssetBundle>` elements. They all follow the same pattern shown above.

The `name` attribute specifies an identifier by which the resource will be referenced everywhere else. This must be unique inside your mod, but the mod loader will handle separating resources with the same name by different mods.

The `path` attribute is the path to the resource file, relative to the `Resources/` directory.

Adding content to a mod

There is a number of things a mod can add to the game:

- [Blocks](#)
- [Entities \(Level Objects\)](#)
- [Custom Code](#)
- [Triggers and Events for the logic system](#)

Elements of the Mod manifest

There is a variety of elements that can be placed in the mod manifest. For a full reference, see [Mod Manifest](#).

Developing Mods

This document aims to give some general workflow-related advice for modders, and it serves as documentation for some of the mod loader features that are related.

Project Layout

Manually installed mods are always placed in the `Mods` folder in `Besiege_Data`.

For relatively small or simple mods, it is possible to just directly place the mod there for developing it as well.

For most mods, one likely wants an extra directory to store additional resources like source code, Photoshop files, Blender files, source control, etc. Storing these inside the mod directory itself is not advised in order to prevent some issues, like them being uploaded to the Workshop along with the rest of the mod.

A recommended structure for these use-cases may look as follows:

```
<root mods folder> (wherever you want to keep your projects, this can be Besiege_Data/Mods)
|- SomeMod Project/ (keep this directory under source control)
|  |- SomeMod/ (this directory is what's loaded by the game)
|     |- Mod.xml
|     |- SomeBlock.xml
|     |- SomeAssembly.dll
|     |- Resources/
|     |- ...
|  |- src/
|     |- IDE Project file / build scripts / ...
|     |- Source files (possibly organized in more folders)
|- SomeOtherMod Project/
|- AThirdMod Project/
|- ...
```

This keeps all files required to develop the mod in one place. You can then set up your build environment such that the compiled assemblies are automatically copied to the mod folder, ideally using relative paths. This also makes sure the the mod is self-contained in one folder, which is useful for checking it into source control, for example for collaborating with other modders.

The last step is then to tell the game where to find the mod. There are two ways to do this:

- By launching it with a `-mod "<root mods folder>/SomeMod Project/SomeMod/` command line argument. You can set up a shortcut with all the command line options you need or set it up in Steam for example.
- By once executing `addmodsdire "<root mods folder>/SomeMod Project/` in the in-game console. This will make the game look for any mods in this directory permanently, until it is removed from the list again. This happens automatically if the mod is created using the `createmod` command.

Project Creation Console Commands

There are 4 console commands to make starting a new mod and adding content to it easier.

`createmod`

```
createmod "<name>" "[root mods folder]"
```

`createmod` always creates a mod according to the recommended structure described in the section above.

The second argument is optional, it corresponds to the `<root mods folder>` described above in the Project Layout section. It should be an absolute path on your computer. If the path contains spaces, place quotes (") around it.

If it is not specified, `Besiege_Data/Mods` is used as the default.

createblock and createentity

```
createblock "<modid | name>" "<name>"
```

```
createentity "<modid | name>" "<name>"
```

Adds a new block or entity to the specified name.

createassembly

```
createassembly "<modid | name>" <compiled | script> <assembly name> <default namespace> [noUnityTools | forceUnityTools]
```

Creates a new assembly and adds it to the mod.

Requires/assumes the recommended project structure, as used by `createmod`.

With `compiled`:

Creates (or updates) a solution and creates a project that can be opened using Visual Studio or MonoDevelop (tested with VS Community 2017 and MonoDevelop 5.9.6, should work with other versions).

The project contains references to appropriate game assemblies and a post-build action to copy the assembly to the mod directory and is set up in a portable way so that it can be checked in to source control and used from multiple machines.

Note: The references are set up using two environment variables that are set up when the game is first started with mods enabled.

If you have not restarted your computer since then, Visual Studio may not immediately be able to find the referenced assemblies. To fix this, restart Visual Studio and make sure to start it through the start menu and not by double-clicking the solution. Alternatively, just restart your computer once after executing the `createassembly` command and the references should work.

Regarding the `unityTools` arguments: For developing code to run in the game, it is helpful to have the Visual Studio Tools for Unity installed. They add a .NET Target Framework Profile that matches exactly what is available in Unity, thereby preventing you from attempting to use features that are not actually present in the runtime.

The `createassembly` command will attempt to detect whether the Unity Profile is available, and use it if it is. If it is not, it will print a warning and use the default profile instead.

The warning can be disabled by passing the `noUnityTools` argument. In that case, the default profile is always used.

If the game doesn't detect the Unity profile even though it is installed, the check can be bypassed by passing the `forceUnityTools` argument.

Debug Mode

Using the `Debug` element of the mod manifest, it is possible to turn on debug mode for a mod. It is generally recommended to turn on debug mode while developing a mod but turning it off for any public releases.

Turning it on will have three main effects:

- Some additional error messages and warnings when validating the XML files in the mod.
- Enables value reloading.
- Makes the mod loader act as if the mod version was changed every time the mod is loaded.

Value reloading describes a feature of the mod loader that allows a mod author to change the definitions of blocks and entities while the game is running. This doesn't work for all of the elements that are part of a block or entity definition, but among others, it is available for Colliders, Adding Points and the transforms of Mesh elements.

Reloaded values will be applied to the block/entity prefabs but also to any blocks and entities already placed.

Combined with turning on debug mode for the blocks or entities themselves (which is separate from the debug mode for the whole mod!) this is very useful for positioning these elements correctly. One can just adjust a value (or even add or remove a collider, etc. completely), save, and immediately check in-game if it had the desired effect.

Reloading is normally triggered automatically when a file inside the mod directory is changed, but in case this doesn't work it can also be triggered manually by pressing the Reload key, Ctrl+H by default.

The last point is to avoid unexpected behaviour while developing mods. A version change triggers various maintenance tasks that should be performed when the mod is changed. During development, these tasks should usually be performed every game restart (e.g. recompiling script assemblies, which are otherwise cached) and enabling Debug means it is not required to change the Version element every time while developing.

Log Output

When developing mods containing custom code, it is very useful to be able to print debug messages and instantly see them in-game.

The in-game console is normally only used for entering commands, but by setting the `show_logs` variable to `true` (enter `show_logs true`) printing all log output to it can be enabled. This will make all messages logged using the various `Debug.Log*` methods (from all mods and the game itself) as well as exceptions with stack traces appear in the console. Additionally, the frame number a message was logged in is also displayed.

If the same exception is thrown more than once without any other messages in between, it won't be logged repeatedly to combat the "exception-spam" problem otherwise frequently seen (a message noting that one or more messages have been omitted is shown instead, the full log can be seen in the log file if necessary).

It is generally recommended to keep this option turned on when developing mods in order to be notified of exceptions that are thrown.

Debugging Code

If you have access to a development build of the game, it is possible to use a real debugger on your mods code. Visual Studio with Unity Tools for Visual Studio has a "Debug->Attach Unity Debugger" menu option and a running development version of the game will appear in the window which is opened by that option.

On its own, the running game/debugger will not have access to the mod's source code, but the .pdb file that is generated by Visual Studio for debug builds of the mod can be converted into a .mdb file using Mono's pdb2mdb utility. Placing this .mdb file next to the .dll file in the mod's directory will allow the debugger to find the debug information.

Combined, this allows complete debugging of mod code using the Visual Studio debugger, including e.g. breakpoints and stepping through code.

Developing Custom Code

As mentioned in the documentation for [custom code](#), the compiled assemblies need to be located somewhere under the mod's root folder.

It is thus useful to automate copying them instead of having to do it manually every time.

Note: This is set up automatically when using the `createassembly` command.

Considering it is generally possible to develop custom code using any IDE that support .NET (or even no IDE at all), it is not possible to explain how to set this up for all possible scenarios. However, here is an example of how to set up a post-build action to do this in Visual Studio:

In the properties of the project, open the "Build Events" section. Entering the following command in the "post-build event

command line" box will copy the target assembly to the correct location (after replacing the placeholder with the correct path to the Besiege installation and mod name of course):

```
copy "$(TargetPath)" "<path-to-besiege>\Mods\<mod-name>\$(TargetFileName)"
```

If the mod is an open-source or otherwise collaborative project, it is advisable to set up the build event such that it does not contain machine/user-specific paths. A relatively easy way of doing this is requiring everybody who is to work on the project set up an environment variable (e.g. BESIEGE_LOCATION) that points to their Besiege installation:

```
copy "$(TargetPath)" "%BESIEGE_LOCATION%Besiege_Data\Mods\<mod-name>\$(TargetFileName)"
```

The Mod Manifest

The mod manifest is the central file of a mod. Every element of the mod needs to be declared inside it so the mod loader knows how to handle it. The file must be named `Mod.xml` and placed inside the root directory of the mod.

The root element of the file is the `Mod` element:

```
<Mod>
  ...
</Mod>
```

The remaining element are described below.

Basic Metadata

Name (String) *(required)*

```
<Name>Some Mod</Name>
```

The name of the mod.

Author (String) *(required)*

```
<Author>Your Name</Author>
```

The name of the author (or authors) that wrote the mod.

Version (Version) *(required)*

```
<Version>1.0.0</Version>
```

Version of the mod. Format: Major.Minor.Build, each component is an integer.

Please do change the version element when releasing new versions of a mod. This triggers some maintenance that should always be done when a new version of a mod is installed to prevent old data lingering.

Description (String) *(required)*

```
<Description>This is some mod that does something.</Description>
```

or

```
<Description>
This is some mod that also does something,
but it needs a few more words to say so.
</Description>
```

Short description of the mod. Displayed in the mod list. Should be no longer than two lines, otherwise it will be shortened in the list. Can be written on multiple lines, the game handles removing preceding and trailing whitespace.

Icon (Texture) *(optional)*

```
<Icon name="some-icon-texture-name" />
```

Square icon of the mod. Displayed in the mod list and, if `WorkshopThumbnail` is not set, used for the workshop when uploaded.

See [Common Elements](#) for a guide on how `Texture` elements are used.

WorkshopThumbnail (Texture) *(optional)*

```
<WorkshopThumbnail name="some-thumbnail-texture-name" />
```

Texture resource used as thumbnail when uploading the mod to the workshop. If this is not present, the Icon texture is used instead.

This can be set to have different icons in the mod list and in the workshop.

See [Common Elements](#) for a guide on how `Texture` elements are used.

MultiplayerCompatible (Boolean) *(required)*

```
<MultiplayerCompatible>True</MultiplayerCompatible>
```

Specifies whether the mod can be used in multiplayer sessions. Writing a mod to work correctly in multiplayer is more complex than writing one to only work in singleplayer, but singleplayer-only mods are more limiting to player of course. Once a singleplayer-only mod was loaded, the player must restart the game before a multiplayer session can be started/joined since the mod can't be unloaded.

LoadInTitleScreen *(optional)*

```
<LoadInTitleScreen />
```

Including this makes the mod be loaded immediately once the game is started. Without it, mods are only loaded once the player actually enters the game. Only use this if there is a good reason for it! It makes the mod significantly less player-friendly, as disabling it (for example to join a multiplayer session without it) will always require restarting the game.

LoadOrder (Integer) *(optional)*

```
<LoadOrder>1</LoadOrder>
```

Can be used to ensure a mod is loaded after another mod. The default value is 0 and mods are loaded in ascending load order, i.e. first mods with load order 0, then 1, etc.

If a mod has a dependency on another mod, use this to ensure it is loaded after the dependency.

Note that the LoadInTitleScreen element overrides the order specified using this, mods with that are always loaded before mods without it. The load order can still be used to control in which order multiple mods with LoadInTitleScreen are loaded.

Debug (Boolean) *(optional)*

```
<Debug>True</Debug>
```

Setting this to true will enable runtime reloading of various values in your mod. It will also provide additional warnings and error messages. For further information on this, see [Debug Mode](#).

Elements of a mod

Assemblies (Assembly[]) *(optional)*

```
<Assemblies>
  <Assembly path="SomeAssembly.dll" />
</Assemblies>
```

Assemblies containing custom code to be loaded into the game. See [Custom Code](#) for more information.

Blocks (Block[]) *(optional)*

```
<Blocks>
  <Block path="SomeBlock.xml" />
</Blocks>
```

Custom blocks that the mod adds to the game. See [Blocks](#) for more information.

Entities (Entity[]) *(optional)*

```
<Entities>
  <Entity path="SomeEntity.xml" />
</Entities>
```

Custom entities that the mod adds to the game. See [Entities](#) for more information.

Triggers (Trigger[]) *(optional)*

```
<Triggers>
  <Trigger>
    ...
  </Trigger>
</Triggers>
```

Custom triggers for the logic system that the mod adds to the game. See [Logic](#) for more information.

Events (Event[]) *(optional)*

```
<Events>
  <Event>
    ...
  </Event>
  <Event path="SomeEvent.xml" />
</Events>
```

Custom events for the logic system that the mod adds to the game. See [Logic](#) for more information.

Keys (Key[]) *(optional)*

```
<Keys>
  <Key name="some-key" defaultModifier="LeftControl" defaultTrigger="L" />
</Keys>
```

Key bindings to be managed by the mod loader. See [Keys](#) for more information.

Resources (Resource[]) *(optional)*

```
<Resources>
  <Mesh name="some-mesh" path="someMesh.obj" />
  <Texture name="some-texture" path="someTexture.png" />
  ...
</Resources>
```

Resources to be managed by the mod loader. See [Resource Handling](#) for more information.

ID (Guid)

The ID element is automatically generated when the mod is loaded the first time and inserted at the end of the manifest. It should not be changed manually.

Blocks

The mod loader provides a way to add custom block types to the game. Note: In this document (and much of the mod loader) "block" and "block type" are used interchangeably in the context of modded blocks.

There is a step-by-step [guide to creating blocks](#) available as well.

Creating a Block

Create blocks using the `createblock` console command. This will create a block XML file and set it up properly as described below.

Basics

Blocks are defined using an XML file containing a `Block` element. This file is referenced in the `Blocks` section of the `mod manifest`.

The manifest reference looks like this:

```
<Mod>
  ....
  <Blocks>
    <Block path="SomeBlock.xml" />
  </Blocks>
</Mod>
```

The `path` attribute is relative to the mod's directory, i.e. the directory that contains the `Mod.xml` file.

A template block XML file can be found at the bottom of this page. The same content is also in the XML file created by `createblock`.

For a full list and description of all the possible child elements of the `Block` element, see [Block element](#).

ID

Each block in a mod must be identified by an integer ID using the `ID` element of `Block`.

The ID must uniquely identify the block among all blocks in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).

It must only be unique in your mod though! The game automatically distinguishes between blocks of different mods.

Do note however, that changing this ID will break compatibility with machines that were saved using the old ID. Since the ID is internal and not user-facing, it is thus recommended to *never change these IDs* once the mod was first published.

Coordinate System

All child elements obviously take local coordinates, i.e. coordinates relative to the block's position and rotation. This local coordinate system is set up such that the Z axis always points "forward", i.e. away from the other block that the block itself was placed onto.

Adding Behaviour to Blocks

There are two ways of adding behaviour to modded blocks: By writing a custom BlockScript or by using pre-defined modules without writing any code.

These methods are not exclusive, a block can have both a custom script and modules attached.

BlockScript

The first step of using a custom BlockScript is loading [custom code](#) into the game. Then declare the BlockScript using the [Script child element](#) of `Block`.

```
<Block>
  ....
  <Script>SomeMod.Blocks.SomeBlock</Script>
</Block>
```

In this example, `SomeMod.Blocks.SomeBlock` is the fully qualified name of a class that extends `Modding.BlockScript`. This class must be defined in one of the assemblies listed in the mod manifest.

For information on the APIs provided by the `BlockScript` class and guidelines on writing block behaviour, see below.

Modules

[Modules](#) are sets of predefined behaviour that can be added to blocks using only their XML file. There are several [modules included in the game](#) by default and mods can also add new modules that can be used by other mods.

See the [Modules article](#) for more information on how to use modules.

Guidelines on writing BlockScripts

For a full list of the properties and methods provided by `BlockScript`, see [the API documentation](#).

Some general pointers on how to write `BlockScripts`:

- Call `AddKey`, `AddSlider`, ... in `SafeAwake`.

They need to always be called on each game instance as well as for both simulation and building clones.

- Use `SafeAwake` instead of `Awake`
- Be careful when overriding methods that are not explicitly marked as callbacks.

It is permissible to do so, however remember to call the corresponding base method, otherwise some of the other callbacks or convenience properties may stop working.

Additionally, when writing block scripts, special care needs to be taken when writing "lifecycle" method. While the normal Unity `Update()`, `FixedUpdate()`, etc. methods are available, their use is discouraged.

It is very important to think about what should be executed on what games instances and when, especially when writing a block that should also work in multiplayer.

In singleplayer, there is of course only one game instance that always handles all physics. This can be treated as a "host" instance.

In multiplayer, generally the host simulates all physics while the other instances (clients) don't and only receive simulation data from the host. The only exception is when a client is in local simulation mode. Then, only with respect to the blocks of that machine, that client acts as a host, while the regular host and all other clients act as clients.

As such, `BlockScript` provides methods to differentiate between updates during building mode and simulation mode. Simulation mode updates are also differentiated between updates on the instance simulating physics for that block (host) and all other instances in MP.

This leads to the following methods:

- `BuildingUpdate`: Called every `Update()` during build mode, on all instances.

- `SimulateUpdateHost`: Called every `Update` during simulation mode, on the instance acting as host for this block.
- `SimulateUpdateClient`: Called every `Update()` during simulation mode, on instances acting as clients for this block.
- `SimulateUpdateAlways`: Called every `Update()` during simulation mode, on all instances.

All of these functions also have `FixedUpdate()` and `LateUpdate()` versions.

Skins

Modded blocks can also be included in skin packs, just like regular blocks.

The mechanism works almost exactly the same way, there is just one difference: To alleviate conflicts between mods, the skin for the block must be identified (in the skin folder name) not by its in-game name (like for normal blocks) but by a combination of the mod ID and the ID assigned to the block within the mod.

For example, next to the `StartingBlock`, `Wheel`, etc. folders in a skin pack, there can also be a `a0e993c0-952e-4516-b661-20e4329ae5b2-0` folder, for the block with ID 0 (`-0`) in the mod with ID `a0e993c0-952e-4516-b661-20e4329ae5b2`.

This skin will then work exactly like for other blocks for players who have the mod installed but just ignored by the game if the mod is not installed.

Sample Block XML File

```
<Block>
  <!-- Block definition file.
    Optional elements are mostly out-commented.
    Remember to insert appropriate values where specified,
    the mod will not load correctly until you do.
    Restart the game to load the block once this file is completed.

    Values that should always be changed are marked with "TODO".

    See the documentation for further information on any of these elements.
  -->

  <!-- Optional. Enables debug mode.
    In debug mode, colliders and adding points are shown visually
    to assist in positioning them correctly.
    (Capsule colliders are shown as cubes, imagine their edges were rounded off.) -->
  <Debug>True</Debug>

  <!-- ID of your block. See "Note on IDs" in Mod.xml.
    The ID must be unique among blocks in your mod.
    It may conflict with blocks of other mods, the mod loader handles this.
    The easiest way of assigning IDs is to use 1, 2, 3, etc.-->
  <ID>%ID%</ID>

  <!-- Name of the block, shown in the user interface. -->
  <Name>%NAME%</Name>

  <!-- TODO: Change the mass to something appropriate -->
  <Mass>0.3</Mass>

  <!-- Additional keywords that can be used to search for this block
    in the search tab of the block bar.

    Blocks can always be searched for by name and author,
    additional keywords can be specified here. -->
  <!--<SearchKeywords>
    <Keyword>Some Keyword</Keyword>
  </SearchKeywords>-->
```

```

<!-- Optional.
    Only has an effect if the OnFlip method in the block script is not overridden.

    Causes the Flipped property for the script to be set correctly. This is also used by
    certain modules, like Spinning or Steering.
    If an Arrow element is included, it is automatically flipped too. -->
<!-- <CanFlip>true</CanFlip> -->

<!-- Specify that this block is a replacement of an old modded block.
    If this block has an equivalent that was created with the old community mod/block loader,
    specifying its id here will make the game load this block when loading machines that contain the old
block. -->
<!-- <Replaces>410</Replaces> -->

<!-- Normally, when a machine with a modded block is loaded, but that block is not loaded, the block will
be ignored.
    If the block has a fallback specified here, the fallback block is loaded instead in this scenario.

    Valid values are entries of the BlockType enum or the numeric ID of a block. Only normal blocks can
be specified as
    fallback, not modded blocks. -->
<!--<Fallback>DoubleWoodenBlock</Fallback>-->

<!-- <Script>Full Name of a BlockScript class, optional.</Script> -->

<!-- Blocks can have certain predefined behaviour added without any custom code.
    These behaviours are called modules.
    The Shooting, Spewing, Spinning, and Steering modules are included by default
    and mods can also add new modules.
    Check the documentation for more information on how to use modules. -->
<!--<Modules>

</Modules>-->

<!-- Include to make block take damage. -->
<!-- <Health>20</Health> -->

<!-- Optional.
    The game generates "stripped" versions of the prefab,
    these have some components and child objects removed and are used in MP where the full
    object is not always necessary.
    If you find that this stripping removes some components or child objects that you added to the
prefab manually
    and need on the stripped version, you can include a list of objects to keep using this. -->
<!-- <KeepWhenStripped>
    <Object>SomeObjectName</Object>
</KeepWhenStripped> -->

<!-- Include to enable block to burn.
    The Trigger element is optional. -->
<!-- <FireInteraction burnDuration="5">
    <SphereTrigger>
        <Position x="0" y="0" z="0.61" />
        <Radius>1.5</Radius>
    </SphereTrigger>
</FireInteraction> -->

<!-- Include to make block freezable. -->
<!-- <IceInteraction /> -->

<!-- Optionally specify type of damage done to entities.
    Can be one of "Blunt", "Sharp", "Fire" -->
<!-- <DamageType>Blunt</DamageType> -->

```

```

    <Mesh name="<!-- TODO: Insert mesh resource name here. -->" <!-- Must be defined as a resource in the
manifest. -->
        <!--<Position x="0.0" y="0.0" z="0.0" />
        <Rotation x="0.0" y="0.0" z="0.0" />
        <Scale x="1.0" y="1.0" z="1.0" /> -->
    </Mesh>

    <Texture name="<!-- TODO: Insert texture resource name here. -->" /> <!-- Must be defined as a resource
in the manifest. -->

    <Icon>
        <Position x="0.0" y="0.0" z="0.0" />
        <Rotation x="0.0" y="0.0" z="0.0" />
        <Scale x="1.0" y="1.0" z="1.0" />
    </Icon>

    <!-- Including this causes a direction arrow, like the one on wheels and other turnable blocks,
        to be displayed. The child elements define how and where it is displayed. -->
    <!--<Arrow>
        <Position x="0" y="0" z="0" />
        <Rotation x="0" y="0" z="0" />
        <Scale x="1" y="1" z="1" />
    </Arrow>-->

    <!-- Optional.
        Both child elements are optional.
        Hammer can be used to specify the position and rotation of the end of the nail at the start of the
hammer animation.
        Colliders can be used to specify a different set of colliders to use for the ghost.
        If it is not present, the colliders of the normal block will be used.
        It is also possible to specify ignoreForGhost attributes for some of the normal colliders to use the
normal set of
        colliders with a few of them removed on the ghost.
        If the Colliders element here is present, all ignoreForGhost attributes are ignored. -->
    <!-- <Ghost>
        <Hammer>
            <Position x="0" y="0" z="0.8" />
            <Rotation x="0" y="0" z="0" />
        </Hammer>
        <Colliders>
            <BoxCollider>
                <Position x="0.0" y="0.0" z="0.0" />
                <Rotation x="0.0" y="0.0" z="0.0" />
                <Scale x="1.0" y="1.0" z="1.0" />
            </BoxCollider>
        </Colliders>
    </Ghost> -->

    <Colliders>
        <!-- TODO: Insert Collider definitions here.
            Examples: -->
        <BoxCollider>
            <Position x="0.0" y="0.0" z="0.0" />
            <Rotation x="0.0" y="0.0" z="0.0" />
            <Scale x="1.0" y="1.0" z="1.0" />
        </BoxCollider>
        <SphereCollider>
            <Position x="0.0" y="0.0" z="0.0" />
            <Radius>1.0</Radius>
        </SphereCollider>
        <CapsuleCollider>
            <Position x="0.0" y="0.0" z="0.0" />
            <Rotation x="0.0" y="0.0" z="0.0" />
            <Capsule direction="X" radius="1.0" height="2.0" />
        </CapsuleCollider>
    </Colliders>

```

```
<BasePoint hasAddingPoint="false">
  <Stickiness enabled="true" radius="0.6" />
  <!-- Can only have motion if sticky -->
  <Motion x="false" y="false" z="false" /> <!-- Optional -->
</BasePoint>

<AddingPoints>
  <!-- TODO: Insert AddingPoint definitions here. Example:-->
  <AddingPoint>
    <Position x="0.0" y="0.0" z="0.0" />
    <Rotation x="0.0" y="0.0" z="0.0" />
    <Stickiness enabled="false" radius="0"/>
  </AddingPoint>
</AddingPoints>
</Block>
```

Modules

The block module system is a way to add behaviour to blocks without having to write any code.

There are some official modules included by default:

- Steering
- Spinning
- Spewing (Particle Systems)
- Shooting

Are you looking for the [documentation for these official modules?](#)

Additional modules can also be added by mods and then be used by other mods.

Using Modules

Blocks can contain a `Modules` element. Inside, one or more modules can be configured.

Note: While it is possible to add multiple modules to a block, not all modules are necessarily compatible with each other. Two modules that both try turning or rotating the block are unlikely to work well in tandem for example.

Here's an example for declaring an official module:

```
<Block>
  ...
  <Modules>
    <Steering>
      [Steering Parameters Here]
    </Steering>
  </Modules>
</Block>
```

Adding modules of other mods is similar:

```
<Block>
  ...
  <Modules>
    <SomeModdedModule
      modid="<id of mod that adds module here>" >
      [Module Parameters Here]
    </SomeModdedModule>
  </Modules>
</Block>
```

Note that you also need to ensure that the mod that adds the module is loaded before yours by setting an appropriate `LoadOrder` value in one (or both) of the mods.

Each module can define its own parameters, so look up the documentation for the respective module to find out how to configure it.

[Documentation for the official modules](#) is also available.

Mapper Types in Modules

Where modules use mapper types in the block mapper, it is usually necessary to first define them in the `ModuleMapperTypes` element of `Block`, for example:

```
<Block>
  <ModuleMapperTypes>
    <Key displayName="Some Name" key="unique-key" default="defaultKey" />
  </ModuleMapperTypes>
</Block>
```

Here's how to declare each of the available mapper types:

```
<!-- The valid values for default are entries of the UnityEngine.KeyCode enum.
      In the simplest case, this means a capital letter, like C -->
<Key displayName="Some Name" key="unique-key" default="defaultKey" />

<!-- unclamped is optional and false by default. If it is set to true, min and max only
      apply to the slider itself but it is possible to type in values outside of these bounds. -->
<Slider displayName="Some Name" key="unique-key" min="0.0" max="10.0"
        default="5.0" unclamped="false">

<Toggle displayName="Some Name" key="unique-key" default="false" />

<Value displayName="Some Name" key="unique-key" default="5.0" />

<ColourSlider displayName="Some Name" key="unique-key"
  r="1.0" g="1.0" b="1.0" snap="false" />
```

Then you can reference the key in a module like this (example from the Steering module):

```
<LeftKey key="unique-key" />
```

This system makes it possible to share sliders, keys, or toggles among modules, if they should be controlled simultaneously.

Writing custom Modules

There are two parts to a module: A class extending `BlockModule` that is used to deserialize the module and its parameters from the block XML file and a class extending `BlockModuleBehaviour<TModule>` that is attached to the block as a component.

The `BlockModule` class (hereafter called `TModule`) is basically a collection of properties that represent the parameters available to users of the module.

It must have an `[XmlAttribute("ModuleName")]` attribute, where `ModuleName` is the name that users of the module should specify in their block XML files.

For information on how to write the class so that it is correctly deserialized, see [Serialization](#).

The `BlockModuleBehaviour` class (hereafter called `TBehaviour`) is used to actually add behaviour to the block. It has access to the deserialized `TModule` via the `Module` property. Apart from that, it has basically the same API available to it as a `BlockScript`.

When both classes are created, the module needs to be registered by calling `CustomModules.AddBlockModule<TModule, TBehaviour>(name, canReload)`. `name` is the same name that is in the `XmlAttribute` attribute. `canReload` specifies whether the modules supports value reloading. See below for more information. This should be called during execution of the mod's `OnLoad` method.

When adding mapper types (other than keys) in modules, prefix the key with a relatively unique value, like the name of your mode, to prevent conflicts with other modules that may eventually be put on the same block by users.

Mapper Types in Custom Modules

To use the `ModuleMapperTypes` system described above, add a field like this to `TModule`:


```
[XmlElement, RequireToValidate]  
public MKeyReference MyKey;
```

Here, `MyKey` is also the name that will be used in the XML file. Other available classes are `MSliderReference`, `MToggleReference`, `MValueReference`, and `MColourSliderReference`.

Then, in `TBehaviour.SafeAwake`:

```
myKey = GetKey(Module.MyKey);
```

`GetKey` returns an `MKey`, just like the normal `AddKey`. Here it is assumed that the `myKey` variable/field was previously declared.

Resources in Custom Modules

While normally, a mod can only access its own resources, modules can also access the resource of the mod that declares the block.

Accessing those resources is similar to the key setup. First create an XML element of type `ResourceReference` or a `MeshReference`, with a `RequireToValidate` attribute.

(`MeshReference` has additional child elements for position, rotation and scale.)

Then, in the behaviour, `GetResource` can be called with the reference and returns a `ModResource` instance that can be cast depending on the resource type.

Reloading for Custom Modules

Custom modules can optionally support the value reloading system. This allows modders using the module to turn on `Debug` for their mod and edit some (or all) of the module's XML parameters during runtime and immediately see the effects in-game.

For details on how to implement reloading, see the [Reloading section in Serialization](#). There are two special cases to consider when implementing reloading for modules specifically: By extending `BlockModule`, your `TModule` type automatically implements `IReloadable`. The two methods of the interface are empty by default and declared in the base class, you can override them if you need them.

Additionally, you can also override `void OnReload()` in the `TBehaviour` class, since reloaded values should also be applied to existing blocks. If the behaviour just accesses the modules' values in methods like `Update`, this override is probably not necessary, but for some values certain actions may need to be taken, for example to update properties of other components that were previously created.

Lastly, tell the mod loader that the module supports reloading by setting the `canReload` argument of `CustomModules.AddBlockModule` to `true`.

Don't forget to document which values can be reloaded and which can't!

Official Modules

This document describes the configuration of all official block modules.

See [Modules](#) for general information about modules.

Steering Module

The steering module allows rotation the block along one axis controlled by user input, similar to the steering block.

Example:

```
<Modules>
  <Steering>
    <LeftKey key="left" />
    <RightKey key="right" />
    <SpeedSlider key="speed" />
    <AutomaticToggle key="automatic" />

    <Axis>Y</Axis>

    <MaxAngularSpeed>300</MaxAngularSpeed>
    <TargetAngleSpeed>0.8</TargetAngleSpeed>

    <HasLimits>True</HasLimits>

    <LimitsDisplay>
      <Position x="0" y="0" z="0" />
      <Rotation x="0" y="0" z="0" />
      <Scale x="0.5" y="0.5" z="0.5" />
    </LimitsDisplay>
    <LimitsDefaultMin>45</LimitsDefaultMin>
    <LimitsDefaultMax>45</LimitsDefaultMax>
    <LimitsHighestAngle>180</LimitsHighestAngle>
  </Steering>
</Modules>
```

Explanation of the available elements:

- `LeftKey`, `RightKey`: Keybindings to control the steering. References to keys that must be defined in the `ModuleMapperTypes` element.
- `SpeedSlider`: Defines the speed range possible. Reference to a slider that must be defined in the `ModuleMapperTypes` element.
- `AutomaticToggle`: Toggle to control whether automatic mode is active. Reference to a toggle that must be defined in the `ModuleMapperTypes` element. If an automatic mode is not desired, just set `showInMapper` to false on the toggle definition.
- `Axis`: Around which axis the block should rotate. Can be X, Y, or Z.
- `MaxAngularSpeed`: Limits how fast the block's Rigidbody can possibly rotate. See `Rigidbody.maxAngularVelocity` in the Unity documentation for more information.
- `TargetAngleSpeed`: Constant speed multiplier.
- `HasLimits`: Whether limits can be used with the block. The following elements are required if `HasLimits` is True, but are not necessary if `HasLimits` is False:
 - `LimitsDisplay`: A copy of the block's visuals is shown in the block mapper for the limits display. This defines how it's shown.
 - `LimitsDefaultMin`, `LimitsDefaultMax`: Default values for the limits.
 - `LimitsHighestAngle`: The highest angle a limit can be set to.

The following values are reloadable if the mod has Debug mode enabled: `MaxAngularSpeed`, `TargetAngleSpeed`, `LimitsHighestAngle`, and `LimitsDisplay`.

Spinning Module

Similarly to the Steering Module, the Spinning Module allows rotation of the block along one axis, controlled by the user. However the Spinning Module behaves like a wheel, rather than a steering hinge.

Example:

```
<Modules>
  <Spinning>
    <Forward key="forward" />
    <Backward key="backward" />

    <SpeedSlider key="speed" />
    <AccelerationSlider key="acceleration" />

    <AutomaticToggle key="automatic" />
    <ToggleModeToggle key="toggle-mode" />

    <Axis>Z</Axis>
    <MaxAngularSpeed>50</MaxAngularSpeed>
  </Spinning>
</Modules>
```

- **Forward**, **Backward**: Keybindings to control the block. References to keys that must be defined in the ModuleMapperTypes element.
- **SpeedSlider**: Defines the speed range possible. Reference to a slider that must be defined in the ModuleMapperTypes element.
- **AccelerationSlider**: Defines the possible accelerations. Reference to a slider that must be defined in the ModuleMapperTypes element. Infinity results in almost instant acceleration and deceleration.
- **AutomaticToggle**: Toggle mapper type for turning automatic mode on and off. Reference to a toggle that must be defined in the ModuleMapperTypes element.
- **ToggleModeToggle**: Toggle mapper type for turning toggle mode on and off. Reference to a toggle that must be defined in the ModuleMapperTypes element.
- **Axis**: The local axis that the block rotates around; X, Y, or Z.
- **MaxAngularSpeed**: Limits how fast the block's Rigidbody can possibly rotate. See Rigidbody.maxAngularVelocity in the Unity documentation for more information. Optional, default is 50.

Axis and **MaxAngularSpeed** are reloadable. If the Axis is changed, any wheels currently in simulation will not be able to rotate anymore until the simulation is restarted.

Shooting Module

The shooting module can be used to add cannon- or crossbow-like behaviour to a block.

Example:

```

<Modules>
  <Shooting>
    <Projectile>
      <Mesh name="mesh" />
      <Texture name="texture" />

      <Colliders>
        <SphereCollider>
          <Position x="0" y="0" z="0" />
          <Radius>1</Radius>
        </SphereCollider>
      </Colliders>

      <Mass>0.1</Mass>
      <Drag>0.0</Drag>
      <AngularDrag>5.0</AngularDrag>
      <IgnoreGravity>False</IgnoreGravity>

      <EntityDamage>150</EntityDamage>
      <BlockDamage>1</BlockDamage>

      <Attaches>false</Attaches>

      <FireInteraction ... />
    </Projectile>

    <FireKey key="fire" />
    <PowerSlider key="power" />
    <RateOfFireSlider key="rate-of-fire" />
    <HoldToShootToggle key="hold-to-shoot" />

    <ProjectileStart>
      <Position x="0.0" y="0.0" z="1.0" />
      <Rotation x="0.0" y="0.0" z="0.0" />
    </ProjectileStart>

    <ShowPlaceholderProjectile>False</ShowPlaceholderProjectile>

    <DefaultAmmo>5</DefaultAmmo>
    <AmmoType>Arrow</AmmoType>

    <ProjectilesExplode>False</ProjectilesExplode>

    <SupportsExplosionGodTool>False</SupportsExplosionGodTool>
    <ProjectilesDespawnImmediately>False</ProjectilesDespawnImmediately>
    <TriggeredByFire>False</TriggeredByFire>

    <RecoilMultiplier>0.1</RecoilMultiplier>

    <PoolSize>10</PoolSize>

    <Sounds>
      <CrossbowSounds />
      <CannonSound />
      <AudioClip name="some-resource-name" />
    </Sounds>
  </Shooting>
</Modules>

```

- **Projectile**: Defines the projectile that is shot.
 - **Mesh**: Mesh of the projectile. Resource defined in the manifest.
 - **Texture**: Texture of the projectile. Resource defined in the manifest.
 - **Colliders**: Colliders of the projectile. See [Colliders](#).

- **Mass**: Mass of the projectile.
 - **Drag**: Drag of the projectile. Optional, default: 0.
 - **AngularDrag**: Angular drag of the projectile. Optional, default: 5.
 - **IgnoreGravity**: Whether the projectile should ignore gravity.
 - **EntityDamage**: How much damage an entity takes on colliding with the projectile. Optional, default: 100.
 - **BlockDamage**: How much damage a block takes on colliding with the projectile. Optional, default: 1.
 - **Attaches**: Whether the projectile attaches to objects it hits. For context, the crossbow arrow attaches but the cannonball doesn't.
 - **FireInteraction**: Determines how the projectile is going to interact with fire. See [FireInteraction](#) for more information. Optional, default is no interaction with fire.
- **FireKey**: Key used for shooting. Reference to a key that must be defined in the ModuleMapperTypes element.
 - **PowerSlider**: Slider to define the power with which the projectile is shot. Reference to a slider that must be defined in the ModuleMapperTypes element.
 - **RateOfFireSlider**: Slider to define the rate of fire with which the projectile is shot. Reference to a slider that must be defined in the ModuleMapperTypes element.
 - **HoldToShootToggle**: Toggle to choose between turning automatic firing on/off or having to hold/press the key to fire. Reference to a toggle that must be defined in the ModuleMapperTypes element.
 - **ProjectileStart**: Defines where projectiles are spawned and in what direction they are shot. This is visualized by a metal spike pointing in the direction of shooting, if Debug is turned on on the block.
 - **ShowPlaceholderProjectile**: Whether to show a copy of the projectile during building and during simulation, while not shooting. The projectile will be shown exactly where the projectile will be spawned. Hint: The ExtralconObjects element of the Block element can be useful when this is used. Optional, default: False.
 - **DefaultAmmo**: How often the block can shoot without reloading.
 - **AmmoType**: What reload events the module should respond to. Module is always reloaded by the **All** type, this can be used to also respond to **Arrow**, **Cannon**, or **Fire**. Set this to **All** if the module should only respond to events with type All. Optional, default: All.
 - **ProjectilesExplode**: Whether projectiles explode on impact. Optional, default: False.
 - **SupportsExplosionGodTool**: Whether or not projectiles should explode on collision if the "Explosive Cannonballs" god tool is enabled. Optional, default: True.
 - **ProjectilesDespawnImmediately**: Whether projectiles immediately disappear after impacting. Optional, default: False.
 - **TriggeredByFire**: Whether the block should shoot automatically when it is on fire / being heated, the way that the cannon does. Optional, default: False.
 - **RecoilMultiplier**: The force applied to the projectile is multiplied by this factor before being applied as recoil to the block itself. That isn't entirely realistic (unless you set it to 1) but can be tweaked for better gameplay. Optional, default: 0.1.
 - **PoolSize**: Projectiles are pre-created and then the objects are reused. This defines how many projectile objects are available in singleplayer. In multiplayer, the size is fixed and the same for all projectile types. Optional, default: 10.
 - **Sounds**: Sounds to play when a projectile is fired. Optional, by default no sound is played. One of the sounds specified is chosen at random every time.

Possible elements: - **CrossbowSounds**, all the sounds that the crossbow uses. - **CannonSound**, the sound that is used by the cannon. - **AudioClip**, a reference to any AudioClip defined as a mod resource.

Everything except `ShowPlaceholderProjectile`, the mapper type references, the pool size and the sounds can be reloaded.

Spewing Module

The spewing module can be used to add particle systems to a block.

Each spewing module has a number of settings applying to all its particle systems, as well as one or more particle system, with associated settings each.

Global Settings

```
<Modules>
  <Spewing>
    <TriggerKey key="trigger" />
    <RangeSlider key="range" />
    <HoldToFireToggle key="hold-to-fire" />
    <ToggleTimeLimit>10</ToggleTimeLimit>
    <BaseAmmo>10</BaseAmmo>
    <AcceptFireAmmo>False</AcceptFireAmmo>
  </Spewing>
</Modules>
```

- `TriggerKey`: Key to trigger the particle systems. Reference to a key that must be defined in the `ModuleMapperTypes` element.
- `RangeSlider`: A slider definition for the range slider. Reference to a slider that must be defined in the `ModuleMapperTypes` element.
- `HoldToFireToggle`: A toggle definition for the Hold To Fire toggle. Reference to a toggle that must be defined in the `ModuleMapperTypes` element.
- `ToggleTimeLimit`: Optional, if specified and the module is not in hold to fire mode, the particle systems will automatically be turned off after the specified amount of time after toggling it on.
- `BaseAmmo`: How long the particles can be fired without reloading, in seconds.
- `AcceptFireAmmo`: If true, will reload using "Fire" and "All" ammunition. If false, will only reload using "All" ammunition. Default is false.

Defining Particle Systems

The definitions for the particle system are placed in the `ParticleSystems` element. There are three types of particles available: `Water`, `Fire`, `Steam`, and `Custom`.

This type just controls the visuals of the particles, not any actual behaviour with regards to fire, that can be configured separately as described below.

If the block has Debug mode enabled, the whole `ParticleSystems` element can be reloaded.

All particle types have some required options:

- `StartPosition`: Where, relative to the block, the particles are emitted.
- `Direction`: in which direction the particles are emitted.
- `StartSpeed`: Base speed of the particles when being emitted. The actual `startSpeed` property is computed as `StartSpeed * Range Slider Value + 0.5`.

For a `Custom` particle system, one more element is needed to specify the texture that should be used:

```
<Texture name="some-texture-resource-name" />.
```

In addition, it is possible to optionally specify these parameters:

- `LifetimeMultiplier`: Scales the lifetime of the particles. The default depends on the type of particle. This value acts as a multiplier on the respective default. The default of this is 1 accordingly.

- **DousesFire**: If included, a fire that collides with any particles will be extinguished. This requires the particle system to have the **Collisions** module enabled.
- **AddForce**: Adds extra force to particle collisions. This requires the particle system to have the **Collisions** module enabled.
- **Collisions**: By default, particle collisions are disabled. This can be used to turn them on and configure them.

Its parameters, as shown below, directly set the corresponding property in Unity.

The values shown below are the default values, and the same that the water particles of the normal water cannon use.

There are two ways of making a particle system ignite new fire:

- **FireTrigger**: A **BoxCollider** element, every burnable object entering the collider while the particles are on will be set on fire.
- **StartsFire**: An empty element, similar to **DousesFire**. Uses particle collisions to start fire, and requires the **Collisions** module to be enabled.

Which one to use depends on how the particle system should behave. The normal flamethrower uses the fire trigger method for example, because its fire does not have any collisions and passes through objects.

If you had something that bounces off of other objects, you should probably use the StartsFire method, otherwise objects that are not ever hit by the particles will get ignited.

Example for two particle systems, one using fire particles and one using water particles. For demonstration, the fire particles actually douse fire and the water particles ignite burnable blocks:

```
<Spewing>
  <ParticleSystems>
    <Fire>
      <StartPosition x="0" y="0" z="-0.5" />
      <Direction x="0" y="0" z="1" />
      <StartSpeed>2.0</StartSpeed>

      <RangeMultiplier>1</RangeMultiplier>
      <LifetimeMultiplier>2.0</LifetimeMultiplier>

      <AddForce>1.5</AddForce>
      <DousesFire />

      <Collisions>
        <Dampen>0.25</Dampen>
        <Bounce>0.6</Bounce>
        <LifetimeLoss>0.01</LifetimeLoss>
        <MinKillSpeed>0.01</MinKillSpeed>
        <MaxKillSpeed>10000</MaxKillSpeed>
        <RadiusScale>0.01</RadiusScale>
      </Collisions>
    </Fire>

    <Water>
      <StartPosition x="0.5" y="0" z="0" />
      <Direction x="1" y="0" z="0" />

      <RangeMultiplier>1</RangeMultiplier>

      <FireTrigger>
        <Position x="5" y="0" z="0" />
        <Rotation x="0" y="0" z="0" />
        <Scale x="9" y="1.5" z="1.5" />
      </FireTrigger>
    </Water>
  </ParticleSystems>
</Spewing>
```


Entities

The mod loader provides a way to add custom entity types to the game.

Note: In this document (and much of the mod loader) "entity" and "entity type" are used interchangeably in the context of modded entities.

There is a step-by-step [guide to creating entities](#) available as well.

Creating an Entity

Create entities using the `createentity` console command. This will create an entity XML file and set it up properly as described below.

Basics

Entities are defined using an XML file containing an `Entity` element. This file is referenced in the `Entity` section of the [mod manifest](#).

The manifest reference looks like this:

```
<Mod>
...
  <Entities>
    <Entity path="SomeEntity.xml" />
  </Entities>
</Mod>
```

The `path` attribute is relative to the mod's directory, i.e. the directory that contains the Mod.xml file.

A template entity XML file can be found at the bottom of this page. The same content is also in the XML file created by `createentity`.

For a full list and description of all the possible child elements of the `Entity` element, see [Entity element](#).

ID

Each entity in a mod must be identified by an integer ID using the `ID` element of `Entity`.

The ID must uniquely identify the entity among all entities in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).

It must only be unique in your mod though! The game automatically distinguishes between entities of different mods.

Do not however, that changing this ID will break compatibility with machines that were saved using the old ID. Since the ID is internal and not user-facing, it is thus recommended to *never change these IDs* once the mod was first published.

Logic

Closely related to entities is the logic system. Mods can also add trigger and event types, see [the Logic documentation](#) for details.

Sample Entity XML File

```
<Entity>
  <!-- Entity definition file.
    Optional elements are mostly out-commented.
    Remember to insert appropriate values where specified,
```

the mod will not load correctly until you do.
Restart the game to load the entity once this file is completed.

Values that should always be changed are marked with "TODO".

See the documentation for further information on any of these elements.

-->

<Name>%NAME%</Name>

<!-- ID of your entity. This ID must be unique among the entities that your mod adds.

(It can conflict with other mods, the mod loader will handle this.)

The easiest way of handling IDs is to just give your entities the IDs 1, 2, 3, etc. -->

<ID>%ID%</ID>

<!-- Optional. Enables debug mode.

In debug mode, colliders are shown visually to assist in positioning them correctly.

(Capsule colliders are shown as cubes, imagine their edges were rounded off.) -->

<Debug>true</Debug>

<!-- Normally, when a level with a modded entity is loaded, but that entity is not loaded, the entity will be ignored.

If the entity has a fallback specified here, the fallback entity is loaded instead in this scenario.

Valid values are numeric entity IDs or the name of an entity, as returned by the .Name property of the Entity class.

Only normal entities can be specified as fallbacks, not modded entities.

<!--<Fallback>Cottage</Fallback>-->

<Mesh name="<!-- TODO: Insert mesh resource name here. -->"> <!-- Must be defined as a resource in the manifest. -->

<!--<Position x="0.0" y="0.0" z="0.0" />

<Rotation x="0.0" y="0.0" z="0.0" />

<Scale x="1.0" y="1.0" z="1.0" />-->

</Mesh>

<Texture name="<!-- TODO: Insert texture resource name here. -->" /> <!-- Must be defined as a resource in the manifest. -->

<Icon name="<!-- TODO: Insert icon texture resource name here. -->" /> <!-- Must be defined as a resource in the manifest. -->

<!-- Optional. Placement offset. -->

<!-- <Offset x="0.0" y="0.0" z="0.0" /> -->

<Category><!-- TODO: Insert Category here. --></Category>

<Colliders>

<!-- Insert collider definitions here. Examples: -->

<BoxCollider>

<Position x="0.0" y="0.0" z="0.0" />

<Rotation x="0.0" y="0.0" z="0.0" />

<Scale x="1.0" y="1.0" z="1.0" />

</BoxCollider>

<SphereCollider>

<Position x="0.0" y="0.0" z="0.0" />

<Radius>1.0</Radius>

</SphereCollider>

<CapsuleCollider>

<Position x="0.0" y="0.0" z="0.0" />

<Rotation x="0.0" y="0.0" z="0.0" />

<Capsule direction="X" radius="1.0" height="2.0" />

</CapsuleCollider>

</Colliders>

<!-- Optional, this is the default -->

```

<!-- <Scale canScale="true" uniformScale="false" /> -->

<!-- Whether the entity can be picked in events that act on entities (e.g. Activate). -->
<!-- Optional, default is true -->
<!-- <CanPick>true</CanPick> -->

<!-- Whether to show the Physics toggle in the properties of the entity.
      Disable it, if your entity shouldn't/can't respond to any physics. -->
<!-- Optional, default is true -->
<!-- <ShowPhysicsToggle>true</ShowPhysicsToggle> -->

<!-- Include to enable entity to burn.
      The Trigger element is optional. -->
<!-- <FireInteraction burnDuration="5">
      <SphereTrigger>
        <Position x="0" y="0.8" z="0" />
        <Radius>1.5</Radius>
      </SphereTrigger>
</FireInteraction> -->

<!-- Include to make entity destructible.
      The BreakForce element is optional, the default is shown below.
      The Sound element is optional, the default is the sound of the Cottage breaking.
      Particles is used to specify the objects the entity breaks into when destroyed. -->
<!-- <Destructible forceToBreak="2">
      <BreakForce power="200" radius="6" />
      <Sound name="" />
      <Particles>
        <Particle>
          <Mesh name="" />
          <Colliders>

            </Colliders>
        </Particle>
        <Particle>
          <Mesh name="" />
          <Colliders>

            </Colliders>
        </Particle>
      </Particles>
</Destructible> -->

<!-- Include to manually specify available event triggers on your entity. -->
<!-- You should always include at least LevelStart, Activate, Deactivate and Variable.
      These are also applied by default, when no triggers are specified. -->
<!-- You can also specify that the entity is compatible with specific triggers of other mods. -->
<!-- <Triggers>
      <Trigger>LevelStart</Trigger>
      <Trigger>Activate</Trigger>
      <Trigger>Deactivate</Trigger>
      <Trigger>Destroy</Trigger>
      <Trigger>Ignite</Trigger>
      <Trigger>Variable</Trigger>
      <ModdedTrigger>
        <ModID>00000000-0000-0000-0000-000000000000</ModID>
        <LocalID>1</LocalID>
      </ModdedTrigger>
</Triggers> -->
</Entity>

```

Custom Code

It is possible to load custom code into the game using the mod loader. The mod loader provides various APIs to make interacting with the game easier. This guide assumes a basic familiarity with C# development.

There are two ways of loading code into the game: By referencing an already compiled DLL file or by distributing the source code and letting the game compile the code.

Both ways have advantages and disadvantages: Letting the game compile the code (this is then called a "Script Assembly") means that the only thing required to write code is a basic plain text editor, although an editor with syntax highlighting is recommended (e.g. Notepad++, Visual Studio Code, or Atom), and it's a bit easier to set up.

Compiling the code manually before passing it to the game requires more setup and larger/more complicated tools (usually Visual Studio, although there are other options, like MonoDevelop). There are significant benefits though, especially for larger projects. Most noticeably, a good autocompletion engine as well as immediate syntax and other error checking is invaluable for quickly writing a lot of code.

Setup

No matter which way of writing code you want to use, the assembly can be set up using the `createassembly` command.

When using compiled assemblies, it is recommended, but not required, to install the Visual Studio Tools for Unity using the Visual Studio installer. If you don't, pass the `noUnityTools` argument to `createassembly` (or ignore the warning it prints).

Especially when using a full IDE and compiled code reading the [Project Layout](#) section is also important.

Note: Always add all assemblies to the `Assemblies` element, even if they don't contain a `ModEntryPoint`. It is only possible to call various mod loader APIs from assemblies listed in the mod manifest.

Executing Code on Startup

If the code is only intended to add blocks behaviour to blocks, this may not be needed. For most mods, it will be required to execute some code whenever the mod is loaded.

This is done by creating a class that inherits from `Modding.ModEntryPoint` and overrides the `OnLoad` method.

Security

There are some restrictions on the available .NET framework classes and methods that are allowed. These are enforced by the mod loader when loading the assembly. This mostly concerns IO methods, as mods are not allowed to access files outside of their mod directory to protect players. Use the `ModIO` class for file IO inside the mod directory.

What is blacklisted:

- `System.IO`, `System.Net`, `System.Xml`, except for classes that cannot be constructed with arbitrary paths. (`Stream`, `{Binary, Text}{Writer, Reader}`, `MemoryStream`)
- `System.Reflection`, `System.Runtime.InteropServices`
- `System.Diagnostics`, `System.Security`, except for `Stopwatch` and `Cryptography`
- `Mono.CSharp`, `Mono.Cecil`, `Mono.CompilerServices`
- `System.CodeDom.Compiler`, `CSharpCompiler`, `IKVM`
- `UnityEngine.WWW`, `InternalModding`

The IO related classes and namespaces are replaced by the `ModIO` class and the resources system.

The other namespaces should not be needed by mods. Reflection can occasionally be useful for mods but cannot be allowed

because it could be used to circumvent all other restrictions.

APIs

There are various APIs provided by the mod loader to interact with the game. These are all contained in the `Modding` namespace.

It is also permissible to use internal game classes contained in other namespaces (or the global namespace). These are relatively nice to work with sometimes, but can also be rather obscure.

One exception is the `InternalModding` namespace which cannot be accessed by mods.

Only APIs in the `Modding` namespace are considered stable. Everything else may change without notice when the game updates, so try doing things with the `Modding` namespace as far as possible, but it's okay to access other classes when needed.

Important APIs

- The `Events` class provides C# events for various things that may happen in the game.
- The `ModNetworking` class provides an API to send messages across the network in multiplayer sessions.
- The `Modding.Common`, `Modding.Blocks` and `Modding.Entities` namespaces provide representation of many in-game objects like Blocks, Entities and Players.
- The `Configuration` class provides a way to store and access persistent data.
- The `ModIO` class provides file-access methods.
- The `ModKeys` class provides an API for keybindings that the user can rebind.

Some other APIs are described in the [API](#) document. Others are closely related to specific features of the mod loader and explained in the documentation for these features.

For concrete documentation explaining these classes and their methods, see [the API documentation](#).

Networking

The networking API can be used to make a mod multiplayer-compatible.

Note: It is possible to write multiplayer-compatible mods without using the networking API because certain things (for example placing blocks, triggering logic chains) are networked automatically by the game.

For many mods it may however be necessary to transmit additional information over the network connection in multiplayer. That is what the networking API is for.

Using the networking API is generally done in three steps:

- Creating message types
- Sending messages
- Acting on received messages

Message Types

Every message sent must conform to a pre-defined type. These are created using the `ModNetworking.CreateMessageType` function, which returns a `MessageType` object.

Message types are global within your mod and it is important that they are always created by the mod in the same order. The recommended way is to create them in `OnLoad` or in `OnPrefabCreation` and then store them in static variables to access throughout the mod.

Example of creating and storing a message type:

```
using Modding;
using Modding.Blocks;

public static class Messages {
    public static MessageType Example;
}

public class YourEntryPoint : ModEntryPoint {
    public override void OnLoad() {
        Messages.Example = ModNetworking.CreateMessageType(DataType.Single, DataType.Block);
    }
}
```

As you can see, you can pass a list of `DataType` values to the `CreateMessageType` method. This is necessary for sending any kind of data along with the message and determines what types of objects and in what order they can be sent and received.

DATATYPE	TYPE	OTHER ACCEPTED TYPES
Boolean	bool	
Color	UnityEngine.Color	
Integer	int	
IntegerArray	int[]	
Single	float	

DATATYPE	TYPE	OTHER ACCEPTED TYPES
SingleArray	float[]	
String	string	
StringArray	string[]	
ByteArray	byte[]	
Vector3	UnityEngine.Vector3	Modding.Serialization.Vector3
Block	Modding.Blocks.Block	BlockBehaviour
Entity	Modding.Levels.Entity	LevelEntity
Machine	Modding.Blocks.PlayerMachine	Machine

Types in the "Other accepted types" column can be passed when sending a message but regardless of what type was passed in, you will always receive objects from the "Type" column.

Every mod may register at most 255 message types.

Sending Messages

The first step of sending a message is to create a `Message` object. This is done by calling the `CreateMessage` method of a previously-created `MessageType`. The method must be passed one object for each of the `DataType`s passed to `CreateMessageType`. These must also appear in the same order as in `CreateMessageType`.

To send the created message, call the appropriate `Send` method in `ModNetworking`:

- `SendToAll(Message)` sends the given message to all connected instances.
- `SendTo(Player, Message)` sends the given message only to the given player.
- `SendInSimulation(Message)` sends the message to all instance participating in the current simulation. This may mean no player (not in simulation, in local simulation) or all players that are currently not in local simulation (in global simulation).

Continuing the example from above:

```
float x = <compute some float>;
Block b = <get some block>;

Message message = Messages.Example.CreateMessage(x, b);
ModNetworking.SendToAll(b);
```

Receiving Messages

To receive messages, a mod must register a callback to at least one of two events: `ModNetworking.MessageReceived` or `ModNetworking.Callbacks[MessageType]`. Both events require callbacks that take exactly one argument: a `Message` object.

The first event is fired for every message sent by the mod.

Using the second form, it is possible to register different callbacks for different message types. If `exampleType` was a `MessageType` instance, callbacks registered to `ModNetworking.Callbacks[exampleType]` will be called for every message with the given type that is received. (Registering callbacks works the same way it does for normal events, i.e.

```
ModNetworking.Callbacks[exampleType] += MyCallbackMethod; or += (msg) => {<code>;})
```

Either way, a callback receives a `Message` object.

The `Message` object exposes who sent the message with the `Sender` property (of type `Player`) and the data attached to it with the `GetData(int)` method.

Indices passed to `GetData` correspond to the position of the objects in the `CreateMessage` (or, equivalently, `CreateMessageType`) call.

Values are returned as `object`s but can simply be cast to the correct types.

Concluding our example:

```
public override void OnLoad() {
    Message.Example = ...; // see above

    ModNetworking.Callbacks[Messages.Example] += msg => {
        float x = (float) msg.GetData(0);
        Block b = (Block) msg.GetData(1);

        Debug.Log("Received a message from " + msg.Sender.Name
            + ", x = " + x
            + ", b = " + b.Guid + "!");
    };
}
```


Serialization

The mod loader needs to deserialize a variety of XML files to load mods.

In some cases, the API used to configure this deserialization may also be used by mods, e.g. when writing custom Block Modules.

This document describes how set up custom types to be deserialized by the system.

Basics

At its core, the mod loader uses the standard .NET `XmlSerializer` class/framework to implement deserialization, but with some added functionality.

Every type that can be deserialized should have either a `[Serializable]` or a `[XmlRoot]` attribute, and usually derive from `Modding.Serialization.Element` (this is automatically the case most of the time, e.g. a class inheriting `BlockModule` also inherits from `Element`).

Within a type that should be deserialized, every public member should at minimum have a `[XmlIgnore]`, `[XmlElement]`, or `[XmlAttribute]` attribute, depending on how it should be treated during deserialization.

The `XmlElement`, `XmlAttribute`, and `XmlRoot` attributes take an optional name parameter that can be used to specify what name the corresponding field/property/type should have in the XML files. If the parameter is not included, the name is based on the C# name of the element.

Validation

The `XmlSerializer` and the mod loader deserialization system provide some logic to validate that the modder or user provided XML files deserialize into a valid object.

This not only includes the XML syntax checked `XmlSerializer` but also some additional features to verify that the resulting object contains valid state, according to what data is valid for a specific type.

Ideally, the valid states can also be described using attributes, though it is also possible to encode more complex validation logic manually.

Validation-related attributes

By default, all `XmlElement`s and `XmlAttribute`s are assumed to be required and deserialization will not succeed when one or more are missing.

To mark an element or attribute as optional, apply a `[DefaultValue(x)]` attribute to it. (The one in `System`, not in `UnityEngine`.)

Important: The value passed to `DefaultValue` is not important! It is not automatically assigned if the value is missing. It also doesn't have to type-check, it is perfectly valid to pass `null` even for non-nullable types.

There are a few ways of handling optional values: A default value can be assigned in the constructor, this will stay if the element/attribute is not present but is overwritten if it is present.

Alternatively, for a field called `SomeField` that should be deserialized, a field `[XmlIgnore] public bool SomeFieldSpecified` can be added to the type. This field will be `true` if the value was included in the XML file and `false` if it was not.

Lastly, when including elements/attributes that also extend `Element`, adding a `[RequireToValidate]` attribute will cause them to be automatically validated too.

Manual validation logic

To manually validate state of your type that cannot be encoded using the attributes described, override the

`bool Validate(string elemName)` method in your class.

This method is called to validate the element after basic deserialization has occurred. It should return `true` if the object is valid, or `false` if it is not.

To run the default attribute-based checks in addition to your custom code, start the method with `if (!base.Validate(elemName)) return false;`.

Whenever possible, the `MissingElement`, `MissingAttribute`, and `InvalidData` methods should be used in an error case, this will ensure that error messages are printed and properly formatted, including line number and file name. These methods can be used like this:

```
if (<some condition>) {  
    // The condition above told us that the "Foo" element is missing.  
    return MissingElement(elemName, "Foo");  
}
```

Lists and Arrays

Lists and arrays require special attributes to deserialize. For basic information about how to deserialize these types, please see the documentation for the normal `XmlSerializer`.

In addition, by default an error is thrown when a list is specified but empty. Lists that can be empty should be marked with the `[CanBeEmpty]` attribute.

Reloading

The serialization system also has features to facilitate the "reloading" of XML files at runtime, i.e. applying changes in the XML file directly to the corresponding objects in-game.

This can be seen in action with, for example, colliders and adding points of blocks, which are reloaded if the Debug element of the mod is set to true.

Reloading can be enabled for your custom deserializable types too, but only when the underlying mod loader feature loading the XML file containing your type supports it. This is the case for modules, for example.

To enable reloading, mark your type with a `Modding.Serialization.Reloadable` attribute.

Then, also add a `Reloadable` attribute to all fields and properties whose values should be dynamically replaced when a reload happens.

Lastly, make your type implement `IReloadable`. This interface requires two methods: `void OnReload(IReloadable newObject)` and `void PreprocessForReloading()`.

In many cases, these can simply be kept empty, but sometimes it may be necessary to do some further processing in order to make reloading work correctly. This can be achieved using the two methods.

`OnReload` is called after the normal reloading has been performed and can be used if any additional values from the new object are needed that can't just be copied directly by the system.

`PreprocessForReloading` is called on the *new* object before the normal reloading process takes place. As the name suggests, it can be used to perform any preprocessing needed to correctly populate all values that should later be copied.

Common Elements

The mod loader contains some classes to deserialize values that are needed frequently.

Vector3

Unity's `Vector3` class does unfortunately not (de)serialize correctly with the system. Use the `Modding.Serialization.Vector3` class instead.

It has the same `x`, `y` and `z` components, but does not offer an additional functionality beyond that. It can however be cast to a Unity `Vector3`, and even supports implicit conversions, so a `Modding.Serialization.Vector3` can be assigned to a field/variable of type `UnityEngine.Vector3` and vice-versa without any explicit casts.

TransformValues

For cases where a position, rotation, and/or scale must be specified, instead of using 2 or more `Vector3`s, the `TransformValues` class can be used instead.

It's usage depends on what values are needed exactly, and whether any default values are present.

In the case that all 3 values are required, none are optional, just add the `TransformValues` field/property, an `XmlElement` attribute and a `RequireToValidate` attribute.

If there are any default values available, don't add the `RequireToValidate` attribute. Instead, override the `Validate` method in your type according to the instructions above.

Then, as custom validation logic, you can set default values on the object and ultimately call `Check` on it. An example is below:

```
protected override bool Validate(string elemName) {
    if (!base.Validate(elemName)) return false;

    SomeTransformValue
        .SetPositionDefault(new Vector3(0f, 0f, 0f))
        .SetRotationDefault(new Vector3(0f, 0f, 0f))
        .SetScaleDefault(new Vector3(0f, 0f, 0f));

    if (!SomeTransformValue.Check("SomeTransformValue")) return false;
}
```

It is possible to specify any combination of defaults, e.g. one for rotation and scale but none for position. The values without default will be treated as required to be specified by the user.

Additionally, the `HasNoScale()` method can also be called to specify that the object does not support a scale at all. In that case, a warning will be printed when a scale child element is included in the XML, but no error.

Lastly, there is a `SetOnTransform(Transform t)` method available to set all three (or, if `HasNoScale` was called, all two) values on a `Transform`. This will set the values as local coordinates, not world coordinates.

Direction

A simple enum that has possible values `X`, `Y`, and `Z` to avoid redefining this frequently.

Also provides two useful extension methods:

`ToAxisVector` returns a unit vector pointing in the direction given by the enum value.

`GetAxisComponent` takes an additional `Vector3` and returns the component corresponding to the enum value.

MapperTypes

There are classes available for defining `MKeys`, `MToggles`, `MSliders`, `MValues`, and `MColourSliders`. These are for example used the `ModuleMapperTypes` system.

The `key`, `displayName`, and `showInMapper` attributes are present on all mapper types, with `key` and `displayName` always being required and `showInMapper` being optional (true by default).

- `<Key displayName="Name" key="key" default="G" />`
- `<Slider displayName="Name" key="key" min="0.0" max="10.0" default="5.0" />` Additional optional attribute: `unclamped`, false by default. If unclamped is set to true, min and max only apply to the slider itself but it is possible to type in values outside of these bounds.
- `<Toggle displayName="Name" key="key" default="true" />`
- `<Value displayName="Name" key="key" default="5.0" />`
- `<ColourSlider displayName="Name" key="key" r="1.0" g="1.0" b="1.0" snap="false" />` `r`, `g`, and `b` define the default colour, an optional `a` alpha value can also be included. `snap` determines if any color should be allowed or if the slider to stick to certain predefined colors.

APIs

This document gives an overview of the various utility APIs the mod loader provides. It does not contain information about major elements, like [Blocks](#) or [Entities](#). The [networking API](#) is explained in its own document. It is also not a method-by-method reference, for that see [the API documentation](#).

Keys

The mod loader supports remappable key-bindings. When a mod supports keyboard shortcuts, it should use this API to handle them, this automatically makes the shortcuts more user-friendly by making them rebindable.

Note: As of now there is no UI for players to remap key-bindings. It is nevertheless recommended to use this API, as it still allows remapping keys through the config file and any mod using it will automatically take advantage of the UI when it is added.

Each key-binding needs to be declared in the mod manifest first:

```
<Mod>
  <Keys>
    <Key name="some-key" defaultModifier="LeftControl" defaultTrigger="L" />
  </Keys>
</Mod>
```

The `name` property must be unique among the keys of a single mod. The name may not contain the `|` character. The `defaultModifier` and `defaultTrigger` values may be any Unity `KeyCode`. See [the Unity documentation](#) for a full list.

`defaultModifier` may be omitted, the default value is `None`. `None` as a modifier value causes the modifier to be ignored. `None` as trigger value is equivalent to unbinding the key, it will never be triggered.

To access the keybindings in code, use the `ModKeys` API. Call `ModKeys.GetKey(name)` to get a reference to the specified key. If you plan on checking the key frequently (e.g. every frame), it is advisable to store the returned reference, as each call to `GetKey` requires two dictionary lookups.

The returned `ModKey` object provides the `IsDown`, `IsPressed` and `IsReleased` properties. These correspond to `GetKey`, `GetKeyDown` and `GetKeyUp` in Unity's `Input` class respectively.

Configuration

The `Configuration` API provides a way to store persistent configuration data. It is not mean for storing large amounts of data, please use normal file I/O for that (see [IO](#)). To use the `Configuration` API, call `Configuration.GetData()`. This returns an `XDataHolder` object, a simple key-value store that is also used to store custom information in levels and machines. Once retrieved, values can be set and read from the object freely. The configuration is automatically saved when the game exits, but is also possible to manually call `Configuration.Save()` to save it instantly.

IO

The game prevents mods from using most IO-related methods from the .NET framework to protect the files and computers of players. It is still possible to do some file IO using the `ModIO` class and whitelisted classes from the framework.

In modded code, it is permissible to use the `Stream`, `Text{Reader, Writer}` and `Binary{Reader, Writer}` classes. Using these on its own, it is not possible to open a file for reading or writing, so the `ModIO` class provides methods to do this.

The methods in `ModIO` are mostly analogous to those in the `File`, `Directory` and `WebClient` classes (in `System.IO` and `System.Net`) of the .NET framework. Parameters and usage are explained in the documentation for those.

There are two directories a mod is allowed to access: Its own root directory, as well as a data directory provided by the mod

loader.

The root directory access can be used to for example load additional resource files outside of the mod loader resource handling system.

The data directory should be used for storing additional persistent files. The reason for not using the mod root directory for this is that, when the mod is installed from the Steam workshop, it is possible that Steam may overwrite any additional files, especially when the mod is updated for example.

The important difference is that any paths taken as arguments by those methods are always interpreted relative to the mod's root directory (the one where the mod manifest is located) and no paths that traverse outside of it are allowed.

Additionally, the `WebClient` methods are implemented as static wrappers instead of exposing a `WebClient` object. Instead of setting events on the object, the static wrappers take (optional) event handler arguments for completion and progress events. It is also not allowed to access local files using the `Download*` methods, even if they are located in the mod directory.

Resources

The `ModResource` API can be used to easily load supported resource types. Resources to be loaded using this system need to be declared in the mod manifest file:

```
<Mod>
  ...
  <Resources>
    <Mesh name="some-name" path="someMesh.obj" />
  </Resources>
</Mod>
```

Available resource types are `Mesh`, `Texture`, `AudioClip`, and `AssetBundle`. The `name` must be unique within a mod, like an ID. The `path` is relative to the `Resources/` directory.

It is possible to manually access these resources using the `ModResource` class. There is a `ModResource.getX(string name)` method for each of the supported types, e.g. `GetMesh`, `GetTexture`, etc. These methods return a `ModX` (`ModMesh`, `ModTexture`, etc.) object, all of which inherit from the basic `ModResource`.

Note that returned textures are non-readable by default, specify `readable="true"` as attribute in the manifest `Resource` section for a texture you need to modify or read in code.

Along with accessors to get the loaded resource (in Unity's type for them), they have the following interface:

- `Name` (String): Returns the name of the resource.
- `Type` (ResourceType): Type of the resource.
- `Loaded` (Boolean): Indicates whether the resource has finished loading. Will also be true if an error occurred while loading the resource.
- `HasError` (Boolean): Indicates whether an error occurred while loading the resource.
- `Error` (String): If an error occurred, an error message.
- `Available` (Boolean): Like `Loaded` but false if an error occurred.
- `OnLoad` (event, Action): Event called when the resource has finished loading. Also called when an error occurs while loading the resource. If an event handler is registered to this event and the resource has already been loaded, the handler is immediately called.

In addition to that, the `ModResource` class also contains the static event `OnResourceLoaded` (`Action<ModResource>`). It behaves like the `OnLoad` event for a single resource but is called for all resources of a mod. It can be used to do generic error handling for example.

For one of the the most common use cases, `ModResources` also have a `SetOnObject` method. This can be used on all resource

types except `ModAssetBundles`.

It will set the resource on the given object once it has finished loading, thereby automatically handling the case that a resource may take a longer time to load than usual.

Additionally, it automatically handles the given object being instantiated as well. If the object is cloned by Unity while the resource has not finished loading, the resource will automatically also be set on all copies of the object once it has loaded.

Console

The `ModConsole` class allows mods to interact with the in-game console.

The `Log` method simply prints the given message to the console on its own line.

`RegisterCommand` can be used for custom console commands. It requires a command name, a handler and some help text.

For every command `someCommand` that is registered, 3 commands actually become available to the user: `someCommand`, `<modName>:someCommand`, and `<modId>:someCommand`.

This is done to allow different mods to add the same command. If this happens, the basic `someCommand` version will automatically be changed to only output some informational text and the user has to call `mod1:someCommand` or `mod2:someCommand` instead, depending on which one they want to call.

The command handler must be a method (or lambda, ...) taking a `string[]`. This array contains the arguments passed on the console. Arguments are separated by spaces, and quoting of an argument containing spaces is automatically handled.

Mods

The `Mods` class provides methods to query the state of other loaded mods.

Using this, it is possible to perform additional actions depending on the presence of other mods, adding integration between mods without creating a hard dependency.

When interacting with classes from assemblies of other mods, one needs to be very careful in order to not create a hard dependency. Make sure methods accessing other assemblies are only loaded/called if the mod is actually present.

Game

The `Game` class provides various methods to interact with the game. The `UI` subclass is documented below.

Simulation State

There are a variety of properties and methods to control the current simulation state: `IsSimulating`, `IsSimulating{Local,Global}`, `{Start,End}Simulation{Local,Global}()`.

These are pretty self-explanatory, note that it is possible to set the properties, this is equivalent to calling the appropriate methods. Calling the methods can throw exceptions for invalid operations, for example trying to start local sim while in global sim or vice-versa.

Additionally, `IsSetToLocalSim` and `ToggleLocalSimState()` control whether the game is currently set to use local simulation or join global simulation.

Multiverse Playlists

The `NextPlaylistLevel()` and `PreviousPlaylistLevel()` methods can be used to programmatically control the multiverse playlist mode. They have to be called on the host, and will just not do anything if no playlist is set up.

The `GetPlaylist()` method can be used to get the current playlist, as a list of paths to level files.

Multiverse Level Progress

The `GetCurrentProgress(MPTeam)` and `GetCurrentProgress()` methods can be used to check the level progress in a multiplayer game, as set by the logic system.

Useful raycasts

`BlockEntityRaycast` and `BlockEntityMouseRaycast` are utility functions to perform raycasts to detect only blocks and entities (using the `Game.BlockEntityLayerMask` mask).

There is one caveat that applies to blocks: The game uses stripped-down versions of blocks that are part of other players' machines on all clients (but not the host). These do not have any colliders and can thus not be detected using raycasts. If it is necessary to perform a raycast that can detect *all* blocks from a client, use the `ModNetworking` API to perform it on the host and transmit the result.

UI

The `UI` subclass provides methods for easily interacting with the game UI from code. For example, specific block tabs or level editor categories can be opened, or block types / level editor objects can be selected.

If the level editor-related methods are called when not in multiplayer, or any method is called while in a menu scene, nothing is done.

GamePrefabs and GameMaterials

The `GamePrefabs` class provides easy access to certain useful game objects that are frequently used in the game. Currently, these are projectiles as well as different explosions.

It contains two methods: `InstantiateExplosion` and `InstantiateProjectile`. Both spawn a `GameObject` of the given type and return it.

The `GameMaterials` class similarly provides access to the shaders used in the game, along with some sample Material setups that can then be customized as desired.

The shaders can simply be accessed in the static `GameMaterials.Shaders` class.

Materials can be accessed using the `GetMaterial` method. It returns *a copy* of the appropriate material specified.

Wrapper classes

The game generally represents objects such as blocks, entities, machines, or players using a variety of internal classes.

Since these are considered not stable and can change with any update to enable new game features or for optimization purposes, the `Common`, `Blocks`, and `Levels` namespaces in `Modding` provide wrapper classes to represent these objects that have a stable (and modder-friendly) API.

For the full documentation of each of these classes, see [Modding.Common](#), [Modding.Blocks](#), and [Modding.Levels](#).

ModUtility

The `ModUtility` class provides some miscellaneous useful methods.

See [the API documentation](#) for a list and detailed information.

Logic System

Mods can extend Besiege's level editor logic system in two ways:

- Adding new triggers
- Adding new events

There are step-by-step guides to creating [triggers](#) and [events](#) available as well.

Triggers

Adding Triggers

Triggers are defined in the mod manifest:

```
<Mod>
  ....
  <Triggers>
    <Trigger>
      <Name>Scream</Name>
      <ID>1</ID>
      <AvailableOn>
        <Entity>1000</Entity> <!-- Cottage -->
        <Entity>Windmill</Entity>
        <ModdedEntity>
          <ModID>00000000-0000-0000-0000-000000000000</ModID>
          <LocalID>0</LocalID>
        </ModdedEntity>
        <!--<AllOfficialEntities />-->
        <!--<AllModdedEntities />-->
      </AvailableOn>
    </Trigger>
  </Triggers>
</Mod>
```

- The `Name` element is displayed in the user interface when using the trigger.
- The `ID` element is used to uniquely identify the trigger within the mod. The ID must uniquely identify the trigger among all triggers in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).
- The `AvailableOn` element specifies what entities the trigger can be added to:
 - `<Entity>someId</Entity>` enables it on the vanilla entity with the given ID or name.
 - `<ModdedEntity>` enables it on a modded entity by specifying the mod's ID and the local ID of the entity within that mod. The `ModID` element may be omitted when the entity is in the same mod as the trigger.
 - `<AllOfficialEntities />` enables it on all vanilla entities.
 - `<AllModdedEntities />` enables it on all modded entities.

Activating Triggers

To activate the custom triggers, it is necessary to add [custom code](#) to the game. The `ModTriggers` class provides an API to interact with custom triggers.

There are two ways of activating triggers: 1) Activating all triggers of a specific type that currently exist in the level. 2) Activating triggers per-entity.

In either case, this will only work correctly when called on the instance running the simulation.

== Activating Triggers Globally Use the `ModTriggers.GetCallback(id)` method to get an `Action` that you can call to activate all instances of the trigger with the specified ID.

== Activating Triggers Per-Entity Create an `OnTriggerChanged` callback, which has the following signature:

```
void OnTriggerChanged(Entity entity, Action activate, bool removed). Then call  
ModTriggers.RegisterCallback(id, callback).
```

Your callback will now be called whenever the trigger with the specified ID is added to or removed from an entity:

- The first argument is the entity which was modified.
- The second argument is an `Action` which you can call to activate the trigger, similar to the one returned by `GetCallback` but only activating the trigger on that entity.
- The third argument indicates whether the trigger was added or removed from the entity. (`false` means added, `true` means removed).

With this approach you are responsible for storing these callbacks for when you need them.

Events

Adding Events

Events are also first declared in the mod manifest.

```
<Mod>  
  ....  
  <Events>  
    <Event>  
      <Name>Some Event</Name>  
      <ID>someId</ID>  
      <Icon name="some-icon" />  
      <Properties>  
        <NumberInput name="numberinput" title="Test" />  
      </Properties>  
    </Event>  
  </Events>  
</Mod>
```

The `Name` and `ID` elements work similarly to their Trigger equivalents. The `Icon` element specifies an icon to be displayed in the mapper when the event is selected.

Unlike Triggers, Events also support various values that can be entered by the player when using the event, these are called Event Properties and declared in the `Properties` element. See the [Event Properties](#) section for details.

Because the declaration for events with many properties can get quite long and complex, it is possible to instead define an event in its own XML file, similar to how Blocks and Entities work. To do so, reference the file in the mod manifest like this:

```
<Mod>  
  ...  
  <Events>  
    <Event path="SomeEvent.xml" />  
  </Events>  
</Mod>
```

The `path` attribute is relative to the mod's directory, i.e. the directory that contains the `Mod.xml` file. The `SomeEvent.xml` file would then have to contain the complete `Event` element as described above.

Event Properties

Events support custom "parameters" that can be set by the users. Think of the official vanilla events and you'll see that almost all of them have some values to be set by players when setting up their level. These are called Event Properties in the mod loader.

The mod loader provides several Event Property types that modded events can use:

- `TextInput`

- NumberInput
- Choice
- Toggle
- TeamButton

An event can contain any combination of these as required and multiple elements of the same type are permissible. In addition, there is the `Picker` element. When specified, the event will also contain a picker, however an event can contain at most one picker.

Lastly, the `Icon` and `Text` elements can be added to give additional information to the player, they are only displayed and otherwise ignored by the API.

Each Property must have a `name` attribute that uniquely identifies it among the properties of the same event. The `name` attribute may not contain `|` and `=` characters.

Apart from that, each property has various attributes that can be used to modify its behaviour and appearance. See [Event Properties](#) for a full reference.

Properties will generally be displayed in the order they are declared in, from top to bottom. It is also possible to group multiple elements in a `Row` XML element to make them appear on the same line. Elements inside a `Row` are required to have an `x` (float) attribute to specify their position. Elements outside `Row` elements are always centered automatically.

Adding Behaviour to Events

The `ModEvents` class contains an API to make your events actually do something.

The API consists of a single method: `ModEvents.RegisterCallback(int id, OnEventExecute callback)`

This method registers a callback that will be called whenever an event of the specified type is activated by the logic system. The `id` parameter corresponds to the `ID` element of the XML declaration.

`OnEventExecute` has the following signature:

`OnEventExecute(LogicChain logic, IDictionary<string, EventProperty> properties)`. `logic` is the LogicChain that the event is a part of. This also provides access to the entity on which the event is set up.

`properties` provides a way to access the values set up in the event: Every Event Property type has a corresponding class in `EventProperty` (e.g. `EventProperty.TextInput`, `EventProperty.Choice` etc.). These all inherit from the base `EventProperty` class.

Using the `properties` dictionary, you can access properties by the `name` defined in the XML declaration. You can then cast the resulting object to the appropriate class which provides ways of accessing the values set up by the player. Again, see [Event Properties](#) for a full reference.

Note that the `name` property is not only used for accessing the data from code, it's also used when saving and loading level files. This means that if you ever change one of the names, this will break compatibility with levels saved using an earlier version of the mod. Considering this and the fact that the names are purely internal and not user-facing, it is recommended to *never change the property names* in any mod that was already published.

Event Properties

This is the full reference for all event properties and the options available on them.

The `name` attribute is omitted here for brevity but still required on all properties (except `Text` and `Icon`), as described in [Logic](#).

Picker

```
<Picker mode="All" />
```

Allows the player to pick an object on the map.

- `mode` (`PickMode`) (*required*): What objects can be picked by the player. Can be `Entity`, `Zone`, `All`, or `None`.

TextInput

```
<TextInput title="Some Text" maxChars="8" default="test" />
```

A simple text input field.

- `title` (String) (*required*): Title to display next to the text field.
- `maxChars` (Integer) (*optional*): How many chars to allow in the text field. Default is 6
- `default` (String) (*optional*): Default text to display in the text field. Default is empty.

C#: `Modding.EventProperty.TextInput`

Access the entered text using the `Text` property. `Text` can also be set to a new value.

NumberInput

```
<NumberInput
  title="Some Number" maxChars="5"
  prefix="$" suffix="k"
  decimals="3" maxDecimals="3"
  splitThousands="false" negativeNumbers="false"
  maxValue="20" minValue="5" />
```

A simple number input field that supports numbers with or without decimals.

- `title` (String) (*required*): Title to display next to the number field.
- `maxChars` (Integer) (*optional*): How many chars to allow in number field. Default is 6
- `prefix` (String) (*optional*): A prefix to always display in front of the entered number. Default is empty.
- `suffix` (String) (*optional*): A suffix to always display behind the entered number. Default is empty.
- `decimals` (Integer) (*optional*): How many decimal places to display by default. Default is 2.
- `maxDecimals` (Integer) (*optional*): How many decimal places to allow maximally. Default is 2.
- `splitThousands` (Boolean) (*optional*): Whether to display a character to split thousands. Default is false.
- `negativeNumbers` (Boolean) (*optional*): Whether to allow negative numbers. Default is true.
- `maxValue` (Float) (*optional*): Largest value that may be entered. Default is +Infinity (i.e. no limit).
- `minValue` (Float) (*optional*): Smallest value that may be entered. Default is -Infinity (i.e. no limit).

C#: `Modding.EventProperty.NumberInput`

Access the entered number using the `Value` property. `Value` can also be set to a new value.

Choice

```
<Choice default="1">
  <Option index="0">Destroy</Option>
  <Option index="1">Deactivate</Option>
</Choice>
```

Allows the player to choose one of multiple options.

- `default` (Integer) (*optional*): Index of the option that should be selected by default. Default is 0.

`Option` child elements:

- `index` (Integer) (*optional*): If one `Option` element has an index specified, all options in the same `Choice` are also required to have it. In the absence of `index` attributes, the indices are determined by order. Options are displayed in the order in which they are listed, but they are saved using the indices. Manually specifying the indices allows preserving backwards-compatibility when adding new options or reordering them.

C#: `Modding.EventProperty.Choice`

A little care is required accessing the state of a `Choice` property: It exposes two relevant properties: `CurrentIndex` and `Options`.

`CurrentIndex` is only an index into the `Options` array, it does *not* correspond to the indices specified using the `index` attributes!

`Options` is an array of `Choice.Option` objects that represent each possible value. Each `Option` has a `Text` and an `Index` property. These are the indices from the XML declaration.

Toggle

```
<Toggle default="true">
  <Icon name="some-icon-texture-name" />
</Toggle>
```

A simple on-off toggle with an icon.

- `default` (Boolean) (*optional*): Default value of the toggle. Default is false.
- `Icon` (Texture) (*required*): What texture to display on the toggle. See [Common Elements](#) on how to use `Texture` elements.

C#: `Modding.EventProperty.Toggle`

Access the state of the toggle using the `Value` property. `Value` can also be set to a new value.

Text

```
Text fontSize="0.3">Some Text:</Text>
```

Displays some text in the properties window. Not editable by the player and not passed to the callback.

The `name` attribute can be omitted here because it is impossible to interact with a `Text`.

- `fontSize` (Float) (*optional*): Size of the text. Default is 0.15

C#: `Modding.EventProperty.Text`

Not exposed through the callbacks, not possible to interact with.

Icon

```
<Icon> <Icon name="some-icon-texture-name" /> </Icon>
```

Displays an icon in the properties window. Not editable by the player and not passed to the callback.

The `name` attribute can be omitted here because it is impossible to interact with an `Icon`.

- `Icon` (Texture) (*required*): What texture to display. See [Common Elements](#) on how to use `Texture` elements.

C#: `Modding.EventProperty.Icon`

Not exposed through the callbacks, not possible to interact with.

Custom Mapper Types

The different element types displayed in the block mapper (and for entities, `GenericDataHolder`'s etc.) are called mapper types. Using the `CustomMapperTypes` API it is possible to add new mapper types to the game.

To add a mapper type two elements are required: The type itself (extending `MCustom`, analogous to `MKey`, `MToggle`, etc.) and a `CustomSelector` that handles displaying the type in the block mapper.

After registering them, custom mapper types can be added to any `SaveableDataHolder` (Blocks, Entities, `GenericDataHolder`) using the `AddCustom` method.

Adding a mapper type is a somewhat complex topic and creating the interface properly can require a lot of work depending on the interface. It is not recommended for beginners or when it only provides marginal benefits over using a combination of the default mapper types.

There is a step-by-step [guide to creating a mapper type](#) available as well.

Creating a MapperType

The base class for all custom mapper types is `MCustom<T>`. The type parameter indicates what kind of value is edited by the mapper type, this can be a primitive type (like `string` or `int`) or a more complex type.

A mapper type contains three values: The default value, the current value and a "load value". The load value is basically used for the undo system and is normally handled automatically.

There are three things required in a class extending `MCustom<T>`: A constructor, `SerializeValue`, and `DeSerializeValue`.

Required overrides

The constructor usually has a signature similar to the default mapper types:

`string displayName, string key, T defaultValue`. Whether or not this structure is followed exactly, it is **required** to call the base constructor with the same signature.

The `XData SerializeValue(T value)` method is used to specify how a value of type `T` should be serialized into an `XData` object which is what is used for saving.

The `key` of the returned `XData` should be the `SerializationKey` property, not the `key` property!

Similarly, `T DeSerializeValue(XData data)` is used to deserialize an `XData` that was serialized by the `SerializeValue` method.

The easiest way of converting to and from an `XData` object is to convert whatever type is being edited to a primitive type supported by the `XData` system (if it is not already) and then convert to `XString`, `XInteger`, etc. (Note that `XStringArray`, etc. are also supported which may make serialization easier for more complex types.)

Both of these methods may be called from the base class constructor, before the constructor of the subclass is called. Make sure these methods are written such that they can handle this.

Overriding these methods is sufficient for value types (i.e. the value is changed by assigning a new one, not by changing fields of the value object). If the underlying type is not a value type, it is likely necessary to override some of the optional override below as well.

Optional overrides

The `Serialize()`, `SerializeDefault()`, `SerializeLoadValue()`, and `DeSerialize(XData)` methods are automatically implemented using the `SerializeValue` and `DeSerializeValue` methods. They can be overridden manually if necessary, but the default implementations should normally suffice.

The `MCustom` base class provides a default `Changed` event, which is always invoked from `InvokeChanged`. If desired, a subclass can add another event with a more appropriate signature. This should then also be called from `InvokeChanged`, but the base method has to be called from the override!

`ResetDefaults` should reset the `value` back to the default specified in the constructor. The default implementation does this by assigning `defaultValue` to `value` which is likely not correct for non-value types.

`ResetValue` and `ApplyValue` set `value` to `loadValue` or `loadValue` to `value` respectively. `ApplyValue` should also call `InvokeChanged` with the new value. The same reservations as for `ResetDefaults` apply.

If the underlying type is not a value type, the constructor should also do some additional work: The default constructor will set `value = loadValue = defaultValue` which is only correct for value types. If a reference type is used, the subclass needs to correctly create copies of the default value after the base constructor has run.

The `isDefaultValue` property has a default implementation comparing `Value` and `defaultValue` using the `.Equals()` method. If this is not appropriate or there is a more suitable comparison available, this can be overridden.

Creating the Selector

Each mapper type has an associated `Selector` class handling the interface in the block mapper. Custom selectors are created by extending the `CustomSelector<T, TMapper>` class, where `T` is the same type parameter as in `MCustom<T>` and `TMapper` is the mapper type (derived from `MCustom`) that this selector is associated to.

To define a selector, override `CreateInterface` and `UpdateInterface`.

Creating an interface

Writing code to create the interface can be somewhat bothersome because it requires correctly positioning and creating UI elements from code. The block mapper uses a UI system based on complete GameObjects instead of either the legacy Unity `GUI` system or the newer `Canvas`-based system.

The `CustomSelector` class provides some helpers to assist creating the UI by using the `Elements` property which has methods like `MakeText` or `MakeBox` to create elements and `AddButton` or `ScaleOnMouse` to add behaviour to elements that is consistent with the other game UI.

All UI objects should be put beneath the `Content` game object in the hierarchy. This is done automatically when using the helper methods from `Elements`.

Also in the interest of consistency, the `Materials` property provides access to materials used elsewhere in the block mapper which make it possible to create UI that fits in to the block mapper.

When creating an interface, also make sure to scale the `Background` element correctly. It determines the complete size of the selector interface, so it must have the correct size to ensure other elements don't overlap. The background scale should ideally be set as the first step in the interface creation, otherwise elements could appear offset.

The coordinate system in the interface is based directly on how Unity handles the elements: This means that (0, 0) is the middle point of the interface and the extent of the coordinate system is determined by the background scale.

Note that the positions given to the `Elements` helper methods also specify the middle point of elements, so that passing a position of (0, 0) will always center them in the selector. (This also applies to text, although the anchor properties of the text element can of course be changed on the returned object if desired.)

The UpdateInterface method

`UpdateInterface` is automatically called when the value of the underlying mapper type changes and should update the UI appropriately, for example by updating displayed texts.

Changing the mapper type's value

The `CustomMapperType` property can be used to access the underlying mapper type.

If the underlying type that is being edited is a value type (i.e. the value object should be swapped out instead of changing properties of it), it can be set using the `Value` property of the mapper type. Setting this will also automatically invoke the `Changed` event of the mapper type.

If the underlying type is edited by changing its properties, call the `InvokeChanged` method manually whenever the object is edited.

In either case, the `OnEdit` method of the `CustomSelector` must also be called every time the value is edited.

Registering the custom mapper type

Custom mapper types must be registered so the game knows about them. This is done by calling the `CustomMapperTypes.AddMapperType<T, TMapper, TSelector>()` method, passing the appropriate types that were created in the earlier steps. This should be called during execution of `OnLoad`.

Reference Values

This document contains some examples of values for various properties such as health or mass that need to be specified when creating blocks and entities.

Health and Mass

BLOCK	HEALTH	MASS
Starting Block	-	0.25
Double Wood	3	0.5
Single Wood	3	0.3
Wooden Pole	3	0.5
Log	3	1.0
Wheel	4	1.0
Large Wheel	4	1.0
Flying Block	2	0.5
Crossbow	4	0.5
Boulder	-	5.0

Break Forces

ENTITY	FORCE TO BREAK	BREAK POWER	BREAK FORCE RADIUS
Cottage	2.0	200	6
Windmill	7.0	200	2
Tent	2.0	200	6
Fence	2.0	200	6

Common XML Elements

Reference for various element types that are used in multiple different places.

Resource References

Mesh

```
<Mesh name="some-mesh-name">
  <Position x="0" y="0" z="0" />
  <Rotation x="0" y="0" z="0" />
  <Scale x="1" y="1" z="1" />
</Mesh>
```

The `name` attribute specifies what mesh to use. It must correspond to the `name` of a `Mesh` declaration in the `Resources` element of the mod manifest.

The child elements are all optional and can be used to adjust the mesh on a case-by-case basis.

See [Resource Handling](#) for more details.

Texture

```
<Texture name="some-texture-name" />
```

The `name` attributes specifies what texture to use. It must correspond to the `name` of a `Texture` declaration in the `Resources` element of the mod manifest.

See [Resource Handling](#) for more details.

Colliders

A `Collider` can be a `BoxCollider`, `SphereCollider`, or `CapsuleCollider` element. On all of these collider types, there are three optional attributes:

- `trigger` (Boolean): Make this collider a trigger.
- `layer` (Integer): Specify what layer to place this collider on.
- `ignoreForGhost` (Boolean): Don't include this collider on the ghost of a block. (Only valid on colliders inside `Block/Colliders`.)

They also all have an optional `Position` child element.

BoxCollider

```
<BoxCollider>
  <Rotation x="0" y="0" z="0" />
  <Scale x="1" y="1" z="1" />
</BoxCollider>
```

SphereCollider

```
<SphereCollider>
  <Radius>1</Radius>
</SphereCollider>
```

CapsuleCollider

```
<CapsuleCollider>
  <Rotation x="0" y="0" z="0" />
  <Capsule direction="X" radius="1.0" height="2.0" />
</CapsuleCollider>
```

The `Capsule` element specifies the shape of the capsule:

- `direction` can be `X`, `Y`, or `Z`
- `radius` and `height` determine the size of the collider

Note that when debug visuals are turned on, capsule colliders are displayed the same as box colliders due to internal restrictions. Just imagine rounded off corners and edges.

FireInteraction

```
<FireInteraction burnDuration="5" igniteOnStart="false">
  <SphereTrigger>
    <Position x="0" y="0" z="0.91" />
    <Radius>1.52</Radius>>
  </SphereTrigger>
</FireInteraction>
```

- `burnDuration` (Float): How long until the block/entity is burnt out.
- `igniteOnStart` (Bool) (*optional*): Whether the object starts on fire. Can be used to for example emulate the Fireball by automatically igniting and setting a very long burn duration. Default: False.
- `SphereTrigger` (*optional*): Manually specify the fire trigger. The fire trigger is a collider that determines how fire spreads from an object: When an object is burning, every other burnable object that is inside the fire trigger will also catch fire after a short time. The default trigger is shown above. You can also specify `BoxTrigger` but only one trigger at a time. The `layer` and `trigger` attributes have no effect here, they are always set automatically.

Block Properties

This is the full reference of elements that can be used inside the `Block` element to fully define a block.

Basic Metadata

ID

Integer, *required*

```
<ID>someNumber</ID>
```

Described in the introduction to blocks.

Note that the game uses some ID to identify each block type internally, this is *not* the same ID. In fact, the internal ID used to identify modded blocks may change across different runs of the game or even while the game is running, so don't rely on it for anything.

Name

String, *required*

```
<Name>Some Block Name</Name>
```

Name of the block, to be displayed in the user interface.

Search Keywords

String[], *optional*

```
<SearchKeywords>
  <Keyword>SomeKeyword</Keyword>
  ...
</SearchKeywords>
```

Manually specify keywords that can be used to find the block using the search feature. If not specified, players can still search for the name of the block.

Mass

Float, *required*

```
<Mass>2.0</Mass>
```

How much mass the block has, used for physics.

See [Reference Values](#) for example values.

CanFlip

Boolean, *optional*

```
<CanFlip>False</CanFlip>
```

Whether the block can be flipped with the F key. Assumed to be false if not specified.

If the block has an `Arrow`, it will automatically be flipped.

Modules that support flipping (e.g. the Steering module) automatically handle flipping.

If a custom block script is used, it can override the `OnFlip(bool playSound)` method to customize flipping behaviour. The method should always return `true` if any flipping was performed, this causes the flip state to be saved and the arrow to be flipped.

If `OnFlip` is overridden in script, the `CanFlip` element does not have any effect.

Replaces

Integer, *optional*

```
<Replaces>someId</Replaces>
```

If a block has an equivalent using the old community mod/block loader, this can be used to mark this as the replacement. This will make game load this block when trying to load a machine saved with the old block. This can only be used to replace old modded blocks, not official blocks or new modded blocks.

Fallback

BlockType, *optional*

```
<Fallback>DoubleWoodenBlock</Fallback>
```

Normally, when a machine with a modded block is loaded, but that block is not loaded, the block will be ignored. If the block has a fallback specified here, the fallback block is loaded instead in this scenario.

Valid values are entries of the BlockType enum or the numeric ID of a block. Only normal blocks can be specified as fallback, not modded blocks.

Health

Float, *optional*

```
<Health>20.0</Health>
```

How much damage the block can take before being destroyed. If this is not specified, the block can not be destroyed.

See [Reference Values](#) for example values.

Script

String, *optional*

```
<Script>SomeNamespace.SomeBlockScript</Script>
```

Behaviour of this block. Described in the introduction to blocks.

Visuals

Mesh

Mesh, *required*

```
<Mesh name="some-mesh-name" />
```

Mesh/Model of the block.

See [Common Elements](#) for a guide on how `Mesh` elements can be used.

Texture

Texture, *required*

```
<Texture name="some-texture-name" />
```

Texture of the block.

See [Common Elements](#) for a guide on how `Texture` elements can be used.

Icon

Transform, *required*

```
<Icon>
  <Position x="0" y="0" z="0" />
  <Rotation x="0" y="0" z="0" />
  <Scale x="1" y="1" z="1" />
</Icon>
```

Block icons in the bottom bar are automatically generated based on the block visuals, use this to position and scale the icon correctly.

ExtraIconObjects

Object[], *optional*

```
<ExtraIconObjects>
  <Object>
    <Mesh name="some-mesh">
      <Position x="0.0" y="0.0" z="0.0" />
      <Rotation x="0.0" y="0.0" z="0.0" />
      <Scale x="1.0" y="1.0" z="1.0" />
    </Mesh>
    <Texture name="some-texture" />
  </Object>
</ExtraIconObjects>
```

In addition to the normal block visual, it is possible to include additional objects in the block icon.

Arrow

Transform, *optional*

```
<Arrow>
  <Position x="0" y="0" z="0" />
  <Rotation x="0" y="0" z="0" />
  <Scale x="1" y="1" z="1" />
</Arrow>
```

If this element is included, an arrow visual (like the one on Wheels and other turnable blocks) will be displayed on the block.

The child elements control its position, rotation and scale.

In-Game Behaviour

Ghost

Ghost, *optional*

```

<Ghost>
  <Hammer>
    <Position x="0" y="0" z="0" />
    <Rotation x="0" y="0" z="0" />
  </Hammer>
  <Colliders>
    ...
  </Colliders>
</Ghost>

```

Can be used to override default ghost behaviour. If the `Ghost` element is included, it is possible to specify one or both of the child elements.

- `Hammer`: Specify position and rotation of the end of the nail at the start of the hammer animation that is played when the block is placed.
- `Colliders` (`Collider[]`): Manually specify another set of colliders for the ghost. If this is not present, the ghost will use the same colliders as the normal block. Alternatively, if the ghost colliders should be the same but with a few colliders removed, it is also possible to use the `ignoreForGhost` attribute on the colliders of the normal block. If this `Colliders` element is included, any `ignoreForGhost` attributes will have no effect.

Colliders

`Collider[]`, *required*

```

<Colliders>
  ...
</Colliders>

```

The colliders of the block. A list of `Collider` elements. See [Common Elements](#) for a guide on how to use `Collider` elements.

AddingPoints

`AddingPoint[]`, *required*

```

<AddingPoints>
  <AddingPoint>
    <Position x="0" y="0" z="0" />
    <Rotation x="0" y="0" z="0" />
    <Stickiness enabled="false" radius="0" />
  <AddingPoint>
    ...
  </AddingPoint>
</AddingPoints>

```

The adding points of the block. An adding point is basically a point onto which other blocks can be placed by the player. They have a position and rotation relative to the block itself. The `Stickiness` element is optional. It can be used to mark adding points as "sticky", this means that they will also create a connection to adding points of other blocks, even if they were not placed directly onto the point, if they are inside the specified radius.

BasePoint

`BasePoint`, *required*

```

<BasePoint hasAddingPoint="true" breakForce="6187.0">
  <Stickiness enabled="true" radius="0.6" />
  <Motion x="false" y="false" z="false" />
</BasePoint>

```

The base adding point of the block.

This is the adding point which is used to place the block. (I.e., the player clicks on an adding point of another block, then the block is placed such that the base point attaches to the clicked adding point.)

`hasAddingPoint` controls whether there is an adding point present at the base point. Should usually be true, to allow players to place a block back on the base point after deleting the block it was previously placed on.

`breakForce` controls how strong the connection is, it will break if a force or torque greater than the specified value is applied. Optional, the default is 6187.

`Stickiness` works the same way it does for a normal adding point.

In addition, the base point can also allow the block to be rotated around it, this is controlled using the `Motion` element. It can be used to create blocks that act like free hinges for example.

IceInteraction

`IceInteraction`, *optional*

```
<IceInteraction />
```

Including this element makes this block freezable.

FireInteraction

`FireInteraction`, *optional*

```
<FireInteraction burnDuration="5">
  ...
</FireInteraction>
```

Including this element makes it possible for this block to catch fire. See [Common Elements](#) for a guide on how to use `FireInteraction` elements.

DamageType

`DamageType`, *optional*

```
<DamageType>Blunt<DamageType>
```

Include to explicitly specify what kind of damage this block does when it damages AI. Can be `Blunt`, `Sharp`, or `Fire`.

Misc. Elements

Debug

`Boolean`, *optional*

```
<Debug>False</Debug>
```

Enables debug mode for this block. In debug mode, colliders and adding points are shown visually in-game to assist in placing them correctly.

KeepWhenStripped

`Object[]`, *optional*

```
<KeepWhenStripped>
  <Object>SomeObjectName</Object>
</KeepWhenStripped>
```

The game generates "stripped" versions of each block which have some components and child objects removed. These are used in MP where non-simulating clients do not need full versions of the blocks of other player's machines. If you find that this

stripping removes some components or child objects that you added to the prefab manually and need on the stripped version too, you can use this element to specify a list of objects that will be kept.

Entity Properties

This is the full reference of elements that can be used inside the `Entity` element to fully define an entity.

Basic Metadata

ID

Integer, *required*

```
<ID>someNumber</ID>
```

Described in the introduction to entities. Note that the game uses some ID to identify each block type internally, this is *not* the same ID. In fact, the internal ID used to identify modded entities may change across different runs of the game or even while the game is running, so don't rely on it for anything.

Name

String, *required*

```
<Name>Some Entity Name</Name>
```

Name of the entity, to be displayed in the user interface.

Category

Category, *required*

```
<Category>Virtual</Category>
```

In which category the entity should be displayed in the level editor UI. Available categories are `Buildings`, `Brick`, `Animals`, `Humans`, `Weaponry`, `EnvironmentFoliage`, `Primitives`, `Weather`, and `Virtual`.

Scale

Scale, *optional*

```
<Scale canScale="true" uniformScale="false" />
```

Specifies how the level editor scale tool should treat the entity. If not specified, the default is as in the example above.

CanPick

Boolean, *optional*

```
<CanPick>True</CanPick>
```

Specifies if the entity can be picked in events that act on entities (e.g. Activate). Default is `True`, only set this to `False` with a good reason as it may break player expectations.

ShowPhysicsToggle

Boolean, *optional*

```
<CanPick>True</CanPick>
```

Specifies whether to show the Physics toggle in the properties of the entity. Disable it if the entity can't/shouldn't respond to any physics. Default is `True`.

Offset

Vector3, *optional*

```
<Offset x="0" y="0" z="0" />
```

Can be used to specify a placement offset for the entity.

Fallback

EntityType, *optional*

```
<Fallback>Cottage</Fallback>
```

Normally, when a level with a modded entity is loaded, but that entity is not loaded, the entity will be ignored. If the entity has a fallback specified here, the fallback entity is loaded instead in this scenario.

Valid values are numeric entity IDs or the name of an entity, as returned by the .Name property of the Entity class. Only normal entities can be specified as fallbacks, not modded entities.

Visuals

Mesh

Mesh, *required*

```
<Mesh name="some-mesh-name" />
```

Mesh/Model of the entity. See [Common Elements](#) for a guide on how `Mesh` elements can be used.

Texture

Texture, *required*

```
<Texture name="some-texture-name" />
```

Texture of the entity. See [Common Elements](#) for a guide on how `Texture` elements can be used.

Icon

Texture, *required*

```
<Icon name="some-icon-texture-name" />
```

Unlike the 3D models used for blocks, the game uses pre-rendered textures for display in the level editor UI. Also unlike for blocks, the game is not currently able to generate these textures automatically. Use this to specify the texture to display.

`Icon` is a texture element, see [Common Elements](#) for a guide on these elements can be used.

In-Game Behaviour

Colliders

Collider[], *required*

```
<Colliders>...</Colliders>
```

The colliders of the entity. A list of `Collider` elements. See [Common Elements](#) for a guide on how to use `Collider` elements.

FireInteraction

FireInteraction, *optional*

```
<FireInteraction burnDuration="5">
  ...
</FireInteraction>
```

Including this element makes it possible for the entity to catch fire. See [Common Elements](#) for a guide on how to use `FireInteraction` elements.

Triggers

`Trigger[]`, *optional*

```
<Triggers>
  <Trigger>LevelStart</Trigger>
  <ModdedTrigger>
    <ModID>someModId</ModID>
    <LocalID>someId</LocalID>
  </ModdedTrigger>
</Triggers>
```

Include to manually specify what triggers can be used on the entity. The `Trigger` element is used to enable official triggers. Available triggers are `LevelStart`, `Activate`, `Deactivate`, `Destroy`, `Ignite`, `Variable`, `Death`, and `Explode`.

The `ModdedTrigger` element is used to enable additional modded triggers, by specifying the ID of the mod that adds them as well as the ID of the trigger within that mod. A modded trigger can be enabled either by including it here or by specifying the entity in the trigger declaration. Either one is enough to enable the trigger. It is possible to omit the `ModID` element when the trigger is in the same mod as the entity.

You should almost always include the `LevelStart`, `Activate`, `Deactivate` and `Variable` triggers, otherwise the player may feel unnecessarily limited. These are also the default if the `Triggers` element is not included.

Destructible

Include a `Destructible` element to make the entity destructible. The `Destructible` element has one attribute, `forceToBreak` (Integer) which specifies how much force is needed to break the entity. The `Destructible` element has various child elements to specify how the entity should break apart. These are described below.

See [Reference Values](#) for sample values for the various force involved.

BreakForce

`BreakForce`, *optional*

```
<BreakForce power="200" radius="6" />
```

Specify what force the the entity applies to surrounding objects when breaking. Default is 200, 6.

Sound

`Sound`, *optional*

```
<Sound name="some-breaking-sound-name" />
```

Specify what sound to play when entity breaks. Default is the sound of the cottage breaking.

Particles

`Particle[]`, *required*

```
<Particles>
  ...
</Particles>
```

A list of particles, which are the objects an entity breaks into when it is destroyed. Each `Particle` child element is defined as follows.

Particle

```
<Particle>
  <Mesh name="some-particle-mesh-name" />
  <Colliders> .... </Colliders>
</Particle>
```

Child elements are self explanatory. See [Common Elements](#) on how to use them. The particles will inherit the texture of the entity itself, so make sure this is reflected in the UV mapping of the meshes.

Misc. Elements

Debug

Boolean, *optional*

```
<Debug>False</Debug>
```

Enables debug mode for this entity. In debug mode, colliders are shown visually in-game to assist in placing them correctly.

Guides / Tutorials

Some guides that show how a mod for Besiege can be created. These are written in a tutorial-like fashion.

- [Starting a mod](#) - Step-by-step walkthrough to starting a new mod.
- [Creating a block](#) - Step-by-step walkthrough to creating a block.
- [Creating an entity](#) - Step-by-step walkthrough to creating an entity.
- [Creating a trigger](#) - Step-by-step walkthrough to creating a trigger.
- [Creating an event](#) - Step-by-step walkthrough to creating an event.

Creating a basic mod

This is a step-by-step tutorial to starting the development of a new mod, ending with a mod that can be loaded by the game but does not have any content.

For complete documentation of the different concepts involved in creating mods, see [Introduction to Modding](#).

createmod

The recommended way of starting a new mod is using the `createmod` console command. That means starting up the game, opening the console and executing `createmod "<name of the mod>"`.

A second parameter can optionally be given to the command, to store the mod outside of the `Besiege_Data/Mods` directory, see the documentation for the command for details.

This will create a directory for your mod in `Besiege_Data/Mods` (or another directory, if specified). The directory has one subdirectory with a `Mod.xml` file in it.

This file is called the mod manifest and it contains all the important information required to load the mod. You'll insert some appropriate values in it in the next section.

The directory containing the manifest is the only directory the game considers as "part" of the mod. All files the mod needs in order to be loaded must be in there, and paths in the mod loader are usually relative to this directory.

The directory above that one can be used to store additional files belonging to the mod that do not concern the game directly, like documentation for users, source code or files from Blender, Maya, Photoshop etc.

Basic Manifest Elements

Open up the Mod.xml file. It has already been populated with all the important elements and examples for how they are used.

At the very least edit `Author` and `Description`. You should also read through the rest of the file to see what options are available.

Note: By default, a mod is assumed to work correctly in multiplayer after it was created. If you intend your mod for singleplayer use only, change the `MultiplayerCompatible` element appropriately.

Other very useful or relevant elements are the `Debug` and `Icon` elements.

See [the mod manifest documentation](#) for a full list of available elements.

Creating a block

This is a step-by-step tutorial to adding a new block to the game. It assumes that the mod that should contain the block already exists. (See [Creating a mod](#) if this is not yet the case.)

It also assumes that the visuals (mesh and texture) already exist.

For complete documentation on creating blocks, see [Blocks](#).

Getting Started

A block is defined using an XML file within the mod's directory.

Create that file using the `createblock` console command, by starting the game, opening the console, and executing `createblock "<name of the mod>" "<name of the block>"`.

This will create a basic block file and add a reference to it to the manifest.

Defining the block's properties

Assigning an ID

The ID must uniquely identify the block among all blocks in the same mod; the easiest way to handle this is to just use sequential ID (1, 2, 3, ...).

```
<ID>1</ID>
```

Some basic properties

```
<Mass>2.0</Mass>
<Health>23.0</Health>
```

Visuals

The visuals of a block are defined using the `Mesh` and `Texture` elements. They are used with the resource system, which means they first need to be defined in the mod manifest.

- Add the mesh and texture to the mod manifest.

```
<Mod>
...
<Resources>
  <Mesh name="some-block-mesh" path="SomeBlock.obj" />
  <Texture name="some-block-tex" path="SomeBlock.png" />
</Resources>
</Mod>
```

- Reference the mesh and texture in the block file.

```
<Block>
...
<Mesh name="some-block-mesh" />
<Texture name="some-block-tex" />
</Block>
```

Remember that resource paths are always relative to the `Resources/` directory, not the mod directory itself.

For more details on how the resource system works, see [Resource Handling](#).

There is an additional option to transform the mesh in case it is incorrectly positioned, rotated or scaled, see [Mesh](#).

Colliders and Adding Points

Every block needs to have colliders for physics, as well as a base adding point which is used to place it on other blocks.

Depending on the block, it may also be desirable to enable other blocks to be placed on it, to do this, additional adding points need to be defined.

Before spending time on correctly placing colliders and adding points, read up on the `Debug` elements of both [the mod manifest](#) and [the block itself](#). Debug mode is very useful for quickly adjusting colliders and adding points.

- Define the block's colliders

```
<Block>
...
<Colliders>
  <BoxCollider>...</BoxCollider>
  <SphereCollider>...</SphereCollider>
  ...
</Colliders>
</Block>
```

For more information on the collider shapes available and how to use them, see [Colliders](#).

- Define the block's base adding point

```
<Block>
...
<BasePoint hasAddingPoint="true">
  <Stickiness enabled="true" radius="0.5" />
  <Motion x="false" y="false" z="false" />
</BasePoint>
</Block>
```

For more information on how the base point works and its properties, see [Base Point](#).

- Define any additional adding points

```
<Block>
...
<AddingPoints>
  <AddingPoint>
    <Position ... />
    <Rotation ... />
  </AddingPoint>
  ...
</AddingPoints>
</Block>
```

These are where other blocks can be attached. For more information, see [Adding Points](#).

Other elements

There is a variety of other elements to define specifics of how the block interacts with the game, for example `DamageType`, `FireInteraction`, or `IceInteraction`. A full list can be found on the [reference page for the Block element](#).

Making the block do something

It is possible to create custom behaviour for modded blocks, please see the page about [custom code](#) and the [introduction to](#)

[blocks](#) for more information.

Creating an entity

This is a step-by-step tutorial to adding a new entity to the game. It assumes that the mod that should contain the entity already exists. (See [Creating a mod](#) if this is not yet the case.)

It also assumes that the visuals (mesh and texture) already exist.

For complete documentation on creating entities, see [Entities](#).

Getting Started

An entity is defined using an XML file within the mod's directory.

Create that file using the `createentity` console command, by starting up the game, opening the console, and executing `createentity "<mod name>" "<entity name>"`.

This will create a basic entity file and add a reference to it to the manifest.

Defining the entity's basic properties

The various elements of the entity's XML file are used to define how the entity behaves and looks in-game. For a full reference, see [Elements of an Entity](#).

Assigning an ID

The ID must uniquely identify the entity among all entities in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).

```
<ID>1</ID>
```

Some basic properties

```
<Category>Buildings</Category>
<Scale canScale="true" uniformScale"false" />
```

These are just examples, there are more properties that can be used.

Visuals

The visuals of an entity are defined using the `Mesh` and `Texture` elements. In addition, mods also need to supply an image to be used as the entity's icon in the level editor UI. These are all used with the resource system, which means they first need to be defined in the mod manifest.

- Add the mesh and texture to the mod manifest.

```
<Mod>
  ...
  <Resources>
    <Mesh name="some-entity-mesh" path="SomeEntity.obj" />
    <Texture name="some-entity-tex" path="SomeEntity.png" />
    <Texture name="some-entity-icon" path="SomeEntity.png" />
  </Resources>
</Mod>
```

- Reference the mesh and texture in the entity file.

```
<Entity>
  ...
  <Mesh name="some-entity-mesh" />
  <Texture name="some-entity-tex" />
  <Icon name="some-entity-icon" />
</Entity>
```

Remember that resource paths are always relative to the `Resources/` directory, not the mod directory itself.

For more details on how the resource system works, see [Resource Handling](#).

There is an additional option to transform the mesh in case it is incorrectly positioned, rotated or scaled, see [Mesh](#).

Colliders

Every entity needs to have colliders for physics.

Before spending time correctly positioning colliders, read up on the `Debug` elements of both [the mod manifest](#) and [the entity itself](#). Debug mode is very useful for quickly and accurately adjusting colliders.

- Define the entity's colliders

```
<Entity>
  ...
  <Colliders>
    <BoxCollider>...</BoxCollider>
    <SphereCollider>...</SphereCollider>
    ...
  </Colliders>
</Entity>
```

For more information on the collider shapes available and how to use them, see [Colliders](#).

Other elements

There are many more elements that can be used to define specifics of how the entity behaves, for example `FireInteraction`, `Offset`, or `Destructible`. A full list can be found on the [reference page for the Entity element](#).

Creating a Trigger

This is a step-by-step tutorial to adding a new trigger for the level editor logic system to the game. It assumes that the mod that should contain the trigger already exists. (See [Creating a mod](#) if this is not yet the case.)

Note that to create a useful trigger (i.e. one that can be activated) requires writing some code that determines when it will be activated. The specific APIs used to do this are discussed here, but the basics of loading code into the game or how to interact with the game world are not.

See [Custom Code](#) for information on how to add code to the game and the documentation of Unity and the `Modding.{Blocks, Levels, Common}` namespaces for information on how to interact with the game world.

Getting Started

Triggers are defined in the mod manifest, within the Triggers section.

- Add the Trigger element to the mod manifest.

```
<Mod>
  ...
  <Triggers>
    <Trigger>
      ...
    </Trigger>
  </Triggers>
</Mod>
```

- Define the name and ID.

There are only 3 elements used to define the trigger. The first two are its name and its ID. The ID must uniquely identify the trigger among all triggers in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).

```
<Trigger>
  <Name>Some Trigger</Name>
  <ID>1</ID>
</Trigger>
```

Usable Entities

Not every trigger is available on every entity. An example from the game itself is the "Ignite" trigger which is only available on entities that can be set on fire.

There are two ways of specifying that a specific modded trigger is available on a specific entity: Either in the trigger definition or in the entity definition. In the following the trigger definition is discussed, for the entity definition see [Triggers](#).

- Define the AvailableOn element.

The `AvailableOn` element is used to specify what entities the trigger is available on. There are 4 forms of specifying entities, all are included and discussed below.

```
<Trigger>
  ...
  <AvailableOn>
    <Entity>1000</Entity>
    <ModdedEntity>
      <ModID>1283901230</ModID>
      <LocalID>1</LocalID>
    </ModdedEntity>
    <AllOfficialEntities />
    <AllModdedEntities />
  </AvailableOn>
</Trigger>
```

The `Entity` element can be used to add the trigger to specific official entities.

The `ModdedEntity` element can be used to add the trigger to specific modded entities. Modded entities are identified by the ID of the mod and the entity's ID within the mod. Note: This does not automatically form a dependency on the other mod, the game just ignores the element if the referenced mod is not loaded. The `ModID` element can be omitted if the entity is in the same mod as the trigger.

The `AllOfficialEntities` and `AllModdedEntities` can be used to automatically add the trigger to all official and modded entities respectively.

Activating the trigger

This is enough to get a trigger to be loaded by the game, but it can't be activated yet.

The `ModTriggers` class provides the API to activate modded triggers.

Triggers can be activated in two ways: Either by activating all triggers of a specific type that are currently in the level or by activating trigger per-entity.

Note that in both cases, activating the trigger only works when done on the game instance that is running the simulation.

To activate all triggers simultaneously, call `ModNetworking.GetCallback(id)`. This returns an `Action` that will activate all instances of the specified trigger when called.

To activate triggers per-entity, call `ModNetworking.RegisterCallback(id, callback)`. `callback` must be a function with 3 arguments of type `Entity`, `Action`, and `bool`.

The callback will be called whenever a trigger of the specified type is added to or removed from an entity. The first argument is the entity modified, and the third arguments indicates whether the trigger was removed or added (`true` is removed, `false` is added).

The second argument of the callback is an `Action` which will activate the trigger only on the passed entity when called (assuming the trigger was added, not removed). When using this approach, store the given `Action`s and call them as appropriate.

Creating an event

This is a step-by-step tutorial to adding a new event for the level editor logic system to the game. It assumes that the mod that should contain the event already exists. (See [Creating a mod](#) if this is not yet the case.)

Note that to create a useful event (i.e. one that does anything) requires writing some code that is called when the event is triggered. The specific APIs used to do this are discussed here, but the basics of loading code into the game or how to interact with the game world are not.

See [Custom Code](#) for information on how to add code to the game and the documentation of Unity and the `Modding.{Blocks, Levels, Common}` namespaces for information on how to interact with the game world.

Getting started

Events can be defined either directly inside the mod manifest or, for longer events, in their own XML file.

- Add the event to the mod manifest.

```
<Mod>
  ...
  <Events>
    <Event> <!-- To define the event directly in the manifest. -->
      ...
    </Event>
    <Event path="SomeEvent.xml" /> <!-- To define the event in its own file. -->
  </Events>
</Mod>
```

When defining an event in its own file, the file must contain an `Event` element as its root element which is equivalent to the `Event` element in the manifest when defining it inline.

- Define name, ID, and icon

3 elements are sufficient to define an event: A name, an ID and an icon. The ID must uniquely identify the event among all events in the same mod; the easiest way to handle this is to just use sequential IDs (1, 2, 3, ...).

```
<Event>
  <Name>Some Event</Name>
  <ID>1</ID>
  <Icon name="some-event-icon" />
</Event>
```

The `Icon` element is a reference to a texture defined in the mod manifest, see [Resource Handling](#) and [the Texture element](#) for more information.

Event Properties

Events can optionally have properties that can be set by the player in the mapper. Examples of properties for events of the game itself are the various values in the Transform event or the team selection in the Add Progress event.

Available types of event properties of modded events are `TextInput`, `NumberInput`, `Choice`, `Toggle`, `TeamButton`, and `Picker`. Event properties are defined in the `Properties` element of an event.

For more information on how to use these properties, see [Event Properties](#).

Making the event do something

This is enough to have the event appear in-game, but it won't do anything when triggered yet.

The `ModEvents.RegisterCallback(id, callback)` method can be used to add behaviour to events. After calling the method, the given callback will be called every time an event of the given type is activated.

The `callback` parameters must be a function taking 2 arguments: a `LogicChain` and an `IDictionary<string, EventProperty>`. The `LogicChain` gives access to the chain that the event is a part of, it can also be used to access the entity on which the event was placed. The `IDictionary<string, EventProperty>` is a mapping of property names to objects of the subclasses of `EventProperty` and can be used to access the values the player set up.

For more information, see [Adding Behaviour to Events](#).

Namespace Modding

Classes

BlockScript

Base class for custom BlockBehaviours. It offers various callbacks and convenience properties to assist in writing the code for your blocks.

Configuration

Provides an easy way to store configuration for mods. Each mod gets an `XDataHolder` to store its data. The Configuration class handles saving and loading the data.

EventProperty

EventProperty is the base class for all different event properties.

EventProperty.Choice

EventProperty class for the Choice property.

EventProperty.Choice.Option

Represents one of the available options.

EventProperty.Icon

EventProperty class for the Icon property. You cannot interact with Icon properties from code.

EventProperty.NumberInput

EventProperty class for the NumberInput property.

EventProperty.Picker

EventProperty class for the Picker property.

EventProperty.TeamButton

EventProperty class for the TeamButton property.

EventProperty.Text

EventProperty class for the Text property. You cannot interact with Text properties from code.

EventProperty.TextInput

EventProperty class for the TextInput property.

EventProperty.Toggle

EventProperty class for the Toggle property.

Events

Provides easy hooks to various things happening in the game.

Game

Provides access to some in-game state and easy methods and properties for changing it.

Game.UI

Provides methods to interact with the game UI.

GameMaterials

Gives access to shaders frequently used in the game, along with example materials that use them.

GameMaterials.BlockGhostShaders

Container for block ghost shaders. Access via GameMaterials.Shaders.BlockGhosts.

GameMaterials.BlockShaders

Container for block shaders. Access via GameMaterials.Shaders.Blocks.

GameMaterials.EntityShaders

Container for entity shaders. Access via GameMaterials.Shaders.Entities.

GameMaterials.MiscShaders

Container for miscellaneous shaders. Access via GameMaterials.Shaders.Misc.

GameMaterials.ParticleShaders

Container for particle shaders. Access via GameMaterials.Shaders.Particles.

GameMaterials.Shaders

Access to commonly-used shaders in the game.

GamePrefabs

Provides access to some useful prefabs of the game.

Message

A network message from/for the ModNetworking class.

MessageType

A message type for messages to be sent with network messages using the ModNetworking class.

ModAssetBundle

An asset bundle mod resource.

Some care must be taken when using this class, see the Remarks section for details.

ModAudioClip

An audio clip mod resource.

ModBlockBehaviour

Base class for all custom behaviours attached to modded blocks by the mod loader. This includes BlockScripts as well as the behaviour of custom modules.

ModConsole

Allows mods to interact with the in-game console. Output to the console using the Log method and add your own console commands using the RegisterCommand method.

ModEntryPoint

Every Assembly can contain one ModEntryPoint. If it does and is listed in the Mod manifest, an entry point will be instantiated by the mod loader and its `OnLoad` method called when the mod is loaded.

ModEvents

Allows registering behaviour for custom events using the RegisterCallback method.

ModIO

Provides methods for IO since large parts of the System.IO namespace are blocked for security reasons.

ModKey

Represents a keybinding that is managed by the mod loader. Keys are defined in the mod manifest.

Properties correspond to properties of the same name in the MKey class. This naming is different to that of the UnityEngine.Input class!

If the modifier is KeyCode.None, the keybinding only considers the trigger.

If the trigger is KeyCode.None, the keybinding is considered to be disabled, and all the properties always return false.

ModKeys

Keybindings managed by the mod loader. They can be rebound by the player via config files and automatically handle the possibility of being a combination of a modifier and a trigger key.

ModMesh

A mesh mod resource.

ModNetworking

Networking system for mods.

ModResource

This class provides access to the resources listed in the Resources section of the mod manifest via static members and is the base class for the objects representing a resource.

Mods

Provides an API for querying the status of other installed mods.

This can for example be used to do additional things based on the presence of other mods, like adding some kind of interaction.

ModTexture

A texture mod resource.

ModTriggers

You can use this class in conjunction with the Triggers element in Mod.xml to define custom triggers for the level editor logic system.

ModUtility

Enums

DataType

Possible data types to be sent with network messages using the ModNetworking class.

Game.UI.BlockTab

Constants representing the official block categories.

GameMaterials.BlockGhostMaterial

Available block ghost example materials. These use shaders from Shaders.BlockGhosts.

[GameMaterials.BlockMaterial](#)

Available block example materials. These use shaders from Shaders.Blocks.

[GameMaterials.EntityMaterial](#)

Available entity example materials. These use shaders from Shaders.Entities.

[GameMaterials.MiscMaterial](#)

Available miscellaneous example materials. These use shaders from Shaders.Misc.

[GameMaterials.ParticleMaterial](#)

Available particle example materials. These use shaders from Shaders.Particles.

[GamePrefabs.ExplosionType](#)

Available types of explosion prefabs.

[GamePrefabs.ProjectileType](#)

Available types of projectile prefabs.

[ModResource.ResourceType](#)

The different resource types that can be managed by the mod loader.

Delegates

[ModEvents.OnEventExecute](#)

Delegate for callbacks registered to `RegisterCallback`.

[ModTriggers.OnTriggerChanged](#)

Used in conjunction with `RegisterCallback`.

Namespace Modding

Classes

BlockScript

Base class for custom BlockBehaviours. It offers various callbacks and convenience properties to assist in writing the code for your blocks.

Configuration

Provides an easy way to store configuration for mods. Each mod gets an `XDataHolder` to store its data. The Configuration class handles saving and loading the data.

EventProperty

EventProperty is the base class for all different event properties.

EventProperty.Choice

EventProperty class for the Choice property.

EventProperty.Choice.Option

Represents one of the available options.

EventProperty.Icon

EventProperty class for the Icon property. You cannot interact with Icon properties from code.

EventProperty.NumberInput

EventProperty class for the NumberInput property.

EventProperty.Picker

EventProperty class for the Picker property.

EventProperty.TeamButton

EventProperty class for the TeamButton property.

EventProperty.Text

EventProperty class for the Text property. You cannot interact with Text properties from code.

EventProperty.TextInput

EventProperty class for the TextInput property.

EventProperty.Toggle

EventProperty class for the Toggle property.

Events

Provides easy hooks to various things happening in the game.

Game

Provides access to some in-game state and easy methods and properties for changing it.

Game.UI

Provides methods to interact with the game UI.

GameMaterials

Gives access to shaders frequently used in the game, along with example materials that use them.

GameMaterials.BlockGhostShaders

Container for block ghost shaders. Access via GameMaterials.Shaders.BlockGhosts.

GameMaterials.BlockShaders

Container for block shaders. Access via GameMaterials.Shaders.Blocks.

GameMaterials.EntityShaders

Container for entity shaders. Access via GameMaterials.Shaders.Entities.

GameMaterials.MiscShaders

Container for miscellaneous shaders. Access via GameMaterials.Shaders.Misc.

GameMaterials.ParticleShaders

Container for particle shaders. Access via GameMaterials.Shaders.Particles.

GameMaterials.Shaders

Access to commonly-used shaders in the game.

GamePrefabs

Provides access to some useful prefabs of the game.

Message

A network message from/for the ModNetworking class.

MessageType

A message type for messages to be sent with network messages using the ModNetworking class.

ModAssetBundle

An asset bundle mod resource.

Some care must be taken when using this class, see the Remarks section for details.

ModAudioClip

An audio clip mod resource.

ModBlockBehaviour

Base class for all custom behaviours attached to modded blocks by the mod loader. This includes BlockScripts as well as the behaviour of custom modules.

ModConsole

Allows mods to interact with the in-game console. Output to the console using the Log method and add your own console commands using the RegisterCommand method.

ModEntryPoint

Every Assembly can contain one ModEntryPoint. If it does and is listed in the Mod manifest, an entry point will be instantiated by the mod loader and its `OnLoad` method called when the mod is loaded.

ModEvents

Allows registering behaviour for custom events using the RegisterCallback method.

ModIO

Provides methods for IO since large parts of the System.IO namespace are blocked for security reasons.

ModKey

Represents a keybinding that is managed by the mod loader. Keys are defined in the mod manifest.

Properties correspond to properties of the same name in the MKey class. This naming is different to that of the UnityEngine.Input class!

If the modifier is KeyCode.None, the keybinding only considers the trigger.

If the trigger is KeyCode.None, the keybinding is considered to be disabled, and all the properties always return false.

ModKeys

Keybindings managed by the mod loader. They can be rebound by the player via config files and automatically handle the possibility of being a combination of a modifier and a trigger key.

ModMesh

A mesh mod resource.

ModNetworking

Networking system for mods.

ModResource

This class provides access to the resources listed in the Resources section of the mod manifest via static members and is the base class for the objects representing a resource.

Mods

Provides an API for querying the status of other installed mods.

This can for example be used to do additional things based on the presence of other mods, like adding some kind of interaction.

ModTexture

A texture mod resource.

ModTriggers

You can use this class in conjunction with the Triggers element in Mod.xml to define custom triggers for the level editor logic system.

ModUtility

Enums

DataType

Possible data types to be sent with network messages using the ModNetworking class.

Game.UI.BlockTab

Constants representing the official block categories.

GameMaterials.BlockGhostMaterial

Available block ghost example materials. These use shaders from Shaders.BlockGhosts.

[GameMaterials.BlockMaterial](#)

Available block example materials. These use shaders from Shaders.Blocks.

[GameMaterials.EntityMaterial](#)

Available entity example materials. These use shaders from Shaders.Entities.

[GameMaterials.MiscMaterial](#)

Available miscellaneous example materials. These use shaders from Shaders.Misc.

[GameMaterials.ParticleMaterial](#)

Available particle example materials. These use shaders from Shaders.Particles.

[GamePrefabs.ExplosionType](#)

Available types of explosion prefabs.

[GamePrefabs.ProjectileType](#)

Available types of projectile prefabs.

[ModResource.ResourceType](#)

The different resource types that can be managed by the mod loader.

Delegates

[ModEvents.OnEventExecute](#)

Delegate for callbacks registered to `RegisterCallback`.

[ModTriggers.OnTriggerChanged](#)

Used in conjunction with `RegisterCallback`.

Class BlockScript

Base class for custom BlockBehaviours. It offers various callbacks and convenience properties to assist in writing the code for your blocks.

Inheritance

[Object](#)

[ModBlockBehaviour](#)

BlockScript

Inherited Members

[ModBlockBehaviour.SafeAwake\(\)](#)

[ModBlockBehaviour.OnPrefabCreation\(\)](#)

[ModBlockBehaviour.OnBlockPlaced\(\)](#)

[ModBlockBehaviour.BuildingUpdate\(\)](#)

[ModBlockBehaviour.SimulateUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateUpdateClient\(\)](#)

[ModBlockBehaviour.BuildingFixedUpdate\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateClient\(\)](#)

[ModBlockBehaviour.BuildingLateUpdate\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateClient\(\)](#)

[ModBlockBehaviour.OnSimulateStart\(\)](#)

[ModBlockBehaviour.OnSimulateStop\(\)](#)

[ModBlockBehaviour.OnStartBurning\(\)](#)

[ModBlockBehaviour.OnStopBurning\(Boolean\)](#)

[ModBlockBehaviour.OnSimulateCollisionEnter\(Collision\)](#)

[ModBlockBehaviour.OnSimulateCollisionStay\(Collision\)](#)

[ModBlockBehaviour.OnSimulateCollisionExit\(Collision\)](#)

[ModBlockBehaviour.OnSimulateTriggerEnter\(Collider\)](#)

[ModBlockBehaviour.OnSimulateTriggerStay\(Collider\)](#)

[ModBlockBehaviour.OnSimulateTriggerExit\(Collider\)](#)

[ModBlockBehaviour.OnSimulateParticleCollision\(GameObject\)](#)

[ModBlockBehaviour.OnSave\(XDataHolder\)](#)

[ModBlockBehaviour.OnLoad\(XDataHolder\)](#)

[ModBlockBehaviour.OnReloadAmmo\(Int32, AmmoType, Boolean, Boolean\)](#)

[ModBlockBehaviour.AddKey\(String, String, KeyCode\)](#)

[ModBlockBehaviour.AddKey\(MKey\)](#)

[ModBlockBehaviour.AddTeam\(String, String, MPTeam\)](#)

[ModBlockBehaviour.AddTeam\(MTeam\)](#)

[ModBlockBehaviour.AddText\(String, String, String\)](#)

[ModBlockBehaviour.AddText\(MText\)](#)

[ModBlockBehaviour.AddValue\(String, String, Single\)](#)

[ModBlockBehaviour.AddValue\(String, String, Single, Single, Single\)](#)

[ModBlockBehaviour.AddValue\(MValue\)](#)

[ModBlockBehaviour.AddSlider\(String, String, Single, Single, Single\)](#)

[ModBlockBehaviour.AddSliderUnclamped\(String, String, Single, Single, Single\)](#)

[ModBlockBehaviour.AddSlider\(MSlider\)](#)

[ModBlockBehaviour.AddColourSlider\(String, String, Color, Boolean\)](#)

[ModBlockBehaviour.AddColourSlider\(MColourSlider\)](#)
[ModBlockBehaviour.AddMenu\(String, Int32, List<String>, Boolean\)](#)
[ModBlockBehaviour.AddMenu\(MMenu\)](#)
[ModBlockBehaviour.AddToggle\(String, String, Boolean\)](#)
[ModBlockBehaviour.AddToggle\(String, String, String, Boolean\)](#)
[ModBlockBehaviour.AddToggle\(MToggle\)](#)
[ModBlockBehaviour.AddLimits\(String, String, Single, Single, Single, FauxTransform\)](#)
[ModBlockBehaviour.AddLimits\(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay\)](#)
[ModBlockBehaviour.AddLimits\(MLimits\)](#)
[ModBlockBehaviour.AddCustom<T>\(MCustom<T>\)](#)
[ModBlockBehaviour.IsBurning](#)
[ModBlockBehaviour.HasBurnedOut](#)
[ModBlockBehaviour.IsFrozen](#)
[ModBlockBehaviour.IsDestroyed](#)
[ModBlockBehaviour.HasRigidbody](#)
[ModBlockBehaviour.Rigidbody](#)
[ModBlockBehaviour.BlockBehaviour](#)
[ModBlockBehaviour.VisualController](#)
[ModBlockBehaviour.Renderer](#)
[ModBlockBehaviour.MainVis](#)
[ModBlockBehaviour.ShowDebugVisuals](#)
[ModBlockBehaviour.Flipped](#)
[ModBlockBehaviour.BlockId](#)
[ModBlockBehaviour.SimPhysics](#)
[ModBlockBehaviour.IsSimulating](#)
[ModBlockBehaviour.IsStripped](#)
[ModBlockBehaviour.Machine](#)
[ModBlockBehaviour.CanFlip](#)
[ModBlockBehaviour.DirectionArrow](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BlockScript : ModBlockBehaviour, ILimitsDisplay
```

Remarks

The documentation for callbacks contains information about their behaviour in multiplayer. When writing your block script, keep in mind that most callbacks are also called if the game instance the script is running on does not handle physics currently.

You can use the `SimPhysics` property to determine whether you should handle physics, or use methods that are only called if physics should / should not be simulated.

Constructors

BlockScript()

Declaration

```
public BlockScript()
```

Methods

GetLimitsDisplay()

Declaration

```
public virtual Transform GetLimitsDisplay()
```

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	

OnFlip()

Override to implement custom flipping behaviour on the block. Return value indicates whether the block was actually flipped, returning true causes the direction arrow to be flipped, if it is present.

Overriding this means the CanFlip XML element has no effect anymore.

Declaration

```
public virtual bool OnFlip()
```

Returns

TYPE	DESCRIPTION
Boolean	

Remarks

If you just want to change behaviour of the block based on whether it is currently flipped, also consider using the Flipped property instead.

Class Configuration

Provides an easy way to store configuration for mods. Each mod gets an `XDataHolder` to store its data. The Configuration class handles saving and loading the data.

Inheritance

[Object](#)

Configuration

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Configuration
```

Constructors

Configuration()

Declaration

```
public Configuration()
```

Methods

GetData()

Gets the XDataHolder instance for the calling mod's configuration. If one does not exist for the mod yet, one is created.

Declaration

```
public static XDataHolder GetData()
```

Returns

TYPE	DESCRIPTION
XDataHolder	

Save()

Manually trigger saving your mod's configuration. Configuration is automatically saved on application exit, you can use this to force saving to file whenever you want.

Declaration

```
public static void Save()
```

Enum DataType

Possible data types to be sent with network messages using the ModNetworking class.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum DataType
```

Fields

NAME	DESCRIPTION
Block	
Boolean	
ByteArray	
Color	
Entity	
Integer	
IntegerArray	
Machine	
Single	
SingleArray	
String	
StringArray	
Vector3	

Class EventProperty

EventProperty is the base class for all different event properties.

Inheritance

- Object
- Element
- EventProperty
- EventProperty.Choice
- EventProperty.Icon
- EventProperty.NumberInput
- EventProperty.Picker
- EventProperty.TeamButton
- EventProperty.Text
- EventProperty.TextInput
- EventProperty.Toggle

Implements

- IValidatable

Inherited Members

- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class EventProperty : Element, IValidatable
```

Constructors

EventProperty()

Declaration

```
protected EventProperty()
```

Fields

Event

Declaration

```
protected EntityEvent Event
```

Field Value

TYPE	DESCRIPTION
EntityEvent	

Logic

Declaration

protected EntityLogic Logic

Field Value

TYPE	DESCRIPTION
EntityLogic	

Name

Name of the property, as specified in the XML declaration.

Declaration

public string Name

Field Value

TYPE	DESCRIPTION
String	

nameRequired

Declaration

protected bool nameRequired

Field Value

TYPE	DESCRIPTION
Boolean	

Row

What row the property is displayed in.

Declaration

public int Row

Field Value

TYPE	DESCRIPTION
Int32	

XSpecified

Whether or not an X coordinate was specified.

Declaration


```
public bool XSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Properties

X

The X coordinate of the event property in its row. 0 if not specified.

Declaration

```
public float X { get; set; }
```

Property Value

TYPE	DESCRIPTION
Single	

Methods

CopyAttributes(EventProperty)

Declaration

```
protected EventProperty CopyAttributes(EventProperty dst)
```

Parameters

TYPE	NAME	DESCRIPTION
EventProperty	dst	

Returns

TYPE	DESCRIPTION
EventProperty	

OnEdit()

Declaration

```
protected virtual void OnEdit()
```

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Validate(String)

Declaration

```
protected override bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class EventProperty.Choice

EventProperty class for the Choice property.

Inheritance

[Object](#)

[Element](#)

[EventProperty](#)

EventProperty.Choice

Implements

[IValidatable](#)

Inherited Members

[EventProperty.Name](#)

[EventProperty.nameRequired](#)

[EventProperty.XSpecified](#)

[EventProperty.Row](#)

[EventProperty.Logic](#)

[EventProperty.Event](#)

[EventProperty.OnEdit\(\)](#)

[EventProperty.CopyAttributes\(EventProperty\)](#)

[EventProperty.Validate\(String\)](#)

[EventProperty.ToString\(\)](#)

[EventProperty.X](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Choice : EventProperty, IValidatable
```

Constructors

Choice()

Declaration

```
public Choice()
```

Fields

Default

Corresponds to the default XML attribute.

Declaration

```
public int Default
```

Field Value

TYPE	DESCRIPTION
Int32	

DefaultSpecified

Whether the default attribute was specified.

Declaration

```
public bool DefaultSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Options

The available options, specified in the XML file.

Declaration

```
[RequireToValidate]  
public EventProperty.Choice.Option[] Options
```

Field Value

TYPE	DESCRIPTION
EventProperty.Choice.Option[]	

Properties

CurrentIndex

Which option is currently selected. This is an index into the Options array, not to be confused with the Index property of the Option class.

Declaration

```
public int CurrentIndex { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Methods

SetIndex(Int32)

Selects the option with the given index. The index here is compared with the Index property of the Option class.

Declaration

```
public void SetIndex(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	Which option to select.

Implements

[IValidatable](#)

Class EventProperty.Choice.Option

Represents one of the available options.

Inheritance

[Object](#)

[Element](#)

EventProperty.Choice.Option

Implements

[IValidatable](#)

Inherited Members

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Option : Element, IValidatable
```

Constructors

Option()

Declaration

```
public Option()
```

Fields

Index

Corresponds to the index XML attribute. This is used only when saving the level, it is not an index into the Options array!

Declaration

```
public int Index
```

Field Value

TYPE	DESCRIPTION
Int32	

IndexSpecified

Whether or not an index was specified.

Declaration

```
public bool IndexSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Text

Option text, as specified in the XML file.

Declaration

```
public string Text
```

Field Value

TYPE	DESCRIPTION
String	

Implements

[IValidatable](#)

Class EventProperty.Icon

EventProperty class for the Icon property. You cannot interact with Icon properties from code.

Inheritance

[Object](#)
[Element](#)
[EventProperty](#)
EventProperty.Icon

Implements

[IValidatable](#)

Inherited Members

[EventProperty.Name](#)
[EventProperty.nameRequired](#)
[EventProperty.XSpecified](#)
[EventProperty.Row](#)
[EventProperty.Logic](#)
[EventProperty.Event](#)
[EventProperty.OnEdit\(\)](#)
[EventProperty.CopyAttributes\(EventProperty\)](#)
[EventProperty.Validate\(String\)](#)
[EventProperty.ToString\(\)](#)
[EventProperty.X](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class Icon : EventProperty, IValidatable
```

Constructors

Icon()

Declaration

```
public Icon()
```

Fields

IconReference

Declaration

```
[RequireToValidate]  
public ResourceReference IconReference
```


Field Value

TYPE	DESCRIPTION
ResourceReference	

IconTexture

Declaration

public ModTexture IconTexture

Field Value

TYPE	DESCRIPTION
ModTexture	

Implements

[IValidatable](#)

Class EventProperty.NumberInput

EventProperty class for the NumberInput property.

Inheritance

- Object
- Element
- EventProperty
- EventProperty.NumberInput

Implements

- IValidatable

Inherited Members

- EventProperty.Name
- EventProperty.nameRequired
- EventProperty.XSpecified
- EventProperty.Row
- EventProperty.Logic
- EventProperty.Event
- EventProperty.OnEdit()
- EventProperty.CopyAttributes(EventProperty)
- EventProperty.Validate(String)
- EventProperty.ToString()
- EventProperty.X
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class NumberInput : EventProperty, IValidatable
```

Constructors

NumberInput()

Declaration

```
public NumberInput()
```

Fields

CharLimit

Corresponds to the maxChars XML attribute.

Declaration

```
public int CharLimit
```

Field Value

TYPE	DESCRIPTION
Int32	

Decimals

Corresponds to the decimals XML attribute.

Declaration

```
public int Decimals
```

Field Value

TYPE	DESCRIPTION
Int32	

DefaultValue

Default value, as specified in the XML file.

Declaration

```
public float DefaultValue
```

Field Value

TYPE	DESCRIPTION
Single	

DefaultValueSpecified

Whether or not a default value was specified.

Declaration

```
public bool DefaultValueSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

MaxDecimals

Corresponds to the maxDecimals XML attribute.

Declaration

```
public int MaxDecimals
```

Field Value

TYPE	DESCRIPTION
Int32	

MaxValue

Corresponds to the max**Value** XML attribute.

Declaration

```
public float MaxValue
```

Field Value

TYPE	DESCRIPTION
Single	

MinValue

Corresponds to the min**Value** XML attribute.

Declaration

```
public float MinValue
```

Field Value

TYPE	DESCRIPTION
Single	

NegativeNumbers

Corresponds to the negative**Numbers** XML attribute.

Declaration

```
public bool NegativeNumbers
```

Field Value

TYPE	DESCRIPTION
Boolean	

Prefix

Corresponds to the prefix XML attribute.

Declaration

```
public string Prefix
```

Field Value

TYPE	DESCRIPTION
String	

SplitThousand

Corresponds to the splitThousands XML attribute.

Declaration

```
public bool SplitThousand
```

Field Value

TYPE	DESCRIPTION
Boolean	

Suffix

Corresponds to the suffix XML attribute.

Declaration

```
public string Suffix
```

Field Value

TYPE	DESCRIPTION
String	

Title

Title text, as specified in the XML file.

Declaration

```
public string Title
```

Field Value

TYPE	DESCRIPTION
String	

Properties

Value

Number entered in the UI.

Declaration

```
public float Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
Single	

Implements

[IValidatable](#)

Class EventProperty.Picker

EventProperty class for the Picker property.

Inheritance

- Object
- Element
- EventProperty
- EventProperty.Picker

Implements

- IValidatable

Inherited Members

- EventProperty.Name
- EventProperty.nameRequired
- EventProperty.XSpecified
- EventProperty.Row
- EventProperty.Logic
- EventProperty.Event
- EventProperty.OnEdit()
- EventProperty.CopyAttributes(EventProperty)
- EventProperty.Validate(String)
- EventProperty.ToString()
- EventProperty.X
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class Picker : EventProperty, IValidatable
```

Constructors

Picker()

Declaration

```
public Picker()
```

Fields

Entities

The entities that were selected.

Declaration

```
public List<Entity> Entities
```

Field Value

TYPE	DESCRIPTION
List<Entity>	

Mode

The pick mode, as specified in the XML file.

Declaration

```
public StatMaster.Mode.PickMode Mode
```

Field Value

TYPE	DESCRIPTION
StatMaster.Mode.PickMode	

Implements

[IValidatable](#)

Class EventProperty.TeamButton

EventProperty class for the TeamButton property.

Inheritance

- Object
- Element
- EventProperty
- EventProperty.TeamButton

Implements

- IValidatable

Inherited Members

- EventProperty.Name
- EventProperty.nameRequired
- EventProperty.XSpecified
- EventProperty.Row
- EventProperty.Logic
- EventProperty.Event
- EventProperty.OnEdit()
- EventProperty.CopyAttributes(EventProperty)
- EventProperty.Validate(String)
- EventProperty.ToString()
- EventProperty.X
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class TeamButton : EventProperty, IValidatable
```

Constructors

TeamButton()

Declaration

```
public TeamButton()
```

Properties

Team

Currently selected team.

Declaration


```
public MPTeam Team { get; set; }
```

Property Value

TYPE	DESCRIPTION
MPTeam	

Implements

[IValidatable](#)

Class EventProperty.Text

EventProperty class for the Text property. You cannot interact with Text properties from code.

Inheritance

[Object](#)

[Element](#)

[EventProperty](#)

EventProperty.Text

Implements

[IValidatable](#)

Inherited Members

[EventProperty.Name](#)

[EventProperty.nameRequired](#)

[EventProperty.XSpecified](#)

[EventProperty.Row](#)

[EventProperty.Logic](#)

[EventProperty.Event](#)

[EventProperty.OnEdit\(\)](#)

[EventProperty.CopyAttributes\(EventProperty\)](#)

[EventProperty.Validate\(String\)](#)

[EventProperty.ToString\(\)](#)

[EventProperty.X](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Text : EventProperty, IValidatable
```

Constructors

Text()

Declaration

```
public Text()
```

Fields

DisplayText

Declaration

```
public string DisplayText
```

Field Value

TYPE	DESCRIPTION
String	

FontSize

Declaration

public float FontSize

Field Value

TYPE	DESCRIPTION
Single	

Implements

[IValidatable](#)

Class EventProperty.TextInput

EventProperty class for the TextInput property.

Inheritance

[Object](#)

[Element](#)

[EventProperty](#)

EventProperty.TextInput

Implements

[IValidatable](#)

Inherited Members

[EventProperty.Name](#)

[EventProperty.nameRequired](#)

[EventProperty.XSpecified](#)

[EventProperty.Row](#)

[EventProperty.Logic](#)

[EventProperty.Event](#)

[EventProperty.OnEdit\(\)](#)

[EventProperty.CopyAttributes\(EventProperty\)](#)

[EventProperty.Validate\(String\)](#)

[EventProperty.ToString\(\)](#)

[EventProperty.X](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class TextInput : EventProperty, IValidatable
```

Constructors

TextInput()

Declaration

```
public TextInput()
```

Fields

DefaultText

Default text, as specified in the XML file.

Declaration

```
public string DefaultText
```

Field Value

TYPE	DESCRIPTION
String	

MaxChars

Maximum characters, as specified in the XML file.

Declaration

```
public int MaxChars
```

Field Value

TYPE	DESCRIPTION
Int32	

Title

Title text, as specified in the XML file.

Declaration

```
public string Title
```

Field Value

TYPE	DESCRIPTION
String	

Properties

Text

Text entered in the UI.

Declaration

```
public string Text { get; set; }
```

Property Value

TYPE	DESCRIPTION
String	

Implements

[IValidatable](#)

Class EventProperty.Toggle

EventProperty class for the Toggle property.

Inheritance

- Object
- Element
- EventProperty
- EventProperty.Toggle

Implements

- IValidatable

Inherited Members

- EventProperty.Name
- EventProperty.nameRequired
- EventProperty.XSpecified
- EventProperty.Row
- EventProperty.Logic
- EventProperty.Event
- EventProperty.OnEdit()
- EventProperty.CopyAttributes(EventProperty)
- EventProperty.Validate(String)
- EventProperty.ToString()
- EventProperty.X
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class Toggle : EventProperty, IValidatable
```

Constructors

Toggle()

Declaration

```
public Toggle()
```

Fields

Default

Corresponds to the default XML attribute.

Declaration

```
public bool Default
```

Field Value

TYPE	DESCRIPTION
Boolean	

DefaultSpecified

Whether a default attribute was specified.

Declaration

```
public bool DefaultSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Icon

The Icon resource used.

Declaration

```
public ModTexture Icon
```

Field Value

TYPE	DESCRIPTION
ModTexture	

IconReference

A reference to the Icon specified for the toggle.

Declaration

```
[RequireToValidate]  
public ResourceReference IconReference
```

Field Value

TYPE	DESCRIPTION
ResourceReference	

Properties

Value

Current value of the toggle.

Declaration

```
public bool Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Implements

[IValidatable](#)

Class Events

Provides easy hooks to various things happening in the game.

Inheritance

[Object](#)

SingleInstance<[Events](#)>

Events

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Events : SingleInstance<Events>
```

Constructors

Events()

Declaration

```
public Events()
```

Properties

Name

Declaration

```
public override string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Overrides

SingleInstance<Modding.Events>.Name

Methods

SetUp()

Declaration

```
public override void SetUp()
```

Overrides

SingleInstance<Modding.Events>.SetUp()

Events

OnBlockInit

Called while a new block is being initialized.

This is called for new blocks placed in building mode but also for block clones being created for simulation.

Blocks will not have loaded their mapper type data yet at this point, which means it is possible to add mapper types to the blocks here.

MP: Always called on all connected instances.

Declaration

```
public static event Action<Block> OnBlockInit
```

Event Type

TYPE	DESCRIPTION
Action<Block>	

OnBlockPlaced

Called when a block was placed on a machine.

This is called for blocks placed in build mode and only once the block has been completely initialized.

Add Mapper Types to blocks here does not work correctly, use `OnBlockInit` for that.

MP: Called only on the instance that places the block.

Declaration

```
public static event Action<Block> OnBlockPlaced
```

Event Type

TYPE	DESCRIPTION
Action<Block>	

OnBlockRemoved

Called when a block is removed from a machine.

The passed block will be destroyed immediately after the event handlers for this were executed.

MP: Called only on the instance that removes the block.

Declaration

```
public static event Action<Block> OnBlockRemoved
```

Event Type

TYPE	DESCRIPTION
Action<Block>	

OnConnect

Called when connecting to a multiplayer session.

Also called when starting up a server.

Declaration

```
public static event Action OnConnect
```

Event Type

TYPE	DESCRIPTION
Action	

OnDisconnect

Called when disconnecting from a multiplayer session.

Also called when shutting down a server.

Declaration

```
public static event Action OnDisconnect
```

Event Type

TYPE	DESCRIPTION
Action	

OnEntityPlaced

Called when an entity (level object) is placed.

MP: Always called on all connected instances.

Declaration

```
public static event Action<Entity> OnEntityPlaced
```

Event Type

TYPE	DESCRIPTION
Action<Entity>	

OnEntityRemoved

Called when an entity is removed.

The passed entity will be destroyed immediately after the event handlers for this were executed.

MP: Always called on all connected instances.

Declaration

```
public static event Action<Entity> OnEntityRemoved
```

Event Type

TYPE	DESCRIPTION
Action<Entity>	

OnLevelDeleted

Called when a level is cleared (deleted).

Called on all connected instances, because the level is always the same for everyone. Also called in the preparation for joining or loading a level.

Declaration

```
public static event Action OnLevelDeleted
```

Event Type

TYPE	DESCRIPTION
Action	

OnLevelLoaded

Called when a level was loaded.

Always called on all connected instances, because the level is always the same for everyone. Also called when joining a server and loading the current level.

Declaration

```
public static event Action<Level> OnLevelLoaded
```

Event Type

TYPE	DESCRIPTION
Action<Level>	

OnLevelSave

Called when a level is begin saved.

Called only on the instance that saves the level.

Declaration

```
public static event Action<Level> OnLevelSave
```

Event Type

TYPE	DESCRIPTION
Action<Level>	

Remarks

The host may automatically save the level to send it to joining players.

OnLevelSetupChanged

Called when the settings of the current level were edited.

Called on all connected instances.

Declaration

```
public static event Action<LevelSetup> OnLevelSetupChanged
```

Event Type

TYPE	DESCRIPTION
Action<LevelSetup>	

OnLevelWonMP

Called when a multiplayer level is won.

Called on the server and all clients that have global sim toggled on when in global sim. If a client has global sim off, but is in building mode, this event does not fire. Called only on the instance that's simulating when in local sim.

Parameter: The list of teams that have won (can also be MPTeam.None).

Declaration

<code>public static event Action<List<MPTeam>> OnLevelWonMP</code>
--

Event Type

TYPE	DESCRIPTION
Action<List<MPTeam>>	

OnLogicChainTriggered

Called when a logic chain is triggered.

MP: Called only on the instance running the simulation.

Declaration

<code>public static event Action<LogicChain> OnLogicChainTriggered</code>

Event Type

TYPE	DESCRIPTION
Action<LogicChain>	

OnLogicEventExecuted

Called when a logic event is executed.

MP: Called only on the instance running the simulation.

Declaration

<code>public static event Action<LogicEvent> OnLogicEventExecuted</code>
--

Event Type

TYPE	DESCRIPTION
Action<LogicEvent>	

OnMachineDestroyed

Called when a machine is destroyed (deleted).

MP: Only called on the instance deleting its machine.

Declaration

```
public static event Action OnMachineDestroyed
```

Event Type

TYPE	DESCRIPTION
Action	

OnMachineLoaded

Called when a machine was loaded.

Custom data that was saved will have been put back into PlayerMachineInfo.MachineData.

MP: Called only on the instance that loaded.

Declaration

```
public static event Action<PlayerMachineInfo> OnMachineLoaded
```

Event Type

TYPE	DESCRIPTION
Action<PlayerMachineInfo>	

OnMachineSave

Called when a machine is being saved.

Custom data can be saved into PlayerMachineInfo.MachineData here, it will then be saved in the machine.

MP: Called only on the instance that saves.

Declaration

```
public static event Action<PlayerMachineInfo> OnMachineSave
```

Event Type

TYPE	DESCRIPTION
Action<PlayerMachineInfo>	

See Also

[MachineData](#)

OnMachineSimulationToggle

Called when the simulation for a machine is toggled on or off.

Declaration

```
public static event Action<PlayerMachine, bool> OnMachineSimulationToggle
```

Event Type

TYPE	DESCRIPTION
Action<PlayerMachine, Boolean>	

Remarks

In singleplayer, this works like `OnSimulationToggle`. It is called whenever simulation is started or stopped.

In multiplayer, the event is always called for every machine entering or exiting simulation, on all connected instances. This includes when machines enter or exit local simulation.

OnPlayerJoin

Called when a player joins a multiplayer session.

Only called on the host.

Declaration

```
public static event Action<Player> OnPlayerJoin
```

Event Type

TYPE	DESCRIPTION
Action<Player>	

OnPlayerLeave

Called when a player leaves a multiplayer session.

Only called on the host.

Declaration

```
public static event Action<Player> OnPlayerLeave
```

Event Type

TYPE	DESCRIPTION
Action<Player>	

OnProgressAddedMP

Called when level progress is added in multiplayer.

Called on the server and all clients that have global sim toggled on when in global sim. If a client has global sim off, but is in building mode, this event does not fire. Called only on the instance that's simulating when in local sim.

Declaration

```
public static event Action<MPTeam, float> OnProgressAddedMP
```

Event Type

TYPE	DESCRIPTION
Action<MPTeam, Single>	

OnSceneWithModsCameraInitialised

Called when the camera has finished its initialisation in a scene where mods should be active (singleplayer levels and multiplayer scene).

Camera initialization is the fade-in / rotate-in effect when loading a level.

Declaration

```
public static event Action OnSceneWithModsCameraInitialised
```

Event Type

TYPE	DESCRIPTION
Action	

OnSceneWithModsLoaded

Called when a scene is loaded in which mods should be active.

These scenes are all singleplayer levels (including sandboxes) and the multiverse scene.

The event is not fired when loading the title screen or a level select screen.

Declaration

```
public static event Action OnSceneWithModsLoaded
```

Event Type

TYPE	DESCRIPTION
Action	

OnSimulationToggle

Called when the simulation for the level is toggled on or off.

Declaration

```
public static event Action<bool> OnSimulationToggle
```

Event Type

TYPE	DESCRIPTION
Action<Boolean>	

Remarks

In singleplayer, this is functionally equivalent to the `OnMachineSimulationToggle` and called whenever simulation is started or stopped.

In multiplayer, behaviour depends on whether the local instance is set to local or global simulation.

For global simulation, this event is called when the level overall starts or stops simulating. It is called on the instance causing the simulation to start and all other instances that are in build mode and set to global simulation.

When an instance is set to local simulation, the event works like in singleplayer. It does not get called when global simulation is started or stopped, but it gets called when the local player starts or stops simulation.

Class Game

Provides access to some in-game state and easy methods and properties for changing it.

Inheritance

[Object](#)

Game

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class Game
```

Fields

BlockEntityLayerMask

Declaration

```
public static LayerMask BlockEntityLayerMask
```

Field Value

TYPE	DESCRIPTION
UnityEngine.LayerMask	

Properties

IsSetToLocalSim

Whether the game is set to local sim instead of global sim. Does NOT indicate whether actually in simulation or not.

Declaration

```
public static bool IsSetToLocalSim { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSimulating

Whether the local instance is currently in any simulation. Settings this is equivalent to calling {Start,End}Simulation{Local,Global}, according to `IsSetToLocalSim`.

Declaration

```
public static bool IsSimulating { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSimulatingGlobal

Whether the local instance is participating in global simulation. Setting this is equivalent to calling {Start,End}SimulationGlobal.

Also indicates simulation in singleplayer.

Declaration

```
public static bool IsSimulatingGlobal { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSimulatingLocal

Whether the local instance is currently in local simulation. Setting this is equivalent to calling {Start,End}SimulationLocal.

Never true in singleplayer; singleplayer simulation is treated as global simulation.

Declaration

```
public static bool IsSimulatingLocal { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSpectator

Whether the local instance is in spectator mode.

Declaration

```
public static bool IsSpectator { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

BlockEntityMouseRaycast(out RaycastHit, Single)

Performs a raycast from the mouse position with a predefined layer mask to detect only blocks and entities.

Note that, in multiplayer, the blocks of machines of other players do not have colliders attached (except on the host). This means they cannot be detected with raycasts.

The paramters correspond to those of `Physics.Raycast`.

Declaration

```
public static bool BlockEntityMouseRaycast(out RaycastHit hit, float distance)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.RaycastHit	hit	
Single	distance	

Returns

TYPE	DESCRIPTION
Boolean	

BlockEntityRaycast(Ray, out RaycastHit, Single)

Performs a raycast with a predefined layer mask to detect only blocks and entities.

Note that, in multiplayer, the blocks of machines of other players do not have colliders attached (except on the host). This means they cannot be detected with raycasts.

The parameters correspond to those of `Physics.Raycast`.

Declaration

```
public static bool BlockEntityRaycast(Ray ray, out RaycastHit hit, float distance)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Ray	ray	
UnityEngine.RaycastHit	hit	
Single	distance	

Returns

TYPE	DESCRIPTION
Boolean	

EndSimulationGlobal()

Ends/Leaves the global simulation. No-op if not currently in any simulation.

Declaration

```
public static void EndSimulationGlobal()
```

Exceptions

TYPE	CONDITION
InvalidOperationException	If currently in local sim.

EndSimulationLocal()

Ends the local simulation. No-op if not currently in any simulation.

Declaration

```
public static void EndSimulationLocal()
```

Exceptions

TYPE	CONDITION
InvalidOperationException	If currently in global sim.

GetCurrentProgress()

Gets the current victory progress (in percent) of all teams.

Returns zero for all teams when not in MP game.

Declaration

```
public static Dictionary<MPTeam, float> GetCurrentProgress()
```

Returns

TYPE	DESCRIPTION
Dictionary <MPTeam, Single >	

GetCurrentProgress(MP Team)

Gets the current victory progress (in percent) of the specified team.

Returns zero for all teams when not in MP game.

Declaration

```
public static float GetCurrentProgress(MPTeam team)
```

Parameters

TYPE	NAME	DESCRIPTION
MPTeam	team	

Returns

TYPE	DESCRIPTION
Single	

GetPlaylist()

Gets a list of all levels in the current playlist.

Empty list if no playlist is set up, not in MP, or not the host.

Declaration

```
public static List<string> GetPlaylist()
```

Returns

TYPE	DESCRIPTION
List<String>	A list of filesystem paths to the playlist levels.

NextPlaylistLevel()

Loads the next playlist level.

No-op if there is no playlist set up, not in MP, or not the host.

First exits simulation if currently simulating.

Declaration

```
public static void NextPlaylistLevel()
```

PreviousPlaylistLevel()

Loads the previous playlist level.

No-op if no playlist is set up, not in MP, or not the host.

First exits simulation if currently simulating.

Declaration

```
public static void PreviousPlaylistLevel()
```

Exceptions

TYPE	CONDITION
Exception	If currently in simulation.

StartSimulationGlobal()

Starts/Joins the global simulation. No-op if already in global simulation. If the game is currently set to local sim, will change this setting first.

Declaration

```
public static void StartSimulationGlobal()
```

Exceptions

TYPE	CONDITION
InvalidOperationException	If currently in local sim.
InvalidOperationException	If currently in spectator mode.

StartSimulationLocal()

Starts local simulation. No-op if already in local sim. If the game is currently set to global sim, will change this settings first.

Declaration

```
public static void StartSimulationLocal()
```

Exceptions

TYPE	CONDITION
InvalidOperationException	If currently in global sim.
InvalidOperationException	If currently in spectator mode.

ToggleLocalSimState()

Toggles whether the game is set to local or global simulation.

Declaration

```
public static void ToggleLocalSimState()
```

Exceptions

TYPE	CONDITION
InvalidOperationException	If currently in any simulation.

Class Game.UI

Provides methods to interact with the game UI.

Inheritance

[Object](#)

Game.UI

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class UI
```

Methods

OpenBlockTab(Game.UI.BlockTab)

Open the specified official block tab.

Declaration

```
public static void OpenBlockTab(Game.UI.BlockTab tab)
```

Parameters

TYPE	NAME	DESCRIPTION
Game.UI.BlockTab	tab	

OpenLevelCategory(StatMaster.Category)

Open the specified level editor category. No-op if not in MP.

Declaration

```
public static void OpenLevelCategory(StatMaster.Category cat)
```

Parameters

TYPE	NAME	DESCRIPTION
StatMaster.Category	cat	

OpenModdedBlockTab(Int32)

Open a modded block tab. These are numbered sequentially, starting at 0.

Declaration

```
public static void OpenModdedBlockTab(int tab)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	tab	

SelectBlockPrefab(BlockType)

Select the specified official block type for building. Also opens the correct tab for the block type.

Declaration

```
public static void SelectBlockPrefab(BlockType type)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockType	type	

SelectBlockPrefab(BlockPrefabInfo)

Select the specified block prefab/type for building. Also opens the correct tab for the block type.

Declaration

```
public static void SelectBlockPrefab(BlockPrefabInfo prefab)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockPrefabInfo	prefab	

SelectBlockPrefab(Guid, Int32)

Select the specified modded block type for building. Modded block types are represented by their (modId, localId) pair. Also opens the correct tab for the block type.

Declaration

```
public static void SelectBlockPrefab(Guid modId, int localId)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	
Int32	localId	

SelectEntityPrefab(EntityPrefabInfo)

Select the specified entity prefab/type for placing in the level editor. Also opens the correct category for the entity type. No-op if not in MP.

Declaration

```
public static void SelectEntityPrefab(EntityPrefabInfo prefab)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityPrefabInfo	prefab	

SelectEntityPrefab(Guid, Int32)

Select the specified modded entity type for placing in the level editor. Modded entity types are represented by their (modId, localId) pair. Also opens the correct category for the entity type. No-op if not in MP.

Declaration

```
public static void SelectEntityPrefab(Guid modId, int localId)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	
Int32	localId	

SelectEntityPrefab(Int32)

Select the specified official entity type for placing in the level editor. Also opens the correct category for the entity type. No-op if not in MP.

Declaration

```
public static void SelectEntityPrefab(int id)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	

Enum Game.Ul.BlockTab

Constants representing the official block categories.

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public enum BlockTab
```

Fields

NAME	DESCRIPTION
Armour	
Basic	
Blocks	
Flight	
Locomotion	
Mechanical	
Search	
Weaponry	

Class GameMaterials

Gives access to shaders frequently used in the game, along with example materials that use them.

Inheritance

[Object](#)

GameMaterials

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class GameMaterials : MonoBehaviour
```

Constructors

GameMaterials()

Declaration

```
public GameMaterials()
```

Methods

GetMaterial(GameMaterials.BlockGhostMaterial)

Gets an instance of the given material.

Declaration

```
public static Material GetMaterial(GameMaterials.BlockGhostMaterial mat)
```

Parameters

TYPE	NAME	DESCRIPTION
GameMaterials.BlockGhostMaterial	mat	What material to return.

Returns

TYPE	DESCRIPTION
UnityEngine.Material	A copy of the example material.

GetMaterial(GameMaterials.BlockMaterial)

Gets an instance of the given material.

Declaration

```
public static Material GetMaterial(GameMaterials.BlockMaterial mat)
```

Parameters

TYPE	NAME	DESCRIPTION
GameMaterials.BlockMaterial	mat	What material to return.

Returns

TYPE	DESCRIPTION
UnityEngine.Material	A copy of the example material.

GetMaterial(GameMaterials.EntityMaterial)

Gets an instance of the given material.

Declaration

```
public static Material GetMaterial(GameMaterials.EntityMaterial mat)
```

Parameters

TYPE	NAME	DESCRIPTION
GameMaterials.EntityMaterial	mat	What material to return.

Returns

TYPE	DESCRIPTION
UnityEngine.Material	A copy of the example material.

GetMaterial(GameMaterials.MiscMaterial)

Gets an instance of the given material.

Declaration

```
public static Material GetMaterial(GameMaterials.MiscMaterial mat)
```

Parameters

TYPE	NAME	DESCRIPTION
GameMaterials.MiscMaterial	mat	What material to return.

Returns

TYPE	DESCRIPTION
UnityEngine.Material	A copy of the example material.

GetMaterial(GameMaterials.ParticleMaterial)

Gets an instance of the given material.

Declaration

```
public static Material GetMaterial(GameMaterials.ParticleMaterial mat)
```

Parameters

TYPE	NAME	DESCRIPTION
GameMaterials.ParticleMaterial	mat	What material to return.

Returns

TYPE	DESCRIPTION
UnityEngine.Material	A copy of the example material.

Enum GameMaterials.BlockGhostMaterial

Available block ghost example materials. These use shaders from Shaders.BlockGhosts.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum BlockGhostMaterial
```

Fields

NAME	DESCRIPTION
Blade	
Bomb	
Cannon	
Decoupler	
Fireball	
SmallWood	
Spike	
Wheel	
WoodPanel	

Class GameMaterials.BlockGhostShaders

Container for block ghost shaders. Access via GameMaterials.Shaders.BlockGhosts.

Inheritance

[Object](#)

GameMaterials.BlockGhostShaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BlockGhostShaders
```

Constructors

BlockGhostShaders()

Declaration

```
public BlockGhostShaders()
```

Properties

TransparentDiffuse

One of the shaders used for ghosts. Example materials: GhostDecoupler, GhostCannon.

Declaration

```
public Shader TransparentDiffuse { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

TransparentDiffuseRim

One of the shaders used for ghosts. Example materials: GhostWheel, GhostBlade.

Declaration

```
public Shader TransparentDiffuseRim { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

TransparentDiffuseRimBump

One of the shaders used for ghosts. Example material: GhostFireball.

Declaration

```
public Shader TransparentDiffuseRimBump { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

TransparentSpecular

One of the shaders used for ghosts. Example material: GhostBomb.

Declaration

```
public Shader TransparentSpecular { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

TransparentSpecularBumped

One of the shaders used for ghosts. Example material: GhostSpike.

Declaration

```
public Shader TransparentSpecularBumped { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

TransparentWithRim

One of shaders used for ghosts. Example materials: GhostSmallWood, GhostWoodPanel.

Declaration

```
public Shader TransparentWithRim { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Enum GameMaterials.BlockMaterial

Available block example materials. These use shaders from Shaders.Blocks.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum BlockMaterial
```

Fields

NAME	DESCRIPTION
Camera	
Fireball	
Pin	
Wheel	
WoodenBlock	

Class GameMaterials.BlockShaders

Container for block shaders. Access via GameMaterials.Shaders.Blocks.

Inheritance

[Object](#)

GameMaterials.BlockShaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BlockShaders
```

Constructors

BlockShaders()

Declaration

```
public BlockShaders()
```

Properties

Camera

Special shader used for the camera block visual. Example material: Camera.

Declaration

```
public Shader Camera { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Fireball

Special shader used for the fireball. Example material: Fireball.

Declaration

```
public Shader Fireball { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Main

Shader used by most blocks. Example materials: WoodenBlock and Wheel.

Declaration

```
public Shader Main { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Pin

Special shader used for the pin block. Example material: Pin.

Declaration

```
public Shader Pin { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Enum GameMaterials.EntityMaterial

Available entity example materials. These use shaders from Shaders.Entities.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum EntityMaterial
```

Fields

NAME	DESCRIPTION
Barrel	
BuildZone	
Bush	
Cottage	
Cube	
DangerSign	
IvyLeaves	
StaticCloud	
Trigger	
Windmill	

Class GameMaterials.EntityShaders

Container for entity shaders. Access via GameMaterials.Shaders.Entities.

Inheritance

[Object](#)

GameMaterials.EntityShaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class EntityShaders
```

Constructors

EntityShaders()

Declaration

```
public EntityShaders()
```

Properties

AlphaBlended

Used for some of the virtual objects. Example materials: BuildZone, Trigger.

Declaration

```
public Shader AlphaBlended { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

BlockShader

Same as Shaders.Blocks.Main. This is also used for some entities. Example materials: Barrel, DangerSign.

Declaration

```
public Shader BlockShader { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Diffuse

Simple diffuse shader used mainly for the primitive level objects. Example material: Cube (Cylinder, Sphre, etc used the same material.)

Declaration

```
public Shader Diffuse { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Foliage

Shader used for foliage objects, like bushes and trees. Example materials: Bush, IvyLeaves.

Declaration

```
public Shader Foliage { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

PBS

Custom PBS shader used for some level objects. Example materials: Cottage, Windmill.

Declaration

```
public Shader PBS { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

VertexLit

Used for the static cloud. Example material: StaticCloud.

Declaration

```
public Shader VertexLit { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Enum GameMaterials.MiscMaterial

Available miscellaneous example materials. Thses use shaders from Shaders.Misc.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum MiscMaterial
```

Fields

NAME	DESCRIPTION
Loading	

Class GameMaterials.MiscShaders

Container for miscellaneous shaders. Access via GameMaterials.Shaders.Misc.

Inheritance

[Object](#)

GameMaterials.MiscShaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MiscShaders
```

Constructors

MiscShaders()

Declaration

```
public MiscShaders()
```

Properties

Loading

Shader used for the loading material for skins, among others. Example material: Loading.

Declaration

```
public Shader Loading { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

RenderTargetCutout

Shader used for making mirrors and portals. Example material: N/A

Declaration

```
public Shader RenderTextureCutout { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Enum GameMaterials.ParticleMaterial

Available particle example materials. Thsee use shaders from Shaders.Particles.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum ParticleMaterial
```

Fields

NAME	DESCRIPTION
Fire	
VacuumDust	
WaterRefract	

Class GameMaterials.ParticleShaders

Container for particle shaders. Access via GameMaterials.Shaders.Particles.

Inheritance

[Object](#)

GameMaterials.ParticleShaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ParticleShaders
```

Constructors

ParticleShaders()

Declaration

```
public ParticleShaders()
```

Properties

Additive

Example material: VacuumDust.

Declaration

```
public Shader Additive { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

AlphaBlended

Example material: Fire.

Declaration

```
public Shader AlphaBlended { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

StainedBumpDistort

Example material: WaterRefract.

Declaration

```
public Shader StainedBumpDistort { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Shader	

Class GameMaterials.Shaders

Access to commonly-used shaders in the game.

Inheritance

[Object](#)

GameMaterials.Shaders

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class Shaders
```

Properties

BlockGhosts

Shaders used for block ghosts.

Declaration

```
public static GameMaterials.BlockGhostShaders BlockGhosts { get; }
```

Property Value

TYPE	DESCRIPTION
GameMaterials.BlockGhostShaders	

Blocks

Shaders used for blocks.

Declaration

```
public static GameMaterials.BlockShaders Blocks { get; }
```

Property Value

TYPE	DESCRIPTION
GameMaterials.BlockShaders	

Entities

Shaders used for level editor entities.

Declaration

```
public static GameMaterials.EntityShaders Entities { get; }
```

Property Value

TYPE	DESCRIPTION
GameMaterials.EntityShaders	

Misc

Other shaders used for miscellaneous purposes.

Declaration

```
public static GameMaterials.MiscShaders Misc { get; }
```

Property Value

TYPE	DESCRIPTION
GameMaterials.MiscShaders	

Particles

Shaders used for particle systems.

Declaration

```
public static GameMaterials.ParticleShaders Particles { get; }
```

Property Value

TYPE	DESCRIPTION
GameMaterials.ParticleShaders	

Class GamePrefabs

Provides access to some useful prefabs of the game.

Inheritance

[Object](#)

GamePrefabs

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class GamePrefabs : MonoBehaviour
```

Constructors

GamePrefabs()

Declaration

```
public GamePrefabs()
```

Properties

Instance

Declaration

```
public static GamePrefabs Instance { get; }
```

Property Value

TYPE	DESCRIPTION
GamePrefabs	

Methods

InstantiateExplosion(GamePrefabs.ExplosionType, Vector3, Quaternion, Transform)

Creates a new explosion game object of the given type.

Declaration

```
public static GameObject InstantiateExplosion(GamePrefabs.ExplosionType type, Vector3 position = null, Quaternion rotation = null, Transform parent = null)
```

Parameters

TYPE	NAME	DESCRIPTION
GamePrefabs.ExplosionType	type	Explosion type.
UnityEngine.Vector3	position	Position of the new object.
UnityEngine.Quaternion	rotation	Rotation of the new object.

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Parent of the new object.

Returns

TYPE	DESCRIPTION
UnityEngine.GameObject	The created object.

InstantiateProjectile(GamePrefabs.ProjectileType, Vector3, Quaternion, Transform)

Creates a new projectile game object of the given type.

Declaration

```
public static GameObject InstantiateProjectile(GamePrefabs.ProjectileType type, Vector3 position = null, Quaternion rotation = null, Transform parent = null)
```

Parameters

TYPE	NAME	DESCRIPTION
GamePrefabs.ProjectileType	type	Projectile type.
UnityEngine.Vector3	position	Position of the new object.
UnityEngine.Quaternion	rotation	Rotation of the new object.
UnityEngine.Transform	parent	Parent of the new object.

Returns

TYPE	DESCRIPTION
UnityEngine.GameObject	The created object.

Enum GamePrefabs.ExlosionType

Available types of explosion prefabs.

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public enum ExlosionType
```

Fields

NAME	DESCRIPTION
Firework	
Large	
Small	

Enum GamePrefabs.ProjectileType

Available types of projectile prefabs.

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public enum ProjectileType
```

Fields

NAME	DESCRIPTION
BowArrow	
CannonBall	
CrossbowArrow	

Class Message

A network message from/for the ModNetworking class.

Inheritance

[Object](#)

Message

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Message
```

Properties

Sender

Declaration

```
public Player Sender { get; }
```

Property Value

TYPE	DESCRIPTION
Player	

Type

Declaration

```
public MessageType Type { get; }
```

Property Value

TYPE	DESCRIPTION
MessageType	

Methods

Encode()

Declaration

```
public byte[] Encode()
```

Returns

TYPE	DESCRIPTION
Byte[]	

GetData(Int32)

Declaration

```
public object GetData(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	

Returns

TYPE	DESCRIPTION
Object	

Class MessageType

A message type for messages to be sent with network messages using the ModNetworking class.

Inheritance

[Object](#)

MessageType

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MessageType
```

Properties

ID

Declaration

```
public byte ID { get; }
```

Property Value

TYPE	DESCRIPTION
Byte	

Methods

CreateMessage(Object[])

Declaration

```
public Message CreateMessage(params object[] objs)
```

Parameters

TYPE	NAME	DESCRIPTION
Object[]	objs	

Returns

TYPE	DESCRIPTION
Message	

Decode(Byte[])

Declaration

```
public Message Decode(byte[] buffer)
```

Parameters

TYPE	NAME	DESCRIPTION
Byte[]	buffer	

Returns

TYPE	DESCRIPTION
Message	

Class ModAssetBundle

An asset bundle mod resource.

Some care must be taken when using this class, see the Remarks section for details.

Inheritance

[Object](#)

[ModResource](#)

ModAssetBundle

Inherited Members

[ModResource.GetTexture\(String\)](#)

[ModResource.GetMesh\(String\)](#)

[ModResource.GetAudioClip\(String\)](#)

[ModResource.GetAssetBundle\(String\)](#)

[ModResource.Get\(ResourceReference\)](#)

[ModResource.CreateTextureResource\(String, String, Boolean, Boolean\)](#)

[ModResource.CreateMeshResource\(String, String, Boolean\)](#)

[ModResource.CreateAudioClipResource\(String, String, Boolean\)](#)

[ModResource.CreateAssetBundleResource\(String, String, Boolean\)](#)

[ModResource.SetOnObject\(GameObject, Action<GameObject>, Action\)](#)

[ModResource.TriggerOnLoad\(\)](#)

[ModResource.AllResourcesLoaded](#)

[ModResource.Type](#)

[ModResource.Name](#)

[ModResource.Available](#)

[ModResource.OnResourceLoaded](#)

[ModResource.OnAllResourcesLoaded](#)

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ModAssetBundle : ModResource
```

Remarks

Sometimes all assets of an AssetBundle are unloaded by Unity if the bundle hasn't been used for a while after it was first loaded.

The recommended way of using this class is to access the `AssetBundle` property to load any assets directly from an `OnLoad` callback. This will ensure that the bundle is correctly loaded at that point.

If you must access the bundle outside of the OnLoad callback and you or players run into issues where the bundle suddenly claims the assets you want to load are not present, use the `LoadAsset` wrappers of this class instead. They will reload the bundle when necessary, but this can incur a hit on framerate, so use OnLoad instead whenever possible.

Properties

AssetBundle

Declaration

```
public AssetBundle AssetBundle { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.AssetBundle	

Error

Declaration

```
public override string Error { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Overrides

[ModResource.Error](#)

HasError

Declaration

```
public override bool HasError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.HasError](#)

Loaded

Declaration

```
public override bool Loaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.Loaded](#)

Methods

LoadAsset<T>(Int32)

Returns the asset with the given index from LoadAllAssets, possibly reloading the bundle.

Use this only when necessary, see the Remarks section of the class documentation for more details.

Declaration

```
public T LoadAsset<T>(int index)
    where T : Object
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	

Returns

TYPE	DESCRIPTION
T	

Type Parameters

NAME	DESCRIPTION
T	

LoadAsset<T>(String)

Returns the asset with the given name, possibly reloading the bundle.

Use this only when necessary, see the Remarks section of the class documentation for more details.

Declaration

```
public T LoadAsset<T>(string name)
    where T : Object
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	

Returns

TYPE	DESCRIPTION
T	

Type Parameters

NAME	DESCRIPTION
T	

Events

OnLoad

Called when the resource has finished loading.

This is also called if there was an error loading the resource.

If a new handler is added to this and the resource has already finished loading, it is immediately called.

Declaration

```
public override event Action OnLoad
```


Event Type

TYPE	DESCRIPTION
Action	

Overrides

ModResource.OnLoad

Operators

Implicit(ModAssetBundle to AssetBundle)

Declaration

```
public static implicit operator AssetBundle(ModAssetBundle bundle)
```

Parameters

TYPE	NAME	DESCRIPTION
ModAssetBundle	bundle	

Returns

TYPE	DESCRIPTION
UnityEngine.AssetBundle	

Class ModAudioClip

An audio clip mod resource.

Inheritance

Object
ModResource
ModAudioClip

Inherited Members

- ModResource.GetTexture(String)
- ModResource.GetMesh(String)
- ModResource.GetAudioClip(String)
- ModResource.GetAssetBundle(String)
- ModResource.Get(ResourceReference)
- ModResource.CreateTextureResource(String, String, Boolean, Boolean)
- ModResource.CreateMeshResource(String, String, Boolean)
- ModResource.CreateAudioClipResource(String, String, Boolean)
- ModResource.CreateAssetBundleResource(String, String, Boolean)
- ModResource.SetOnObject(GameObject, Action<GameObject>, Action)
- ModResource.TriggerOnLoad()
- ModResource.AllResourcesLoaded
- ModResource.Type
- ModResource.Name
- ModResource.Available
- ModResource.OnResourceLoaded
- ModResource.OnAllResourcesLoaded
- ModResource.OnLoad

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class ModAudioClip : ModResource
```

Properties

AudioClip

Declaration

```
public AudioClip AudioClip { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.AudioClip	

Error

Declaration

```
public override string Error { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Overrides

[ModResource.Error](#)

HasError

Declaration

```
public override bool HasError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.HasError](#)

Loaded

Declaration

```
public override bool Loaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.Loaded](#)

Operators

Implicit(ModAudioClip to AudioClip)

Declaration

```
public static implicit operator AudioClip(ModAudioClip sound)
```

Parameters

TYPE	NAME	DESCRIPTION
ModAudioClip	sound	

Returns

TYPE	DESCRIPTION
UnityEngine.AudioClip	

Class ModBlockBehaviour

Base class for all custom behaviours attached to modded blocks by the mod loader. This includes BlockScripts as well as the behaviour of custom modules.

Inheritance

- Object
- ModBlockBehaviour
- BlockScript
- BlockModuleBehaviour<TModule>

Namespace: **Modding**
Assembly: Assembly-CSharp.dll

Syntax

```
public class ModBlockBehaviour : MonoBehaviour
```

Constructors

ModBlockBehaviour()

Declaration

```
public ModBlockBehaviour()
```

Properties

BlockBehaviour

The BlockBehaviour object for this block.

Declaration

```
public BlockBehaviour BlockBehaviour { get; }
```

Property Value

TYPE	DESCRIPTION
BlockBehaviour	

BlockId

The current effective ID of this block.

Declaration

```
public int BlockId { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

CanFlip

Whether this block has specified that it can flip.

Declaration

```
public bool CanFlip { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

DirectionArrow

Declaration

```
public Transform DirectionArrow { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

Flipped

Whether the block is currently flipped from its default.

Declaration

```
public bool Flipped { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

HasBurnedOut

Whether this block was on fire and is now burned out.

Declaration

```
public bool HasBurnedOut { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

HasRigidbody

Whether this block has a Rigidbody attached.

Declaration

```
public bool HasRigidbody { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsBurning

Whether this block is currently on fire.

Declaration

```
public bool IsBurning { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsDestroyed

Whether this block has been destroyed.

Declaration

```
public bool IsDestroyed { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsFrozen

Whether this block has been frozen.

Declaration

```
public bool IsFrozen { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSimulating

Whether the block is currently part of the simulation.

Declaration

```
public bool IsSimulating { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsStripped

Whether this instance of the block is a stripped version.

Declaration

```
public bool IsStripped { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Machine

The machine this block is a part of.

Declaration

```
public PlayerMachine Machine { get; }
```

Property Value

TYPE	DESCRIPTION
PlayerMachine	

MainVis

The object containing the main visuals for this block.

Declaration

```
public Transform MainVis { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

Renderer

Main Renderer of this block.

Declaration

```
public Renderer Renderer { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Renderer	

Rigidbody

Rigidbody attached to this block, if any.

Declaration

```
public Rigidbody Rigidbody { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Rigidbody	

ShowDebugVisuals

Value of the Debug element of the block in the XML definition.

Declaration

```
public bool ShowDebugVisuals { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

SimPhysics

Whether this instance of the block script should handle simulation physics.

Declaration

```
public bool SimPhysics { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

VisualController

The BlockVisualController handling this block.

Declaration

```
public BlockVisualController VisualController { get; }
```

Property Value

TYPE	DESCRIPTION
BlockVisualController	

Methods

AddColourSlider(MColourSlider)

Registers a colour slider. Needs to be called in the SafeAwake() method.

This will register this slider for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MColourSlider AddColourSlider(MColourSlider slider)
```


Parameters

TYPE	NAME	DESCRIPTION
MColourSlider	slider	Slider to register

Returns

TYPE	DESCRIPTION
MColourSlider	The slider registered

AddColourSlider(String, String, Color, Boolean)

Registers a colour slider. Needs to be called in the SafeAwake() method.

This will register this slider for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MColourSlider AddColourSlider(string displayName, string key, Color defaultValue, bool snapColors)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
UnityEngine.Color	defaultValue	Default value
Boolean	snapColors	

Returns

TYPE	DESCRIPTION
MColourSlider	The slider registered

AddCustom<T>(MCustom<T>)

Registers a custom mapper type. Needs to be called in the SafeAwake() method.

This will register this mapper type for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MCustom<T> AddCustom<T>(MCustom<T> custom)
```

Parameters

TYPE	NAME	DESCRIPTION
MCustom<T>	custom	Mapper type to register

Returns

TYPE	DESCRIPTION
MCustom<T>	The mapper type registered

Type Parameters

NAME	DESCRIPTION
T	

AddKey(MKey)

Registers a key. Needs to be called in the SafeAwake() method.

This will register this key for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

<pre>public MKey AddKey(MKey key)</pre>

Parameters

TYPE	NAME	DESCRIPTION
MKey	key	Key to register

Returns

TYPE	DESCRIPTION
MKey	The key registered

AddKey(String, String, KeyCode)

Registers a key. Needs to be called in the SafeAwake() method.

This will register this key for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

<pre>public MKey AddKey(string displayName, string key, KeyCode defaultKey)</pre>

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper

TYPE	NAME	DESCRIPTION
String	key	Key used for saving. Changing this breaks existing save files
UnityEngine.KeyCode	defaultKey	Default key code

Returns

TYPE	DESCRIPTION
MKey	The key registered

AddLimits(MLimits)

Registers a limit. Needs to be called in the SafeAwake() method.

This will register this limit for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MLimits AddLimits(MLimits limits)
```

Parameters

TYPE	NAME	DESCRIPTION
MLimits	limits	Limits to register

Returns

TYPE	DESCRIPTION
MLimits	The limits registered

AddLimits(String, String, Single, Single, Single, FauxTransform)

Registers a limit. Needs to be called in the SafeAwake() method.

This will register this limit for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MLimits AddLimits(string displayName, string key, float defaultMin, float defaultMax, float highestAngle, FauxTransform iconInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper

TYPE	NAME	DESCRIPTION
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultMin	Default min angle
Single	defaultMax	Default max angle
Single	highestAngle	The highest angle allowed
FauxTransform	iconInfo	The orientation and position of the icon

Returns

TYPE	DESCRIPTION
MLimits	The limits registered

AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)

Registers a limit. Needs to be called in the SafeAwake() method.

This will register this limit for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MLimits AddLimits(string displayName, string key, float defaultMin, float defaultMax, float highestAngle, FauxTransform iconInfo, ILimitsDisplay limitsDisplay)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultMin	Default min angle
Single	defaultMax	Default max angle
Single	highestAngle	The highest angle allowed

TYPE	NAME	DESCRIPTION
FauxTransform	iconInfo	The orientation and position of the icon
ILimitsDisplay	limitsDisplay	

Returns

TYPE	DESCRIPTION
MLimits	The limits registered

AddMenu(MMenu)

Registers a menu. Needs to be called in the SafeAwake() method.

This will register this menu for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MMenu AddMenu(MMenu menu)
```

Parameters

TYPE	NAME	DESCRIPTION
MMenu	menu	Menu to register

Returns

TYPE	DESCRIPTION
MMenu	The menu registered

AddMenu(String, Int32, List<String>, Boolean)

Registers a menu. Needs to be called in the SafeAwake() method.

This will register this menu for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MMenu AddMenu(string key, int defaultIndex, List<string> items, bool footerMenu = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	key	Key used for saving. Changing this breaks existing save files
Int32	defaultIndex	Default selection

TYPE	NAME	DESCRIPTION
List<String>	items	Available options
Boolean	footerMenu	Whether to display this menu at the bottom

Returns

TYPE	DESCRIPTION
MMenu	The menu registered

AddSlider(MSlider)

Registers a slider. Needs to be called in the SafeAwake() method.

This will register this slider for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MSlider AddSlider(MSlider slider)
```

Parameters

TYPE	NAME	DESCRIPTION
MSlider	slider	Slider to register

Returns

TYPE	DESCRIPTION
MSlider	The slider registered

AddSlider(String, String, Single, Single, Single)

Registers a slider. Needs to be called in the SafeAwake() method.

This will register this slider for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MSlider AddSlider(string displayName, string key, float defaultValue, float min, float max)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper

TYPE	NAME	DESCRIPTION
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultValue	Default value
Single	min	Minimum value
Single	max	Maximum value

Returns

TYPE	DESCRIPTION
MSlider	The slider registered

AddSliderUnclamped(String, String, Single, Single, Single)

Registers an unclamped slider. Needs to be called in the SafeAwake() method.

The slider's range will be determined by the min and max arguments but, unlike with AddSlider, it is possible to type in numbers outside that range when using AddSliderUnclamped.

This will register this slider for automatically saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MSlider AddSliderUnclamped(string displayName, string key, float defaultValue, float min, float max)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultValue	Default value
Single	min	Minimum slider value
Single	max	Maximum slider value

Returns

TYPE	DESCRIPTION
MSlider	

AddTeam(MTeam)

Registers a team. Needs to be called in the SafeAwake() method.

This will register this team for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MTeam AddTeam(MTeam team)
```

Parameters

TYPE	NAME	DESCRIPTION
MTeam	team	Team to register

Returns

TYPE	DESCRIPTION
MTeam	The team registered

AddTeam(String, String, MPTeam)

Registers a team. Needs to be called in the SafeAwake() method.

This will register this team for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MTeam AddTeam(string displayName, string key, MPTeam defaultTeam)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
MPTeam	defaultTeam	Default team

Returns

TYPE	DESCRIPTION
MTeam	The key registered

AddText(MText)

Registers text. Needs to be called in the SafeAwake() method.

This will register this text for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MText AddText(MText text)
```

Parameters

TYPE	NAME	DESCRIPTION
MText	text	Text to register

Returns

TYPE	DESCRIPTION
MText	The text registered

AddText(String, String, String)

Registers text. Needs to be called in the SafeAwake() method.

This will register this text for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MText AddText(string displayName, string key, string defaultText)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
String	defaultText	Default text

Returns

TYPE	DESCRIPTION
MText	The text registered

AddToggle(MToggle)

Registers a toggle. Needs to be called in the SafeAwake() method.

This will register this toggle for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MToggle AddToggle(MToggle toggle)
```

Parameters

TYPE	NAME	DESCRIPTION
MToggle	toggle	Toggle to register

Returns

TYPE	DESCRIPTION
MToggle	The toggle registered

AddToggle(String, String, Boolean)

Registers a toggle. Needs to be called in the SafeAwake() method.

This will register this toggle for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MToggle AddToggle(string displayName, string key, bool defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
Boolean	defaultValue	Default value

Returns

TYPE	DESCRIPTION
MToggle	The toggle registered

AddToggle(String, String, String, Boolean)

Registers a toggle. Needs to be called in the SafeAwake() method.

This will register this toggle for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MToggle AddToggle(string displayName, string key, string tooltipText, bool defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
String	tooltipText	text to appear in tooltip when hovering toggle
Boolean	defaultValue	Default value

Returns

TYPE	DESCRIPTION
MToggle	The toggle registered

AddValue(MValue)

Registers a value holder. Needs to be called in the SafeAwake() method.

This will register this value holder for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MValue AddValue(MValue valueHolder)
```

Parameters

TYPE	NAME	DESCRIPTION
MValue	valueHolder	Value holder to register

Returns

TYPE	DESCRIPTION
MValue	The value holder registered

AddValue(String, String, Single)

Registers a value holder. Needs to be called in the SafeAwake() method.

This will register this value holder for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MValue AddValue(string displayName, string key, float defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultValue	Default value

Returns

TYPE	DESCRIPTION
MValue	The value holder registered

AddValue(String, String, Single, Single, Single)

Registers a value holder. Needs to be called in the SafeAwake() method.

This will register this value holder for automatic saving and loading in the LoadMapperValues and SaveMapperValues methods.

Declaration

```
public MValue AddValue(string displayName, string key, float defaultValue, float min, float max)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	Name displayed in block mapper
String	key	Key used for saving. Changing this breaks existing save files
Single	defaultValue	Default value
Single	min	Minimum value
Single	max	Maximum value

Returns

TYPE	DESCRIPTION
MValue	The value holder registered

BuildingFixedUpdate()

Like `BuildingUpdate` but called every `FixedUpdate`.

Declaration

```
public virtual void BuildingFixedUpdate()
```

BuildingLateUpdate()

Like `BuildingUpdate` but called every `LateUpdate`.

Declaration

```
public virtual void BuildingLateUpdate()
```

BuildingUpdate()

Called every frame while in build mode.

MP: Called on the server and all clients, on each block that is part of a machine in build mode.

Declaration

```
public virtual void BuildingUpdate()
```

OnBlockPlaced()

Called when a block is placed, on that block's instance.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnBlockPlaced()
```

OnLoad(XDataHolder)

Declaration

```
public virtual void OnLoad(XDataHolder data)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataHolder	data	

OnPrefabCreation()

Called on the prefab once it's been created and fully initialized. This happens when the first level that is not a menu is loaded.

This is also called on the stripped block prefab, right after the call on the normal one.

Use this to modify the prefab if you can't achieve what you want using the options exposed via the XML file.

Declaration

```
public virtual void OnPrefabCreation()
```

OnReloadAmmo(ref Int32, AmmoType, Boolean, Boolean)

Override to make it possible to reload the block using the logic system.

The parameters correspond directly to the options for the reload event in the UI.

Declaration

```
public virtual void OnReloadAmmo(ref int units, AmmoType type, bool setAmmo, bool eachBlock)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	units	Number entered in the UI. Should be the number of shots where applicable. The amount of ammo "used up" by this block should be subtracted from this.
AmmoType	type	Ammunition type, set with the icon in the UI.
Boolean	setAmmo	True if the equal sign is selected, false if the addition sign is selected.
Boolean	eachBlock	True if "each weapon" is selected, false if "machine" is selected.

OnSave(XDataHolder)

Declaration

```
public virtual void OnSave(XDataHolder data)
```

Parameters

TYPE	NAME	DESCRIPTION
XDataHolder	data	

OnSimulateCollisionEnter(Collision)

Simulation-only wrapper around OnCollisionEnter.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateCollisionEnter(Collision collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collision	collision	

OnSimulateCollisionExit(Collision)

Simulation-only wrapper around OnCollisionExit.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateCollisionExit(Collision collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collision	collision	

OnSimulateCollisionStay(Collision)

Simulation-only wrapper around OnCollisionStay.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateCollisionStay(Collision collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collision	collision	

OnSimulateParticleCollision(GameObject)

Simulation-only wrapper around OnParticleCollision.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateParticleCollision(GameObject other)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	other	

OnSimulateStart()

Called on the simulation clone of the block when the simulation was started.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateStart()
```

OnSimulateStop()

Called on the simulation clone of the block when the simulation is stopped.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateStop()
```

OnSimulateTriggerEnter(Collider)

Simulation-only wrapper around OnTriggerEnter.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateTriggerEnter(Collider collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collider	collision	

OnSimulateTriggerExit(Collider)

Simulation-only wrapper around OnTriggerExit.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateTriggerExit(Collider collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collider	collision	

OnSimulateTriggerStay(Collider)

Simulation-only wrapper around OnTriggerStay.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnSimulateTriggerStay(Collider collision)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collider	collision	

OnStartBurning()

Called when the block is set on fire.

MP: Called on the server and all clients.

Declaration

```
public virtual void OnStartBurning()
```

OnStopBurning(Boolean)

Called when the block stops burning.

MP: Called on the server and all clients.

Declaration


```
public virtual void OnStopBurning(bool doused)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	doused	true if the block stopped burning because it was extinguished, false if it stopped because it burned out.

SafeAwake()

Called every time a new instance of the script is initialized, after some necessary initialization by the game was done.

Use this instead of the normal Unity Awake() method.

Declaration

```
public virtual void SafeAwake()
```

SimulateFixedUpdateAlways()

Like `SimulateUpdateAlways` but called every FixedUpdate.

Declaration

```
public virtual void SimulateFixedUpdateAlways()
```

SimulateFixedUpdateClient()

Like `SimulateUpdateClient` but called every FixedUpdate.

Declaration

```
public virtual void SimulateFixedUpdateClient()
```

SimulateFixedUpdateHost()

Like `SimulateUpdateHost` but called every FixedUpdate.

Declaration

```
public virtual void SimulateFixedUpdateHost()
```

SimulateLateUpdateAlways()

Like `SimulateUpdateAlways` but called every LateUpdate.

Declaration

```
public virtual void SimulateLateUpdateAlways()
```

SimulateLateUpdateClient()

Like `SimulateUpdateClient` but called every LateUpdate.

Declaration

```
public virtual void SimulateLateUpdateClient()
```

SimulateLateUpdateHost()

Like `SimulateUpdateHost` but called every LateUpdate.

Declaration

```
public virtual void SimulateLateUpdateHost()
```

SimulateUpdateAlways()

Called every frame while in simulation mode.

MP: Called on each instance connected (including the host).

Declaration

```
public virtual void SimulateUpdateAlways()
```

SimulateUpdateClient()

Called every frame while in simulation mode.

Not called in singleplayer.

MP: Called on every game instance receiving the simulation state from another instance, i.e. on the instances that should not simulate physics for this block. This means connected clients for each machine in global sim, as well as the server and clients for each machine in local sim on another instance.

Declaration

```
public virtual void SimulateUpdateClient()
```

SimulateUpdateHost()

Called every frame while in simulation mode.

MP: Called on the game instance that is handling physics for this block. This is the host for machines in global sim, or a client when simulating their own machine locally.

Declaration

```
public virtual void SimulateUpdateHost()
```

Class ModConsole

Allows mods to interact with the in-game console. Output to the console using the Log method and add your own console commands using the RegisterCommand method.

Inheritance

Object

ModConsole

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModConsole
```

Methods

Log(String)

Outputs the given string to the console.

Declaration

```
public static void Log(string log)
```

Parameters

TYPE	NAME	DESCRIPTION
String	log	String to output.

Log(String, Object[])

Formats the given string and the given objects like string.Format does and outputs it to the console.

Declaration

```
public static void Log(string format, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
String	format	Format string.
Object[]	args	Objects to format.

RegisterCommand(String, CommandHandler, String)

Registers a new console command.

Declaration

```
public static void RegisterCommand(string command, CommandHandler handler, string help)
```

Parameters

TYPE	NAME	DESCRIPTION
String	command	Name of the command to register.
CommandHandler	handler	Function to call when the command is executed.
String	help	Help text to display in the help command output.

Remarks

Usually, the command can be executed by the user by entering `command`, `<mod name>:command`, or `<mod id>:command`.

One mod may not attempt to register the same command twice, otherwise an `ArgumentException` is thrown.

A mod may not register a command that is already implemented in the base game, otherwise an `ArgumentException` is thrown.

If two or more mods register the same command, only the command syntax including the mod name or id can be used. The user will be pointed to the registered variants when attempting to use such an 'overloaded' command.

Similarly, if two or more mods with the same name register the same command, only the syntax including the mod id can be used.

The command name may not contain a colon (':') and the help text may not be empty, otherwise an `ArgumentException` is thrown.

The handler will be called whenever the command is executed. Its only parameter is a string array containing the arguments passed on the console. Arguments are delimited by spaces. Arguments that should contain spaces can be wrapped in quotes, this is automatically handled by the system and the arguments array will only contain one string for such an argument.

Class ModEntryPoint

Every Assembly can contain one ModEntryPoint. If it does and is listed in the Mod manifest, an entry point will be instantiated by the mod loader and its `OnLoad` method called when the mod is loaded.

Inheritance

Object

ModEntryPoint

Namespace: `Modding`

Assembly: `Assembly-CSharp.dll`

Syntax

```
public abstract class ModEntryPoint
```

Constructors

ModEntryPoint()

Declaration

```
protected ModEntryPoint()
```

Methods

IsRequiredForLevel(Level)

Called to determine if the mod is a necessary part of a level. If this returns true, a warning will be presented to the user when trying to load the level without the mod installed.

If the method is not overridden, it always returns false.

Note: This is intended for when the mod changes other objects or logic elements such that it will also be necessary to load them correctly again. If the level contains a modded entity or logic element, the corresponding mod will automatically be marked as required, disregarding what this method returns.

Declaration

```
public virtual bool IsRequiredForLevel(Level editor)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>Level</code>	editor	

Returns

TYPE	DESCRIPTION
<code>Boolean</code>	

IsRequiredForMachine(PlayerMachineInfo)

Called to determine if the mod is a necessary part of a machine. If this returns true, a warning will be presented to the user when trying to load the machine without the mod installed.

If the method is not overridden, it always returns false.

Note: This is intended for when the mod changes other blocks such that it will also be necessary to load them correctly again. If the machine contains a modded block, the corresponding mod will automatically be marked as required, disregarding what this

method returns.

Declaration

```
public virtual bool IsRequiredForMachine(PlayerMachineInfo machine)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachineInfo	machine	The machine.

Returns

TYPE	DESCRIPTION
Boolean	If the mod is required for the machine.

OnBlockPrefabCreation(Int32, GameObject, GameObject)

Enables customization of block prefabs.

Declaration

```
public virtual void OnBlockPrefabCreation(int blockId, GameObject prefab, GameObject ghost)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	blockId	Local ID of the block
UnityEngine.GameObject	prefab	Prefab object
UnityEngine.GameObject	ghost	Ghost object

Remarks

Called for every block the mod adds, after its prefab has been created. Allows making changes to the prefab and the ghost that are not possible using the XML declarations.

This is called before the OnPrefabCreation method of BlockScript, so when writing the block script you may assume that any changes made here have already been applied.

Note: Unlike OnPrefabCreation of BlockScript, this is called before a block has been fully initialized. Most notably, the block does not have an ID assigned yet, which may lead to unexpected behaviour.

If you encounter such behaviour, try using OnPrefabCreation instead of this!

OnEntityPrefabCreation(Int32, GameObject)

Enables customization of entity prefabs.

Declaration

```
public virtual void OnEntityPrefabCreation(int entityId, GameObject prefab)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	entityId	Local ID of the entity.
UnityEngine.GameObject	prefab	Prefab object

Remarks

Called for every entity the mod adds, after its prefab has been created. Allows making changes to the prefab that are not possible using the XML declarations.

OnLoad()

Called when the mod is first loaded.

Declaration

```
public virtual void OnLoad()
```

Remarks

Modes marked with the LoadInTitleScreen element are usually loaded when the game is started.

All other mods are only loaded once the the player enters either the multiverse or one of the singleplayer levels.

Note that even mods marked with LoadInTitle can be possibly be loaded later than the first game start, for example if a disabled LoadInTitle mod is enabled in the Multiverse Join screen, it will be loaded then.

Class ModEvents

Allows registering behaviour for custom events using the RegisterCallback method.

Inheritance

[Object](#)

ModEvents

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModEvents
```

Methods

RegisterCallback(Int32, ModEvents.OnEventExecute)

Register an `OnEventExecute` callback for your event. The callback will be called every time an instance of your event is activated by the logic system.

MP: The callback is only called on the instance simulating (i.e. the server in global sim, the simulating client in local sim).

Declaration

```
public static void RegisterCallback(int id, ModEvents.OnEventExecute callback)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	ID of your event, as defined in the XML declaration.
ModEvents.OnEventExecute	callback	Callback to register.

Delegate ModEvents.OnEventExecute

Delegate for callbacks registered to `RegisterCallback`.

Namespace: `Modding`

Assembly: `Assembly-CSharp.dll`

Syntax

```
public delegate void OnEventExecute(LogicChain logic, IDictionary<string, EventProperty> properties);
```

Parameters

TYPE	NAME	DESCRIPTION
<code>LogicChain</code>	logic	The logic chain the event is a part of.
<code>IDictionary<String, EventProperty></code>	properties	Values of the event's properties.

Remarks

Properties contains the values of all the properties specified in the Event's definition.

Constructors

`OnEventExecute(Object, IntPtr)`

Declaration

```
public OnEventExecute(object object, IntPtr method)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>Object</code>	object	
<code>IntPtr</code>	method	

Methods

`BeginInvoke(LogicChain, IDictionary<String, EventProperty>, AsyncCallback, Object)`

Declaration

```
public virtual IAsyncResult BeginInvoke(LogicChain logic, IDictionary<string, EventProperty> properties, AsyncCallback callback, object object)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>LogicChain</code>	logic	
<code>IDictionary<String, EventProperty></code>	properties	
<code>AsyncCallback</code>	callback	

TYPE	NAME	DESCRIPTION
Object	object	

Returns

TYPE	DESCRIPTION
IAsyncResult	

EndInvoke(IAsyncResult)

Declaration

```
public virtual void EndInvoke(IAsyncResult result)
```

Parameters

TYPE	NAME	DESCRIPTION
IAsyncResult	result	

Invoke(LogicChain, IDictionary<String, EventProperty>)

Declaration

```
public virtual void Invoke(LogicChain logic, IDictionary<string, EventProperty> properties)
```

Parameters

TYPE	NAME	DESCRIPTION
LogicChain	logic	
IDictionary<String, EventProperty>	properties	

Class ModIO

Provides methods for IO since large parts of the System.IO namespace are blocked for security reasons.

Inheritance

Object

ModIO

Namespace: **Modding**

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModIO
```

Remarks

You can only access files located in your mod's directory or its data directory. with this class. All paths are relative to these directories. Which directory is used as base is determined by the data (boolean) parameter that all methods that take a path have. It defaults to false, i.e. using the mod root directory.

Use the data directory for files that are meant to be stored persistently, as file in the mod directory could be overwritten by Steam at any time if the mod is installed from the workshop.

Apart from that, methods are mostly analogous to the methods of the same name in the `System.IO.File`, `System.IO.Directory`, and `System.Net.WebClient` classes. The methods corresponding to WebClient methods take event handlers as (optional) arguments, instead of exposing the WebClient object itself. See documentation of that for explanation of parameters.

Methods

AppendText(String, Boolean)

Declaration

```
public static TextWriter AppendText(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
TextWriter	

CreateDirectory(String, Boolean)

Declaration

```
public static void CreateDirectory(string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	path	

TYPE	NAME	DESCRIPTION
Boolean	data	

CreateText(String, Boolean)

Declaration

```
public static TextWriter CreateText(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
TextWriter	

DeleteDirectory(String, Boolean, Boolean)

Declaration

```
public static void DeleteDirectory(string path, bool recursive, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	path	
Boolean	recursive	
Boolean	data	

DeleteFile(String, Boolean)

Declaration

```
public static void DeleteFile(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

DeserializeXml<T>(String, Boolean, Boolean)

Deserializes an XML file using the Modding.Serialization system for optional validation.

Declaration

```
public static T DeserializeXml<T>(string filePath, bool data = false, bool validate = true)
    where T : Element
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	Path to the XML file.
Boolean	data	Whether the path is relative to the data directory or the mod directory.
Boolean	validate	Whether the deserialized object should be validated.

Returns

TYPE	DESCRIPTION
T	The deserialized object or null if an error was encountered.

Type Parameters

NAME	DESCRIPTION
T	Type of object to be deserialized.

Remarks

In case of errors, they are logged to the in-game console and `null` is returned.

DownloadData(String)

Declaration

```
public static byte[] DownloadData(string url)
```

Parameters

TYPE	NAME	DESCRIPTION
String	url	

Returns

TYPE	DESCRIPTION
Byte[]	

DownloadDataAsync(Uri, DownloadDataCompletedEventHandler, DownloadProgressChangedEventHandler)

Declaration

```
public static void DownloadDataAsync(Uri uri, DownloadDataCompletedEventHandler downloadComplete,
DownloadProgressChangedEventHandler progressChanged = null)
```

Parameters

TYPE	NAME	DESCRIPTION
Uri	uri	
DownloadDataCompletedEventHandler	downloadComplete	
DownloadProgressChangedEventHandler	progressChanged	

DownloadFile(String, String, AsyncCompletedEventHandler, DownloadProgressChangedEventHandler, Boolean)

Declaration

```
public static void DownloadFile(string url, string filePath, AsyncCompletedEventHandler downloadCompleted = null, DownloadProgressChangedEventHandler progressChanged = null, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	url	
String	filePath	
AsyncCompletedEventHandler	downloadCompleted	
DownloadProgressChangedEventHandler	progressChanged	
Boolean	data	

DownloadFileAsync(Uri, String, AsyncCompletedEventHandler, DownloadProgressChangedEventHandler, Boolean)

Declaration

```
public static void DownloadFileAsync(Uri uri, string filePath, AsyncCompletedEventHandler downloadComplete = null, DownloadProgressChangedEventHandler progressChanged = null, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
Uri	uri	
String	filePath	
AsyncCompletedEventHandler	downloadComplete	
DownloadProgressChangedEventHandler	progressChanged	
Boolean	data	

DownloadString(String)

Declaration

```
public static string DownloadString(string url)
```

Parameters

TYPE	NAME	DESCRIPTION
String	url	

Returns

TYPE	DESCRIPTION
String	

DownloadStringAsync(Uri, DownloadStringCompletedEventHandler, DownloadProgressChangedEventHandler)

Declaration

```
public static void DownloadStringAsync(Uri uri, DownloadStringCompletedEventHandler downloadComplete,
DownloadProgressChangedEventHandler progressChanged = null)
```

Parameters

TYPE	NAME	DESCRIPTION
Uri	uri	
DownloadStringCompletedEventHandler	downloadComplete	
DownloadProgressChangedEventHandler	progressChanged	

ExistsDirectory(String, Boolean)

Declaration

```
public static bool ExistsDirectory(string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	path	
Boolean	data	

Returns

TYPE	DESCRIPTION
Boolean	

ExistsFile(String, Boolean)

Declaration

```
public static bool ExistsFile(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	

TYPE	NAME	DESCRIPTION
Boolean	data	

Returns

TYPE	DESCRIPTION
Boolean	

GetDirectories(String, Boolean)

Declaration

```
public static string[] GetDirectories(string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	path	
Boolean	data	

Returns

TYPE	DESCRIPTION
String[]	

GetFiles(String, Boolean)

Declaration

```
public static string[] GetFiles(string dirPath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	dirPath	
Boolean	data	

Returns

TYPE	DESCRIPTION
String[]	

GetFiles(String, String, Boolean)

Declaration

```
public static string[] GetFiles(string dirPath, string searchPattern, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	dirPath	
String	searchPattern	
Boolean	data	

Returns

TYPE	DESCRIPTION
String[]	

Open(String, FileMode, Boolean, FileAccess, FileShare)

Declaration

```
public static Stream Open(string filePath, FileMode mode, bool data = false, FileAccess access =
FileAccess.Read | FileAccess.Write | FileAccess.ReadWrite, FileShare share = FileShare.None)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
FileMode	mode	
Boolean	data	
FileAccess	access	
FileShare	share	

Returns

TYPE	DESCRIPTION
Stream	

OpenFolderInFileBrowser(String, Boolean)

Declaration

```
public static void OpenFolderInFileBrowser(string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	path	
Boolean	data	

OpenRead(String, Boolean)

Declaration

```
public static Stream OpenRead(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
Stream	

OpenText(String, Boolean)

Declaration

```
public static TextReader OpenText(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
TextReader	

ReadAllBytes(String, Boolean)

Declaration

```
public static byte[] ReadAllBytes(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
Byte[]	

ReadAllLines(String, Boolean)

Declaration

```
public static string[] ReadAllLines(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
String[]	

ReadAllLines(String, Encoding, Boolean)

Declaration

```
public static string[] ReadAllLines(string filePath, Encoding encoding, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Encoding	encoding	
Boolean	data	

Returns

TYPE	DESCRIPTION
String[]	

ReadAllText(String, Boolean)

Declaration

```
public static string ReadAllText(string filePath, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Boolean	data	

Returns

TYPE	DESCRIPTION
String	

ReadAllText(String, Encoding, Boolean)

Declaration

```
public static string ReadAllText(string filePath, Encoding encoding, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Encoding	encoding	
Boolean	data	

Returns

TYPE	DESCRIPTION
String	

SerializeXml<T>(T, String, Boolean)

Serializes an Element-derived object to an XML file. If the file path already exists, the file is overwritten.

The serialized XML output will contain lineNumber and linePosition attributes that cannot be avoided due to limitations of the serialization system. These attributes will not interfere with deserialization.

Note: The mod loader itself makes heavy use of the Deserialization system but doesn't directly use XML serialization. Because of this there are multiple tricks for deserialization that make it more reliable while serialization hasn't received the same attention. Use at your own risk.

Declaration

```
public static void SerializeXml<T>(T obj, string filePath, bool data = false)
    where T : Element
```

Parameters

TYPE	NAME	DESCRIPTION
T	obj	
String	filePath	
Boolean	data	

Type Parameters

NAME	DESCRIPTION
T	

WriteAllBytes(String, Byte[], Boolean)

Declaration

```
public static void WriteAllBytes(string filePath, byte[] bytes, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
Byte[]	bytes	
Boolean	data	

WriteAllLines(String, String[], Boolean)

Declaration

```
public static void WriteAllLines(string filePath, string[] lines, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
String[]	lines	
Boolean	data	

WriteAllText(String, String, Boolean)

Declaration

```
public static void WriteAllText(string filePath, string text, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	filePath	
String	text	
Boolean	data	

Class ModKey

Represents a keybinding that is managed by the mod loader. Keys are defined in the mod manifest.

Properties correspond to properties of the same name in the MKey class. This naming is different to that of the UnityEngine.Input class!

If the modifier is KeyCode.None, the keybinding only considers the trigger.

If the trigger is KeyCode.None, the keybinding is considered to be disabled, and all the properties always return false.

Inheritance

[Object](#)

ModKey

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ModKey
```

Constructors

ModKey()

Declaration

```
public ModKey()
```

Properties

IsDown

Whether the key (combinations) is currently pressed.

Declaration

```
public bool IsDown { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsPressed

Whether the key (combination) was pressed down this frame.

Declaration

```
public bool IsPressed { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsReleased

Whether the key (combination) was released this frame.

Declaration

```
public bool IsReleased { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Class ModKeys

Keybindings managed by the mod loader. They can be rebound by the player via config files and automatically handle the possibility of being a combination of a modifier and a trigger key.

Inheritance

[Object](#)

ModKeys

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModKeys
```

Methods

GetKey(String)

Get a reference to the specified ModKey defined in the mod manifest.

Declaration

```
public static ModKey GetKey(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the key

Returns

TYPE	DESCRIPTION
ModKey	The ModKey instance

Class ModMesh

A mesh mod resource.

Inheritance

[Object](#)
[ModResource](#)
ModMesh

Inherited Members

- [ModResource.GetTexture\(String\)](#)
- [ModResource.GetMesh\(String\)](#)
- [ModResource.GetAudioClip\(String\)](#)
- [ModResource.GetAssetBundle\(String\)](#)
- [ModResource.Get\(ResourceReference\)](#)
- [ModResource.CreateTextureResource\(String, String, Boolean, Boolean\)](#)
- [ModResource.CreateMeshResource\(String, String, Boolean\)](#)
- [ModResource.CreateAudioClipResource\(String, String, Boolean\)](#)
- [ModResource.CreateAssetBundleResource\(String, String, Boolean\)](#)
- [ModResource.SetOnObject\(GameObject, Action<GameObject>, Action\)](#)
- [ModResource.TriggerOnLoad\(\)](#)
- [ModResource.AllResourcesLoaded](#)
- [ModResource.Type](#)
- [ModResource.Name](#)
- [ModResource.Available](#)
- [ModResource.OnResourceLoaded](#)
- [ModResource.OnAllResourcesLoaded](#)
- [ModResource.OnLoad](#)

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class ModMesh : ModResource
```

Properties

Error

Declaration

```
public override string Error { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Overrides

[ModResource.Error](#)

HasError

Declaration

```
public override bool HasError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.HasError](#)

Loaded

Declaration

```
public override bool Loaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.Loaded](#)

Mesh

Declaration

```
public Mesh Mesh { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Mesh	

Methods

SetOnObject(GameObject, MeshReference, Action<GameObject>, Action)

SetOnObject is a utility method that easily allows handling long resource loading times. This overload also sets the transform values of a MeshReference.

Declaration

```
public void SetOnObject(GameObject go, MeshReference r, Action<GameObject> postSetAction = null, Action prefabPostSetAction = null)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	go	Object on which to set the resource.
MeshReference	r	MeshReference whose transform values should be set.
Action <UnityEngine.GameObject>	postSetAction	Callback for every object that is set.

TYPE	NAME	DESCRIPTION
Action	prefabPostSetAction	Callback for the original object.

Remarks

When SetOnObject is called on an object, an OnLoad handler for the resource is installed that will automatically set the appropriate value (renderer.material.mainTexture for a ModTexture, meshFilter.mesh for a ModMesh, and audioSource.clip for a ModAudioClip) when the resource is done loading.

However, the major benefit of using SetOnObject is that it will also automatically handle the given GameObject being instantiated/copied. All instances of the object will also automatically receive the resource.

The two optional callbacks can be used to perform additional actions that are necessary to correctly set the resource. The first one is called for every object on which the resource is set (including the original object), and the second one only for the object originally passed into the function.

In this overload, an additional MeshReference object is passed in. Its transform values (Position, Rotation, Scale) will also automatically be set, in addition to the mesh.

Operators

Implicit(ModMesh to Mesh)

Declaration

```
public static implicit operator Mesh(ModMesh mesh)
```

Parameters

TYPE	NAME	DESCRIPTION
ModMesh	mesh	

Returns

TYPE	DESCRIPTION
UnityEngine.Mesh	

Class ModNetworking

Networking system for mods.

Inheritance

[Object](#)

ModNetworking

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModNetworking
```

Remarks

Defining MessageTypes

To use network messages from your mod, you first need to define the possible message types. To define a message type, call `CreateMessageType`. If you want to attach data to your messages, pass corresponding `DataType` parameters. The function returns a `MessageType` object which you will need to keep around.

Sending Messages

To create messages, call `CreateMessage` on the corresponding `MessageType` object. Pass along values for all data parameters you have defined when creating the message type. To send a message once you have created it, call `SendTo` or `SendToAll` and pass the `Message` object. The receive callbacks will not be called on the instance sending the message.

Receiving Messages

There's two ways to act on received messages:

1. You can add a handler to the `MessageReceived` event. It will get called for every message your mod receives.
2. You can add per-`MessageType` event handlers using the `Callbacks` property. The `Player` property of the `Message` object you receive is the sender of the message.

Examples

Create a message type:

```
public static MessageType TestType;
void OnLoad() {
    TestType = ModNetworking.CreateMessageType(DataType.Single);
}
```

Register event handlers:

```
ModNetworking.MessageReceived += message => {
    // This will get called for all message types.
};
ModNetworking.Callbacks[TestType] += message => {
    // This will only get called for TestType messages.
};
```

Send a message:

```
var message = TestType.CreateMessage(20f);
ModNetworking.SendToAll(message);
```

Fields

Callbacks

A set of events for each registered message type. Index by message type, then use the result like a normal event, i.e.

```
+= someHandler;
```

Declaration

```
public static ModNetworking.CallbacksWrapper Callbacks
```

Field Value

TYPE	DESCRIPTION
Modding.ModNetworking.CallbacksWrapper	

Properties

IsNetworkingReady

Whether the connection has been initialized far enough so that mod networking can work. If any networking methods are used while this is false, they will not have any effect.

Declaration

```
public static bool IsNetworkingReady { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

CreateMessageType(DataType[])

Creates a new message type with the specified data attached. Note that order matters in creating message type, if the message types are created in different order on different instances, receiving messages will not work correctly.

Declaration

```
public static MessageType CreateMessageType(params DataType[] types)
```

Parameters

TYPE	NAME	DESCRIPTION
DataType[]	types	

Returns

TYPE	DESCRIPTION
MessageType	

SendInSimulation(Message)

Sends a message to every instance affected by the current simulation.

Declaration

```
public static void SendInSimulation(Message message)
```

Parameters

TYPE	NAME	DESCRIPTION
Message	message	

Remarks

Does not do anything when the local instance is in local simulation. Otherwise, sends the message to every player that is participating in global simulation or is in build mode.

SendTo(Player, Message)

Sends a message to a specific player.

Declaration

```
public static void SendTo(Player player, Message message)
```

Parameters

TYPE	NAME	DESCRIPTION
Player	player	
Message	message	

SendToAll(Message)

Sends a message to all instances in the current multiplayer game.

Declaration

```
public static void SendToAll(Message message)
```

Parameters

TYPE	NAME	DESCRIPTION
Message	message	

SendToHost(Message)

Sends a message to the host.

Declaration

```
public static void SendToHost(Message message)
```

Parameters

TYPE	NAME	DESCRIPTION
Message	message	

Events

MessageReceived

Event that fires whenever a message is received. Called for all message types.

Declaration

```
public static event Action<Message> MessageReceived
```

Event Type

TYPE	DESCRIPTION
Action<Message>	

Class ModResource

This class provides access to the resources listed in the Resources section of the mod manifest via static members and is the base class for the objects representing a resource.

Inheritance

- [Object](#)
- [ModResource](#)
- [ModAssetBundle](#)
- [ModAudioClip](#)
- [ModMesh](#)
- [ModTexture](#)

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class ModResource
```

Constructors

ModResource()

Declaration

```
protected ModResource()
```

Properties

AllResourcesLoaded

Determines whether all the resources of the calling mod have finished loading.

Declaration

```
public static bool AllResourcesLoaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Available

Whether the resource is ready for use, that is, it has finished loading without errors.

Declaration

```
public virtual bool Available { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Error

A description of the error that occurred while loading this resource.

An empty string if there was no error.

Declaration

```
public abstract string Error { get; }
```

Property Value

TYPE	DESCRIPTION
String	

HasError

Whether there was an error loading the resource.

Declaration

```
public abstract bool HasError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

See Also

[Error](#)

Loaded

Whether this resource has finished loading. Is also true if there was an error loading the resource.

Declaration

```
public abstract bool Loaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Name

The name of this resource.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Type

The type of this resource.

Declaration

```
public ModResource.ResourceType Type { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
ModResource.ResourceType	

Methods

CreateAssetBundleResource(String, String, Boolean)

Creates a new AssetBundle resource at runtime.

The resource will then work just like one that was declared in the Mod.xml file.

The path is either relative to the mod's data directory, or the mod's *root* directory. It is *not* relative to the Resources/ directory, unlike the paths in the Mod.xml file.

Declaration

```
public static ModAssetBundle CreateAssetBundleResource(string name, string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.
String	path	Relative path to the resource.
Boolean	data	Whether the path is relative to the data directory.

Returns

TYPE	DESCRIPTION
ModAssetBundle	Newly created ModAssetBundle resource.

CreateAudioClipResource(String, String, Boolean)

Creates a new AudioClip resource at runtime.

The resource will then work just like one that was declared in the Mod.xml file.

The path is either relative to the mod's data directory, or the mod's *root* directory. It is *not* relative to the Resources/ directory, unlike the paths in the Mod.xml file.

Declaration

```
public static ModAudioClip CreateAudioClipResource(string name, string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.
String	path	Relative path to the resource.
Boolean	data	Whether the path is relative to the data directory.

Returns

TYPE	DESCRIPTION
ModAudioClip	Newly created ModAudioClip resource.

CreateMeshResource(String, String, Boolean)

Creates a new Mesh resource at runtime.

The resource will then work just like one that was declared in the Mod.xml file.

The path is either relative to the mod's data directory, or the mod's *root* directory. It is *not* relative to the Resources/ directory, unlike the paths in the Mod.xml file.

Declaration

```
public static ModMesh CreateMeshResource(string name, string path, bool data = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.
String	path	Relative path to the resource.
Boolean	data	Whether the path is relative to the data directory.

Returns

TYPE	DESCRIPTION
ModMesh	Newly created ModMesh resource.

CreateTextureResource(String, String, Boolean, Boolean)

Creates a new Texture resource at runtime.

The resource will then work just like one that was declared in the Mod.xml file.

The path is either relative to the mod's data directory, or the mod's *root* directory. It is *not* relative to the Resources/ directory, unlike the paths in the Mod.xml file.

Declaration

```
public static ModTexture CreateTextureResource(string name, string path, bool data = false, bool readable = false)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.
String	path	Relative path to the resource.
Boolean	data	Whether the path is relative to the data directory.
Boolean	readable	Whether the texture should be loaded as readable.

Returns

TYPE	DESCRIPTION
ModTexture	Newly created ModTexture resource.

Get(ResourceReference)

Returns the resource specified by the given reference.

Declaration

```
public static ModResource Get(ResourceReference reference)
```

Parameters

TYPE	NAME	DESCRIPTION
ResourceReference	reference	Reference to the resource to fetch.

Returns

TYPE	DESCRIPTION
ModResource	The resource, or null if no resource with the given name could be found.

Exceptions

TYPE	CONDITION
InvalidOperationException	If called from an assembly not listed in the mod manifest.

GetAssetBundle(String)

Returns the asset bundle resource with the given name.

The resource must be declared in the mod manifest's Resources section.

Declaration

```
public static ModAssetBundle GetAssetBundle(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.

Returns

TYPE	DESCRIPTION
ModAssetBundle	The resource.

Exceptions

TYPE	CONDITION
InvalidOperationException	If called from an assembly not listed in the mod manifest.
ArgumentException	If the resource with the given name is not an asset bundle.
ArgumentException	If no resource with the given name exists.

GetAudioClip(String)

Returns the audio clip resource with the given name.

The resource must be declared in the mod manifest's Resources section.

Declaration

```
public static ModAudioClip GetAudioClip(string name)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.

Returns

TYPE	DESCRIPTION
ModAudioClip	The resource.

Exceptions

TYPE	CONDITION
InvalidOperationException	If called from an assembly not listed in the mod manifest.
ArgumentException	If the resource with the given name is not an audio clip.
ArgumentException	If no resource with the given name exists.

GetMesh(String)

Returns the mesh resource with the given name.

The resource must be declared in the mod manifest's Resources section.

Declaration

```
public static ModMesh GetMesh(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.

Returns

TYPE	DESCRIPTION
ModMesh	The resource.

Exceptions

TYPE	CONDITION
InvalidOperationException	If called from an assembly not listed in the mod manifest.

TYPE	CONDITION
ArgumentException	If the resource with the given name is not a mesh.
ArgumentException	If no resource with the given name exists.

GetTexture(String)

Returns the texture resource with the given name.

The resource must be declared in the mod manifest's Resources section.

Declaration

```
public static ModTexture GetTexture(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the resource.

Returns

TYPE	DESCRIPTION
ModTexture	The resource.

Exceptions

TYPE	CONDITION
InvalidOperationException	If called from an assembly not listed in the mod manifest.
ArgumentException	If the resource with the given name is not a texture.
ArgumentException	If no resource with the given name exists.

SetOnObject(GameObject, Action<GameObject>, Action)

SetOnObject is a utility method that easily allows handling longer resource loading times, as well as objects that should receive a resource that is not loaded yet being instantiated.

Declaration

```
public void SetOnObject(GameObject go, Action<GameObject> postSetAction = null, Action prefabPostSetAction = null)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	go	Object on which to set the resource.
Action <UnityEngine.GameObject>	postSetAction	Callback for every object that is set.
Action	prefabPostSetAction	Callback for the original object.

Remarks

When SetOnObject is called on an object, an OnLoad handler for the resource is installed that will automatically set the appropriate value (renderer.material.mainTexture for a ModTexture, meshFilter.mesh for a ModMesh, and audioSource.clip for a ModAudioClip) when the resource is done loading.

However, the major benefit of using SetOnObject is that it will also automatically handle the given GameObject being instantiated/copied. All instances of the object will also automatically receive the resource.

The two optional callbacks can be used to perform additional actions that are necessary to correctly set the resource. The first one is called for every object on which the resource is set (including the original object), and the second one only for the object originally passed into the function.

Asset bundle resources can not be set on objects.

Exceptions

TYPE	CONDITION
InvalidOperationException	If this resource is an asset bundle.

TriggerOnLoad()

Declaration

```
protected void TriggerOnLoad()
```

Events

OnAllResourcesLoaded

Event that fires once all resources of the mod have finished loading.

If all resources have already finished loading when a callback is registered, it is called immediately.

Declaration

```
public static event Action OnAllResourcesLoaded
```

Event Type

TYPE	DESCRIPTION
Action	

OnLoad

Called when the resource has finished loading.

This is also called if there was an error loading the resource.

If a new handler is added to this and the resource has already finished loading, it is immediately called.

Declaration

```
public virtual event Action OnLoad
```

Event Type

TYPE	DESCRIPTION
Action	

OnResourceLoaded

Event that fires whenever a resource of your mod finishes loading.

Note: The event also fires if there was an error loading the resource.

If there are already any resources that have finished loading when you register a callback, the callback will be called once for each of them.

Declaration

```
public static event Action<ModResource> OnResourceLoaded
```

Event Type

TYPE	DESCRIPTION
Action < ModResource >	

Enum ModResource.ResourceType

The different resource types that can be managed by the mod loader.

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum ResourceType
```

Fields

NAME	DESCRIPTION
AssetBundle	
AudioClip	
Mesh	
Texture	

Class Mods

Provides an API for querying the status of other installed mods.

This can for example be used to do additional things based on the presence of other mods, like adding some kind of interaction.

Inheritance

[Object](#)

Mods

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class Mods
```

Methods

GetVersion(Guid)

Returns the version of the mod specified by the given id. If there is no corresponding mod loaded, returns null.

Declaration

```
public static Version GetVersion(Guid id)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	id	

Returns

TYPE	DESCRIPTION
Version	

IsModLoaded(Guid)

Returns whether a mod with the given id is currently loaded. Note that depending on the LoadOrder setup, this doesn't necessarily mean that the mod won't still be loaded after the calling one.

Declaration

```
public static bool IsModLoaded(Guid id)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	id	

Returns

TYPE	DESCRIPTION
Boolean	

Events

OnModLoaded

Called every time a new mod is loaded.

When registering a new handler for this, it is immediately called once for each currently loaded mod.

Declaration

```
public static event Action<Guid> OnModLoaded
```

Event Type

TYPE	DESCRIPTION
Action<Guid>	

Class ModTexture

A texture mod resource.

Inheritance

[Object](#)
[ModResource](#)
ModTexture

Inherited Members

- [ModResource.GetTexture\(String\)](#)
- [ModResource.GetMesh\(String\)](#)
- [ModResource.GetAudioClip\(String\)](#)
- [ModResource.GetAssetBundle\(String\)](#)
- [ModResource.Get\(ResourceReference\)](#)
- [ModResource.CreateTextureResource\(String, String, Boolean, Boolean\)](#)
- [ModResource.CreateMeshResource\(String, String, Boolean\)](#)
- [ModResource.CreateAudioClipResource\(String, String, Boolean\)](#)
- [ModResource.CreateAssetBundleResource\(String, String, Boolean\)](#)
- [ModResource.SetOnObject\(GameObject, Action<GameObject>, Action\)](#)
- [ModResource.TriggerOnLoad\(\)](#)
- [ModResource.AllResourcesLoaded](#)
- [ModResource.Type](#)
- [ModResource.Name](#)
- [ModResource.Available](#)
- [ModResource.OnResourceLoaded](#)
- [ModResource.OnAllResourcesLoaded](#)
- [ModResource.OnLoad](#)

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class ModTexture : ModResource
```

Properties

Error

Declaration

```
public override string Error { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Overrides

[ModResource.Error](#)

HasError

Declaration

```
public override bool HasError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.HasError](#)

Loaded

Declaration

```
public override bool Loaded { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

[ModResource.Loaded](#)

Texture

Declaration

```
public Texture2D Texture { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Texture2D	

Methods

SetOnObject(GameObject, Material, Action<GameObject>, Action)

SetOnObject is a utility method that easily allows handling long resource loading times. This overload also sets a given Material.

Declaration

```
public void SetOnObject(GameObject go, Material mat, Action<GameObject> postSetAction = null, Action prefabPostSetAction = null)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	go	Object on which to set the resource.
UnityEngine.Material	mat	Material which should be set on the renderers.
Action <UnityEngine.GameObject>	postSetAction	Callback for every object that is set.

TYPE	NAME	DESCRIPTION
Action	prefabPostSetAction	Callback for the original object.

Remarks

When SetOnObject is called on an object, an OnLoad handler for the resource is installed that will automatically set the appropriate value (renderer.material.mainTexture for a ModTexture, meshFilter.mesh for a ModMesh, and audioSource.clip for a ModAudioClip) when the resource is done loading.

However, the major benefit of using SetOnObject is that it will also automatically handle the given GameObject being instantiated/copied. All instances of the object will also automatically receive the resource.

The two optional callbacks can be used to perform additional actions that are necessary to correctly set the resource. The first one is called for every object on which the resource is set (including the original object), and the second one only for the object originally passed into the function.

In this overload, an additional Material is passed in, which will also be set on the renderers, in addition to the texture.

Operators

Explicit(Texture2D to ModTexture)

Declaration

public static explicit operator ModTexture(Texture2D texture)

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Texture2D	texture	

Returns

TYPE	DESCRIPTION
ModTexture	

Implicit(ModTexture to Texture2D)

Declaration

public static implicit operator Texture2D(ModTexture texture)

Parameters

TYPE	NAME	DESCRIPTION
ModTexture	texture	

Returns

TYPE	DESCRIPTION
UnityEngine.Texture2D	

Class ModTriggers

You can use this class in conjunction with the Triggers element in Mod.xml to define custom triggers for the level editor logic system.

Inheritance

Object

ModTriggers

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModTriggers
```

Remarks

Mod.xml is used to define the properties of the triggers.. This class is used to activate your triggers, using either GetCallback or RegisterCallback. See their documentation for more details.

Multiplayer Note: You an only call the activation callbacks on the instance running a simulation (i.e. the server for global sim or the respective client for local sim). The effects of the activation are automatically propagated across the network. Calling the callbacks on a client will throw an InvalidOperationException.

Methods

GetCallback(Int32)

GetCallback returns an [Action](#) that you can later call to activate your trigger on all entities that have it set up.

Declaration

```
public static Action GetCallback(int id)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	ID of the trigger you want to activate.

Returns

TYPE	DESCRIPTION
Action	Callback to activate the trigger.

RegisterCallback(Int32, ModTriggers.OnTriggerChanged)

RegisterCallback registers an OnTriggerChanged callback for one of your triggers.

This callback will be called by the mod loader whenever the specified trigger is added to or removed from an entity. If the trigger was added, one of its arguments is another callback which you can call to activate the trigger, analogous to the one returned from GetCallback, except that this callback will only activate the trigger on the specified entity.

Declaration

```
public static void RegisterCallback(int id, ModTriggers.OnTriggerChanged callback)
```


Parameters

TYPE	NAME	DESCRIPTION
Int32	id	ID of the trigger.
ModTriggers.OnTriggerChanged	callback	OnTriggerChanged callback to register.

Remarks

The callback will be called on all connected instances. See documentation on ModTriggers for how your code should behave in MP.

Note that the callback will only be called for the non-simulation entites, never for a simulation clone.

Delegate ModTriggers.OnTriggerChanged

Used in conjunction with RegisterCallback.

Namespace: [Modding](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public delegate void OnTriggerChanged(Entity entity, Action activate, bool removed);
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	entity	The entity that was modified.
Action	activate	Callback to activate the trigger on this entity. Null if removed.
Boolean	removed	Whether the trigger was removed from the entity and not added.

Constructors

OnTriggerChanged(Object, IntPtr)

Declaration

```
public OnTriggerChanged(object object, IntPtr method)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	object	
IntPtr	method	

Methods

BeginInvoke(Entity, Action, Boolean, AsyncCallback, Object)

Declaration

```
public virtual IAsyncResult BeginInvoke(Entity entity, Action activate, bool removed, AsyncCallback callback, object object)
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	entity	
Action	activate	
Boolean	removed	

TYPE	NAME	DESCRIPTION
AsyncCallback	callback	
Object	object	

Returns

TYPE	DESCRIPTION
IAsyncResult	

EndInvoke(IAsyncResult)

Declaration

```
public virtual void EndInvoke(IAsyncResult result)
```

Parameters

TYPE	NAME	DESCRIPTION
IAsyncResult	result	

Invoke(Entity, Action, Boolean)

Declaration

```
public virtual void Invoke(Entity entity, Action activate, bool removed)
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	entity	
Action	activate	
Boolean	removed	

Class ModUtility

Inheritance

[Object](#)

ModUtility

Namespace: [Modding](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ModUtility
```

Methods

[CopyComponent<T>\(T, GameObject\)](#)

Creates a copy of the given component on the destination game object.

Adds a component of the same type to the destination game object and then copies its field's values as described in [CopyComponentValues](#).

Declaration

```
public static T CopyComponent<T>(T original, GameObject destination)
    where T : Component
```

Parameters

TYPE	NAME	DESCRIPTION
T	original	Original component.
UnityEngine.GameObject	destination	Destination game object.

Returns

TYPE	DESCRIPTION
T	The newly-created copied component.

Type Parameters

NAME	DESCRIPTION
T	Type of the component to copy.

See Also

[CopyComponentValues<T>\(T, T\)](#)

[CopyComponentValues<T>\(T, T\)](#)

Copies the values of fields of one component onto another one.

All values of all public fields and private fields marked with the [SerializeField](#) attribute are directly copied.

This is somewhat similar to the copy Unity does when instantiating objects but it doesn't attempt to also modify references to other components on the same game object or the game object itself etc.

This means this won't always copy all components entirely correctly but it is still often useful. It is always possible to copy or adjust more values by hand after calling this method.

Declaration

```
public static T CopyComponentValues<T>(T original, T destination)
    where T : Component
```

Parameters

TYPE	NAME	DESCRIPTION
T	original	Original component.
T	destination	Destination of the copy.

Returns

TYPE	DESCRIPTION
T	The destination.

Type Parameters

NAME	DESCRIPTION
T	Type of the component to copy.

GetWindowId()

Returns an ID that is guaranteed to be unique among the IDs supplied by this function to all callers.

Can be used as a window ID for Unity's Legacy GUI system to avoid conflicts with other mods.

Declaration

```
public static int GetWindowId()
```

Returns

TYPE	DESCRIPTION
Int32	

Namespace Modding.Blocks

Classes

[Block](#)

Represents a specific Block placed in the game. Can be a building or simulation block.

[BlockInfo](#)

Meta-data about a block, used mainly for saving and loading.

[BlockPrefabInfo](#)

Represents a block type.

[PlayerMachine](#)

Represents a Machine.

[PlayerMachineInfo](#)

Class Block

Represents a specific Block placed in the game. Can be a building or simulation block.

Inheritance

Object

Block

Namespace: [Modding.Blocks](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Block
```

Properties

BlockScript

BlockScript attached to this block.

Always null for non-modded blocks. Can also be null for modded blocks, depending on whether the block has one.

Declaration

```
public BlockScript BlockScript { get; }
```

Property Value

TYPE	DESCRIPTION
BlockScript	

BuildingBlock

A reference to the building version of the current block.

This returns the same object when accessed on a building block.

Declaration

```
public Block BuildingBlock { get; }
```

Property Value

TYPE	DESCRIPTION
Block	

GameObject

The game object of this block.

Declaration

```
public GameObject GameObject { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

Guid

Unique identifier for each block. This value is the same for a building block and its simulation clone.

MP: This value is consistent across all instances in MP.

Declaration

```
public Guid Guid { get; }
```

Property Value

TYPE	DESCRIPTION
Guid	

Health

The health of this block, 0 if the block doesn't support health.

Declaration

```
public float Health { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

InternalObject

The internal representation of this block.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public BlockBehaviour InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
BlockBehaviour	

InWind

Whether this block is currently affected by wind.

Declaration

```
public bool InWind { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsArmor

Whether this block is an armor block.

Declaration

```
public bool IsArmor { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsBurning

Whether this block is currently burning.

Declaration

```
public bool IsBurning { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Machine

The machine that this block is a part of.

Declaration

```
public PlayerMachine Machine { get; }
```

Property Value

TYPE	DESCRIPTION
PlayerMachine	

MaxHealth

Maximum health of this block, 0 if the block doesn't support health.

Declaration

```
public float MaxHealth { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

Prefab

An object representing the type of this block.

Declaration

```
public BlockPrefabInfo Prefab { get; }
```

Property Value

TYPE	DESCRIPTION
BlockPrefabInfo	

SimBlock

A reference to the simulation clone of the current block.

This is null in building mode and returns the same object when accessed on a simulation block.

Declaration

<pre>public Block SimBlock { get; }</pre>

Property Value

TYPE	DESCRIPTION
Block	

Team

The team that the player this block belongs to is a part of.

Declaration

<pre>public MPTeam Team { get; }</pre>
--

Property Value

TYPE	DESCRIPTION
MPTeam	

Methods

Equals(Object)

Determines if the other object refers to the same block.

Declaration

<pre>public override bool Equals(object obj)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	Object to compare with this object.

Returns

TYPE	DESCRIPTION
Boolean	True if the other object refers to the same block.

Overrides

Object.Equals(Object)

From(BlockBehaviour)

Returns a Block instance from its internal representation.

Declaration

```
public static Block From(BlockBehaviour behaviour)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockBehaviour	behaviour	

Returns

TYPE	DESCRIPTION
Block	

From(BlockScript)

Returns a Block instance from the behaviour of a modded block.

Declaration

```
public static Block From(BlockScript script)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockScript	script	

Returns

TYPE	DESCRIPTION
Block	

From(Guid)

Returns a Block instance from a Guid.

This returns the building block corresponding to the Guid.

Declaration

```
public static Block From(Guid guid)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	guid	

Returns

TYPE	DESCRIPTION
Block	

From(GameObject)

Returns a Block instance from a block GameObject.

Null if the game object is not a block.

Declaration

```
public static Block From(GameObject obj)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	obj	

Returns

TYPE	DESCRIPTION
Block	

GetHashCode()

Computes a hash code.

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	A hash code for this object.

Overrides

[Object.GetHashCode\(\)](#)

SetOnFire(Boolean)

Sets the "on fire" status of this block.

No-op if the block does not support burning.

Declaration

```
public void SetOnFire(bool onFire)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
Boolean	onFire	Whether or not the block should be on fire.

ToString()

Returns a string identifying this block.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	String containing the block type and the block’s GUID.

Overrides

[Object.ToString\(\)](#)

Operators

Equality(Block, Block)

Compares two blocks for equality.

This tests whether objects refer to the same block, not if they are the same object.

Declaration

```
public static bool operator ==(Block left, Block right)
```

Parameters

TYPE	NAME	DESCRIPTION
Block	left	
Block	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(Block, Block)

Declaration

```
public static bool operator !=(Block left, Block right)
```

Parameters

TYPE	NAME	DESCRIPTION
Block	left	
Block	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class BlockInfo

Meta-data about a block, used mainly for saving and loading.

Inheritance

[Object](#)

BlockInfo

Namespace: [Modding.Blocks](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BlockInfo
```

Properties

Data

Declaration

```
public XDataHolder Data { get; }
```

Property Value

TYPE	DESCRIPTION
XDataHolder	

Guid

Unique identifier of the block.

Declaration

```
public Guid Guid { get; }
```

Property Value

TYPE	DESCRIPTION
Guid	

InternalObject

Internal representation of this block's metadata.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public BlockInfo InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
BlockInfo	

Position

Position in world-space of the block.

Declaration

```
public Vector3 Position { get; set; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Rotation

Rotation of the block.

Declaration

```
public Quaternion Rotation { get; set; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Quaternion	

Scale

Scale of the block.

Declaration

```
public Vector3 Scale { get; set; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Type

Type ID of the block.

Declaration

```
public int Type { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Methods

Equals(Object)

Tests if the other object refers to the same block info as this one.

Declaration

```
public override bool Equals(object obj)
```


Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(BlockInfo)

Returns a BlockInfo instance from its internal representation.

Declaration

```
public static BlockInfo From(BlockInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockInfo	info	

Returns

TYPE	DESCRIPTION
BlockInfo	

From(Block)

Creates a BlockInfo instance based on an existing block.

Declaration

```
public static BlockInfo From(Block block)
```

Parameters

TYPE	NAME	DESCRIPTION
Block	block	

Returns

TYPE	DESCRIPTION
BlockInfo	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

[ToString\(\)](#)

Returns a string representation of this block info, containing its GUID.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

[Equality\(BlockInfo, BlockInfo\)](#)

Declaration

```
public static bool operator ==(BlockInfo left, BlockInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockInfo	left	
BlockInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

[Inequality\(BlockInfo, BlockInfo\)](#)

Declaration

```
public static bool operator !=(BlockInfo left, BlockInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockInfo	left	

TYPE	NAME	DESCRIPTION
BlockInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class BlockPrefabInfo

Represents a block type.

Inheritance

[Object](#)

BlockPrefabInfo

Namespace: [Modding.Blocks](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BlockPrefabInfo
```

Properties

GameObject

"Prefab" / Template GameObject of this block type.

Declaration

```
public GameObject GameObject { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

GhostObject

"Prefab" / Template GameObject for the ghost of this block type.

Declaration

```
public GameObject GhostObject { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

InternalObject

Internal representation of this block type.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public BlockPrefab InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
BlockPrefab	

Name

Name of the block type.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Type

Integer ID of this block type.

Declaration

```
public int Type { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Methods

Equals(BlockPrefabInfo)

Declaration

```
protected bool Equals(BlockPrefabInfo other)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockPrefabInfo	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(BlockPrefab)

Returns a BlockPrefab instance from its internal representation.

Declaration

```
public static BlockPrefabInfo From(BlockPrefab prefab)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockPrefab	prefab	

Returns

TYPE	DESCRIPTION
BlockPrefabInfo	

FromId(Int32)

Get a BlockPrefab based on its internal id. Null if no prefab with the specified id can be found.

Declaration

```
public static BlockPrefabInfo FromId(int id)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	

Returns

TYPE	DESCRIPTION
BlockPrefabInfo	

FromIdModded(Guid, Int32)

Get a modded BlockPrefab with the specified modId and localId. Null if not corresponding prefab can be found.

Declaration

```
public static BlockPrefabInfo FromIdModded(Guid modId, int localId)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	
Int32	localId	

Returns

TYPE	DESCRIPTION
BlockPrefabInfo	

GetAll()

Returns all currently loaded block types.

Declaration

```
public static BlockPrefabInfo[] GetAll()
```

Returns

TYPE	DESCRIPTION
BlockPrefabInfo []	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

GetOfficial(BlockType)

Get an official BlockPrefab from its BlockType value.

Declaration

```
public static BlockPrefabInfo GetOfficial(BlockType type)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockType	type	

Returns

TYPE	DESCRIPTION
BlockPrefabInfo	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(BlockPrefabInfo, BlockPrefabInfo)

Declaration

```
public static bool operator ==(BlockPrefabInfo left, BlockPrefabInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockPrefabInfo	left	
BlockPrefabInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(BlockPrefabInfo, BlockPrefabInfo)

Declaration

```
public static bool operator !=(BlockPrefabInfo left, BlockPrefabInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockPrefabInfo	left	
BlockPrefabInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class PlayerMachine

Represents a Machine.

Inheritance

[Object](#)

PlayerMachine

Namespace: [Modding.Blocks](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class PlayerMachine
```

Properties

BlockCount

The total amount of blocks in this machine.

Declaration

```
public int BlockCount { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Bounds

The bounds of this machine.

Declaration

```
public Bounds Bounds { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Bounds	

BuildingBlocks

The building blocks of this machine.

Declaration

```
public ReadOnlyCollection<Block> BuildingBlocks { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	

BuildingMachine

Parent transform of the building blocks of this machine.

Declaration

```
public Transform BuildingMachine { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

ClusterCount

The total number of clusters in this machine.

Declaration

```
public int ClusterCount { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

CurtainMode

Whether this machine is in Curtain mode.

Declaration

```
public bool CurtainMode { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

ExplodingCannonballs

Whether this machine is in Exploding Cannonballs mode.

Declaration

```
public bool ExplodingCannonballs { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

GhostMode

Whether this machine is currently displayed as ghost on the local instance.

Declaration

```
public bool GhostMode { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Remarks

Machines of other players who are in local simulation are displayed in ghost mode.

Health

The health of this machine.

0 if the health system is not turned on for this level/machine.

0 in singleplayer.

Declaration

```
public float Health { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

InfiniteAmmo

Whether this machine is in Infinite Ammo mode.

Declaration

```
public bool InfiniteAmmo { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

InternalObject

The internal representation of this machine.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public Machine InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
Machine	

InternalObjectServer

Internal representation of this machine.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public ServerMachine InternalObjectServer { get; }
```

Property Value

TYPE	DESCRIPTION
ServerMachine	

MachineData

Custom data associated to the machine.

Declaration

```
public XDataHolder MachineData { get; }
```

Property Value

TYPE	DESCRIPTION
XDataHolder	

Remarks

The keys for the data here are not automatically seperated by mod, a unique prefix for each mod is advisable here.

Keep in mind that one can't rely on any data begin present here, as machines may have been saved without the mod installed and should still load.

Mass

The total mass of this machine.

Declaration

```
public float Mass { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

MiddlePosition

The middle of this machine.

Declaration

```
public Vector3 MiddlePosition { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Name

Name of the machine.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Player

The Player that this machine belongs to.

Null in single player.

Declaration

```
public Player Player { get; }
```

Property Value

TYPE	DESCRIPTION
Player	

Position

Position of this machine in world-space.

Declaration

```
public Vector3 Position { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Rotation

Rotation of this machine.

Declaration

```
public Quaternion Rotation { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Quaternion	

SimulationBlocks

The simulation clones of the blocks of this machine.

Declaration

```
public ReadOnlyCollection<Block> SimulationBlocks { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection < Block >	

SimulationMachine

Parent transform the simulation clone of this machine.

Declaration

```
public Transform SimulationMachine { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

Size

The size of this machine.

Declaration

```
public Vector3 Size { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Unbreakable

Whether this machine is in Invincibility mode.

Declaration

```
public bool Unbreakable { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

AddBlock(BlockType, Vector3, Quaternion, Boolean)

Adds an official block to this machine. Can only be used on the instance that owns this machine.

Declaration

```
public Block AddBlock(BlockType type, Vector3 position, Quaternion rotation, bool flipped = false)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockType	type	The type of the block.
UnityEngine.Vector3	position	Position of the block.
UnityEngine.Quaternion	rotation	Rotation of the block.
Boolean	flipped	Whether the block should be flipped.

Returns

TYPE	DESCRIPTION
Block	A Block on success, null on failure.

Remarks

Valid values for the type parameter are the entries in the BlockType enum.

Position and Rotation are considered to be local to the machine.

Note: Adding draggable blocks like braces using this will not work correctly.

AddBlock(Guid, Int32, Vector3, Quaternion, Boolean)

Adds a modded block to this machine. Can only be used on the instance that owns this machine.

Declaration

```
public Block AddBlock(Guid mod, int localId, Vector3 position, Quaternion rotation, bool flipped = false)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	mod	Id of the mod.
Int32	localId	Id of the block within the mod.
UnityEngine.Vector3	position	Position of the block.
UnityEngine.Quaternion	rotation	Rotation of the block.
Boolean	flipped	Whether the block should be flipped.

Returns

TYPE	DESCRIPTION
Block	A Block on success, null on failure.

Remarks

Modded blocks are uniquely identified by the pair (modId, blockId) where blockId is unique within the mod itself. The block will be added if the corresponding mod is loaded, otherwise null is returned.

Position and Rotation are considered to be local to the machine.

Note: Adding draggable blocks like braces using this will not work correctly.

AddBlock(Int32, Vector3, Quaternion, Boolean)

Adds a block to this machine. Can only be used on the instance that owns this machine.

Declaration

<pre>public Block AddBlock(int type, Vector3 position, Quaternion rotation, bool flipped = false)</pre>

Parameters

TYPE	NAME	DESCRIPTION
Int32	type	Id of the block type.
UnityEngine.Vector3	position	Position of the block.
UnityEngine.Quaternion	rotation	Rotation of the block.
Boolean	flipped	Whether the block should be flipped.

Returns

TYPE	DESCRIPTION
Block	A Block on success, null on failure.

Remarks

The Id parameters is in terms of the internal Ids. These are the same values given by BlockPrefab.Type. Note that the Ids of modded blocks may change at any time. Use the overload with seperate modId and localId for reliable placement of modded blocks.

Position and Rotation are considered to be local to the machine.

Note: Adding draggable blocks like braces using this will not work correctly.

Equals(PlayerMachine)

Declaration

```
protected bool Equals(PlayerMachine other)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachine	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(Machine)

Create a Machine from its internal representation.

Declaration

```
public static PlayerMachine From(Machine machine)
```

Parameters

TYPE	NAME	DESCRIPTION
Machine	machine	

Returns

TYPE	DESCRIPTION
PlayerMachine	

From(GameObject)

Create a Machine from its GameObject. Null if the game object is not a machine.

Declaration

```
public static PlayerMachine From(GameObject go)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	go	

Returns

TYPE	DESCRIPTION
PlayerMachine	

GetBlockBehavioursOfType<T>(Guid, Int32)

Gets a list of the behaviours attached to all the blocks with the specified modded block type that are part of this machine.

Returns simulation blocks if the machine is currently simulating and building blocks if not.

If T is not a correct behaviour type for the specified block type, an `InvalidCastException` will be thrown.

Note: The returned objects are not part of the stable API.

Declaration

```
public ReadOnlyCollection<T> GetBlockBehavioursOfType<T>(Guid modId, int localId)
    where T : BlockBehaviour
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	Guid of the mod adding the block type.
Int32	localId	Local ID of the block type within the mod.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<T>	Collection of block behaviours.

Type Parameters

NAME	DESCRIPTION
T	BlockBehaviour type associated to the block type.

GetBlockBehavioursOfType<T>(Guid, Int32, Boolean)

Gets a list of the behaviours attached to all the blocks with the specified modded block type that are part of this machine.

The last argument controls whether to return simulation or building blocks.

If T is not a correct behaviour type for the specified block type, an `InvalidCastException` will be thrown.

Note: The returned objects are not part of the stable API.

Declaration

```
public ReadOnlyCollection<T> GetBlockBehavioursOfType<T>(Guid modId, int localId, bool simulation)
    where T : BlockBehaviour
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	Guid of the mod adding the block type.
Int32	localId	Local ID of the block type within the mod.
Boolean	simulation	True to return simulation blocks, false to return building blocks.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<T>	Collection of block behaviours.

Type Parameters

NAME	DESCRIPTION
T	BlockBehaviour type associated to the block type.

GetBlockBehavioursOfType<T>(Int32)

Gets a list of the behaviours attached to all the blocks with the specified type that are part of this machine.

Returns simulation blocks if the machine is currently simulating and building blocks if not.

If T is not a correct behaviour type for the specified block type, an `InvalidCastException` will be thrown.

Note: The returned objects are not part of the stable API.

Declaration

```
public ReadOnlyCollection<T> GetBlockBehavioursOfType<T>(int type)
    where T : BlockBehaviour
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
Int32	type	ID of the block type.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<T>	Collection of block behaviours.

Type Parameters

NAME	DESCRIPTION
T	BlockBehaviour type associated to the block type.

GetBlockBehavioursOfType<T>(Int32, Boolean)

Gets a list of the behaviours attached to all the blocks with the specified type that are part of this machine.

The last argument controls whether simulation or building blocks are returned.

If T is not a correct behaviour type for the specified block type, an InvalidCastException will be thrown.

Note: The returned objects are not part of the stable API.

Declaration

```
public ReadOnlyCollection<T> GetBlockBehavioursOfType<T>(int type, bool simulation)
    where T : BlockBehaviour
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	type	ID of the block type.
Boolean	simulation	True to return simulation blocks, false to return building blocks.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<T>	Collection of block behaviours.

Type Parameters

NAME	DESCRIPTION
T	BlockBehaviour type associated to the block type.

GetBlocksOfType(BlockType)

Gets all blocks of the specified type that are part of this machine. Returns simulation blocks if the machine is currently in simulation, buildings blocks if not.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(BlockType type)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockType	type	Block type.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetBlocksOfType(BlockType, Boolean)

Gets all blocks of the specified type that are part of this machine. The last arguments controls whether simulation or building blocks are returned.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(BlockType type, bool simulation)
```

Parameters

TYPE	NAME	DESCRIPTION
BlockType	type	Block type.
Boolean	simulation	True to return simulation blocks, false to return building blocks.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetBlocksOfType(Guid, Int32)

Gets all blocks of the specified modded type that are part of this machine. Returns simulation blocks if the machine is currently in simulation, buildings blocks if not.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(Guid modId, int localId)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	ID of the mod that adds the block type.
Int32	localId	Local ID of the block type within the mod.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetBlocksOfType(Guid, Int32, Boolean)

Gets all blocks of the specified modded type that are part of this machine. The last argument controls whether simulation or building blocks are returned.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(Guid modId, int localId, bool simulation)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	ID of the mod that adds the block type.
Int32	localId	Local ID of the block type within the mod.
Boolean	simulation	True to return simulation blocks, false to return building blocks.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetBlocksOfType(Int32)

Gets all blocks of the specified type that are part of this machine. Returns simulation blocks if the machine is currently in simulation, buildings blocks if not.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(int type)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	type	Internal ID of the block type.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetBlocksOfType(Int32, Boolean)

Gets all blocks of the specified type that are part of this machine. The last argument controls whether simulation or building blocks are returned.

Declaration

```
public ReadOnlyCollection<Block> GetBlocksOfType(int type, bool simulation)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	type	Internal ID of the block type.
Boolean	simulation	True to return simulation blocks, false to return building blocks.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Block>	Collection of blocks.

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

Object.GetHashCode()

GetLocal()

Returns the machine of the local player in multiplayer, or the local singleplayer machine. Can return null if the local player does not currently have a machine.

Declaration

```
public static PlayerMachine GetLocal()
```

Returns

TYPE	DESCRIPTION
PlayerMachine	

RemoveBlock(Block)

Removes a block from this machine. Can only be used on the instance that owns this machine.

Declaration

```
public void RemoveBlock(Block block)
```

Parameters

TYPE	NAME	DESCRIPTION
Block	block	The block to remove.

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(PlayerMachine, PlayerMachine)

Declaration

```
public static bool operator ==(PlayerMachine left, PlayerMachine right)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachine	left	
PlayerMachine	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(PlayerMachine, PlayerMachine)

Declaration

```
public static bool operator !=(PlayerMachine left, PlayerMachine right)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachine	left	
PlayerMachine	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class PlayerMachineInfo

Inheritance

[Object](#)

PlayerMachineInfo

Namespace: [Modding.Blocks](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class PlayerMachineInfo
```

Properties

Blocks

Meta-data of the blocks of this machine.

Declaration

```
public ReadOnlyCollection<BlockInfo> Blocks { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection < BlockInfo >	

InternalObject

Internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public MachineInfo InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
MachineInfo	

MachineData

Custom data associated to this machine.

Declaration

```
public XDataHolder MachineData { get; }
```

Property Value

TYPE	DESCRIPTION
XDataHolder	

Remarks

The keys for the data here are not automatically seperated by mod, a unique prefix for each mod is advisable.

Keep in mind that one can't rely on any data being present here, as machines may have been saved without the mod installed and should still load.

That means always checking for the presence of data before accessing it and having sensible default values in case no data is present.

Name

Name of the machine.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Position

Position of this machine.

Declaration

```
public Vector3 Position { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Vector3	

Rotation

Rotation of this machine.

Declaration

```
public Quaternion Rotation { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Quaternion	

Methods

Equals(PlayerMachineInfo)

Declaration

```
protected bool Equals(PlayerMachineInfo other)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachineInfo	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(MachineInfo)

Create a MachineInfo from its internal representation.

Declaration

```
public static PlayerMachineInfo From(MachineInfo info)
```

Parameters

TYPE	NAME	DESCRIPTION
MachineInfo	info	

Returns

TYPE	DESCRIPTION
PlayerMachineInfo	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(PlayerMachineInfo, PlayerMachineInfo)

Declaration

```
public static bool operator ==(PlayerMachineInfo left, PlayerMachineInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachineInfo	left	
PlayerMachineInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(PlayerMachineInfo, PlayerMachineInfo)

Declaration

```
public static bool operator !=(PlayerMachineInfo left, PlayerMachineInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerMachineInfo	left	
PlayerMachineInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Namespace Modding.Common

Classes

[Player](#)

Represents a player in multiplayer.

There are no players in singleplayer.

Class Player

Represents a player in multiplayer.

There are no players in singleplayer.

Inheritance

[Object](#)

Player

Namespace: [Modding.Common](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Player
```

Properties

ActivePrefab

The entity type the player has selected in the level editor.

Can be null if the player is not actively using the level editor.

Declaration

```
public EntityPrefabInfo ActivePrefab { get; }
```

Property Value

TYPE	DESCRIPTION
EntityPrefabInfo	

InLocalSim

Whether this player is currently in local simulation mode.

Declaration

```
public bool InLocalSim { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

InternalObject

Internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public PlayerData InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
PlayerData	

IsHost

Whether this is the host of the current multiplayer session.

Declaration

```
public bool IsHost { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsLocalPlayer

Whether this is the local player.

Declaration

```
public bool IsLocalPlayer { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSpectator

Whether this player is a spectator.

Declaration

```
public bool IsSpectator { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Machine

The machine of this player (if it's not a spectator).

Declaration

```
public PlayerMachine Machine { get; }
```

Property Value

TYPE	DESCRIPTION
PlayerMachine	

Name

Name of the player.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

NetworkId

ID of the player used for network communications.

Declaration

```
public ushort NetworkId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	

Ping

The ping of this player.

Declaration

```
public int Ping { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Selection

What entities the local player has selected in the level editor.

If this is not the local player, null.

Declaration

```
public ReadOnlyCollection<Entity> Selection { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection < Entity >	

SteamId

SteamID of the player.

Empty if not connected via Steam.

Declaration

```
public string SteamId { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Team

What team this player is a part of.

Declaration

```
public MPTeam Team { get; }
```

Property Value

TYPE	DESCRIPTION
MPTeam	

VoteState

Whether this player has voted to start simulation.

Declaration

```
public bool VoteState { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

Equals(Player)

Declaration

```
protected bool Equals(Player other)
```

Parameters

TYPE	NAME	DESCRIPTION
Player	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(PlayerData)

Create a Player object from its internal representation.

Declaration

```
public static Player From(PlayerData data)
```

Parameters

TYPE	NAME	DESCRIPTION
PlayerData	data	

Returns

TYPE	DESCRIPTION
Player	

From(UInt16)

Returns a Player object from its network id.

Declaration

```
public static Player From(ushort networkId)
```

Parameters

TYPE	NAME	DESCRIPTION
UInt16	networkId	

Returns

TYPE	DESCRIPTION
Player	

GetAllPlayers()

Get a list of players currently connected.

Declaration

```
public static List<Player> GetAllPlayers()
```

Returns

TYPE	DESCRIPTION
List<Player>	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

GetHost()

Returns a Player object representing the host of the current multiplayer session.

Declaration

```
public static Player GetHost()
```

Returns

TYPE	DESCRIPTION
Player	

GetLocalPlayer()

Returns a Player object representing the local player. Can be null when in singleplayer.

Declaration

```
public static Player GetLocalPlayer()
```

Returns

TYPE	DESCRIPTION
Player	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(Player, Player)

Declaration

```
public static bool operator ==(Player left, Player right)
```

Parameters

TYPE	NAME	DESCRIPTION
Player	left	
Player	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(Player, Player)

Declaration

```
public static bool operator !=(Player left, Player right)
```

Parameters

TYPE	NAME	DESCRIPTION
Player	left	
Player	right	

Returns

TYPE	DESCRIPTION
Boolean	

Namespace Modding.Levels

Classes

[Entity](#)

Represents an Entity.

[EntityBehaviour](#)

Represents the behaviour of an entity.

Mostly this handles the logic system.

[EntityPrefabInfo](#)

Represents an Entity prefab / type.

[Level](#)

Represents a custom-made level.

Note: Due to intenal restrictions, the Level class can only represent the level that is currently loaded.

[LevelSetup](#)

Represents the level seutp, as set up in the Level Settings UI.

[LogicChain](#)

Represents a logic chain set up on an entity.

[LogicEvent](#)

Represents a single logic event that is part of a logic chain.

Class Entity

Represents an Entity.

Inheritance

[Object](#)

Entity

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Entity
```

Properties

Behaviour

The behaviour associated with this entity.

This mostly handles the logic system.

Declaration

```
public EntityBehaviour Behaviour { get; }
```

Property Value

TYPE	DESCRIPTION
EntityBehaviour	

BuildEntity

A reference to the building mode version of this entity.

Declaration

```
public Entity BuildEntity { get; }
```

Property Value

TYPE	DESCRIPTION
Entity	

GameObject

GameObject of this entity.

Declaration

```
public GameObject GameObject { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

Id

Unique identifier of this entity.

This is the same for the building mode entity and its simulation clone.

Declaration

```
public long Id { get; }
```

Property Value

TYPE	DESCRIPTION
Int64	

InternalObject

Internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public LevelEntity InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
LevelEntity	

IsBuildZone

Whether this entity is a build zone.

Declaration

```
public bool IsBuildZone { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsBurning

Whether this entity is currently burning.

Declaration

```
public bool IsBurning { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsDestroyed

Whether this entity was destroyed.

Declaration

```
public bool IsDestroyed { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsSelected

Whether this entity is selected by the local player.

Declaration

```
public bool IsSelected { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Name

Name of this entity.

The prefab name by default, or a custom name set in the BlockMapper.

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
String	

PhysicsEnabled

Whether this entity has its physics enabled.

Declaration

```
public bool PhysicsEnabled { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Prefab

The prefab / type of this entity.

Declaration

```
public EntityPrefabInfo Prefab { get; }
```

Property Value

TYPE	DESCRIPTION
EntityPrefabInfo	

SimEntity

A reference to the simulation clone of this entity.

Null in building mode.

Note that building zones do not have simulation clones, SimEntity is always null for them.

Declaration

<pre>public Entity SimEntity { get; }</pre>

Property Value

TYPE	DESCRIPTION
Entity	

Methods

Equals(Entity)

Declaration

<pre>protected bool Equals(Entity other)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
Entity	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

<pre>public override bool Equals(object obj)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(LevelEntity)

Create an Entity from its internal representation.

Declaration

```
public static Entity From(LevelEntity levelEntity)
```

Parameters

TYPE	NAME	DESCRIPTION
LevelEntity	levelEntity	

Returns

TYPE	DESCRIPTION
Entity	

From(Int64)

Create an Entity object from its id.

Declaration

```
public static Entity From(long id)
```

Parameters

TYPE	NAME	DESCRIPTION
Int64	id	

Returns

TYPE	DESCRIPTION
Entity	

From(GameObject)

Create an Entity object from its GameObject.

Declaration

```
public static Entity From(GameObject entityObject)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	entityObject	

Returns

TYPE	DESCRIPTION
Entity	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

Select(Boolean)

Marks the entity as selected for the local user.

Note that this does not actually select the entity for any of the normal tools, it just adds the visual effect.

Declaration

```
public void Select(bool selected)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	selected	

SetOnFire(Boolean)

Set the on fire state of this entity.

No-op if the entity cannot burn.

Declaration

```
public void SetOnFire(bool onFire)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	onFire	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

Object.ToString()

Operators

Equality(Entity, Entity)

Declaration

```
public static bool operator ==(Entity left, Entity right)
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	left	
Entity	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(Entity, Entity)

Declaration

```
public static bool operator !=(Entity left, Entity right)
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	left	
Entity	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class EntityBehaviour

Represents the behaviour of an entity.

Mostly this handles the logic system.

Inheritance

[Object](#)

EntityBehaviour

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class EntityBehaviour
```

Properties

Entity

The entity associated with this behaviour.

Declaration

```
public Entity Entity { get; }
```

Property Value

TYPE	DESCRIPTION
Entity	

HasLogic

Whether this entity has any logic chains set up.

Declaration

```
public bool HasLogic { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

HasRunningLogic

Whether this entity has any logic chains currently running.

Declaration

```
public bool HasRunningLogic { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

InternalObject

The internal representation of this objet.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public GenericEntity InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
GenericEntity	

LogicChains

All LogicChains that are set up on this entity.

Declaration

```
public ReadOnlyCollection<LogicChain> LogicChains { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<LogicChain>	

RunningLogicChains

The LogicChains that are currently running.

Declaration

```
public ReadOnlyCollection<LogicChain> RunningLogicChains { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<LogicChain>	

Variables

All variables local to this entity.

Note: Variables can't be changes using this, use SetVariable for that.

Declaration

```
public Dictionary<string, float> Variables { get; }
```

Property Value

TYPE	DESCRIPTION
Dictionary<String, Single>	

Methods

Activate()

Activate this entity.

Declaration

```
public void Activate()
```

Deactivate()

Deactivate this entity.

Declaration

```
public void Deactivate()
```

Equals(EntityBehaviour)

Declaration

```
protected bool Equals(EntityBehaviour other)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityBehaviour	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

Object.Equals(Object)

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

SetVariable(String, Single)

Set a variable (that is local to this entity) to a new value.

Declaration

```
public void SetVariable(string name, float value)
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	
Single	value	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

TriggerLogicChains(LogicChain)

Start all logic chains with the same trigger as the specified chain.

Declaration

```
public void TriggerLogicChains(LogicChain chain)
```

Parameters

TYPE	NAME	DESCRIPTION
LogicChain	chain	

TriggerLogicChains(String)

Start all logic chains with the specified trigger.

Declaration

```
public void TriggerLogicChains(string triggerId)
```

Parameters

TYPE	NAME	DESCRIPTION
String	triggerId	

Operators

Equality(EntityBehaviour, EntityBehaviour)

Declaration

<pre>public static bool operator ==(EntityBehaviour left, EntityBehaviour right)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
EntityBehaviour	left	
EntityBehaviour	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(EntityBehaviour, EntityBehaviour)

Declaration

<pre>public static bool operator !=(EntityBehaviour left, EntityBehaviour right)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
EntityBehaviour	left	
EntityBehaviour	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class EntityPrefabInfo

Represents an Entity prefab / type.

Inheritance

[Object](#)

EntityPrefabInfo

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class EntityPrefabInfo
```

Properties

AvailableTriggers

Identifiers for the triggers available on this entity type.

Declaration

```
public ReadOnlyCollection<string> AvailableTriggers { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<String>	

CanDoDamage

Whether this entity type can do damage to player machines.

Declaration

```
public bool CanDoDamage { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanPick

Whether this entity type can be picked for logic events.

Declaration

```
public bool CanPick { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanScale

Whether this entity type can be scaled.

Declaration

```
public bool CanScale { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Category

Level Editor category of this entity type.

Declaration

```
public StatMaster.Category Category { get; }
```

Property Value

TYPE	DESCRIPTION
StatMaster.Category	

Destructable

Whether this entity type can be destroyed.

Declaration

```
public bool Destructable { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

GameObject

The Prefab / Template game object for this entity type.

Declaration

```
public GameObject GameObject { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

Icon

GUI Icon of this entity type.

Declaration

```
public Texture2D Icon { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Texture2D	

Id

Unique ID of this entity type.

Declaration

```
public int Id { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Inflammable

Whether this entity type can burn.

Declaration

```
public bool Inflammable { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

InternalObject

The internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public LevelPrefab InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
LevelPrefab	

Keywords

Search keywords of this entity type.

Declaration

```
public ReadOnlyCollection<string> Keywords { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<String>	

Name

Name of this entity type.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

UniformScale

Whether this entity type always has to be uniformly scaled.

Declaration

```
public bool UniformScale { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

Equals(EntityPrefabInfo)

Declaration

```
protected bool Equals(EntityPrefabInfo other)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityPrefabInfo	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(LevelPrefab)

Create an EntityPrefab from its internal representation.

Declaration

```
public static EntityPrefabInfo From(LevelPrefab levelPrefab)
```

Parameters

TYPE	NAME	DESCRIPTION
LevelPrefab	levelPrefab	

Returns

TYPE	DESCRIPTION
EntityPrefabInfo	

FromId(Int32)

Get the EntityPrefab with the specified id. Null if it could not be found.

Declaration

```
public static EntityPrefabInfo FromId(int id)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	

Returns

TYPE	DESCRIPTION
EntityPrefabInfo	

FromIdModded(Guid, Int32)

Get the modded EntityPrefab with the specified mod and local ids. Null if it cannot be found.

Declaration

```
public static EntityPrefabInfo FromIdModded(Guid modId, int localId)
```


Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	
Int32	localId	

Returns

TYPE	DESCRIPTION
EntityPrefabInfo	

GetAll()

Returns all currently loaded entity types.

Declaration

```
public static EntityPrefabInfo[] GetAll()
```

Returns

TYPE	DESCRIPTION
EntityPrefabInfo []	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(EntityPrefabInfo, EntityPrefabInfo)

Declaration

```
public static bool operator ==(EntityPrefabInfo left, EntityPrefabInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityPrefabInfo	left	
EntityPrefabInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(EntityPrefabInfo, EntityPrefabInfo)

Declaration

```
public static bool operator !=(EntityPrefabInfo left, EntityPrefabInfo right)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityPrefabInfo	left	
EntityPrefabInfo	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class Level

Represents a custom-made level.

Note: Due to intenal restrictions, the Level class can only represent the level that is currently loaded.

Inheritance

[Object](#)

Level

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Level
```

Properties

CustomData

Custom data associated with this level.

Declaration

```
public XDataHolder CustomData { get; }
```

Property Value

TYPE	DESCRIPTION
XDataHolder	

Remarks

The keys for the data here are not automatically seperated by mod, a unique prefix for each mod is advisable here.

Keep in mind that one can't rely on any data begin present here, as machines may have been saved without the mod installed and should still load.

Entities

All entities placed in the level.

Declaration

```
public ReadOnlyCollection<Entity> Entities { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<Entity>	

GlobalVariables

The variables with global scope.

Note that you cannot change variables using this, use the SetVariable method for that.

Declaration

```
public Dictionary<string, float> GlobalVariables { get; }
```

Property Value

TYPE	DESCRIPTION
Dictionary<String, Single>	

InternalLevel

Part of the internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public CustomLevel InternalLevel { get; }
```

Property Value

TYPE	DESCRIPTION
CustomLevel	

InternalObject

The internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public LevelEditor InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
LevelEditor	

Selection

The entities of this level that are currently selected by the local player.

Declaration

```
public ReadOnlyCollection<Entity> Selection { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<Entity>	

Setup

The settings of the level.

Declaration

```
public LevelSetup Setup { get; }
```

Property Value

TYPE	DESCRIPTION
LevelSetup	

Methods

AddEntity(Guid, Int32, Vector3, Quaternion, Vector3, Boolean)

Declaration

```
public void AddEntity(Guid modId, int localId, Vector3 position, Quaternion rotation, Vector3 scale, bool showEffect = true)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	modId	
Int32	localId	
UnityEngine.Vector3	position	
UnityEngine.Quaternion	rotation	
UnityEngine.Vector3	scale	
Boolean	showEffect	

AddEntity(Int32, Vector3, Quaternion, Vector3, Boolean)

Adds an entity to the level.

Declaration

```
public void AddEntity(int id, Vector3 position, Quaternion rotation, Vector3 scale, bool showEffect = true)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	id	Id of the entity type, as given by EntityPrefab.Id or in the ObjectTypes enums.
UnityEngine.Vector3	position	Position of the entity.
UnityEngine.Quaternion	rotation	Rotation of the entity.
UnityEngine.Vector3	scale	Scale of the entity.
Boolean	showEffect	Whether to show the placement visual effect.

Remarks

Unlike `PlayerMachine.AddBlock` this doesn't return the added entity, but it is possible to call it on any connected instance.

Note that the `Id` of a modded entity can change at any point, use the overload with seperate `modId` and `localId` for reliable placement of modded entities.

You can use entries of the enums in `ObjectTypes` as IDs.

GetCurrentLevel()

Get the level currently open in the level editor. Null if there is no open level.

Declaration

```
public static Level GetCurrentLevel()
```

Returns

TYPE	DESCRIPTION
Level	

GetEntitiesOfType(Guid, Int32)

Gets all entities of the specified modded type in the level.

Declaration

```
public ReadOnlyCollection<Entity> GetEntitiesOfType(Guid modId, int localId)
```

Parameters

TYPE	NAME	DESCRIPTION
Guid	<code>modId</code>	Guid of the mod that adds the entity.
Int32	<code>localId</code>	Local ID of the entity within the mod.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Entity>	Collection of entities.

GetEntitiesOfType(Int32)

Gets all entities of the specified type in the level.

Declaration

```
public ReadOnlyCollection<Entity> GetEntitiesOfType(int id)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
Int32	id	Id of the entity type/perfab.

Returns

TYPE	DESCRIPTION
ReadOnlyCollection<Entity>	Collection of entities.

RemoveEntity(Entity)

Removes an entity from the level.

Declaration

```
public void RemoveEntity(Entity entity)
```

Parameters

TYPE	NAME	DESCRIPTION
Entity	entity	The entity to remove.

SetVariable(String, Single)

Set a global variable.

Declaration

```
public void SetVariable(string var, float val)
```

Parameters

TYPE	NAME	DESCRIPTION
String	var	
Single	val	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Class LevelSetup

Represents the level seutp, as set up in the Level Settings UI.

Inheritance

[Object](#)

LevelSetup

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class LevelSetup
```

Properties

AllowCopyMachines

Whether copying machines of other players is allowed.

Declaration

```
public bool AllowCopyMachines { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

AllowExcessPlayers

Whether additional player can join that have no build zones already placed.

Declaration

```
public bool AllowExcessPlayers { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

BlockCountLimit

The maximum amount of blocks allowed on a machine.

Declaration

```
public int BlockCountLimit { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

CurtainMode

Whether curtain mode is turned on.

Declaration

```
public bool CurtainMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Environment

Which environment is used in the level.

Declaration

```
public LevelSettings.LevelEnvironment Environment { get; set; }
```

Property Value

TYPE	DESCRIPTION
LevelSettings.LevelEnvironment	

HidePlayerLabels

Whether player labels should be hidden or shown.

Declaration

```
public bool HidePlayerLabels { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

InternalObject

The internal representation of this object.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public LevelSettings InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
LevelSettings	

MaxPlayers

Maxmium amount of players that can play the level together.

Declaration

```
public int MaxPlayers { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

MinPlayers

Minimum amount of players necessary to play the level.

Declaration

```
public int MinPlayers { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

MusicID

The ID of the soundtrack playing in the level.

Declaration

```
public int MusicID { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Name

Name of the level.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

VoteMode

Whether vote mod is turned on.

Declaration

```
public bool VoteMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

From(LevelSettings)

Declaration

```
public static LevelSetup From(LevelSettings settings)
```

Parameters

TYPE	NAME	DESCRIPTION
LevelSettings	settings	

Returns

TYPE	DESCRIPTION
LevelSetup	

GetCurrent()

Declaration

```
public static LevelSetup GetCurrent()
```

Returns

TYPE	DESCRIPTION
LevelSetup	

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Class LogicChain

Represents a logic chain set up on an entity.

Inheritance

[Object](#)

LogicChain

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class LogicChain
```

Properties

Entity

The entity this logic chain belongs to.

Declaration

```
public Entity Entity { get; }
```

Property Value

TYPE	DESCRIPTION
Entity	

Events

The events that are part of this logic chain.

Declaration

```
public ReadOnlyCollection<LogicEvent> Events { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection < LogicEvent >	

InternalObject

The internal representation of this logic chain.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public EntityLogic InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
EntityLogic	

IsModdedTrigger

Whether the trigger of this chain is added to the game by a mod.

Declaration

```
public bool IsModdedTrigger { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

TriggerIdentifier

The unique identifier of the trigger of this chain.

Declaration

```
public string TriggerIdentifier { get; }
```

Property Value

TYPE	DESCRIPTION
String	

TriggerName

The display name of the trigger of this chain.

Declaration

```
public string TriggerName { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Methods

Equals(LogicChain)

Declaration

```
protected bool Equals(LogicChain other)
```

Parameters

TYPE	NAME	DESCRIPTION
LogicChain	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(EntityLogic)

Create a LogicChain from its internal representation.

Declaration

```
public static LogicChain From(EntityLogic logic)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityLogic	logic	

Returns

TYPE	DESCRIPTION
LogicChain	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality([LogicChain](#), [LogicChain](#))

Declaration

```
public static bool operator ==(LogicChain left, LogicChain right)
```

Parameters

TYPE	NAME	DESCRIPTION
LogicChain	left	
LogicChain	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality([LogicChain](#), [LogicChain](#))

Declaration

```
public static bool operator !=(LogicChain left, LogicChain right)
```

Parameters

TYPE	NAME	DESCRIPTION
LogicChain	left	
LogicChain	right	

Returns

TYPE	DESCRIPTION
Boolean	

Class LogicEvent

Represents a single logic event that is part of a logic chain.

Inheritance

[Object](#)

LogicEvent

Namespace: [Modding.Levels](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class LogicEvent
```

Remarks

Events can be either modded or official events. They mostly behave the same way, but accessing data the player set up in the mapper is different:

For modded events, the `EventProperties` property exposes the EventProperties set up by the mod, take a look at the documentation for modded events for more information.

There is currently no stable API to access data for official events, but it's still possible using some unstable classes: The `InternalObject` has an `eventData` field which can be cast to the appropriate subclass of `EventContainer` to get access to the event-specific data.

Properties

Chain

The LogicChain this event is a part of.

Declaration

```
public LogicChain Chain { get; }
```

Property Value

TYPE	DESCRIPTION
LogicChain	

EventProperties

EventProperties of modded events. Always empty for official events.

Declaration

```
public IDictionary<string, EventProperty> EventProperties { get; }
```

Property Value

TYPE	DESCRIPTION
IDictionary < String , EventProperty >	

EventType

Which event type this is.

This is unique for offical events, but always Modded for all modded events.

Declaration

```
public EventContainer.EventType EventType { get; }
```

Property Value

TYPE	DESCRIPTION
EventContainer.EventType	

Identifier

Unique identifier of this event.

Declaration

```
public string Identifier { get; }
```

Property Value

TYPE	DESCRIPTION
String	

InternalObject

The internal representation of this event.

WARNING: This is not part of the stable API and subject to change!

Declaration

```
public EntityEvent InternalObject { get; }
```

Property Value

TYPE	DESCRIPTION
EntityEvent	

IsModdedEvent

Whether this event type is added to the game by a mod.

Declaration

```
public bool IsModdedEvent { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Name

Display name of this event. This is the same text that is displayed in the mapper when setting up the event.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

PickerEntities

The entities that are selected in the picker of this event, if it has one.

Declaration

<pre>public ReadOnlyCollection<Entity> PickerEntities { get; }</pre>
--

Property Value

TYPE	DESCRIPTION
ReadOnlyCollection<Entity>	

Team

The assigned team. This value is correct for official events, as well as modded events that contain only one team button. If a modded event supports multiple teams, this value will be the first one while the other can be accessed using the Teams property.

Declaration

<pre>public MPTeam Team { get; }</pre>
--

Property Value

TYPE	DESCRIPTION
MPTeam	

Teams

All assigned teams. Includes the team returned by Team at the first position but also any other teams.

Declaration

<pre>public MPTeam[] Teams { get; }</pre>

Property Value

TYPE	DESCRIPTION
MPTeam[]	

Methods

Equals(LogicEvent)

Declaration

<pre>protected bool Equals(LogicEvent other)</pre>
--

Parameters

TYPE	NAME	DESCRIPTION
LogicEvent	other	

Returns

TYPE	DESCRIPTION
Boolean	

Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Object.Equals\(Object\)](#)

From(EntityEvent, LogicChain)

Create a LogicEvent from its internal representation.

Declaration

```
public static LogicEvent From(EntityEvent evt, LogicChain chain)
```

Parameters

TYPE	NAME	DESCRIPTION
EntityEvent	evt	
LogicChain	chain	

Returns

TYPE	DESCRIPTION
LogicEvent	

GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[Object.GetHashCode\(\)](#)

ToString()

Declaration

public override string ToString()

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

Operators

Equality(LogicEvent, LogicEvent)

Declaration

public static bool operator ==(LogicEvent left, LogicEvent right)

Parameters

TYPE	NAME	DESCRIPTION
LogicEvent	left	
LogicEvent	right	

Returns

TYPE	DESCRIPTION
Boolean	

Inequality(LogicEvent, LogicEvent)

Declaration

public static bool operator !=(LogicEvent left, LogicEvent right)

Parameters

TYPE	NAME	DESCRIPTION
LogicEvent	left	
LogicEvent	right	

Returns

TYPE	DESCRIPTION
Boolean	

Namespace Modding.Mapper

Classes

CustomMapperTypes

Mapper types are the elements available in the block mapper (also used for mod settings etc.).

It is possible to define custom mapper types, these must be registered in this class.

See the documentation about custom mapper types for detailed usage information.

CustomSelector<T, TMapper>

Base class for the selectors of custom mapper types.

MCustom<T>

Base class for all custom mod-added mapper types.

SelectorElements

Utility class for creating elements in custom mapper types.

Access this via the `Elements` property in the `CustomSelector` subclass.

SelectorMaterials

Utility class with references to useful materials for creating custom mapper types.

Access this via the `Materials` property in the `CustomSelector` subclass.

Class CustomMapperTypes

Mapper types are the elements available in the block mapper (also used for mod settings etc.).

It is possible to define custom mapper types, these must be registered in this class.

See the documentation about custom mapper types for detailed usage information.

Inheritance

[Object](#)

CustomMapperTypes

Namespace: [Modding.Mapper](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class CustomMapperTypes
```

Methods

AddMapperType<T, TMapper, TSelector>()

Registers a new custom mapper type.

Declaration

```
public static void AddMapperType<T, TMapper, TSelector>()  
    where TMapper : MCustom<T> where TSelector : CustomSelector<T, TMapper>
```

Type Parameters

NAME	DESCRIPTION
T	Type of base value being edited.
TMapper	MCustom wrapper type around T.
TSelector	CustomSelector type.

Class CustomSelector<T, TMapper>

Base class for the selectors of custom mapper types.

Inheritance

[Object](#)

Selectors.Selector

CustomSelector<T, TMapper>

Namespace: [Modding.Mapper](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class CustomSelector<T, TMapper> : Selector where TMapper : MCustom<T>
```

Type Parameters

NAME	DESCRIPTION
T	Underlying type of the mapper type.
TMapper	Mapper type this selector is associated to.

Constructors

CustomSelector()

Declaration

```
protected CustomSelector()
```

Fields

Elements

Provides utility methods for creating common interface elements.

Declaration

```
protected SelectorElements Elements
```

Field Value

TYPE	DESCRIPTION
SelectorElements	

Materials

Provides access to materials commonly found in mapper UI.

Declaration

```
protected SelectorMaterials Materials
```

Field Value

TYPE	DESCRIPTION
SelectorMaterials	

Properties

Background

Background object of the selector.

Declaration

```
protected Transform Background { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

Content

Hierarchy parent of the created UI.

Declaration

```
protected Transform Content { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Transform	

CustomMapperType

Gives access to the underlying mapper type.

Declaration

```
protected IMapper CustomMapperType { get; }
```

Property Value

TYPE	DESCRIPTION
IMapper	

Methods

CreateInterface()

Create the UI here. The UI should all be child objects to Content.

Declaration

```
protected abstract void CreateInterface()
```

Init()

Used internally to initialise the selector object. Should not be called manually.

Declaration

```
public override sealed void Init()
```

Overrides

Selectors.Selector.Init()

UpdateInterface()

Update the UI here.

Declaration

```
protected abstract void UpdateInterface()
```

UpdateVisual()

Used internally to trigger updating the interface. Should not be called manually.

Declaration

```
protected override sealed void UpdateVisual()
```

Overrides

Selectors.Selector.UpdateVisual()

Class MCustom<T>

Base class for all custom mod-added mapper types.

Inheritance

[Object](#)

[MapperType](#)

MCustom<T>

Namespace: [Modding.Mapper](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class MCustom<T> : MapperType
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

MCustom(String, String, T)

Declaration

```
protected MCustom(string displayName, string key, T defaultValue)
```

Parameters

TYPE	NAME	DESCRIPTION
String	displayName	
String	key	
T	defaultValue	

Fields

defaultValue

The default value this mapper type was created with.

Declaration

```
protected T defaultValue
```

Field Value

TYPE	DESCRIPTION
T	

loadValue

Load value of the mapper type. This is used for the undo system, and is normally handled completely automatically.

Declaration

```
protected T loadValue
```

Field Value

TYPE	DESCRIPTION
T	

value

Current value of the mapper type.

Declaration

```
protected T value
```

Field Value

TYPE	DESCRIPTION
T	

Properties

isDefaultValue

Default implementation compares value and defaultValue using `.Equals()`. If this is not suitable or ideal for type T, override this.

Declaration

```
public override bool isDefaultValue { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Overrides

MapperType.isDefaultValue

SerializationKey

This key should be used for all serialization of the mapper type.

Declaration

```
protected string SerializationKey { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Value

Provides public access to the underlying value. Setting this automatically invokes the Changed event.

Declaration

```
public virtual T Value { get; set; }
```

Property Value

TYPE	DESCRIPTION
T	

Methods

ApplyValue()

Applies the load value to the value and invokes the Changed event.

Declaration

```
public override void ApplyValue()
```

Overrides

MapperType.ApplyValue()

DeSerialize(XData)

Deserializes the given data into both value and load value.

Declaration

```
public override void DeSerialize(XData raw)
```

Parameters

TYPE	NAME	DESCRIPTION
XData	raw	

Overrides

MapperType.DeSerialize(XData)

DeSerializeValue(XData)

Override to specify how to deserialize from XData to a T.

Declaration

```
public abstract T DeSerializeValue(XData data)
```

Parameters

TYPE	NAME	DESCRIPTION
XData	data	

Returns

TYPE	DESCRIPTION
T	

Remarks

This is used to provide default implementations for the other DeSerialize* methods.

InvokeChanged(T)

Be careful if you override this, it is called from the base constructor.

Declaration

```
public virtual void InvokeChanged(T value)
```

Parameters

TYPE	NAME	DESCRIPTION
T	value	

ResetDefaults()

Resets the mapper type's value back to the default value.

Declaration

```
public override void ResetDefaults()
```

Overrides

MapperType.ResetDefaults()

ResetValue()

Resets the value back to the load value. Used for undo.

Declaration

```
public override void ResetValue()
```

Overrides

MapperType.ResetValue()

Serialize()

Serializes the current value.

Declaration

```
public override XData Serialize()
```

Returns

TYPE	DESCRIPTION
XData	

Overrides

MapperType.Serialize()

SerializeDefault()

Serializes the default value.

Declaration

```
public override XData SerializeDefault()
```

Returns

TYPE	DESCRIPTION
XData	

Overrides

MapperType.SerializeDefault()

SerializeLoadValue()

Serializes the load value.

Declaration

```
public override XData SerializeLoadValue()
```

Returns

TYPE	DESCRIPTION
XData	

Overrides

MapperType.SerializeLoadValue()

SerializeValue(T)

Override to specify how to serialize a T into an XData. Be careful: This is called from the MCustom constructor, which means it is called before the derived class' constructor is called! Use SerializationKey as the key for the XData object!

Declaration

```
public abstract XData SerializeValue(T value)
```

Parameters

TYPE	NAME	DESCRIPTION
T	value	

Returns

TYPE	DESCRIPTION
XData	

Remarks

This is used to provide default implementations for the other Serialize* methods.

Events

Changed

Invoked whenever the underlying value of the mapper type is changed.

Declaration

```
public event Action<T> Changed
```

Event Type

TYPE	DESCRIPTION
Action<T>	

Class SelectorElements

Utility class for creating elements in custom mapper types.

Access this via the `Elements` property in the `CustomSelector` subclass.

Inheritance

[Object](#)

SelectorElements

Namespace: `Modding.Mapper`

Assembly: `Assembly-CSharp.dll`

Syntax

```
public class SelectorElements
```

Methods

AddButton(Transform)

Adds button behaviour to an object. Adds a UIButton component and a BoxCollider if there is not already. The returned UIButton has a Click event that can be used.

Declaration

```
public UIButton AddButton(Transform t)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	t	Object to add the button behaviour to

Returns

TYPE	DESCRIPTION
UIButton	The UIButton behaviour

MakeBox(Vector3, Vector2, Material)

Creates a rectangle with rounded corners, using the specified material. Useful for making backgrounds for buttons or other elements.

Declaration

```
public Transform MakeBox(Vector3 position, Vector2 size, Material material)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	position	Position of the center of the box

TYPE	NAME	DESCRIPTION
UnityEngine.Vector2	size	Size of the box
UnityEngine.Material	material	Material to use, see the Materials property

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	A reference to the created object

MakeText(Vector3, String, Single)

Displays text at the given position.

Declaration

```
public DynamicText MakeText(Vector3 position, string content, float fontSize = 0.175F)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	position	Where to display the text
String	content	Initial text to display
Single	fontSize	Text size

Returns

TYPE	DESCRIPTION
DynamicText	A DynamicText object that can be used to modify the text.

MakeTexture(Vector3, Vector2, ModTexture)

Displays a texture in the interface.

Declaration

```
public MeshRenderer MakeTexture(Vector3 position, Vector2 size, ModTexture texture)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	position	Position of the center of the texture
UnityEngine.Vector2	size	Size of the texture in the interface
ModTexture	texture	Texture to display

Returns

TYPE	DESCRIPTION
UnityEngine.MeshRenderer	Renderer that renders the texture

MakeTexture(Vector3, Vector2, Texture)

Displays a texture in the interface.

Declaration

```
public MeshRenderer MakeTexture(Vector3 position, Vector2 size, Texture texture)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	position	Position of the center of the texture
UnityEngine.Vector2	size	Size of the texture in the interface
UnityEngine.Texture	texture	Texture to display

Returns

TYPE	DESCRIPTION
UnityEngine.MeshRenderer	Renderer that renders the texture

ScaleOnMouse(Transform, Transform)

Adds ScaleOnMouseOver behaviour to an object. This will make the toScale object scale appropriately when the mouse hovers over or clicks on the hover object, making it behave like other buttons in the game.

Declaration

```
public void ScaleOnMouse(Transform hover, Transform toScale)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	hover	
UnityEngine.Transform	toScale	

Class SelectorMaterials

Utility class with references to useful materials for creating custom mapper types.

Access this via the `Materials` property in the `CustomSelector` subclass.

Inheritance

[Object](#)

SelectorMaterials

Namespace: `Modding.Mapper`

Assembly: `Assembly-CSharp.dll`

Syntax

```
public class SelectorMaterials
```

Properties

DarkBackground

Darker grey, used for example as background for the skin selector.

Declaration

```
public Material DarkBackground { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Material	

DarkElement

Dark material, used for example as background of buttons.

Declaration

```
public Material DarkElement { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Material	

LightBackground

Light grey, used for example as background of sliders.

Declaration

```
public Material LightBackground { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Material	

RedHighlight

Red material, used for example in active toggles.

Declaration

```
public Material RedHighlight { get; }
```

Property Value

TYPE	DESCRIPTION
UnityEngine.Material	

Namespace Modding.Modules

Classes

[BlockModule](#)

One of the two base classes for block modules. This class is used for deserializing the modules from block XML files.

[BlockModuleBehaviour<TModule>](#)

One of the two base classes for block modules. This class is the base for the behaviours attached to block game objects.

[CustomModules](#)

Block Modules can be used as plug-and-play behaviour for custom blocks without writing any code.

This class can be used to add new modules to use for other mods (or in the same mod of course).

See the documentation about Modules for detailed usage information.

Class BlockModule

One of the two base classes for block modules. This class is used for deserializing the modules from block XML files.

Inheritance

- [Object](#)
- [Element](#)
- [BlockModule](#)
- [ShootingModule](#)
- [SpewingModule](#)
- [SpinningModule](#)
- [SteeringModule](#)

Implements

- [IValidatable](#)
- [IReloadable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.Validate\(String\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Modules](#)

Assembly: [Assembly-CSharp.dll](#)

Syntax

```
public abstract class BlockModule : Element, IValidatable, IReloadable
```

Constructors

BlockModule()

Declaration

```
protected BlockModule()
```

Fields

ModId

modid attribute for custom modules.

Declaration

```
public Guid ModId
```

Field Value

TYPE	DESCRIPTION
Guid	

Methods

OnReload(IReloadable)

Declaration

```
public virtual void OnReload(IReloadable newModule)
```

Parameters

TYPE	NAME	DESCRIPTION
IReloadable	newModule	

PreprocessForReloading()

Declaration

```
public virtual void PreprocessForReloading()
```

Implements

[IValidatable](#)

[IReloadable](#)

Class BlockModuleBehaviour<TModule>

One of the two base classes for block modules. This class is the base for the behaviours attached to block game objects.

Inheritance

[Object](#)

[ModBlockBehaviour](#)

BlockModuleBehaviour<TModule>

[ShootingModuleBehaviour](#)

[SpewingModuleBehaviour](#)

[SpinningModuleBehaviour](#)

[SteeringModuleBehaviour](#)

Inherited Members

[ModBlockBehaviour.SafeAwake\(\)](#)

[ModBlockBehaviour.OnPrefabCreation\(\)](#)

[ModBlockBehaviour.OnBlockPlaced\(\)](#)

[ModBlockBehaviour.BuildingUpdate\(\)](#)

[ModBlockBehaviour.SimulateUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateUpdateClient\(\)](#)

[ModBlockBehaviour.BuildingFixedUpdate\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateFixedUpdateClient\(\)](#)

[ModBlockBehaviour.BuildingLateUpdate\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateAlways\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateHost\(\)](#)

[ModBlockBehaviour.SimulateLateUpdateClient\(\)](#)

[ModBlockBehaviour.OnSimulateStart\(\)](#)

[ModBlockBehaviour.OnSimulateStop\(\)](#)

[ModBlockBehaviour.OnStartBurning\(\)](#)

[ModBlockBehaviour.OnStopBurning\(Boolean\)](#)

[ModBlockBehaviour.OnSimulateCollisionEnter\(Collision\)](#)

[ModBlockBehaviour.OnSimulateCollisionStay\(Collision\)](#)

[ModBlockBehaviour.OnSimulateCollisionExit\(Collision\)](#)

[ModBlockBehaviour.OnSimulateTriggerEnter\(Collider\)](#)

[ModBlockBehaviour.OnSimulateTriggerStay\(Collider\)](#)

[ModBlockBehaviour.OnSimulateTriggerExit\(Collider\)](#)

[ModBlockBehaviour.OnSimulateParticleCollision\(GameObject\)](#)

[ModBlockBehaviour.OnSave\(XDataHolder\)](#)

[ModBlockBehaviour.OnLoad\(XDataHolder\)](#)

[ModBlockBehaviour.OnReloadAmmo\(Int32, AmmoType, Boolean, Boolean\)](#)

[ModBlockBehaviour.AddKey\(String, String, KeyCode\)](#)

[ModBlockBehaviour.AddKey\(MKey\)](#)

[ModBlockBehaviour.AddTeam\(String, String, MPTeam\)](#)

[ModBlockBehaviour.AddTeam\(MTeam\)](#)

[ModBlockBehaviour.AddText\(String, String, String\)](#)

[ModBlockBehaviour.AddText\(MText\)](#)

[ModBlockBehaviour.AddValue\(String, String, Single\)](#)

[ModBlockBehaviour.AddValue\(String, String, Single, Single, Single\)](#)

[ModBlockBehaviour.AddValue\(MValue\)](#)

[ModBlockBehaviour.AddSlider\(String, String, Single, Single, Single\)](#)

ModBlockBehaviour.AddSliderUnclamped(String, String, Single, Single, Single)
ModBlockBehaviour.AddSlider(MSlider)
ModBlockBehaviour.AddColourSlider(String, String, Color, Boolean)
ModBlockBehaviour.AddColourSlider(MColourSlider)
ModBlockBehaviour.AddMenu(String, Int32, List<String>, Boolean)
ModBlockBehaviour.AddMenu(MMenu)
ModBlockBehaviour.AddToggle(String, String, Boolean)
ModBlockBehaviour.AddToggle(String, String, String, Boolean)
ModBlockBehaviour.AddToggle(MToggle)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)
ModBlockBehaviour.AddLimits(MLimits)
ModBlockBehaviour.AddCustom<T>(MCustom<T>)
ModBlockBehaviour.IsBurning
ModBlockBehaviour.HasBurnedOut
ModBlockBehaviour.IsFrozen
ModBlockBehaviour.IsDestroyed
ModBlockBehaviour.HasRigidbody
ModBlockBehaviour.Rigidbody
ModBlockBehaviour.BlockBehaviour
ModBlockBehaviour.VisualController
ModBlockBehaviour.Renderer
ModBlockBehaviour.MainVis
ModBlockBehaviour.ShowDebugVisuals
ModBlockBehaviour.Flipped
ModBlockBehaviour.BlockId
ModBlockBehaviour.SimPhysics
ModBlockBehaviour.IsSimulating
ModBlockBehaviour.IsStripped
ModBlockBehaviour.Machine
ModBlockBehaviour.CanFlip
ModBlockBehaviour.DirectionArrow

Namespace: **Modding.Modules**
Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class BlockModuleBehaviour<TModule> : ModBlockBehaviour, IModuleBehaviour where TModule : BlockModule
```

Type Parameters

NAME	DESCRIPTION
TModule	BlockModule class for this module.

Constructors

BlockModuleBehaviour()

Declaration

```
protected BlockModuleBehaviour()
```

Properties

Module

Access to the BlockModule extending class instance.

Declaration

```
public TModule Module { get; }
```

Property Value

TYPE	DESCRIPTION
TModule	

ModuleGuid

Required for internal purposes. Should not be used from mods, not stable API.

Declaration

```
public string ModuleGuid { get; set; }
```

Property Value

TYPE	DESCRIPTION
String	

RawModule

Required for internal purposes, same value as Module but as object. Should not be used from mods, not stable API.

Declaration

```
public object RawModule { get; set; }
```

Property Value

TYPE	DESCRIPTION
Object	

Methods

GetColourSlider(MColourSliderReference)

Get an MColourSlider from an MColourSLiderReference.

Declaration

```
public MColourSlider GetColourSlider(MColourSliderReference slider)
```

Parameters

TYPE	NAME	DESCRIPTION
MColourSliderReference	slider	

Returns

TYPE	DESCRIPTION
MColourSlider	

GetKey(MKeyReference)

Get an MKey form an MKeyReference.

Declaration

```
public MKey GetKey(MKeyReference key)
```

Parameters

TYPE	NAME	DESCRIPTION
MKeyReference	key	

Returns

TYPE	DESCRIPTION
MKey	

GetResource(ResourceReference)

Get a shared resource from a ModuleResourceReference.

Declaration

```
public ModResource GetResource(ResourceReference reference)
```

Parameters

TYPE	NAME	DESCRIPTION
ResourceReference	reference	

Returns

TYPE	DESCRIPTION
ModResource	

GetSlider(MSliderReference)

Get an MSlider from an MSliderReference.

Declaration

```
public MSlider GetSlider(MSliderReference slider)
```

Parameters

TYPE	NAME	DESCRIPTION
MSliderReference	slider	

Returns

TYPE	DESCRIPTION
MSlider	

GetToggle(MToggleReference)

Get an MToggle from an MToggleReference.

Declaration

```
public MToggle GetToggle(MToggleReference toggle)
```

Parameters

TYPE	NAME	DESCRIPTION
MToggleReference	toggle	

Returns

TYPE	DESCRIPTION
MToggle	

GetValue(MValueReference)

Get an MValue from an MValueReference.

Declaration

```
public MValue GetValue(MValueReference value)
```

Parameters

TYPE	NAME	DESCRIPTION
MValueReference	value	

Returns

TYPE	DESCRIPTION
MValue	

OnReload()

Declaration

```
public virtual void OnReload()
```

Class CustomModules

Block Modules can be used as plug-and-play behaviour for custom blocks without writing any code.

This class can be used to add new modules to use for other mods (or in the same mod of course).

See the documentation about Modules for detailed usage information.

Inheritance

[Object](#)

CustomModules

Namespace: [Modding.Modules](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class CustomModules
```

Methods

AddBlockModule<TModule, TBehaviour>(String, Boolean)

Registers a new block modules.

Declaration

```
public static void AddBlockModule<TModule, TBehaviour>(string name, bool canReload)
    where TModule : BlockModule where TBehaviour : BlockModuleBehaviour<TModule>
```

Parameters

TYPE	NAME	DESCRIPTION
String	name	Name of the module.
Boolean	canReload	

Type Parameters

NAME	DESCRIPTION
TModule	BlockModule class.
TBehaviour	BlockModuleBehaviour class.

Namespace Modding.Modules.Official

Classes

[ParticleHelper](#)

[ParticleHelper.CollisionSettings](#)

[ParticleHelper.CustomParticles](#)

[ParticleHelper.FireParticles](#)

[ParticleHelper.ParticleDefinition](#)

[ParticleHelper.ParticleSystemsInformation](#)

[ParticleHelper.SteamParticles](#)

[ParticleHelper.WaterParticles](#)

[ShootingModule](#)

[ShootingModule.CannonSound](#)

[ShootingModule.CrossbowSounds](#)

[ShootingModule.ModdedProjectileColliderComponent](#)

[ShootingModule.Projectile](#)

[ShootingModuleBehaviour](#)

[ShootingModuleSoundHolder](#)

[SpewingModule](#)

[SpewingModuleBehaviour](#)

[SpewingModuleFireParticleTrigger](#)

[SpinningModule](#)

[SpinningModuleBehaviour](#)

[SteeringModule](#)

[SteeringModuleBehaviour](#)

Class ParticleHelper

Inheritance

Object

ParticleHelper

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class ParticleHelper
```

Methods

CreateParticleSystems<T>(Transform, BlockModuleBehaviour<T>, ParticleHelper.ParticleDefinition[])

Declaration

```
public static ParticleHelper.ParticleSystemsInformation CreateParticleSystems<T>(Transform parent,
BlockModuleBehaviour<T> behaviour, ParticleHelper.ParticleDefinition[] particleSystems)
    where T : BlockModule
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	
BlockModuleBehaviour<T>	behaviour	
ParticleHelper.ParticleDefinition[]	particleSystems	

Returns

TYPE	DESCRIPTION
ParticleHelper.ParticleSystemsInformation	

Type Parameters

NAME	DESCRIPTION
T	

OnReloadModule<T>(Transform, BlockModuleBehaviour<T>, ParticleHelper.ParticleDefinition[], ParticleHelper.ParticleSystemsInformation)

Declaration

```
public static ParticleHelper.ParticleSystemsInformation OnReloadModule<T>(Transform parent,
BlockModuleBehaviour<T> behaviour, ParticleHelper.ParticleDefinition[] particleSystems,
ParticleHelper.ParticleSystemsInformation info)
    where T : BlockModule
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	

TYPE	NAME	DESCRIPTION
BlockModuleBehaviour<T>	behaviour	
ParticleHelper.ParticleDefinition[]	particleSystems	
ParticleHelper.ParticleSystemsInformation	info	

Returns

TYPE	DESCRIPTION
ParticleHelper.ParticleSystemsInformation	

Type Parameters

NAME	DESCRIPTION
T	

ParticlesOff(ParticleHelper.ParticleSystemsInformation, Boolean)

Declaration

```
public static void ParticlesOff(ParticleHelper.ParticleSystemsInformation info, bool SimPhysics)
```

Parameters

TYPE	NAME	DESCRIPTION
ParticleHelper.ParticleSystemsInformation	info	
Boolean	SimPhysics	

ParticlesOn(ParticleHelper.ParticleSystemsInformation, Boolean)

Declaration

```
public static void ParticlesOn(ParticleHelper.ParticleSystemsInformation info, bool SimPhysics)
```

Parameters

TYPE	NAME	DESCRIPTION
ParticleHelper.ParticleSystemsInformation	info	
Boolean	SimPhysics	

SetParticleRange<T>(ParticleHelper.ParticleSystemsInformation, BlockModuleBehaviour<T>, Single)

Declaration

```
public static void SetParticleRange<T>(ParticleHelper.ParticleSystemsInformation info, BlockModuleBehaviour<T>
behaviour, float value)
    where T : BlockModule
```

Parameters

TYPE	NAME	DESCRIPTION
ParticleHelper.ParticleSystemsInformation	info	
BlockModuleBehaviour<T>	behaviour	
Single	value	

Type Parameters

NAME	DESCRIPTION
T	

Class ParticleHelper.CollisionSettings

Inheritance

[Object](#)
[Element](#)

ParticleHelper.CollisionSettings

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.Validate\(String\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class CollisionSettings : Element, IValidatable
```

Constructors

CollisionSettings()

Declaration

```
public CollisionSettings()
```

Fields

Bounce

Declaration

```
public float Bounce
```

Field Value

TYPE	DESCRIPTION
Single	

Dampen

Declaration

```
public float Dampen
```

Field Value

TYPE	DESCRIPTION
Single	

LifetimeLoss

Declaration

public float LifetimeLoss

Field Value

TYPE	DESCRIPTION
Single	

MaxKillSpeed

Declaration

public float MaxKillSpeed

Field Value

TYPE	DESCRIPTION
Single	

MinKillSpeed

Declaration

public float MinKillSpeed

Field Value

TYPE	DESCRIPTION
Single	

RadiusScale

Declaration

public float RadiusScale

Field Value

TYPE	DESCRIPTION
Single	

Implements

[IValidatable](#)

Class ParticleHelper.CustomParticles

Inheritance

[Object](#)
[Element](#)
[ParticleHelper.ParticleDefinition](#)
ParticleHelper.CustomParticles

Implements

[IValidatable](#)

Inherited Members

[ParticleHelper.ParticleDefinition.StartPosition](#)
[ParticleHelper.ParticleDefinition.Direction](#)
[ParticleHelper.ParticleDefinition.StartSpeed](#)
[ParticleHelper.ParticleDefinition.LifetimeMultiplier](#)
[ParticleHelper.ParticleDefinition.FireTrigger](#)
[ParticleHelper.ParticleDefinition.DousesFire](#)
[ParticleHelper.ParticleDefinition.StartsFire](#)
[ParticleHelper.ParticleDefinition.Collisions](#)
[ParticleHelper.ParticleDefinition.AddForce](#)
[ParticleHelper.ParticleDefinition.Validate\(String\)](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class CustomParticles : ParticleHelper.ParticleDefinition, IValidatable
```

Constructors

CustomParticles()

Declaration

```
public CustomParticles()
```

Fields

Texture

Declaration

```
[RequireToValidate]  
public ResourceReference Texture
```

Field Value

TYPE	DESCRIPTION
ResourceReference	

Implements

[IValidatable](#)

Class ParticleHelper.FireParticles

Inheritance

[Object](#)

[Element](#)

[ParticleHelper.ParticleDefinition](#)

ParticleHelper.FireParticles

Implements

[IValidatable](#)

Inherited Members

[ParticleHelper.ParticleDefinition.StartPosition](#)

[ParticleHelper.ParticleDefinition.Direction](#)

[ParticleHelper.ParticleDefinition.StartSpeed](#)

[ParticleHelper.ParticleDefinition.LifetimeMultiplier](#)

[ParticleHelper.ParticleDefinition.FireTrigger](#)

[ParticleHelper.ParticleDefinition.DousesFire](#)

[ParticleHelper.ParticleDefinition.StartsFire](#)

[ParticleHelper.ParticleDefinition.Collisions](#)

[ParticleHelper.ParticleDefinition.AddForce](#)

[ParticleHelper.ParticleDefinition.Validate\(String\)](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class FireParticles : ParticleHelper.ParticleDefinition, IValidatable
```

Constructors

FireParticles()

Declaration

```
public FireParticles()
```

Implements

[IValidatable](#)

Class ParticleHelper.ParticleDefinition

Inheritance

- [Object](#)
- [Element](#)
- [ParticleHelper.ParticleDefinition](#)
- [ParticleHelper.CustomParticles](#)
- [ParticleHelper.FireParticles](#)
- [ParticleHelper.SteamParticles](#)
- [ParticleHelper.WaterParticles](#)

Implements

- [IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Modules.Official](#)
Assembly: [Assembly-CSharp.dll](#)

Syntax

```
public class ParticleDefinition : Element, IValidatable
```

Constructors

ParticleDefinition()

Declaration

```
public ParticleDefinition()
```

Fields

AddForce

Declaration

```
public float AddForce
```

Field Value

TYPE	DESCRIPTION
Single	

Collisions

Declaration

```
[RequireToValidate]  
public ParticleHelper.CollisionSettings Collisions
```

Field Value

TYPE	DESCRIPTION
ParticleHelper.CollisionSettings	

Direction

Declaration

public Vector3 Direction

Field Value

TYPE	DESCRIPTION
Vector3	

DousesFire

Declaration

[RequireToValidate] public Element DousesFire
--

Field Value

TYPE	DESCRIPTION
Element	

FireTrigger

Declaration

[RequireToValidate] public BoxModCollider FireTrigger
--

Field Value

TYPE	DESCRIPTION
BoxModCollider	

LifetimeMultiplier

Declaration

public float LifetimeMultiplier

Field Value

TYPE	DESCRIPTION
Single	

StartPosition

Declaration

public Vector3 StartPosition

Field Value

TYPE	DESCRIPTION
Vector3	

StartsFire

Declaration

<pre>[RequireToValidate] public Element StartsFire</pre>
--

Field Value

TYPE	DESCRIPTION
Element	

StartSpeed

Declaration

<pre>public float StartSpeed</pre>

Field Value

TYPE	DESCRIPTION
Single	

Methods

Validate(String)

Declaration

<pre>protected override bool Validate(string elementName)</pre>

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class ParticleHelper.ParticleSystemsInformation

Inheritance

[Object](#)

ParticleHelper.ParticleSystemsInformation

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ParticleSystemsInformation
```

Constructors

ParticleSystemsInformation()

Declaration

```
public ParticleSystemsInformation()
```

Fields

fireTriggers

Declaration

```
public Dictionary<ParticleHelper.ParticleDefinition, Transform> fireTriggers
```

Field Value

TYPE	DESCRIPTION
Dictionary < ParticleHelper.ParticleDefinition , UnityEngine.Transform >	

fireVisuals

Declaration

```
public Dictionary<ParticleHelper.ParticleDefinition, Transform> fireVisuals
```

Field Value

TYPE	DESCRIPTION
Dictionary < ParticleHelper.ParticleDefinition , UnityEngine.Transform >	

particles

Declaration

```
public Dictionary<ParticleHelper.ParticleDefinition, ParticleSystem[]> particles
```

Field Value

TYPE	DESCRIPTION
Dictionary < ParticleHelper.ParticleDefinition , UnityEngine.ParticleSystem[] >	

Class ParticleHelper.SteamParticles

Inheritance

[Object](#)

[Element](#)

[ParticleHelper.ParticleDefinition](#)

ParticleHelper.SteamParticles

Implements

[IValidatable](#)

Inherited Members

[ParticleHelper.ParticleDefinition.StartPosition](#)

[ParticleHelper.ParticleDefinition.Direction](#)

[ParticleHelper.ParticleDefinition.StartSpeed](#)

[ParticleHelper.ParticleDefinition.LifetimeMultiplier](#)

[ParticleHelper.ParticleDefinition.FireTrigger](#)

[ParticleHelper.ParticleDefinition.DousesFire](#)

[ParticleHelper.ParticleDefinition.StartsFire](#)

[ParticleHelper.ParticleDefinition.Collisions](#)

[ParticleHelper.ParticleDefinition.AddForce](#)

[ParticleHelper.ParticleDefinition.Validate\(String\)](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class SteamParticles : ParticleHelper.ParticleDefinition, IValidatable
```

Constructors

SteamParticles()

Declaration

```
public SteamParticles()
```

Implements

[IValidatable](#)

Class ParticleHelper.WaterParticles

Inheritance

[Object](#)

[Element](#)

[ParticleHelper.ParticleDefinition](#)

ParticleHelper.WaterParticles

Implements

[IValidatable](#)

Inherited Members

[ParticleHelper.ParticleDefinition.StartPosition](#)

[ParticleHelper.ParticleDefinition.Direction](#)

[ParticleHelper.ParticleDefinition.StartSpeed](#)

[ParticleHelper.ParticleDefinition.LifetimeMultiplier](#)

[ParticleHelper.ParticleDefinition.FireTrigger](#)

[ParticleHelper.ParticleDefinition.DousesFire](#)

[ParticleHelper.ParticleDefinition.StartsFire](#)

[ParticleHelper.ParticleDefinition.Collisions](#)

[ParticleHelper.ParticleDefinition.AddForce](#)

[ParticleHelper.ParticleDefinition.Validate\(String\)](#)

[Element.Validate\(\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class WaterParticles : ParticleHelper.ParticleDefinition, IValidatable
```

Constructors

WaterParticles()

Declaration

```
public WaterParticles()
```

Implements

[IValidatable](#)

Class ShootingModule

Inheritance

[Object](#)
[Element](#)
[BlockModule](#)

ShootingModule

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

[BlockModule.ModId](#)
[BlockModule.OnReload\(IReloadable\)](#)
[BlockModule.PreprocessForReloading\(\)](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[Reloadable]
public class ShootingModule : BlockModule, IValidatable, IReloadable
```

Constructors

ShootingModule()

Declaration

```
public ShootingModule()
```

Fields

AmmoType

Declaration

```
[Reloadable]
public AmmoType AmmoType
```

Field Value

TYPE	DESCRIPTION
AmmoType	

DefaultAmmo

Declaration

[Reloadable]
public int DefaultAmmo

Field Value

TYPE	DESCRIPTION
Int32	

FireKey

Declaration

[RequireToValidate]
public MKeyReference FireKey

Field Value

TYPE	DESCRIPTION
MKeyReference	

HoldToShootToggle

Declaration

[RequireToValidate]
public MToggleReference HoldToShootToggle

Field Value

TYPE	DESCRIPTION
MToggleReference	

PoolSize

Declaration

public int PoolSize

Field Value

TYPE	DESCRIPTION
Int32	

PowerSlider

Declaration

[RequireToValidate]
public MSliderReference PowerSlider

Field Value

TYPE	DESCRIPTION
MSliderReference	

ProjectileInfo

Declaration

```
[RequireToValidate]
[Reloadable]
public ShootingModule.Projectile ProjectileInfo
```

Field Value

TYPE	DESCRIPTION
ShootingModule.Projectile	

ProjectilesDespawnImmediately

Declaration

```
[Reloadable]
public bool ProjectilesDespawnImmediately
```

Field Value

TYPE	DESCRIPTION
Boolean	

ProjectilesExplode

Declaration

```
[Reloadable]
public bool ProjectilesExplode
```

Field Value

TYPE	DESCRIPTION
Boolean	

ProjectileStart

Declaration

```
[Reloadable]
public TransformValues ProjectileStart
```

Field Value

TYPE	DESCRIPTION
TransformValues	

RateOfFireSlider

Declaration

```
[RequireToValidate]
public MSliderReference RateOfFireSlider
```

Field Value

TYPE	DESCRIPTION
MSliderReference	

RecoilMultiplier

Declaration

<code>[Reloadable]</code> <code>public float RecoilMultiplier</code>

Field Value

TYPE	DESCRIPTION
Single	

ShowPlaceholderProjectile

Declaration

<code>public bool ShowPlaceholderProjectile</code>
--

Field Value

TYPE	DESCRIPTION
Boolean	

Sounds

Declaration

<code>[RequireToValidate]</code> <code>[CanBeEmpty]</code> <code>public object[] Sounds</code>
--

Field Value

TYPE	DESCRIPTION
Object[]	

SupportsExplosionGodTool

Declaration

<code>[Reloadable]</code> <code>public bool SupportsExplosionGodTool</code>
--

Field Value

TYPE	DESCRIPTION
Boolean	

TriggeredByFire

Declaration

[Reloadable]
public bool TriggeredByFire

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

GetProjectilePlaceholder(ModBlockBehaviourHandler)

Declaration

```
public GameObject GetProjectilePlaceholder(ModBlockBehaviourHandler behaviour)
```

Parameters

TYPE	NAME	DESCRIPTION
InternalModding.Blocks.ModBlockBehaviourHandler	behaviour	

Returns

TYPE	DESCRIPTION
UnityEngine.GameObject	

GetProjectilePrefab(ModBlockBehaviourHandler)

Declaration

```
public GameObject GetProjectilePrefab(ModBlockBehaviourHandler behaviour)
```

Parameters

TYPE	NAME	DESCRIPTION
InternalModding.Blocks.ModBlockBehaviourHandler	behaviour	

Returns

TYPE	DESCRIPTION
UnityEngine.GameObject	

OnReload(ModBlockBehaviourHandler)

Declaration

```
public void OnReload(ModBlockBehaviourHandler behaviour)
```

Parameters

TYPE	NAME	DESCRIPTION
InternalModding.Blocks.ModBlockBehaviourHandler	behaviour	

Validate(String)

Declaration

```
protected override bool Validate(string elemName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

[IReloadable](#)

Class ShootingModule.CannonSound

Inheritance

[Object](#)

ShootingModule.CannonSound

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class CannonSound
```

Constructors

CannonSound()

Declaration

```
public CannonSound()
```

Class ShootingModule.CrossbowSounds

Inheritance

[Object](#)

ShootingModule.CrossbowSounds

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class CrossbowSounds
```

Constructors

CrossbowSounds()

Declaration

```
public CrossbowSounds()
```

Class ShootingModule.ModdedProjectileColliderComponent

Inheritance

Object

ShootingModule.ModdedProjectileColliderComponent

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ModdedProjectileColliderComponent : MonoBehaviour
```

Constructors

ModdedProjectileColliderComponent()

Declaration

```
public ModdedProjectileColliderComponent()
```

Fields

projectileScript

Declaration

```
public ProjectileScript projectileScript
```

Field Value

TYPE	DESCRIPTION
ProjectileScript	

Methods

OnCollisionEnter(Collision)

Declaration

```
public void OnCollisionEnter(Collision col)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Collision	col	

Class ShootingModule.Projectile

Inheritance

[Object](#)
[Element](#)

ShootingModule.Projectile

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.Validate\(String\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Projectile : Element, IValidatable
```

Constructors

Projectile()

Declaration

```
public Projectile()
```

Fields

AngularDrag

Declaration

```
public float AngularDrag
```

Field Value

TYPE	DESCRIPTION
Single	

Attaches

Declaration

```
public bool Attaches
```

Field Value

TYPE	DESCRIPTION
Boolean	

BlockDamage

Declaration

public float BlockDamage

Field Value

TYPE	DESCRIPTION
Single	

Colliders

Declaration

[RequireToValidate] public ModCollider[] Colliders

Field Value

TYPE	DESCRIPTION
ModCollider[]	

Drag

Declaration

public float Drag

Field Value

TYPE	DESCRIPTION
Single	

EntityDamage

Declaration

public float EntityDamage

Field Value

TYPE	DESCRIPTION
Single	

FireInteraction

Declaration

[RequireToValidate] public FireInteraction FireInteraction

Field Value

TYPE	DESCRIPTION
InternalModding.Common.FireInteraction	

IgnoreGravity

Declaration

public bool IgnoreGravity

Field Value

TYPE	DESCRIPTION
Boolean	

Mass

Declaration

public float Mass

Field Value

TYPE	DESCRIPTION
Single	

Mesh

Declaration

[RequireToValidate] public MeshReference Mesh
--

Field Value

TYPE	DESCRIPTION
MeshReference	

Texture

Declaration

[RequireToValidate] public ResourceReference Texture

Field Value

TYPE	DESCRIPTION
ResourceReference	

Implements

[IValidatable](#)

Class ShootingModuleBehaviour

Inheritance

Object

ModBlockBehaviour

BlockModuleBehaviour<ShootingModule>

ShootingModuleBehaviour

Inherited Members

BlockModuleBehaviour<ShootingModule>.GetKey(MKeyReference)

BlockModuleBehaviour<ShootingModule>.GetSlider(MSliderReference)

BlockModuleBehaviour<ShootingModule>.GetToggle(MToggleReference)

BlockModuleBehaviour<ShootingModule>.GetValue(MValueReference)

BlockModuleBehaviour<ShootingModule>.GetColourSlider(MColourSliderReference)

BlockModuleBehaviour<ShootingModule>.GetResource(ResourceReference)

BlockModuleBehaviour<ShootingModule>.RawModule

BlockModuleBehaviour<ShootingModule>.ModuleGuid

BlockModuleBehaviour<ShootingModule>.Module

ModBlockBehaviour.OnPrefabCreation()

ModBlockBehaviour.OnBlockPlaced()

ModBlockBehaviour.BuildingUpdate()

ModBlockBehaviour.SimulateUpdateHost()

ModBlockBehaviour.SimulateUpdateClient()

ModBlockBehaviour.BuildingFixedUpdate()

ModBlockBehaviour.SimulateFixedUpdateAlways()

ModBlockBehaviour.SimulateFixedUpdateHost()

ModBlockBehaviour.SimulateFixedUpdateClient()

ModBlockBehaviour.BuildingLateUpdate()

ModBlockBehaviour.SimulateLateUpdateAlways()

ModBlockBehaviour.SimulateLateUpdateHost()

ModBlockBehaviour.SimulateLateUpdateClient()

ModBlockBehaviour.OnSimulateStart()

ModBlockBehaviour.OnSimulateStop()

ModBlockBehaviour.OnStartBurning()

ModBlockBehaviour.OnStopBurning(Boolean)

ModBlockBehaviour.OnSimulateCollisionEnter(Collision)

ModBlockBehaviour.OnSimulateCollisionStay(Collision)

ModBlockBehaviour.OnSimulateCollisionExit(Collision)

ModBlockBehaviour.OnSimulateTriggerEnter(Collider)

ModBlockBehaviour.OnSimulateTriggerStay(Collider)

ModBlockBehaviour.OnSimulateTriggerExit(Collider)

ModBlockBehaviour.OnSimulateParticleCollision(GameObject)

ModBlockBehaviour.OnSave(XDataHolder)

ModBlockBehaviour.OnLoad(XDataHolder)

ModBlockBehaviour.AddKey(String, String, KeyCode)

ModBlockBehaviour.AddKey(MKey)

ModBlockBehaviour.AddTeam(String, String, MPTeam)

ModBlockBehaviour.AddTeam(MTeam)

ModBlockBehaviour.AddText(String, String, String)

ModBlockBehaviour.AddText(MText)

ModBlockBehaviour.AddValue(String, String, Single)

ModBlockBehaviour.AddValue(String, String, Single, Single, Single)

ModBlockBehaviour.AddValue(MValue)
ModBlockBehaviour.AddSlider(String, String, Single, Single, Single)
ModBlockBehaviour.AddSliderUnclamped(String, String, Single, Single, Single)
ModBlockBehaviour.AddSlider(MSlider)
ModBlockBehaviour.AddColourSlider(String, String, Color, Boolean)
ModBlockBehaviour.AddColourSlider(MColourSlider)
ModBlockBehaviour.AddMenu(String, Int32, List<String>, Boolean)
ModBlockBehaviour.AddMenu(MMenu)
ModBlockBehaviour.AddToggle(String, String, Boolean)
ModBlockBehaviour.AddToggle(String, String, String, Boolean)
ModBlockBehaviour.AddToggle(MToggle)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)
ModBlockBehaviour.AddLimits(MLimits)
ModBlockBehaviour.AddCustom<T>(MCustom<T>)
ModBlockBehaviour.IsBurning
ModBlockBehaviour.HasBurnedOut
ModBlockBehaviour.IsFrozen
ModBlockBehaviour.IsDestroyed
ModBlockBehaviour.HasRigidbody
ModBlockBehaviour.Rigidbody
ModBlockBehaviour.BlockBehaviour
ModBlockBehaviour.VisualController
ModBlockBehaviour.Renderer
ModBlockBehaviour.MainVis
ModBlockBehaviour.ShowDebugVisuals
ModBlockBehaviour.Flipped
ModBlockBehaviour.BlockId
ModBlockBehaviour.SimPhysics
ModBlockBehaviour.IsSimulating
ModBlockBehaviour.IsStripped
ModBlockBehaviour.Machine
ModBlockBehaviour.CanFlip
ModBlockBehaviour.DirectionArrow

Namespace: **Modding.Modules.Official**

Assembly: Assembly-CSharp.dll

Syntax

```
public class ShootingModuleBehaviour : BlockModuleBehaviour<ShootingModule>, IModuleBehaviour, IFireEffect
```

Constructors

ShootingModuleBehaviour()

Declaration

```
public ShootingModuleBehaviour()
```

Fields

AmmoLeft

Declaration

```
public int AmmoLeft
```

Field Value

TYPE	DESCRIPTION
Int32	

FireKey

Declaration

```
public MKey FireKey
```

Field Value

TYPE	DESCRIPTION
MKey	

HoldToShootToggle

Declaration

```
public MToggle HoldToShootToggle
```

Field Value

TYPE	DESCRIPTION
MToggle	

PowerSlider

Declaration

```
public MSlider PowerSlider
```

Field Value

TYPE	DESCRIPTION
MSlider	

projectilePlaceholder

Declaration

```
public GameObject projectilePlaceholder
```

Field Value

TYPE	DESCRIPTION
UnityEngine.GameObject	

projectileStart

Declaration

```
public Transform projectileStart
```

Field Value

TYPE	DESCRIPTION
UnityEngine.Transform	

RateOfFire

Declaration

```
public MSlider RateOfFire
```

Field Value

TYPE	DESCRIPTION
MSlider	

Methods

OnIgnite(FireTag, Collider, Boolean)

Declaration

```
public bool OnIgnite(FireTag t, Collider c, bool pyroMode)
```

Parameters

TYPE	NAME	DESCRIPTION
FireTag	t	
UnityEngine.Collider	c	
Boolean	pyroMode	

Returns

TYPE	DESCRIPTION
Boolean	

OnReload()

Declaration

```
public override void OnReload()
```

Overrides

Modding.Modules.BlockModuleBehaviour<Modding.Modules.Official.ShootingModule>.OnReload()

OnReloadAmmo(ref Int32, AmmoType, Boolean, Boolean)

Declaration

```
public override void OnReloadAmmo(ref int units, AmmoType type, bool setAmmo, bool eachBlock)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	units	

TYPE	NAME	DESCRIPTION
AmmoType	type	
Boolean	setAmmo	
Boolean	eachBlock	

Overrides

[ModBlockBehaviour.OnReloadAmmo\(ref Int32, AmmoType, Boolean, Boolean\)](#)

SafeAwake()

Declaration

```
public override void SafeAwake()
```

Overrides

[ModBlockBehaviour.SafeAwake\(\)](#)

SimulateUpdateAlways()

Declaration

```
public override void SimulateUpdateAlways()
```

Overrides

[ModBlockBehaviour.SimulateUpdateAlways\(\)](#)

Class ShootingModuleSoundHolder

Inheritance

Object

ShootingModuleSoundHolder

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class ShootingModuleSoundHolder : MonoBehaviour
```

Constructors

ShootingModuleSoundHolder()

Declaration

```
public ShootingModuleSoundHolder()
```

Fields

CannonClip

Declaration

```
public AudioClip CannonClip
```

Field Value

TYPE	DESCRIPTION
UnityEngine.AudioClip	

CrossBowClips

Declaration

```
public AudioClip[] CrossBowClips
```

Field Value

TYPE	DESCRIPTION
UnityEngine.AudioClip[]	

Class SpewingModule

Inheritance

[Object](#)
[Element](#)
[BlockModule](#)
SpewingModule

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

[BlockModule.ModId](#)
[BlockModule.OnReload\(IReloadable\)](#)
[BlockModule.PreprocessForReloading\(\)](#)
[Element.Validate\(\)](#)
[Element.Validate\(String\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: [Assembly-CSharp.dll](#)

Syntax

```
[Reloadable]
public class SpewingModule : BlockModule, IValidatable, IReloadable
```

Constructors

SpewingModule()

Declaration

```
public SpewingModule()
```

Fields

AcceptFireAmmo

Declaration

```
[Reloadable]
public bool AcceptFireAmmo
```

Field Value

TYPE	DESCRIPTION
Boolean	

BaseAmmo

Declaration

```
[Reloadable]
public float BaseAmmo
```

Field Value

TYPE	DESCRIPTION
Single	

HoldToFireToggle

Declaration

```
[RequireToValidate]
public MToggleReference HoldToFireToggle
```

Field Value

TYPE	DESCRIPTION
MToggleReference	

ParticleSystems

Declaration

```
[CanBeEmpty]
[RequireToValidate]
[Reloadable]
public ParticleHelper.ParticleDefinition[] ParticleSystems
```

Field Value

TYPE	DESCRIPTION
ParticleHelper.ParticleDefinition[]	

RangeSlider

Declaration

```
[RequireToValidate]
public MSliderReference RangeSlider
```

Field Value

TYPE	DESCRIPTION
MSliderReference	

ToggleTimeLimit

Declaration

```
public float ToggleTimeLimit
```

Field Value

TYPE	DESCRIPTION
Single	

ToggleTimeLimitSpecified

Declaration

public bool ToggleTimeLimitSpecified

Field Value

TYPE	DESCRIPTION
Boolean	

TriggerKey

Declaration

[RequireToValidate] public MKeyReference TriggerKey
--

Field Value

TYPE	DESCRIPTION
MKeyReference	

Implements

[IValidatable](#)

[IReloadable](#)

Class SpewingModuleBehaviour

Inheritance

Object

ModBlockBehaviour

BlockModuleBehaviour<SpewingModule>

SpewingModuleBehaviour

Inherited Members

BlockModuleBehaviour<SpewingModule>.GetKey(MKeyReference)

BlockModuleBehaviour<SpewingModule>.GetSlider(MSliderReference)

BlockModuleBehaviour<SpewingModule>.GetToggle(MToggleReference)

BlockModuleBehaviour<SpewingModule>.GetValue(MValueReference)

BlockModuleBehaviour<SpewingModule>.GetColourSlider(MColourSliderReference)

BlockModuleBehaviour<SpewingModule>.GetResource(ResourceReference)

BlockModuleBehaviour<SpewingModule>.RawModule

BlockModuleBehaviour<SpewingModule>.ModuleGuid

BlockModuleBehaviour<SpewingModule>.Module

ModBlockBehaviour.OnPrefabCreation()

ModBlockBehaviour.OnBlockPlaced()

ModBlockBehaviour.BuildingUpdate()

ModBlockBehaviour.SimulateUpdateHost()

ModBlockBehaviour.SimulateUpdateClient()

ModBlockBehaviour.BuildingFixedUpdate()

ModBlockBehaviour.SimulateFixedUpdateAlways()

ModBlockBehaviour.SimulateFixedUpdateHost()

ModBlockBehaviour.SimulateFixedUpdateClient()

ModBlockBehaviour.BuildingLateUpdate()

ModBlockBehaviour.SimulateLateUpdateAlways()

ModBlockBehaviour.SimulateLateUpdateHost()

ModBlockBehaviour.SimulateLateUpdateClient()

ModBlockBehaviour.OnSimulateStart()

ModBlockBehaviour.OnSimulateStop()

ModBlockBehaviour.OnStartBurning()

ModBlockBehaviour.OnStopBurning(Boolean)

ModBlockBehaviour.OnSimulateCollisionEnter(Collision)

ModBlockBehaviour.OnSimulateCollisionStay(Collision)

ModBlockBehaviour.OnSimulateCollisionExit(Collision)

ModBlockBehaviour.OnSimulateTriggerEnter(Collider)

ModBlockBehaviour.OnSimulateTriggerStay(Collider)

ModBlockBehaviour.OnSimulateTriggerExit(Collider)

ModBlockBehaviour.OnSimulateParticleCollision(GameObject)

ModBlockBehaviour.OnSave(XDataHolder)

ModBlockBehaviour.OnLoad(XDataHolder)

ModBlockBehaviour.AddKey(String, String, KeyCode)

ModBlockBehaviour.AddKey(MKey)

ModBlockBehaviour.AddTeam(String, String, MPTeam)

ModBlockBehaviour.AddTeam(MTeam)

ModBlockBehaviour.AddText(String, String, String)

ModBlockBehaviour.AddText(MText)

ModBlockBehaviour.AddValue(String, String, Single)

ModBlockBehaviour.AddValue(String, String, Single, Single, Single)

ModBlockBehaviour.AddValue(MValue)
ModBlockBehaviour.AddSlider(String, String, Single, Single, Single)
ModBlockBehaviour.AddSliderUnclamped(String, String, Single, Single, Single)
ModBlockBehaviour.AddSlider(MSlider)
ModBlockBehaviour.AddColourSlider(String, String, Color, Boolean)
ModBlockBehaviour.AddColourSlider(MColourSlider)
ModBlockBehaviour.AddMenu(String, Int32, List<String>, Boolean)
ModBlockBehaviour.AddMenu(MMenu)
ModBlockBehaviour.AddToggle(String, String, Boolean)
ModBlockBehaviour.AddToggle(String, String, String, Boolean)
ModBlockBehaviour.AddToggle(MToggle)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)
ModBlockBehaviour.AddLimits(MLimits)
ModBlockBehaviour.AddCustom<T>(MCustom<T>)
ModBlockBehaviour.IsBurning
ModBlockBehaviour.HasBurnedOut
ModBlockBehaviour.IsFrozen
ModBlockBehaviour.IsDestroyed
ModBlockBehaviour.HasRigidbody
ModBlockBehaviour.Rigidbody
ModBlockBehaviour.BlockBehaviour
ModBlockBehaviour.VisualController
ModBlockBehaviour.Renderer
ModBlockBehaviour.MainVis
ModBlockBehaviour.ShowDebugVisuals
ModBlockBehaviour.Flipped
ModBlockBehaviour.BlockId
ModBlockBehaviour.SimPhysics
ModBlockBehaviour.IsSimulating
ModBlockBehaviour.IsStripped
ModBlockBehaviour.Machine
ModBlockBehaviour.CanFlip
ModBlockBehaviour.DirectionArrow

Namespace: **Modding.Modules.Official**

Assembly: Assembly-CSharp.dll

Syntax

```
public class SpewingModuleBehaviour : BlockModuleBehaviour<SpewingModule>, IModuleBehaviour
```

Constructors

SpewingModuleBehaviour()

Declaration

```
public SpewingModuleBehaviour()
```

Methods

OnReload()

Declaration

```
public override void OnReload()
```

Overrides

[Modding.Modules.BlockModuleBehaviour](#) <[Modding.Modules.Official.SpewingModule](#)>.OnReload()

OnReloadAmmo(ref Int32, AmmoType, Boolean, Boolean)

Declaration

```
public override void OnReloadAmmo(ref int units, AmmoType type, bool setAmmo, bool eachBlock)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	units	
AmmoType	type	
Boolean	setAmmo	
Boolean	eachBlock	

Overrides

[ModBlockBehaviour.OnReloadAmmo\(ref Int32, AmmoType, Boolean, Boolean\)](#)

SafeAwake()

Declaration

```
public override void SafeAwake()
```

Overrides

[ModBlockBehaviour.SafeAwake\(\)](#)

SimulateUpdateAlways()

Declaration

```
public override void SimulateUpdateAlways()
```

Overrides

[ModBlockBehaviour.SimulateUpdateAlways\(\)](#)

Class SpewingModuleFireParticleTrigger

Inheritance

[Object](#)

SpewingModuleFireParticleTrigger

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class SpewingModuleFireParticleTrigger : MonoBehaviour
```

Constructors

SpewingModuleFireParticleTrigger()

Declaration

```
public SpewingModuleFireParticleTrigger()
```

Fields

Controller

Declaration

```
public FireController Controller
```

Field Value

TYPE	DESCRIPTION
FireController	

Methods

OnParticleCollision(GameObject)

Declaration

```
public void OnParticleCollision(GameObject other)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.GameObject	other	

Class SpinningModule

Inheritance

[Object](#)
[Element](#)
[BlockModule](#)
SpinningModule

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

[BlockModule.ModId](#)
[BlockModule.OnReload\(IReloadable\)](#)
[BlockModule.PreprocessForReloading\(\)](#)
[Element.Validate\(\)](#)
[Element.Validate\(String\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: [Assembly-CSharp.dll](#)

Syntax

```
[Reloadable]
public class SpinningModule : BlockModule, IValidatable, IReloadable
```

Constructors

SpinningModule()

Declaration

```
public SpinningModule()
```

Fields

AccelerationSlider

Declaration

```
[RequireToValidate]
public MSliderReference AccelerationSlider
```

Field Value

TYPE	DESCRIPTION
MSliderReference	

AutomaticToggle

Declaration

```
[RequireToValidate]
public MToggleReference AutomaticToggle
```

Field Value

TYPE	DESCRIPTION
MToggleReference	

Axis

Declaration

```
[Reloadable]
public Direction Axis
```

Field Value

TYPE	DESCRIPTION
Direction	

Backward

Declaration

```
[RequireToValidate]
public MKeyReference Backward
```

Field Value

TYPE	DESCRIPTION
MKeyReference	

Forward

Declaration

```
[RequireToValidate]
public MKeyReference Forward
```

Field Value

TYPE	DESCRIPTION
MKeyReference	

MaxAngularSpeed

Declaration

```
[Reloadable]
public float MaxAngularSpeed
```

Field Value

TYPE	DESCRIPTION
Single	

SpeedSlider

Declaration

<code>[RequireToValidate]</code> <code>public MSliderReference SpeedSlider</code>
--

Field Value

TYPE	DESCRIPTION
MSliderReference	

ToggleModeToggle

Declaration

<code>[RequireToValidate]</code> <code>public MToggleReference ToggleModeToggle</code>

Field Value

TYPE	DESCRIPTION
MToggleReference	

Implements

[IValidatable](#)

[IReloadable](#)

Class SpinningModuleBehaviour

Inheritance

Object

ModBlockBehaviour

BlockModuleBehaviour<SpinningModule>

SpinningModuleBehaviour

Inherited Members

BlockModuleBehaviour<SpinningModule>.GetKey(MKeyReference)

BlockModuleBehaviour<SpinningModule>.GetSlider(MSliderReference)

BlockModuleBehaviour<SpinningModule>.GetToggle(MToggleReference)

BlockModuleBehaviour<SpinningModule>.GetValue(MValueReference)

BlockModuleBehaviour<SpinningModule>.GetColourSlider(MColourSliderReference)

BlockModuleBehaviour<SpinningModule>.GetResource(ResourceReference)

BlockModuleBehaviour<SpinningModule>.RawModule

BlockModuleBehaviour<SpinningModule>.ModuleGuid

BlockModuleBehaviour<SpinningModule>.Module

ModBlockBehaviour.OnPrefabCreation()

ModBlockBehaviour.OnBlockPlaced()

ModBlockBehaviour.BuildingUpdate()

ModBlockBehaviour.SimulateUpdateAlways()

ModBlockBehaviour.SimulateUpdateClient()

ModBlockBehaviour.BuildingFixedUpdate()

ModBlockBehaviour.SimulateFixedUpdateAlways()

ModBlockBehaviour.SimulateFixedUpdateClient()

ModBlockBehaviour.BuildingLateUpdate()

ModBlockBehaviour.SimulateLateUpdateAlways()

ModBlockBehaviour.SimulateLateUpdateHost()

ModBlockBehaviour.SimulateLateUpdateClient()

ModBlockBehaviour.OnSimulateStop()

ModBlockBehaviour.OnStartBurning()

ModBlockBehaviour.OnStopBurning(Boolean)

ModBlockBehaviour.OnSimulateCollisionEnter(Collision)

ModBlockBehaviour.OnSimulateCollisionStay(Collision)

ModBlockBehaviour.OnSimulateCollisionExit(Collision)

ModBlockBehaviour.OnSimulateTriggerEnter(Collider)

ModBlockBehaviour.OnSimulateTriggerStay(Collider)

ModBlockBehaviour.OnSimulateTriggerExit(Collider)

ModBlockBehaviour.OnSimulateParticleCollision(GameObject)

ModBlockBehaviour.OnSave(XDataHolder)

ModBlockBehaviour.OnLoad(XDataHolder)

ModBlockBehaviour.OnReloadAmmo(Int32, AmmoType, Boolean, Boolean)

ModBlockBehaviour.AddKey(String, String, KeyCode)

ModBlockBehaviour.AddKey(MKey)

ModBlockBehaviour.AddTeam(String, String, MPTeam)

ModBlockBehaviour.AddTeam(MTeam)

ModBlockBehaviour.AddText(String, String, String)

ModBlockBehaviour.AddText(MText)

ModBlockBehaviour.AddValue(String, String, Single)

ModBlockBehaviour.AddValue(String, String, Single, Single, Single)

ModBlockBehaviour.AddValue(MValue)

ModBlockBehaviour.AddSlider(String, String, Single, Single, Single)
ModBlockBehaviour.AddSliderUnclamped(String, String, Single, Single, Single)
ModBlockBehaviour.AddSlider(MSlider)
ModBlockBehaviour.AddColourSlider(String, String, Color, Boolean)
ModBlockBehaviour.AddColourSlider(MColourSlider)
ModBlockBehaviour.AddMenu(String, Int32, List<String>, Boolean)
ModBlockBehaviour.AddMenu(MMenu)
ModBlockBehaviour.AddToggle(String, String, Boolean)
ModBlockBehaviour.AddToggle(String, String, String, Boolean)
ModBlockBehaviour.AddToggle(MToggle)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)
ModBlockBehaviour.AddLimits(MLimits)
ModBlockBehaviour.AddCustom<T>(MCustom<T>)
ModBlockBehaviour.IsBurning
ModBlockBehaviour.HasBurnedOut
ModBlockBehaviour.IsFrozen
ModBlockBehaviour.IsDestroyed
ModBlockBehaviour.HasRigidbody
ModBlockBehaviour.Rigidbody
ModBlockBehaviour.BlockBehaviour
ModBlockBehaviour.VisualController
ModBlockBehaviour.Renderer
ModBlockBehaviour.MainVis
ModBlockBehaviour.ShowDebugVisuals
ModBlockBehaviour.Flipped
ModBlockBehaviour.BlockId
ModBlockBehaviour.SimPhysics
ModBlockBehaviour.IsSimulating
ModBlockBehaviour.IsStripped
ModBlockBehaviour.Machine
ModBlockBehaviour.CanFlip
ModBlockBehaviour.DirectionArrow

Namespace: **Modding.Modules.Official**
Assembly: Assembly-CSharp.dll

Syntax

```
public class SpinningModuleBehaviour : BlockModuleBehaviour<SpinningModule>, IModuleBehaviour
```

Constructors

SpinningModuleBehaviour()

Declaration

```
public SpinningModuleBehaviour()
```

Fields

AccelerationSlider

Declaration

```
public MSlider AccelerationSlider
```

Field Value

TYPE	DESCRIPTION
MSlider	

AutomaticToggle

Declaration

public MToggle AutomaticToggle

Field Value

TYPE	DESCRIPTION
MToggle	

BackwardKey

Declaration

public MKey BackwardKey

Field Value

TYPE	DESCRIPTION
MKey	

ForwardKey

Declaration

public MKey ForwardKey

Field Value

TYPE	DESCRIPTION
MKey	

SpeedSlider

Declaration

public MSlider SpeedSlider

Field Value

TYPE	DESCRIPTION
MSlider	

ToggleMode

Declaration

public MToggle ToggleMode

Field Value

TYPE	DESCRIPTION
MToggle	

WheelEquivalenceMultiplier

Declaration

```
public float WheelEquivalenceMultiplier
```

Field Value

TYPE	DESCRIPTION
Single	

Methods

OnReload()

Declaration

```
public override void OnReload()
```

Overrides

Modding.Modules.BlockModuleBehaviour<Modding.Modules.Official.SpinningModule>.OnReload()

OnSimulateStart()

Declaration

```
public override void OnSimulateStart()
```

Overrides

[ModBlockBehaviour.OnSimulateStart\(\)](#)

SafeAwake()

Declaration

```
public override void SafeAwake()
```

Overrides

[ModBlockBehaviour.SafeAwake\(\)](#)

SimulateFixedUpdateHost()

Declaration

```
public override void SimulateFixedUpdateHost()
```

Overrides

[ModBlockBehaviour.SimulateFixedUpdateHost\(\)](#)

SimulateUpdateHost()

Declaration

```
public override void SimulateUpdateHost()
```

Overrides

[ModBlockBehaviour.SimulateUpdateHost\(\)](#)

Class SteeringModule

Inheritance

[Object](#)
[Element](#)
[BlockModule](#)

SteeringModule

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

[BlockModule.ModId](#)
[BlockModule.OnReload\(IReloadable\)](#)
[BlockModule.PreprocessForReloading\(\)](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Modules.Official](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[Reloadable]
public class SteeringModule : BlockModule, IValidatable, IReloadable
```

Constructors

SteeringModule()

Declaration

```
public SteeringModule()
```

Fields

AutomaticToggle

Declaration

```
[RequireToValidate]
public MToggleReference AutomaticToggle
```

Field Value

TYPE	DESCRIPTION
MToggleReference	

Axis

Declaration

```
public Direction Axis
```

Field Value

TYPE	DESCRIPTION
Direction	

HasLimits

Declaration

```
public bool HasLimits
```

Field Value

TYPE	DESCRIPTION
Boolean	

LeftKey

Declaration

```
[RequireToValidate]  
public MKeyReference LeftKey
```

Field Value

TYPE	DESCRIPTION
MKeyReference	

LimitsDefaultMax

Declaration

```
public float LimitsDefaultMax
```

Field Value

TYPE	DESCRIPTION
Single	

LimitsDefaultMaxSpecified

Declaration

```
public bool LimitsDefaultMaxSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

LimitsDefaultMin

Declaration


```
public float LimitsDefaultMin
```

Field Value

TYPE	DESCRIPTION
Single	

LimitsDefaultMinSpecified

Declaration

```
public bool LimitsDefaultMinSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

LimitsDisplay

Declaration

```
[RequireToValidate]  
[Reloadable]  
public TransformValues LimitsDisplay
```

Field Value

TYPE	DESCRIPTION
TransformValues	

LimitsHighestAngle

Declaration

```
[Reloadable]  
public float LimitsHighestAngle
```

Field Value

TYPE	DESCRIPTION
Single	

LimitsHighestAngleSpecified

Declaration

```
[Reloadable]  
public bool LimitsHighestAngleSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

MaxAngularSpeed

Declaration

[Reloadable] public float MaxAngularSpeed
--

Field Value

TYPE	DESCRIPTION
Single	

RightKey

Declaration

[RequireToValidate] public MKeyReference RightKey
--

Field Value

TYPE	DESCRIPTION
MKeyReference	

SpeedSlider

Declaration

[RequireToValidate] public MSliderReference SpeedSlider
--

Field Value

TYPE	DESCRIPTION
MSliderReference	

TargetAngleSpeed

Declaration

[Reloadable] public float TargetAngleSpeed

Field Value

TYPE	DESCRIPTION
Single	

Methods

Validate(String)

Declaration

protected override bool Validate(string elemName)

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

[IReloadable](#)

Class SteeringModuleBehaviour

Inheritance

Object

ModBlockBehaviour

BlockModuleBehaviour<SteeringModule>

SteeringModuleBehaviour

Inherited Members

BlockModuleBehaviour<SteeringModule>.GetKey(MKeyReference)

BlockModuleBehaviour<SteeringModule>.GetSlider(MSliderReference)

BlockModuleBehaviour<SteeringModule>.GetToggle(MToggleReference)

BlockModuleBehaviour<SteeringModule>.GetValue(MValueReference)

BlockModuleBehaviour<SteeringModule>.GetColourSlider(MColourSliderReference)

BlockModuleBehaviour<SteeringModule>.GetResource(ResourceReference)

BlockModuleBehaviour<SteeringModule>.RawModule

BlockModuleBehaviour<SteeringModule>.ModuleGuid

BlockModuleBehaviour<SteeringModule>.Module

ModBlockBehaviour.OnPrefabCreation()

ModBlockBehaviour.OnBlockPlaced()

ModBlockBehaviour.BuildingUpdate()

ModBlockBehaviour.SimulateUpdateAlways()

ModBlockBehaviour.SimulateUpdateClient()

ModBlockBehaviour.BuildingFixedUpdate()

ModBlockBehaviour.SimulateFixedUpdateAlways()

ModBlockBehaviour.SimulateFixedUpdateHost()

ModBlockBehaviour.SimulateFixedUpdateClient()

ModBlockBehaviour.BuildingLateUpdate()

ModBlockBehaviour.SimulateLateUpdateAlways()

ModBlockBehaviour.SimulateLateUpdateHost()

ModBlockBehaviour.SimulateLateUpdateClient()

ModBlockBehaviour.OnSimulateStart()

ModBlockBehaviour.OnSimulateStop()

ModBlockBehaviour.OnStartBurning()

ModBlockBehaviour.OnStopBurning(Boolean)

ModBlockBehaviour.OnSimulateCollisionEnter(Collision)

ModBlockBehaviour.OnSimulateCollisionStay(Collision)

ModBlockBehaviour.OnSimulateCollisionExit(Collision)

ModBlockBehaviour.OnSimulateTriggerEnter(Collider)

ModBlockBehaviour.OnSimulateTriggerStay(Collider)

ModBlockBehaviour.OnSimulateTriggerExit(Collider)

ModBlockBehaviour.OnSimulateParticleCollision(GameObject)

ModBlockBehaviour.OnSave(XDataHolder)

ModBlockBehaviour.OnLoad(XDataHolder)

ModBlockBehaviour.OnReloadAmmo(Int32, AmmoType, Boolean, Boolean)

ModBlockBehaviour.AddKey(String, String, KeyCode)

ModBlockBehaviour.AddKey(MKey)

ModBlockBehaviour.AddTeam(String, String, MPTeam)

ModBlockBehaviour.AddTeam(MTeam)

ModBlockBehaviour.AddText(String, String, String)

ModBlockBehaviour.AddText(MText)

ModBlockBehaviour.AddValue(String, String, Single)

ModBlockBehaviour.AddValue(String, String, Single, Single, Single)
ModBlockBehaviour.AddValue(MValue)
ModBlockBehaviour.AddSlider(String, String, Single, Single, Single)
ModBlockBehaviour.AddSliderUnclamped(String, String, Single, Single, Single)
ModBlockBehaviour.AddSlider(MSlider)
ModBlockBehaviour.AddColourSlider(String, String, Color, Boolean)
ModBlockBehaviour.AddColourSlider(MColourSlider)
ModBlockBehaviour.AddMenu(String, Int32, List<String>, Boolean)
ModBlockBehaviour.AddMenu(MMenu)
ModBlockBehaviour.AddToggle(String, String, Boolean)
ModBlockBehaviour.AddToggle(String, String, String, Boolean)
ModBlockBehaviour.AddToggle(MToggle)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform)
ModBlockBehaviour.AddLimits(String, String, Single, Single, Single, FauxTransform, ILimitsDisplay)
ModBlockBehaviour.AddLimits(MLimits)
ModBlockBehaviour.AddCustom<T>(MCustom<T>)
ModBlockBehaviour.IsBurning
ModBlockBehaviour.HasBurnedOut
ModBlockBehaviour.IsFrozen
ModBlockBehaviour.IsDestroyed
ModBlockBehaviour.HasRigidbody
ModBlockBehaviour.Rigidbody
ModBlockBehaviour.BlockBehaviour
ModBlockBehaviour.VisualController
ModBlockBehaviour.Renderer
ModBlockBehaviour.MainVis
ModBlockBehaviour.ShowDebugVisuals
ModBlockBehaviour.Flipped
ModBlockBehaviour.BlockId
ModBlockBehaviour.SimPhysics
ModBlockBehaviour.IsSimulating
ModBlockBehaviour.IsStripped
ModBlockBehaviour.Machine
ModBlockBehaviour.CanFlip
ModBlockBehaviour.DirectionArrow

Namespace: **Modding.Modules.Official**

Assembly: Assembly-CSharp.dll

Syntax

```
public class SteeringModuleBehaviour : BlockModuleBehaviour<SteeringModule>, IModuleBehaviour
```

Constructors

SteeringModuleBehaviour()

Declaration

```
public SteeringModuleBehaviour()
```

Methods

OnReload()

Declaration

```
public override void OnReload()
```

Overrides

Modding.Modules.BlockModuleBehaviour<Modding.Modules.Official.SteeringModule>.OnReload()

SafeAwake()

Declaration

```
public override void SafeAwake()
```

Overrides

[ModBlockBehaviour.SafeAwake\(\)](#)

SimulateUpdateHost()

Declaration

```
public override void SimulateUpdateHost()
```

Overrides

[ModBlockBehaviour.SimulateUpdateHost\(\)](#)

Start()

Declaration

```
public void Start()
```

Namespace Modding.Serialization

Classes

[BoxModCollider](#)

ModCollider for BoxColliders.

[CanBeEmptyAttribute](#)

Can be used to tell the serialization system that an XML list/array is allowed to be empty.

[CapsuleModCollider](#)

ModCollider for CapsuleColliders.

Note: Debug visuals for CapsuleColliders are displayed as boxes.

[CapsuleModCollider.CapsuleWrapper](#)

[DirectionExtensions](#)

[Element](#)

[MapperTypeDefinition](#)

Base class for mapper type XML definitions.

[MapperTypeReference](#)

Base class for references to defined mapper types.

[MColourSliderDefinition](#)

Used to define an MColourSlider in an XML file.

[MColourSliderReference](#)

Used to reference an MColourSlider in an XML file.

[MeshReference](#)

[MeshTexturePair](#)

[MKeyDefinition](#)

Used to define an MKey in an XML file.

[MKeyReference](#)

Used to reference an MKey in an XML file.

[ModCollider](#)

Abstract base class for all collider types supported by mods.

[MSliderDefinition](#)

Used to define an MSlider in an XML file.

[MSliderReference](#)

Used to reference an MSlider in an XML file.

[MToggleDefinition](#)

Used to define an MToggle in an XML file.

[MToggleReference](#)

Used to reference an MToggle in an XML file.

[MValueDefinition](#)

Used to define an MValue in an XML file.

[MValueReference](#)

Used to reference an MValue in an XML file.

[ReloadableAttribute](#)

[RequireToValidateAttribute](#)

Can be used to mark XML elements/attributes that extend the Element class to be automatically recursively validated.

[ResourceReference](#)

[SphereModCollider](#)

ModCollider for SphereColliders.

[TransformValues](#)

Utility class to easily include Position, Rotation, and/or Scale for serialization. See the Common Elements serialization documentation for details on how to use it.

[VanillaBlockType](#)

[VanillaEntityType](#)

Structs

[Vector3](#)

Unity's Vector3 does not (de)serialize correctly, so this wrapper is used to do it instead.

Interfaces

[IReloadable](#)

[IValidatable](#)

The base interface a class must implement for it to be validated using the generic validation system.

You should normally use the Element class instead of implementing this interface yourself.

Enums

[Direction](#)

Class BoxModCollider

ModCollider for BoxColliders.

Inheritance

[Object](#)
[Element](#)
[ModCollider](#)
BoxModCollider

Implements

[IValidatable](#)

Inherited Members

[ModCollider.Position](#)
[ModCollider.Layer](#)
[ModCollider.LayerSpecified](#)
[ModCollider.Trigger](#)
[ModCollider.IgnoreForGhost](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class BoxModCollider : ModCollider, IValidatable
```

Constructors

BoxModCollider()

Declaration

```
public BoxModCollider()
```

Properties

Rotation

Rotation of the collider object.

Declaration

```
public Vector3 Rotation { get; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

Scale

Scale of the collider object.

Declaration

```
public Vector3 Scale { get; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

Methods

CreateCollider(Transform)

Creates a game object with a collider attached, set up according to the ModCollider instance.

Declaration

```
public override Collider CreateCollider(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Collider	The new Collider component.

Overrides

[ModCollider.CreateCollider\(Transform\)](#)

CreateVisual(Transform)

Creates a debug visual object to represent this collider.

Declaration

```
public override Transform CreateVisual(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	The Transform of the new object.

Overrides

[ModCollider.CreateVisual\(Transform\)](#)

Validate(String)

Declaration

```
protected override bool Validate(string elemName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class CanBeEmptyAttribute

Can be used to tell the serialization system that an XML list/array is allowed to be empty.

Inheritance

[Object](#)

[Attribute](#)

CanBeEmptyAttribute

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]  
public class CanBeEmptyAttribute : Attribute, _Attribute
```

Constructors

CanBeEmptyAttribute()

Declaration

```
public CanBeEmptyAttribute()
```

Class CapsuleModCollider

ModCollider for CapsuleColliders.

Note: Debug visuals for CapsuleColliders are displayed as boxes.

Inheritance

- Object
- Element
- ModCollider
- CapsuleModCollider

Implements

- IValidatable

Inherited Members

- ModCollider.Position
- ModCollider.Layer
- ModCollider.LayerSpecified
- ModCollider.Trigger
- ModCollider.IgnoreForGhost
- Element.Validate()
- Element.Validate(String)
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**

Assembly: Assembly-CSharp.dll

Syntax

```
public class CapsuleModCollider : ModCollider, IValidatable
```

Constructors

CapsuleModCollider()

Declaration

```
public CapsuleModCollider()
```

Fields

Capsule

Definition of the capsule properties, used for deserialization.

Declaration

```
[RequireToValidate]  
public CapsuleModCollider.CapsuleWrapper Capsule
```

Field Value

TYPE	DESCRIPTION
CapsuleModCollider.CapsuleWrapper	

Properties

Dir

Along which axis the capsule points.

Declaration

```
public Direction Dir { get; }
```

Property Value

TYPE	DESCRIPTION
Direction	

Height

Height of the collider. (Parallel to the direction)

Declaration

```
public float Height { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

Radius

Radius of the collider. (Perpendicular to the direction)

Declaration

```
public float Radius { get; }
```

Property Value

TYPE	DESCRIPTION
Single	

Rotation

Rotation of the collider.

Declaration

```
public Vector3 Rotation { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

Methods

CreateCollider(Transform)

Creates a game object with a collider attached, set up according to the ModCollider instance.

Declaration

```
public override Collider CreateCollider(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Collider	The new Collider component.

Overrides

[ModCollider.CreateCollider\(Transform\)](#)

CreateVisual(Transform)

Creates a debug visual object to represent this collider.

Declaration

```
public override Transform CreateVisual(Transform overallParent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	overallParent	

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	The Transform of the new object.

Overrides

[ModCollider.CreateVisual\(Transform\)](#)

Implements

[IValidatable](#)

Class CapsuleModCollider.CapsuleWrapper

Inheritance

[Object](#)
[Element](#)

CapsuleModCollider.CapsuleWrapper

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class CapsuleWrapper : Element, IValidatable
```

Constructors

CapsuleWrapper()

Declaration

```
public CapsuleWrapper()
```

Fields

Direction

Along which axis the capsule points.

Declaration

```
public Direction Direction
```

Field Value

TYPE	DESCRIPTION
Direction	

Height

Height of the collider. (Parallel to the direction)

Declaration

```
public float Height
```


Field Value

TYPE	DESCRIPTION
Single	

Radius

Radius of the collider. (Perpendicular to the direction)

Declaration

<code>public float Radius</code>

Field Value

TYPE	DESCRIPTION
Single	

Methods

Validate(String)

Declaration

<code>protected override bool Validate(string elemName)</code>
--

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Enum Direction

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public enum Direction
```

Fields

NAME	DESCRIPTION
X	
Y	
Z	

Extension Methods

- [DirectionExtensions.ToAxisVector\(\)](#)
- [DirectionExtensions.GetAxisComponent\(Vector3\)](#)

Class DirectionExtensions

Inheritance

[Object](#)

DirectionExtensions

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public static class DirectionExtensions
```

Methods

GetAxisComponent(Direction, Vector3)

Declaration

```
public static float GetAxisComponent(this Direction dir, Vector3 other)
```

Parameters

TYPE	NAME	DESCRIPTION
Direction	dir	
UnityEngine.Vector3	other	

Returns

TYPE	DESCRIPTION
Single	

ToAxisVector(Direction)

Declaration

```
public static Vector3 ToAxisVector(this Direction dir)
```

Parameters

TYPE	NAME	DESCRIPTION
Direction	dir	

Returns

TYPE	DESCRIPTION
UnityEngine.Vector3	

Class Element

Inheritance

- [Object](#)
- [Element](#)
- [EventProperty](#)
- [EventProperty.Choice.Option](#)
- [BlockModule](#)
- [ParticleHelper.CollisionSettings](#)
- [ParticleHelper.ParticleDefinition](#)
- [ShootingModule.Projectile](#)
- [CapsuleModCollider.CapsuleWrapper](#)
- [MapperTypeDefinition](#)
- [MapperTypeReference](#)
- [MeshTexturePair](#)
- [ModCollider](#)
- [ResourceReference](#)
- [TransformValues](#)
- [VanillaBlockType](#)
- [VanillaEntityType](#)

Implements

- [IValidatable](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class Element : IValidatable
```

Constructors

Element()

Declaration

```
public Element()
```

Properties

AttributesUsed

All the attributes applied to this element in the XML file. Used for validating deserialized objects.

Declaration

```
public string AttributesUsed { get; }
```

Property Value

TYPE	DESCRIPTION
String	

ElementsUsed

All the elements applied as child elements to this element in the XML file. Used for validating deserialized objects.

Declaration

```
public string ElementsUsed { get; }
```

Property Value

TYPE	DESCRIPTION
String	

FileName

File name of the file this object was deserialized from. Used for generating error messages when deserializing.

Declaration

```
public string FileName { get; }
```

Property Value

TYPE	DESCRIPTION
String	

LineNumber

Line number of this XML element. Used for generating error messages when deserializing.

Declaration

```
public int LineNumber { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

LinePosition

Position within its line of this XML element. Used for generating error messages when deserializing.

Declaration

```
public int LinePosition { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Methods

InvalidData(String, String)

Return this if invalid data is specified during validation.

Declaration

```
protected bool InvalidData(string elemName, string error)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	Name of this element.
String	error	Error message.

Returns

TYPE	DESCRIPTION
Boolean	False

MissingAttribute(String, String)

Return this if a required attribute is missing during validation.

Declaration

```
protected bool MissingAttribute(string elemName, string missing)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	Name of this element.
String	missing	Name of the missing attribute.

Returns

TYPE	DESCRIPTION
Boolean	False

MissingElement(String, String)

Return this if a required element is missing during validation.

Declaration

```
protected bool MissingElement(string elemName, string missing)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	Name of this element.

TYPE	NAME	DESCRIPTION
String	missing	Name of the missing element.

Returns

TYPE	DESCRIPTION
Boolean	False

Validate()

Validation with a default name.

Declaration

```
protected virtual bool Validate()
```

Returns

TYPE	DESCRIPTION
Boolean	True if the element is valid, false if not.

Validate(String)

Basic method to override for custom validation logic. Should call base method to also do the normal validation tasks.

Declaration

```
protected virtual bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	Name of the element in the current context, for error messages.

Returns

TYPE	DESCRIPTION
Boolean	true if the element is valid, false if not.

Warn(String, String)

Print a formatted warning during validation.

Declaration

```
protected void Warn(string elemName, string warning)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	Name of this element.
String	warning	Warning message.

Implements

[IValidatable](#)

Interface IReloadable

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public interface IReloadable
```

Methods

OnReload(IReloadable)

Declaration

```
void OnReload(IReloadable newObject)
```

Parameters

TYPE	NAME	DESCRIPTION
IReloadable	newObject	

PreprocessForReloading()

Declaration

```
void PreprocessForReloading()
```

Interface IValidatable

The base interface a class must implement for it to be validated using the generic validation system.

You should normally use the Element class instead of implementing this interface yourself.

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public interface IValidatable
```

Properties

AttributesUsed

Declaration

```
string AttributesUsed { get; }
```

Property Value

TYPE	DESCRIPTION
String	

ElementsUsed

Declaration

```
string ElementsUsed { get; }
```

Property Value

TYPE	DESCRIPTION
String	

FileName

Declaration

```
string FileName { get; }
```

Property Value

TYPE	DESCRIPTION
String	

LineNumber

Declaration

```
int LineNumber { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

LinePosition

Declaration

```
int LinePosition { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Class MapperTypeDefinition

Base class for mapper type XML definitions.

Inheritance

- Object
- Element
- MapperTypeDefinition
- MColourSliderDefinition
- MKeyDefinition
- MSliderDefinition
- MToggleDefinition
- MValueDefinition

Implements

- IValidatable

Inherited Members

- Element.Validate()
- Element.Validate(String)
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**

Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class MapperTypeDefinition : Element, IValidatable
```

Constructors

MapperTypeDefinition()

Declaration

```
protected MapperTypeDefinition()
```

Fields

DisplayName

Name of this mapper type to display in the UI.

Declaration

```
public string DisplayName
```

Field Value

TYPE	DESCRIPTION
String	

Key

Used as key in the save data as well as identifying this mapper type in references.

Declaration

```
public string Key
```

Field Value

TYPE	DESCRIPTION
String	

ShowInMapper

Whether to show this mapper type in the UI. Optional, defaults to true.

Declaration

```
public bool ShowInMapper
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public abstract MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Implements

[IValidatable](#)

Class MapperTypeReference

Base class for references to defined mapper types.

Inheritance

- Object
- Element
- MapperTypeReference
- MColourSliderReference
- MKeyReference
- MSliderReference
- MToggleReference
- MValueReference

Implements

- IValidatable

Inherited Members

- Element.Validate()
- Element.Validate(String)
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**

Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class MapperTypeReference : Element, IValidatable
```

Constructors

MapperTypeReference()

Declaration

```
protected MapperTypeReference()
```

Fields

Key

Key of the mapper type to reference.

Declaration

```
public string Key
```

Field Value

TYPE	DESCRIPTION
String	

Implements

[IValidatable](#)

Class MColourSliderDefinition

Used to define an MColourSlider in an XML file.

Inheritance

[Object](#)
[Element](#)
[MapperTypeDefinition](#)
MColourSliderDefinition

Implements

[IValidatable](#)

Inherited Members

[MapperTypeDefinition.Key](#)
[MapperTypeDefinition.DisplayName](#)
[MapperTypeDefinition.ShowInMapper](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class MColourSliderDefinition : MapperTypeDefinition, IValidatable
```

Constructors

MColourSliderDefinition()

Declaration

```
public MColourSliderDefinition()
```

Fields

A

Alpha component of the default colour, between 0 and 1.

Declaration

```
public float A
```

Field Value

TYPE	DESCRIPTION
Single	

B

Blue component of the default colour, between 0 and 1.

Declaration

```
public float B
```

Field Value

TYPE	DESCRIPTION
Single	

G

Green component of the default colour, between 0 and 1.

Declaration

```
public float G
```

Field Value

TYPE	DESCRIPTION
Single	

R

Red component of the default colour, between 0 and 1.

Declaration

```
public float R
```

Field Value

TYPE	DESCRIPTION
Single	

SnapColors

Whether to snap to predefined colour values.

Declaration

```
public bool SnapColors
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public override MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Overrides

[MapperTypeDefinition.Create\(SaveableDataHolder\)](#)

Validate(String)

Declaration

```
protected override bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class MColourSliderReference

Used to reference an MColourSlider in an XML file.

Inheritance

[Object](#)

[Element](#)

[MapperTypeReference](#)

MColourSliderReference

Implements

[IValidatable](#)

Inherited Members

[MapperTypeReference.Key](#)

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MColourSliderReference : MapperTypeReference, IValidatable
```

Constructors

MColourSliderReference()

Declaration

```
public MColourSliderReference()
```

Implements

[IValidatable](#)

Class MeshReference

Inheritance

[Object](#)
[Element](#)
[ResourceReference](#)

MeshReference

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

[ResourceReference.Name](#)
[ResourceReference.Validate\(String\)](#)
[Element.Validate\(\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
[Reloadable]
public class MeshReference : ResourceReference, IValidatable, IReloadable
```

Constructors

MeshReference()

Declaration

```
public MeshReference()
```

Fields

Position

Declaration

```
[Reloadable]
public Vector3 Position
```

Field Value

TYPE	DESCRIPTION
Vector3	

Rotation

Declaration

```
[Reloadable]
public Vector3 Rotation
```

Field Value

TYPE	DESCRIPTION
Vector3	

Scale

Declaration

```
[Reloadable]
public Vector3 Scale
```

Field Value

TYPE	DESCRIPTION
Vector3	

Methods

OnReload(IReloadable)

Declaration

```
public void OnReload(IReloadable newObject)
```

Parameters

TYPE	NAME	DESCRIPTION
IReloadable	newObject	

PreprocessForReloading()

Declaration

```
public void PreprocessForReloading()
```

SetTransformValues(Transform)

Declaration

```
public void SetTransformValues(Transform t)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	t	

Implements

[IValidatable](#)

[IReloadable](#)

Class MeshTexturePair

Inheritance

[Object](#)
[Element](#)

MeshTexturePair

Implements

[IValidatable](#)
[IReloadable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.Validate\(String\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[Reloadable]
public class MeshTexturePair : Element, IValidatable, IReloadable
```

Constructors

MeshTexturePair()

Declaration

```
public MeshTexturePair()
```

Fields

Mesh

Declaration

```
public ModMesh Mesh
```

Field Value

TYPE	DESCRIPTION
ModMesh	

MeshReference

Declaration

```
[Reloadable]
public MeshReference MeshReference
```

Field Value

TYPE	DESCRIPTION
MeshReference	

Texture

Declaration

<code>public ModTexture Texture</code>
--

Field Value

TYPE	DESCRIPTION
ModTexture	

TextureReference

Declaration

<code>public ResourceReference TextureReference</code>
--

Field Value

TYPE	DESCRIPTION
ResourceReference	

Methods

OnReload(IReloadable)

Declaration

<code>public void OnReload(IReloadable newObject)</code>
--

Parameters

TYPE	NAME	DESCRIPTION
IReloadable	newObject	

PreprocessForReloading()

Declaration

<code>public void PreprocessForReloading()</code>

Implements

[IValidatable](#)

[IReloadable](#)

Class MKeyDefinition

Used to define an MKey in an XML file.

Inheritance

[Object](#)
[Element](#)
[MapperTypeDefinition](#)
MKeyDefinition

Implements

[IValidatable](#)

Inherited Members

[MapperTypeDefinition.Key](#)
[MapperTypeDefinition.DisplayName](#)
[MapperTypeDefinition.ShowInMapper](#)
[Element.Validate\(\)](#)
[Element.Validate\(String\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class MKeyDefinition : MapperTypeDefinition, IValidatable
```

Constructors

MKeyDefinition()

Declaration

```
public MKeyDefinition()
```

Fields

Default

Default key binding.

Declaration

```
public KeyCode Default
```

Field Value

TYPE	DESCRIPTION
UnityEngine.KeyCode	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public override MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Overrides

[MapperTypeDefinition.Create\(SaveableDataHolder\)](#)

Implements

[IValidatable](#)

Class MKeyReference

Used to reference an MKey in an XML file.

Inheritance

[Object](#)

[Element](#)

[MapperTypeReference](#)

MKeyReference

Implements

[IValidatable](#)

Inherited Members

[MapperTypeReference.Key](#)

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MKeyReference : MapperTypeReference, IValidatable
```

Constructors

MKeyReference()

Declaration

```
public MKeyReference()
```

Implements

[IValidatable](#)

Class ModCollider

Abstract base class for all collider types supported by mods.

Inheritance

- Object
- Element
- ModCollider
- BoxModCollider
- CapsuleModCollider
- SphereModCollider

Implements

- IValidatable

Inherited Members

- Element.Validate()
- Element.Validate(String)
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**
Assembly: Assembly-CSharp.dll

Syntax

```
public abstract class ModCollider : Element, IValidatable
```

Constructors

ModCollider()

Declaration

```
protected ModCollider()
```

Properties

IgnoreForGhost

Whether the collider should be ignored for the ghost colliders.

Declaration

```
public bool IgnoreForGhost { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Layer

Layer of the collider, if manually specified.

Declaration

```
public int Layer { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

LayerSpecified

Declaration

```
public bool LayerSpecified { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Position

Local position of the collider.

Declaration

```
public Vector3 Position { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

Trigger

Whether the collider is a trigger.

Declaration

```
public bool Trigger { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

CreateCollider(Transform)

Creates a game object with a collider attached, set up according to the ModCollider instance.

Declaration

```
public abstract Collider CreateCollider(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Collider	The new Collider component.

CreateVisual(Transform)

Creates a debug visual object to represent this collider.

Declaration

<pre>public abstract Transform CreateVisual(Transform parent)</pre>

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	The Transform of the new object.

Implements

[IValidatable](#)

Class MSliderDefinition

Used to define an MSlider in an XML file.

Inheritance

[Object](#)
[Element](#)
[MapperTypeDefinition](#)
MSliderDefinition

Implements

[IValidatable](#)

Inherited Members

[MapperTypeDefinition.Key](#)
[MapperTypeDefinition.DisplayName](#)
[MapperTypeDefinition.ShowInMapper](#)
[Element.Validate\(\)](#)
[Element.Validate\(String\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class MSliderDefinition : MapperTypeDefinition, IValidatable
```

Constructors

MSliderDefinition()

Declaration

```
public MSliderDefinition()
```

Fields

Default

Default value of the slider.

Declaration

```
public float Default
```

Field Value

TYPE	DESCRIPTION
Single	

Max

Maximum value of the slider.

Declaration

```
public float Max
```

Field Value

TYPE	DESCRIPTION
Single	

Min

Minimum value of the slider.

Declaration

```
public float Min
```

Field Value

TYPE	DESCRIPTION
Single	

Unclamped

If set to true, it is possible to type in values outside of the normal maximum and minimum. In this case max and min only apply to the slider itself.

Declaration

```
public bool Unclamped
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public override MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Overrides

[MapperTypeDefinition.Create\(SaveableDataHolder\)](#)

Implements

[IValidatable](#)

Class MSliderReference

Used to reference an MSlider in an XML file.

Inheritance

[Object](#)

[Element](#)

[MapperTypeReference](#)

MSliderReference

Implements

[IValidatable](#)

Inherited Members

[MapperTypeReference.Key](#)

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MSliderReference : MapperTypeReference, IValidatable
```

Constructors

MSliderReference()

Declaration

```
public MSliderReference()
```

Implements

[IValidatable](#)

Class MToggleDefinition

Used to define an MToggle in an XML file.

Inheritance

[Object](#)
[Element](#)
[MapperTypeDefinition](#)
MToggleDefinition

Implements

[IValidatable](#)

Inherited Members

[MapperTypeDefinition.Key](#)
[MapperTypeDefinition.DisplayName](#)
[MapperTypeDefinition.ShowInMapper](#)
[Element.Validate\(\)](#)
[Element.Validate\(String\)](#)
[Element.MissingElement\(String, String\)](#)
[Element.MissingAttribute\(String, String\)](#)
[Element.InvalidData\(String, String\)](#)
[Element.Warn\(String, String\)](#)
[Element.LineNumber](#)
[Element.LinePosition](#)
[Element.AttributesUsed](#)
[Element.ElementsUsed](#)
[Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class MToggleDefinition : MapperTypeDefinition, IValidatable
```

Constructors

MToggleDefinition()

Declaration

```
public MToggleDefinition()
```

Fields

Default

Default state of the toggle.

Declaration

```
public bool Default
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public override MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Overrides

[MapperTypeDefinition.Create\(SaveableDataHolder\)](#)

Implements

[IValidatable](#)

Class MToggleReference

Used to reference an MToggle in an XML file.

Inheritance

[Object](#)

[Element](#)

[MapperTypeReference](#)

MToggleReference

Implements

[IValidatable](#)

Inherited Members

[MapperTypeReference.Key](#)

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MToggleReference : MapperTypeReference, IValidatable
```

Constructors

MToggleReference()

Declaration

```
public MToggleReference()
```

Implements

[IValidatable](#)

Class MValueDefinition

Used to define an MValue in an XML file.

Inheritance

- Object
- Element
- MapperTypeDefinition
- MValueDefinition

Implements

- IValidatable

Inherited Members

- MapperTypeDefinition.Key
- MapperTypeDefinition.DisplayName
- MapperTypeDefinition.ShowInMapper
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**
Assembly: Assembly-CSharp.dll

Syntax

```
public class MValueDefinition : MapperTypeDefinition, IValidatable
```

Constructors

MValueDefinition()

Declaration

```
public MValueDefinition()
```

Fields

Default

Default value.

Declaration

```
public float Default
```

Field Value

TYPE	DESCRIPTION
Single	

Max

Maximum value.

Declaration

```
public float Max
```

Field Value

TYPE	DESCRIPTION
Single	

MaxSpecified

Declaration

```
public bool MaxSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Min

Minimum value.

Declaration

```
public float Min
```

Field Value

TYPE	DESCRIPTION
Single	

MinSpecified

Declaration

```
public bool MinSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Create(SaveableDataHolder)

Adds this mapper type to the given holder.

Declaration

```
public override MapperType Create(SaveableDataHolder holder)
```

Parameters

TYPE	NAME	DESCRIPTION
SaveableDataHolder	holder	

Returns

TYPE	DESCRIPTION
MapperType	The created MapperType instance.

Overrides

[MapperTypeDefinition.Create\(SaveableDataHolder\)](#)

Validate(String)

Declaration

protected override bool Validate(string elementName)
--

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class MValueReference

Used to reference an MValue in an XML file.

Inheritance

[Object](#)

[Element](#)

[MapperTypeReference](#)

MValueReference

Implements

[IValidatable](#)

Inherited Members

[MapperTypeReference.Key](#)

[Element.Validate\(\)](#)

[Element.Validate\(String\)](#)

[Element.MissingElement\(String, String\)](#)

[Element.MissingAttribute\(String, String\)](#)

[Element.InvalidData\(String, String\)](#)

[Element.Warn\(String, String\)](#)

[Element.LineNumber](#)

[Element.LinePosition](#)

[Element.AttributesUsed](#)

[Element.ElementsUsed](#)

[Element.FileName](#)

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public class MValueReference : MapperTypeReference, IValidatable
```

Constructors

MValueReference()

Declaration

```
public MValueReference()
```

Implements

[IValidatable](#)

Class ReloadableAttribute

Inheritance

[Object](#)

[Attribute](#)

ReloadableAttribute

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Property | AttributeTargets.Field)]
public class ReloadableAttribute : Attribute, _Attribute
```

Constructors

ReloadableAttribute()

Declaration

```
public ReloadableAttribute()
```

Fields

CallOnNewBeforeOnReload

Declaration

```
public Action<IReloadable> CallOnNewBeforeOnReload
```

Field Value

TYPE	DESCRIPTION
Action<IReloadable>	

Class RequireToValidateAttribute

Can be used to mark XML elements/attributes that extend the Element class to be automatically recursively validated.

Inheritance

[Object](#)

[Attribute](#)

RequireToValidateAttribute

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class RequireToValidateAttribute : Attribute, _Attribute
```

Constructors

RequireToValidateAttribute()

Declaration

```
public RequireToValidateAttribute()
```

Class ResourceReference

Inheritance

[Object](#)
[Element](#)

ResourceReference
[MeshReference](#)

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class ResourceReference : Element, IValidatable
```

Constructors

ResourceReference()

Declaration

```
public ResourceReference()
```

Fields

Name

Declaration

```
public string Name
```

Field Value

TYPE	DESCRIPTION
String	

Methods

Validate(String)

Declaration

```
protected override bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class SphereModCollider

ModCollider for SphereColliders.

Inheritance

- Object
- Element
- ModCollider
- SphereModCollider

Implements

- IValidatable

Inherited Members

- ModCollider.Position
- ModCollider.Layer
- ModCollider.LayerSpecified
- ModCollider.Trigger
- ModCollider.IgnoreForGhost
- Element.Validate()
- Element.MissingElement(String, String)
- Element.MissingAttribute(String, String)
- Element.InvalidData(String, String)
- Element.Warn(String, String)
- Element.LineNumber
- Element.LinePosition
- Element.AttributesUsed
- Element.ElementsUsed
- Element.FileName

Namespace: **Modding.Serialization**

Assembly: Assembly-CSharp.dll

Syntax

```
public class SphereModCollider : ModCollider, IValidatable
```

Constructors

SphereModCollider()

Declaration

```
public SphereModCollider()
```

Properties

Radius

Radius of the collider.

Declaration

```
public float Radius { get; set; }
```

Property Value

TYPE	DESCRIPTION
Single	

Methods

CreateCollider(Transform)

Creates a game object with a collider attached, set up according to the ModCollider instance.

Declaration

```
public override Collider CreateCollider(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Collider	The new Collider component.

Overrides

[ModCollider.CreateCollider\(Transform\)](#)

CreateVisual(Transform)

Creates a debug visual object to represent this collider.

Declaration

```
public override Transform CreateVisual(Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	parent	Transform the new object should be a child of.

Returns

TYPE	DESCRIPTION
UnityEngine.Transform	The Transform of the new object.

Overrides

[ModCollider.CreateVisual\(Transform\)](#)

Validate(String)

Declaration

```
protected override bool Validate(string elemName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class TransformValues

Utility class to easily include Position, Rotation, and/or Scale for serialization. See the Common Elements serialization documentation for details on how to use it.

Inheritance

[Object](#)
[Element](#)

TransformValues

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class TransformValues : Element, IValidatable
```

Constructors

TransformValues()

Declaration

```
public TransformValues()
```

Fields

Position

Declaration

```
public Vector3 Position
```

Field Value

TYPE	DESCRIPTION
Vector3	

PositionSpecified

Declaration

```
public bool PositionSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Rotation

Declaration

```
public Vector3 Rotation
```

Field Value

TYPE	DESCRIPTION
Vector3	

RotationSpecified

Declaration

```
public bool RotationSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Scale

Declaration

```
public Vector3 Scale
```

Field Value

TYPE	DESCRIPTION
Vector3	

ScaleSpecified

Declaration

```
public bool ScaleSpecified
```

Field Value

TYPE	DESCRIPTION
Boolean	

Methods

Check(String)

Declaration

```
public bool Check(string elemName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

HasNoScale()

Declaration

```
public TransformValues HasNoScale()
```

Returns

TYPE	DESCRIPTION
TransformValues	

SetOnTransform(Transform)

Declaration

```
public void SetOnTransform(Transform t)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Transform	t	

SetPositionDefault(Vector3)

Declaration

```
public TransformValues SetPositionDefault(Vector3 position)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	

Returns

TYPE	DESCRIPTION
TransformValues	

SetRotationDefault(Vector3)

Declaration

```
public TransformValues SetRotationDefault(Vector3 rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	rotation	

Returns

TYPE	DESCRIPTION
TransformValues	

SetScaleDefault(Vector3)

Declaration

```
public TransformValues SetScaleDefault(Vector3 scale)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	scale	

Returns

TYPE	DESCRIPTION
TransformValues	

ToFauxTransform()

Declaration

```
public FauxTransform ToFauxTransform()
```

Returns

TYPE	DESCRIPTION
FauxTransform	

Validate(String)

Declaration

```
protected override bool Validate(string elemName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elemName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class VanillaBlockType

Inheritance

[Object](#)
[Element](#)

VanillaBlockType

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class VanillaBlockType : Element, IValidatable
```

Constructors

VanillaBlockType()

Declaration

```
public VanillaBlockType()
```

Fields

Text

Declaration

```
public string Text
```

Field Value

TYPE	DESCRIPTION
String	

Methods

Get()

Declaration

```
public BlockType Get()
```

Returns

TYPE	DESCRIPTION
BlockType	

Validate(String)

Declaration

```
protected override bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Class VanillaEntityType

Inheritance

[Object](#)
[Element](#)

VanillaEntityType

Implements

[IValidatable](#)

Inherited Members

- [Element.Validate\(\)](#)
- [Element.MissingElement\(String, String\)](#)
- [Element.MissingAttribute\(String, String\)](#)
- [Element.InvalidData\(String, String\)](#)
- [Element.Warn\(String, String\)](#)
- [Element.LineNumber](#)
- [Element.LinePosition](#)
- [Element.AttributesUsed](#)
- [Element.ElementsUsed](#)
- [Element.FileName](#)

Namespace: [Modding.Serialization](#)
Assembly: Assembly-CSharp.dll

Syntax

```
public class VanillaEntityType : Element, IValidatable
```

Constructors

VanillaEntityType()

Declaration

```
public VanillaEntityType()
```

Fields

Text

Declaration

```
public string Text
```

Field Value

TYPE	DESCRIPTION
String	

Methods

Get()

Declaration

```
public int Get()
```

Returns

TYPE	DESCRIPTION
Int32	

Validate(String)

Declaration

```
protected override bool Validate(string elementName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	elementName	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[Element.Validate\(String\)](#)

Implements

[IValidatable](#)

Struct Vector3

Unity's Vector3 does not (de)serialize correctly, so this wrapper is used to do it instead.

Namespace: [Modding.Serialization](#)

Assembly: Assembly-CSharp.dll

Syntax

```
public struct Vector3
```

Constructors

Vector3(Single, Single, Single)

Declaration

```
public Vector3(float pX, float pY, float pZ)
```

Parameters

TYPE	NAME	DESCRIPTION
Single	pX	
Single	pY	
Single	pZ	

Vector3(Vector3)

Declaration

```
public Vector3(Vector3 o)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	o	

Fields

one

Declaration

```
public static readonly Vector3 one
```

Field Value

TYPE	DESCRIPTION
Vector3	

x

Declaration

```
public float x
```

Field Value

TYPE	DESCRIPTION
Single	

y

Declaration

public float y

Field Value

TYPE	DESCRIPTION
Single	

z

Declaration

public float z

Field Value

TYPE	DESCRIPTION
Single	

zero

Declaration

public static readonly Vector3 zero

Field Value

TYPE	DESCRIPTION
Vector3	

Methods

ToString()

Declaration

public override string ToString()

Returns

TYPE	DESCRIPTION
String	

Overrides

ValueType.ToString()

Operators

Implicit(Vector3 to Vector3)

Declaration

```
public static implicit operator Vector3(Vector3 sV)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	sV	

Returns

TYPE	DESCRIPTION
UnityEngine.Vector3	

Implicit(Vector3 to Vector3)

Declaration

```
public static implicit operator Vector3(Vector3 v)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Vector3	v	

Returns

TYPE	DESCRIPTION
Vector3	