Infohazard.Core

1.4.2

Generated by Doxygen 1.9.5

1 Infohazard.Core Documentation

1.1 Table of Contents

- Infohazard.Core Documentation
  - Table of Contents
  - Introduction
  - Documentation and Support
  - License

1.2 Introduction

Infohazard.Core is a collection of systems and utilities that I've found super helpful in making many different kinds of games, so I hope you find it helpful too! This document will cover setup and basic usage of the code in Infohazard.Core. You can find full API documentation here and in the code.

1.3 Documentation and Support

`API Docs`

`Tutorial Playlist`

`Discord`

1.4 License

If Infohazard.Core is acquired from the Unity Asset Store, you must follow the Unity Asset Store license. The open-source repository uses the `MIT license`. You are welcome to have your own packages or assets depend on this package.

1.5 Installation

1.5.1 Method 1 - Package Manager

Using the Package Manager is the easiest way to install the package to your project. Simply install the project as a git URL. Note that if you go this route, you will not be able to make any edits to the package.

1. In Unity, open the Package Manager (Window > Package Manager).
2. Click the '+' button in the top right of the window.
3. Click "Add package from git URL...".
4. Paste in `https://github.com/infohazardgames/Infohazard.Core.git`.
5. Click Add.

1.5.2 Method 2 - Git Submodule

Using a git submodule is an option if you are using git for your project source control. This method will enable you to make changes to the package, but those changes will need to be tracked in a separate git repository.

1. Close the Unity Editor.
2. Using your preferred git client or the command line, add `https://github.com/infohazardgames/↩ Infohazard.Core.git` as a submodule in your project's Packages folder.
3. Re-open the Unity Editor.

If you wish to make changes when you use this method, you'll need to fork the package repo. Once you've made your changes, you can submit a pull request to get those changes merged back to this repository if you wish.

1. Fork this repository. Open your newly created fork, and copy the git URL.
2. In your project's Packages folder, open the package repository.
3. Change the `origin` remote to the copied URL.
4. Make your changes, commit, and push.
5. (Optional) Open your fork again, and create a pull request.

1.5.3 Method 3 - Add To Assets

If you wish to make changes to the library without dealing with a git submodule (or you aren't using git), you can simply copy the files into your project's Assets folder.

1. In the main page for this repo, click on Code > Download Zip.
2. Extract the zip on your computer.
3. Make an Infohazard.Core folder under your project's Assets folder.
4. Copy the `Editor` and `Runtime` folders from the extracted zip to the newly created folder.

1.5.4 Method 4 - Asset Store

If you'd rather use the asset store than the package manager, you can get the project `here`. Simply add it to the project as you would any other asset.

1.6 Setup

1.6.1 General Setup

The only setup required beyond installation is to add references to the Infohazard.Core assembly if you are using an assembly definition. If you are using the default assemblies (such as Assembly-CSharp), nothing is needed here. You may also wish to have your editor assembly (if you have one) reference Infohazard.Core.Editor. In order to reference the generated GameTag file, you must also add a reference to Infohazard.Core.Data.

1.6.2 SRP Setup

If you are using a scriptable render pipeline (URP, HDRP, etc) and wish to run the demos, you will need to upgrade the materials using your render pipeline's material upgrade system. The materials you'll need to upgrade are in `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Materials` and `Assets/↩ Plugins/Infohazard/Demos/Shared Demo Assets/Materials`.

1.7 Demos

The following demo scenes are provided to cover the various features of Infohazard.Core.

1.7.1 Attributes

This demo, located at `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Scenes/Demo↩ _Attributes.unity`, demonstrates how you can use the various attributes provided in Infohazard.Core to improve your parameter editing experience.

1.7.2 Pooling and Timing

This demo, located at `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Scenes/Demo↩ _PoolingAndTiming.unity`, shows how to use the object pooling system to improve performance when spawning many objects, and how to use a timer to control the timing of this spawning (as well as using the pause system).

1.7.3 Quartic Solver

This demo, located at `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Scenes/Demo_↩ QuarticSolver.unity`, demonstrates how to use the math utility to solve a quartic equation. This functionality is useful for scenarios such as aiming at an accelerating target.

1.7.4 Tags and Trigger Volume

This demo, located at `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Scenes/Demo↩ _TagsAndTriggerVolume.unity`, deomonstrates how to use the TriggerVolume script to set up triggers in the editor, as well as how to use a tag mask to filter which objects are considered by the trigger.

1.7.5 Unique name

This demo, located at `Assets/Plugins/Infohazard/Demos/Infohazard.Core/Scenes/Demo↩ _UniqueName.unity`, demonstrates how to use the UniqueName system to reference and find specific objects without having a direct reference (such as if they are in another scene).

1.8 Features Guide

1.8.1 Attributes

The package provides several PropertyAttributes that you can use in your scripts to customize how serialized fields are drawn in the inspector. To use any of these attributes, simply add `[AttributeName]` in front of a serialized field in a script. You can also check out the drawers for these attributes in the Editor/Attributes directory.

1.8.1.1 AssetDropdown The `[AssetDropdown]` attribute is used to show a dropdown menu on a field whose type is a UnityEngine.Object reference. It will find all assets in your project that match this type and display them as options in the dropdown. The standard drag/drop interface still works as well. This functionality is also available the `[Expandable]` attribute.

1.8.1.2 ConditionalDraw The `[ConditionalDraw]` attribute is used to conditionally hide a serialized field in the inspector depending on some other condition. The supplied condition should be the name of another serialized field in the same script. You can also optionally pass in a value to compare that condition field with, and whether they must be equal or unequal to display the field.

1.8.1.3 DrawSingleChildProperty The `[DrawSingleChildProperty]` attribute is used to render a hierarchy of fields as just a single value. Say you have a struct called Data with a string field called _name. Adding `[DrawSingleChildProperty("_name")]` to a field of type Data would cause just the _name field to be drawn in the inspector.

1.8.1.4 EditNameOnlyAttribute The `[EditNameOnly]` attribute displays a Unity Object reference field as a text entry, which is used to control the name of the referenced object. If the reference is set to null, the standard drag-drop box is used.

1.8.1.5 ExpandableAttribute The `[Expandable]` attribute is used to optionally draw the child properties of a Unity Object reference field, such that the referenced object can be edited without changing the inspector context. If the type of the field is not a GameObject or Component and is not abstract, new instances can also be created from the inspector. If the type specified is abstract, has child classes, or a required interface is specified, then the type of object to be created can be chosen from a dropdown. Additionally, a dropdown is provided to select assets from the project, similar to `[AssetDropdown]`.

1.8.1.6 MustImplementAttribute

The `[MustImplement]` attribute is used on a Unity Object reference field to restrict values to implement one or more interfaces. While Unity does not support serializing references to interfaces directly, you can use this attribute on a field of type Object to allow both ScriptableObjects and Mono↩ Behaviours that implement your interface. This functionality is also provided in the `[Expandable]` attribute.

1.8.1.7 TypeSelectAttribute

The `[TypeSelect]` attribute is used on a string field to show a dropdown where any valid C# type can be selected. The selected type is saved in the string as its full class name. This attribute is useful with the TypeUtility class to find the selected type.

1.8.1.8 HelpBox

The `[HelpBox]` attribute is used to add additional information on a serialized field for the user. It will add a clickable ? button that toggles a help box containing the given text. Note that unlike other attributes, `[HelpBox]` requires the use of a custom editor. This is to enable support for additional custom drawers (since Unity only allows one custom drawer on each serialized field). To support help boxes, draw the SerializedProperty in your custom editor using `CoreDrawers.DrawPropertyWithHelpBoxSupport(property);`. You can safely use this on all properties regardless of whether they actually contain a help box.

1.8.2 Data Structures

1.8.2.1 ListQueue

The `ListQueue<T>` class is an implementation of a queue data structure similar to C↩ ::'s Queue class. The main difference is that a ListQueue implements the IList interface and allows you to access any element of the queue whenever you want, while still maintaining O(1) performance for normal Enqueue and Dequeue operations (as long as there is capacity available).

1.8.3 Pooling

The library provides a simple object pooling system under the Pooling directory. Object pooling means that instead of instantiating and destroying GameObjects as needed, we deactivate them and reactivate them to avoid constantly allocating and deallocating memory.

1.8.3.1 Spawnable

To start working with the pooling system, simply add the Spawnable script to your prefabs and then instantiate/destroy them using Spawnable.Spawn and Spawnable.Despawn. Note that Awake/OnDestroy will only be called when the objects are actually created and destroyed; if you want an event when the object is spawned, use the OnSpawned and OnDespawned messages. Note that if a prefab does not contain a Spawnable script, using Spawnable.Spawn and Spawnable.Despawn is the same as Instantiate and Destroy. This allows you to spawn and despawn objects without worrying about whether they're pooled or not. All of my libraries use this system, so they are compatible with pooling.

1.8.3.2 PoolManager

The pooling system needs a PoolManager to work. You can either place one Pool↩ Manager per scene, have a global instance that is never destroyed, or simply let the system create the manager itself. To create it manually, just add this script to an empty GameObject and you're good to go. You can use the ClearInactiveObjects() method on PoolManager.Instance to destroy any inactive pooled objects, such as when you change scenes. PooledTrail, PooledParticleEffect Attach these scripts to TrailRenderer and ParticleSystem GameObjects to make them play nicely with the pooling system.

1.8.4 Timing

The Timing directory contains some useful utilities to deal with in-game time.

1.8.4.1 PassiveTimer

PassiveTimer is a serializable data type used to create timers in your scripts. You can use it to create ability cooldowns and durations, weapon reloads, and other common game functionality. Simply call Initialize() when your script initializes, then use the timer's various methods. See the API docs for more info.

1.8.4.2 Pause

The Pause system is built to do exactly that - pause the game. Just set Pause.IsPaused to true and time will freeze, then set it to false to resume time at its previous speed. The game will automatically unpause if you change scenes. You should avoid running game logic if the game is paused.

1.8.4.3 TimeToLive

This script destroys or despawns an object some number of seconds after it is spawned. It is compatible with the pooling system.

1.8.5 Tags

This system is meant to make working with GameObject tags in your scripts much easier and less error-prone.

1.8.5.1 Tag

This class provides string constants for all default tags. So instead of writing target.CompareTag("↩
Player") you could write target.CompareTag(Tag.Player).

1.8.5.2 TagMask

Using a serialized field of type TagMask allows users to pick tags in the editor from a dropdown instead of typing them, just like the Unity-provided LayerMask.. It also allows selecting multiple tags without using an array. Extension methods are provided for common tag operations so that a TagMask can be used in place of a string tag. For example, you can say target.CompareTag(tagMask), which will return true if target's tag is equal to any of the tags in tagMask.

1.8.5.3 GameTag

This file is generated based on your custom tags, and lives in your project rather than in the package. You should be prompted to generate it if the system detects you have custom tags, or you can use the Tools > Infohazard > Generate > Update GameTag.cs command. Once this file is generated, your game tags will automatically be available for selection in a TagMask, and you can refer to them as constants in code through the generated GameTag class (you will need to reference the Infohazard.Core.Data assembly if you are using assembly definitions).

1.8.6 Unique Names

The UniqueName system enables you to assign names to objects that can be referenced across scenes and assets. You can then easily find the active object using that name (if one exists). This system uses ScriptableObjects to store the names so that you can easily see what names are available to reference, and can avoid having to type out the names and potentially make mistakes. Furthermore, these unique names can be changed without breaking references, since they are stored as object references.

1.8.6.1 UniqueNameList

A UniqueNameList asset is how you start creating a list of unique names. You can have multiple UniqueNameLists in your project, or you can use just one. This is purely for organizational purposes. You can create a UniqueNameList using Assets > Create > Infohazard > Unique Name List.

1.8.6.2 UniqueNameListEntry

A UniqueNameListEntry is the actual unique name, which is organized under a UniqueNameList and used both by objects with unique names and objects using the system to find named objects. UniqueNameListEntries should be created through the UniqueNameList inspector.

1.8.6.3 UniqueNamedObject Attach this script to a GameObject to assign a unique name to it, and make it findable in the system. You can find one of these objects using the static method UniqueNamedObject.TryGet↩
Object, passing in either a string or a UniqueNameListEntry.

1.8.7 Utility

The Utility section contains a bunch of static methods to help with all kinds of common operations. See the API docs for each file for more info.

1.8.7.1 DebugUtility Contains methods to draw a cube using Debug.DrawLine, and to pause the editor after a certain number of frames.

1.8.7.2 EnumerableUtility Contains methods that combine common LINQ calls such as Select and Where into a single enumeration for better code optimization.

1.8.7.3 GameObjectUtility Contains various methods for working with GameObjects and Transforms, such as destroying all the children of an object, setting an object's layer recursively, and getting a path containing an object's ancestor names.

1.8.7.4 MathUtility Contains many useful math operations, such as constructing a Quaternion from any two axes, getting a vector with one component changed, solving polynomials up to degree 4 (quartic), and getting the point where two lines are closest to each other.

1.8.7.5 RandomUtility Contains extension methods to System.Random such as generating 64-bit numbers. StringUtility String processing methods such as splitting a CamelCase string to have spaces between words.

1.8.7.6 TypeUtility Provides methods to get a list of all loaded types using reflection, and to find a type based on its name.

1.8.8 Miscellaneous

The remaining functionality provided by Infohazard.Core doesn't fall nicely into one of the previous categories, but was still useful enough to include.

1.8.8.1 ProgressBar Used to create health bars and other types of progress bars without using a Slider. It supports images that fill the bar using either the “filled” image type or by manipulating the RectTransform anchors.

1.8.8.2 SceneControl Provides a static method to quit the game that works in a standalone build as well as in the editor. Also provides some methods to navigate to scenes. This is useful if you're building a super quick main menu (such as in the last half hour of a game jam) and need to hook up your buttons as fast as possible.

1.8.8.3 SceneRef A serializable type that allows you to have assignable scene references in your scripts without making the user type the scene name. Instead, they can simply drag in a scene asset. At runtime, you still access the scene by its name. Using a SceneRef also enables the reference to be maintained if a scene is renamed.

1.8.8.4 Singleton

You can inherit from this script in managers or other scripts that need to exist in the scene exactly once. A static Instance accessor is automatically provided, which will do a lazy search for the correct instance the first time it is used, or if the previous instance was destroyed. After that it will just return a cached instance.

1.8.8.5 SingletonAsset

Similar to Singleton, but for ScriptableObjects. You specify a path in your subclass where the instance should live (this must be under a Resources folder) and the editor will automatically handle loading and even creating this asset for you when needed.

1.8.8.6 TriggerVolume

A script that makes it easy to add events to a trigger collider. Provides both UnityEvents (assignable in the inspector) and normal C# events for when an object enters or leaves the trigger, and when all objects have left the trigger.

1.8.8.7 SimpleStateMachine

A basic code-only state machine implementation. Handles states and transitions with callbacks and transition conditions, as well as manual direct transitions. Each state has a key which can be any type you desire (typically an enum you create).

1.9 Integrations

Infohazard.Core directly supports integration with the following assets and packages:

1.9.1 Addressables

Infohazard.Core has an extension package to support spawning Addressable prefabs through the spawn/pooling system. This extension can be very helpful when working with Addressables, even if you don't want to actually pool them.

Regardless of how you install the Addressables integration, you will need to install `UniTask`, either via the Package Manager or importing the .unitypackage.

If you downloaded Infohazard.Core from the Unity Asset Store, you can find the Addressables integration package at `Assets/Plugins/Infohazard/Infohazard.Core/Integrations/Infohazard.Core.↩ Addressables.unitypackage`. Double click that package in Unity to extract the files into your project.

If you are using Infohazard.Core as a package or submodule, you can install the Addressables integration package from `Github`.

2 Changelog

All notable changes to this project will be documented in this file.

The format is based on `Keep a Changelog`, and this project adheres to `Semantic Versioning`.

2.1 [1.4.1] - 2025-3-5

2.1.1 Added

- Added methods to check if a GameObject's relative path is unique.
- Added methods to recursively copy values in SerializedProperties.
- Added option for an Expandable attribute to show only the main line and not the expansion.

2.1.2 Changed

- GameObject path methods now URL-encode the path to avoid issues with the slash character.

2.1.3 Fixed

- Fixed ExpandableAttribute property not showing as overridden in a prefab instance.
- Fixed FindFieldInfo not actually working on nested fields.

2.2 [1.4.0] - 2025-2-11

2.2.1 Added

- Added attribute to require a referenced object to implement an interface.
- Added an interface constraint and the asset dropdown to the `Expandable` attribute.
- Added ability to add custom naming logic for objects created through the Expandable attribute.
- Added the SerializableRange struct (used by HyperNav).
- Added a method to record profiling data via the profiler window through code (as opposed to only recording to a file).
- Added an extension IReadOnlyList.IndexOf.
- Added optional folder parameter for GetAssetsOfType.
- Added DebugUtility.DrawDebugSphere.
- Added HelpBox attribute and drawing code.
- Added a simple general-use state machine.
- Added a ColliderCast utility for easily configuring sphere, capsule, and box casts.

2.2.2 Changed

- Renamed SpawnRef.Valid to SpawnRef.IsValid.
- Minimum Unity version changed to 2022.3.
- Use correct linear velocity property for Rigidbody depending on Unity version, so you won't see deprecated API warnings.

2.2.3 Fixed

- TriggerVolume now registers exit events if all of an occupant's colliders are disabled.
- TriggerVolume will now fire exit events if it becomes disabled.
- Fixed a bug with expandable foldouts overlapping the label.
- Fixed Spawnable trying to clear velocity on a kinematic rigidbody.

2.3 [1.3.0] - 2023-6-21

2.3.1 Added

- Addressables integration package.
- Made pooling system more modular in order to support Addressables package.
- Added PassiveTimer.TimeSinceIntervalEnded.
- Added ability for TriggerVolume to enable/disable a list of colliders when it its enabled/disabled.
- Expandable attribute has a new ShowChildTypes option, which allows selecting a subclass of the given type. This means it can now be used on fields with an abstract type.
- Expandable attribute now supports non-ScriptableObject assets such as materials.
- Added events when a UniqueNamedObject is registered or deregistered.
- Added ability to include the GameObject's scene in SpawnParams.At.

2.3.2 Changed

- ListQueue now implements IReadOnlyList.
- Singleton.Instance can now return a destroyed instance if there is not a new instance to replace it.
- Exposed TriggerVolume occupants list.

2.3.3 Fixed

- Fixed Expandable attribute not working when a type had a namespace.
- Fixed ListQueue enumerable constructor not working properly.
- Fixed PassiveTimer.IsIntervalEnded not respecting time mode.
- Fixed SpawnParams.At including global position/rotation when the parented is set to true.

2.4 [1.2.0] - 2022-11-22

2.4.1 Added

- DebugUtility.DrawDebugCircle.
- MathUtility.GetPerpendicularVector.

2.4.2 Fixed

- Fixed TagMaskEditor messing up values when editing multiple objects.

2.5 [1.1.0] - 2022-11-13

2.5.1 Changed

- Made GameTag.cs generation far move robust.
  - Handle when there are more than 64 tags.
  - Handle all invalid characters in variable names.
  - Handle variable names that are keywords.
  - Handle tag strings containing quotation marks.
- Removed the warning message when generating GameTag.cs as it's no longer relevant.

2.6 [1.0.0] - 2022-10-08

2.6.1 Added

- Initial release, all files and documentation added.

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 Namespace Documentation

5.1 Infohazard Namespace Reference

5.2 Infohazard.Core Namespace Reference

Classes

- class AssetDropdownAttribute

  Attribute that draws an object reference as a dropdown that searches through the project.
- struct ComponentID

  A serializable type used to uniquely identify a component relative to a root GameObject.
- class ConditionalDrawAttribute

  Attribute draws a property when a given condition is true.
- class DebugUtility

  Contains various static methods relating to debugging and diagnostics.
- class DefaultPoolHandler

  An IPoolHandler that is passed a prefab directly.
- class DrawSingleChildPropertyAttribute

  Attribute draws only a single child property of a property.
- class EditNameOnlyAttribute

  Attribute that draws only the name of an Object reference field.
- class EnumerableUtility

  Contains various static methods for working with sequences, extending the functionality of LINQ.
- class ExpandableAttribute

  Attribute that enables editing properties of a referenced object.
- class GameObjectUtility

  Contains utility methods for dealing with GameObjects and Transforms.
- class HelpBoxAttribute

  Used to add a detailed help box that can be toggled in the inspector. This can be used to provide more information than a simple tooltip.
- interface IPersistedInstance

  This is a hack so that PoolManager can send messages to PersistedGameObjects.
- class ListQueue

  A FIFO data structure similar to a Queue, except that it implements all List operations.
- class MathUtility

  Contains utility methods for working with mathematical types and solving math equations.
- class MustImplementAttribute

  Applied to object reference fields to ensure that an assigned Object must implement one or more interfaces.
- struct PassiveTimer

  A lightweight timer that does not need to be updated each frame.
- class Pause

  Manages pausing and unpausing of the game.
- class Pool

  Provides a simple pool with an interface similar to the official Unity pool added in 2021.
- class PooledParticleEffect

A component that can be attached to ParticleSystem GameObjects to make them work correctly with pooling.
- class PooledTrail

  A component that can be attached to TrailRenderer GameObjects to make them work correctly with pooling.
- class PoolManager

  The singleton manager class that handles object pooling.
- class ProgressBar

  Used to create health bars and other types of progress bars without using a Slider.
- class RandomUtility

  Contains extensions to builtin randomization functionality.
- class SceneControl

  Provides some methods to navigate to scenes.
- class Singleton

  Base class that makes it easier to write scripts that always have exactly one instance.
- class SingletonAsset

  Base class that makes it easier to write ScriptableObjects that always have exactly one instance in your project.
- class SingletonAssetBase

  Base class of SingletonAsset$<T>$. For internal use only.
- class Spawnable

  Attach this component to a prefab to enable it to use the pooling system.
- struct SpawnParams

  Used to pass spawn parameters to various object creation/initialization methods.
- class SpawnRef

  SpawnRef for spawning a GameObject directly.
- class SpawnRefBase

  Only used internally.
- class StringUtility

  Contains string processing utilities.
- class Tag

  Provides string constants for builtin Unity tags.
- struct TagMask

  Used to select tags in the inspector, including the ability to select multiple tags.
- class TagMaskUtility

  Static operations on Tag enum values.
- class TimeToLive

  Despawns a GameObject after a set amount of time.
- class TriggerVolume

  A script that makes it easy to add events to a trigger collider.
- class TypeSelectAttribute

  Attribute that draws string fields as a dropdown where a Type can be selected.
- class TypeUtility

  Contains utilities for working with C# reflection types and getting a type by its name.
- class UniqueNamedObject

  This script is used to assign a unique name to an object, which can then be used to find that object.
- class UniqueNameList

  A list used to organize unique names used by objects.
- class UniqueNameListEntry

  A unique name asset, usable by a UniqueNamedObject.

5.3 Infohazard.Core.Editor Namespace Reference

Classes

- class CoreDrawers

  Contains extensions to EditorGUI for drawing custom properties.
- class CoreEditorUtility

  Contains several editor-only utilities.
- interface IObjectReferenceFieldAssignmentValidator

  Used to validate and modify values that a user is attempting to assign to an object reference field.
- class ObjectReferenceFieldAssignmentValidator

  IObjectReferenceFieldAssignmentValidator that uses Unity validation internally, and allows adding additional validation with a simplified interface.
- class ObjectValidators

  Provides methods for validating objects.
- class TagsGenerator

  Class used to generate the GameTag.cs file to use your custom tags in code.

5.4 Infohazard.Core.StateMachine Namespace Reference

6 Class Documentation

6.1 Infohazard.Core.AssetDropdownAttribute Class Reference

Attribute that draws an object reference as a dropdown that searches through the project.

6.1.1 Detailed Description

Attribute that draws an object reference as a dropdown that searches through the project.

The documentation for this class was generated from the following file:

- Runtime/Attributes/AssetDropdownAttribute.cs

6.2 Infohazard.Core.ComponentID Struct Reference

A serializable type used to uniquely identify a component relative to a root GameObject.

Public Member Functions

- ComponentID (Transform root, Component target)

  Construct a ComponentID with the given root and target by calculating the path to the target.
- override string ToString ()
- T Get$<$ T $>$ (Transform root)

  Get the referenced component.
- bool Equals (ComponentID other)
- override bool Equals (object obj)
- override int GetHashCode ()

Properties

- string Path `[get]`

  Path from this GameObject to the GameObject that holds the referenced component.
- string Type `[get]`

  Type name of the referenced component.

6.2.1 Detailed Description

A serializable type used to uniquely identify a component relative to a root GameObject.

The root GameObject that holds the ComponentID may not be an actual transform.root, but rather just an ancestor of the referenced component. This type is mostly used in order to persist the states of individual components, keyed by their ID.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 ComponentID()

```
Infohazard.Core.ComponentID.ComponentID (
        Transform root,
        Component target )
```

Construct a ComponentID with the given root and target by calculating the path to the target.

Parameters

root	The GameObject from which the reference originates.
target	The component that is being referenced.

6.2.3 Member Function Documentation

6.2.3.1 Equals() [1/2]

```
bool Infohazard.Core.ComponentID.Equals (
        ComponentID other )
```

6.2.3.2 Equals() [2/2]

```
override bool Infohazard.Core.ComponentID.Equals (
        object obj )
```

6.2.3.3 Get< T >()

```
T Infohazard.Core.ComponentID.Get< T > (
        Transform root )
```

Get the referenced component.

Parameters

root	The GameObject from which the reference originates.

Template Parameters

T	Type to cast the component to.

Returns

The referenced component, or null if not found.

Type Constraints

T* : *Component

6.2.3.4 GetHashCode() `override int Infohazard.Core.ComponentID.GetHashCode ( )`

6.2.3.5 ToString() `override string Infohazard.Core.ComponentID.ToString ( )`

6.2.4 Property Documentation

6.2.4.1 Path `string Infohazard.Core.ComponentID.Path [get]`

Path from this GameObject to the GameObject that holds the referenced component.

6.2.4.2 Type `string Infohazard.Core.ComponentID.Type [get]`

Type name of the referenced component.

The documentation for this struct was generated from the following file:

- Runtime/Misc/ComponentID.cs

6.3 Infohazard.Core.ConditionalDrawAttribute Class Reference

Attribute draws a property when a given condition is true.

Public Member Functions

- ConditionalDrawAttribute (string boolCondition)

  ConditionalAttribute that requires a serialized boolean field to be true.
- ConditionalDrawAttribute (string condition, object value, bool isEqual=true)

  ConditionalAttribute that requires a serialized field named condition to be equal or not equal to value.

Properties

- string Condition `[get]`

  The serialized field to check.
- object Value `[get]`

  The value to compare the Condition field to.
- bool IsEqual `[get]`

  Whether the value of the condition field should be equal to the given value in order to draw.

6.3.1 Detailed Description

Attribute draws a property when a given condition is true.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ConditionalDrawAttribute() [1/2]

```
Infohazard.Core.ConditionalDrawAttribute.Conditional←
DrawAttribute (
            string boolCondition )
```

ConditionalAttribute that requires a serialized boolean field to be true.

Parameters

boolCondition	The name of a boolean field.

6.3.2.2 ConditionalDrawAttribute() [2/2]

```
Infohazard.Core.ConditionalDrawAttribute.Conditional←
DrawAttribute (
            string condition,
            object value,
            bool isEqual = true )
```

ConditionalAttribute that requires a serialized field named condition to be equal or not equal to value.

Parameters

condition	The name of a field.
value	The value to compare to.
isEqual	Whether to check for equality or inequality.

6.3.3 Property Documentation

6.3.3.1 Condition `string Infohazard.Core.ConditionalDrawAttribute.Condition [get]`

The serialized field to check.

6.3.3.2 IsEqual `bool Infohazard.Core.ConditionalDrawAttribute.IsEqual [get]`

Whether the value of the condition field should be equal to the given value in order to draw.

6.3.3.3 Value `object Infohazard.Core.ConditionalDrawAttribute.Value [get]`

The value to compare the Condition field to.

The documentation for this class was generated from the following file:

- Runtime/Attributes/ConditionalDrawAttribute.cs

6.4 Infohazard.Core.Editor.CoreDrawers Class Reference

Contains extensions to EditorGUI for drawing custom properties.

Static Public Member Functions

- static void ValidatedObjectField (Rect position, SerializedProperty property, Type objType, GUIContent label, GUIStyle style, IObjectReferenceFieldAssignmentValidator validator)

  Draw an object field with a custom validator applied.
- static void ObjectFieldWithInterfaces (Rect position, SerializedProperty property, Type objType, GUIContent label, GUIStyle style, IReadOnlyList< Type > interfaces)

  Draw an object field, ensuring that any assigned value implements a list of interfaces.
- static void CreateNewInstanceDropDown (Type requiredType, Rect buttonRect, IReadOnlyList< Type > interfaces, Action< Type > createAction)

  Draw a button which, when pressed, will show a dropdown of all types that can be assigned to the given property. When the user clicks one of the options, their supplied callback will be triggered.
- static void AssetDropdown (SerializedProperty property, Type type, Rect buttonRect, GUIContent content, IReadOnlyList< Type > interfaces=null)

  Draw a dropdown for selecting an asset to assign to a property.

6.4.1 Detailed Description

Contains extensions to EditorGUI for drawing custom properties.

6.4.2 Member Function Documentation

6.4.2.1 AssetDropdown()

```
static void Infohazard.Core.Editor.CoreDrawers.AssetDropdown (
            SerializedProperty property,
            Type type,
            Rect buttonRect,
            GUIContent content,
            IReadOnlyList< Type > interfaces = null )  [static]
```

Draw a dropdown for selecting an asset to assign to a property.

Parameters

property	The property that will be assigned.
type	Type of required object.
buttonRect	Rect in which to show the button.
content	Content to show in the button.
interfaces	Interfaces that the object must implement.

6.4.2.2 CreateNewInstanceDropDown()

```
static void Infohazard.Core.Editor.CoreDrawers.Create←
NewInstanceDropDown (
            Type requiredType,
            Rect buttonRect,
            IReadOnlyList< Type > interfaces,
            Action< Type > createAction )  [static]
```

Draw a button which, when pressed, will show a dropdown of all types that can be assigned to the given property. When the user clicks one of the options, their supplied callback will be triggered.

Parameters

requiredType	Type of required object.
buttonRect	Rect in which to show the button.
interfaces	Interfaces that the object must implement.
createAction	Action to invoke when the user chooses a type.

6.4.2.3 ObjectFieldWithInterfaces()

```
static void Infohazard.Core.Editor.CoreDrawers.ObjectField←
WithInterfaces (
            Rect position,
            SerializedProperty property,
            Type objType,
            GUIContent label,
            GUIStyle style,
            IReadOnlyList< Type > interfaces )  [static]
```

Draw an object field, ensuring that any assigned value implements a list of interfaces.

Parameters

position	Position and size of the field.
property	Property being edited.
objType	Type of the field.
label	Label to display.
style	Style to use.
interfaces	Interfaces that must be implemented.

6.4.2.4 ValidatedObjectField()

```
static void Infohazard.Core.Editor.CoreDrawers.ValidatedObject↩
Field (
            Rect position,
            SerializedProperty property,
            Type objType,
            GUIContent label,
            GUIStyle style,
            IObjectReferenceFieldAssignmentValidator validator )  [static]
```

Draw an object field with a custom validator applied.

Parameters

position	Position and size of the field.
property	Property being edited.
objType	Type of the field.
label	Label to display.
style	Style to use.
validator	Validator to apply.

The documentation for this class was generated from the following file:

- Editor/Utility/CoreDrawers.cs

6.5 Infohazard.Core.Editor.CoreEditorUtility Class Reference

Contains several editor-only utilities.

Static Public Member Functions

- static object GetValue (this SerializedProperty property)

  Get the value of a SerializedProperty of any type. Does not work for serializable objects or gradients. Enum values are returned as ints.
- static object FindValue (this SerializedProperty prop)

  Find the value of a given property using reflection and reading the field directly.
- static T FindValue< T > (this SerializedProperty prop)

  Find the value of a given property using Reflection and reading the field directly.

- static FieldInfo FindFieldInfo (this SerializedProperty prop)

  Find the FieldInfo of a given property using Reflection and reading the field directly. If the property is an array element, return null.
- static List< string > GetDefinesList (BuildTargetGroup group)

  Get the Unity PlayerSettings list of #define symbols for the given build target.
- static void SetSymbolDefined (string symbol, bool value, BuildTargetGroup group)

  Sets whether the given symbol is defined in the PlayerSettings for the given build target.
- static string GetResourcePath (Object obj)

  Get the path an object lives in relative to a resource folder.
- static string GetResourcePath (string path)

  Get the given path relative to a resource folder.
- static string GetPathRelativeToAssetsFolder (string path)

  Convert the given path to be relative to the Assets folder.
- static void DoLazyDropdown< T > (Rect position, GUIContent content, Func< T[ ]> optionsFunc, Func< T, string > stringifier, Action< T > setter)

  Create a dropdown button in the IMGUI, whose options will not be calculated until it is clicked.
- static IEnumerable< T > GetAssetsOfType< T > (string folder=null)

  Find all the assets of the given type in the project.
- static IEnumerable< Object > GetAssetsOfType (string type)

  Find all the assets of the given type in the project.
- static string GetTypeName (this SerializedProperty property)

  Get the type full name (including assembly) of the type of the underlying field for the given property.
- static void CopyFrom (this SerializedProperty destination, SerializedObject source)

  Perform a recursive copy of the values from a SerializedObject to a struct-type SerializedProperty.
- static void CopyFrom (this SerializedProperty destination, SerializedProperty source)

  Perform a recursive copy of the values from one SerializedProperty to another. For compound types, values in one of the properties but not the other are ignored.
- static Type GetFieldType (this PropertyDrawer drawer)

  Get the type of the field the property drawer is drawing. If the type is an array or list, returns the element type.
- static void EnsureDataFolderExists ()

  Ensure that the DataFolder directory exists in your project, and contains an assembly definition.
- static bool ExecuteProcess (string command, string args, bool showMessages)

  Launch an external process using the given command and arguments, and wait for it to complete.
- static GameObject InstantiatePrefabInScene (string path, string name)

  Instantiate the given prefab in the scene, as a child of the selected GameObject if there is one.
- static GameObject CreateGameObjectInScene (string name)

  Create an empty GameObject in the scene, as a child of the selected GameObject if there is one.
- static T CreateGameObjectInSceneWithComponent< T > (string name)

  Create a GameObject in the scene with a given component, as a child of the selected GameObject if there is one.
- static Object CreateAndSaveNewAsset (string name, Type type, string defaultSavePath, Object copy↩ Object=null, Object parentObject=null, Action< Object, string > saveAction=null)

  Prompts the user to browse for a path to save a new asset at. Upon confirmation, a new asset is created and saved.

Static Public Attributes

- const string DataFolder = "Assets/Infohazard.Core.Data/"

  Folder where all generated files used by the Infohazard libraries should live.

6.5.1 Detailed Description

Contains several editor-only utilities.

6.5.2 Member Function Documentation

6.5.2.1 CopyFrom() [1/2]

```
static void Infohazard.Core.Editor.CoreEditorUtility.CopyFrom (
            this SerializedProperty destination,
            SerializedObject source ) [static]
```

Perform a recursive copy of the values from a SerializedObject to a struct-type SerializedProperty.

Parameters

destination	Destination property to update. Must by a serializable class or struct.
source	Source object to read.

6.5.2.2 CopyFrom() [2/2]

```
static void Infohazard.Core.Editor.CoreEditorUtility.CopyFrom (
            this SerializedProperty destination,
            SerializedProperty source ) [static]
```

Perform a recursive copy of the values from one SerializedProperty to another. For compound types, values in one of the properties but not the other are ignored.

Parameters

destination	Destination property to update.
source	Source property to read.

6.5.2.3 CreateAndSaveNewAsset()

```
static Object Infohazard.Core.Editor.CoreEditorUtility.←
CreateAndSaveNewAsset (
            string name,
            Type type,
            string defaultSavePath,
            Object copyObject = null,
            Object parentObject = null,
            Action< Object, string > saveAction = null ) [static]
```

Prompts the user to browse for a path to save a new asset at. Upon confirmation, a new asset is created and saved.

By using a custom save action, you can, for example, add the created object to another asset rather than saving to the project directly. If a custom save action is used, the file browser will not be shown.

Parameters

name	The default filename of the created object.
type	Type of object to create.
defaultSavePath	Default path to save the asset.
parentObject	The object containing the created object (can be null).
saveAction	Action to take to save the asset. Can be null for regular asset save.
copyObject	Copy this object to create the new asset (can be null).

6.5.2.4 CreateGameObjectInScene()

```
static GameObject Infohazard.Core.Editor.CoreEditorUtility.↩
CreateGameObjectInScene (
              string name )  [static]
```

Create an empty GameObject in the scene, as a child of the selected GameObject if there is one.

Parameters

name	Name to assign to the object.

Returns

The created GameObject.

6.5.2.5 CreateGameObjectInSceneWithComponent< T >()

```
static T Infohazard.Core.Editor.Core↩
EditorUtility.CreateGameObjectInSceneWithComponent< T > (
              string name )  [static]
```

Create a GameObject in the scene with a given component, as a child of the selected GameObject if there is one.

Parameters

name	Name to assign to the object.

Template Parameters

T	The type of component to add.

Returns

The created component.

Type Constraints

T* : *Component

6.5.2.6 DoLazyDropdown< T >()

```
static void Infohazard.Core.Editor.CoreEditorUtility.DoLazy↩
Dropdown< T > (
              Rect position,
              GUIContent content,
              Func< T[]> optionsFunc,
              Func< T, string > stringifier,
              Action< T > setter )  [static]
```

Create a dropdown button in the IMGUI, whose options will not be calculated until it is clicked.

For a normal dropdown, you'd need to know all of the options before the button was clicked. With lazy dropdown, they are not evaluated until needed. They are also re-evaluated every time the dropdown is opened, preventing the need for cache invalidation.

Parameters

position	Position to draw the dropdown button.
content	Current value to show in the dropdown button (when not selected).
optionsFunc	Function that will calculate and return the options.
stringifier	Function that converts options to strings for display.
setter	Function that sets the value when an option is chosen.

Template Parameters

T	Type of the options before they are converted to strings.

6.5.2.7 EnsureDataFolderExists() `static void Infohazard.Core.Editor.CoreEditorUtility.Ensure↵ DataFolderExists ( ) [static]`

Ensure that the DataFolder directory exists in your project, and contains an assembly definition.

6.5.2.8 ExecuteProcess() `static bool Infohazard.Core.Editor.CoreEditorUtility.ExecuteProcess (`

```
string command,
string args,
bool showMessages ) [static]
```

Launch an external process using the given command and arguments, and wait for it to complete.

Parameters

command	The command (executable file) to run.
args	The argument string to pass to the command.
showMessages	Whether to display a dialog box if the command fails.

Returns

Whether the command succeeded.

6.5.2.9 FindFieldInfo() `static FieldInfo Infohazard.Core.Editor.CoreEditorUtility.FindFieldInfo (`

```
this SerializedProperty prop ) [static]
```

Find the FieldInfo of a given property using Reflection and reading the field directly. If the property is an array element, return null.

Parameters

prop	The property to read

Returns

The FieldInfo of the property.

6.5.2.10 FindValue() `static object Infohazard.Core.Editor.CoreEditorUtility.FindValue ( this SerializedProperty prop ) [static]`

Find the value of a given property using reflection and reading the field directly.

Parameters

prop	The property to read.

Returns

The value of the property.

6.5.2.11 FindValue< T >() `static T Infohazard.Core.Editor.CoreEditorUtility.FindValue< T > ( this SerializedProperty prop ) [static]`

Find the value of a given property using Reflection and reading the field directly.

Template Parameters

T	The type to cast the value to.

Parameters

prop	The property to read

Returns

The value of the property.

6.5.2.12 GetAssetsOfType() `static IEnumerable< Object > Infohazard.Core.Editor.CoreEditor↩ Utility.GetAssetsOfType ( string type ) [static]`

Find all the assets of the given type in the project.

Only assets whose root object is the given type are included.

Parameters

type	The type name of assets to find. Use the class name only, without namespace.

Returns

A sequence of all the found assets.

6.5.2.13 GetAssetsOfType< T >()

```
static IEnumerable< T > Infohazard.Core.Editor.CoreEditorUtility.GetAssetsOfT
T > (
            string folder = null )  [static]
```

Find all the assets of the given type in the project.

Only assets whose root object is the given type are included.

Parameters

folder	The folder to search in, or null for the entire project.

Template Parameters

T	The type of asset to find.

Returns

A sequence of all the found assets.

Type Constraints

T* : *Object

6.5.2.14 GetDefinesList()

```
static List< string > Infohazard.Core.Editor.CoreEditorUtility.Get↩
DefinesList (
            BuildTargetGroup group )  [static]
```

Get the Unity PlayerSettings list of #define symbols for the given build target.

Parameters

group	The build target.

Returns

A list of all defined symbols for that group.

6.5.2.15 GetFieldType() `static Type Infohazard.Core.Editor.CoreEditorUtility.GetFieldType ( this PropertyDrawer drawer ) [static]`

Get the type of the field the property drawer is drawing. If the type is an array or list, returns the element type.

Parameters

drawer	Drawer to get the type of.

Returns

Type of the drawer's field.

6.5.2.16 GetPathRelativeToAssetsFolder() `static string Infohazard.Core.Editor.CoreEditor←Utility.GetPathRelativeToAssetsFolder ( string path ) [static]`

Convert the given path to be relative to the Assets folder.

Accepts absolute paths, paths staring with "Assets/", and paths starting with "/"". </remarks> <param name="path">

Returns

6.5.2.17 GetResourcePath() [1/2] `static string Infohazard.Core.Editor.CoreEditorUtility.Get←ResourcePath ( Object obj ) [static]`

Get the path an object lives in relative to a resource folder.

The result path can be used with Resources.Load.

Parameters

obj	The object to get the path for.

Returns

The path relative to a resource folder, or null.

6.5.2.18 GetResourcePath() `[2/2]` `static string Infohazard.Core.Editor.CoreEditorUtility.Get←ResourcePath ( string path ) [static]`

Get the given path relative to a resource folder.

The result path can be used with Resources.Load.

Parameters

path	The path to search for.

Returns

The path relative to a resource folder, or null.

6.5.2.19 GetTypeName() `static string Infohazard.Core.Editor.CoreEditorUtility.GetTypeName ( this SerializedProperty property ) [static]`

Get the type full name (including assembly) of the type of the underlying field for the given property.

Due to how Unity works, this will not include the namespace. If you need to access the property type from a PropertyDrawer, use PropertyDrawer.fieldInfo.FieldType instead.

Parameters

property	Property to get type of.

Returns

Type name of the property's underlying type, without namespace.

6.5.2.20 GetValue() `static object Infohazard.Core.Editor.CoreEditorUtility.GetValue ( this SerializedProperty property ) [static]`

Get the value of a SerializedProperty of any type. Does not work for serializable objects or gradients. Enum values are returned as ints.

Parameters

property	The property to read.

Returns

The value of the property, or null if not readable.

6.5.2.21 InstantiatePrefabInScene()

```
static GameObject Infohazard.Core.Editor.CoreEditorUtility.↵
InstantiatePrefabInScene (
            string path,
            string name ) [static]
```

Instantiate the given prefab in the scene, as a child of the selected GameObject if there is one.

Parameters

path	Prefab path to load.
name	Name to assign to the object, or null.

Returns

The instantiated GameObject.

6.5.2.22 SetSymbolDefined()

```
static void Infohazard.Core.Editor.CoreEditorUtility.SetSymbol↵
Defined (
            string symbol,
            bool value,
            BuildTargetGroup group ) [static]
```

Sets whether the given symbol is defined in the PlayerSettings for the given build target.

Parameters

symbol	The symbol to set.
value	Whether the symbol should be defined.
group	The build target.

6.5.3 Member Data Documentation

6.5.3.1 DataFolder `const string Infohazard.Core.Editor.CoreEditorUtility.DataFolder = "Assets/Infohazard.↵ Core.Data/" [static]`

Folder where all generated files used by the Infohazard libraries should live.

Call EnsureDataFolderExists to make sure this folder exists before using it.

The documentation for this class was generated from the following file:

- Editor/Utility/CoreEditorUtility.cs

6.6 Infohazard.Core.DebugUtility Class Reference

Contains various static methods relating to debugging and diagnostics.

Static Public Member Functions

- static void DrawDebugBounds (Bounds bounds, Color color, float duration=0.0f, bool depthTest=true)

  Draw the given Bounds in the scene view.
- static void DrawDebugCircle (Vector3 point, Vector3 normal, float radius, Color color, float duration=0.0f, bool depthTest=true, int pointCount=32)

  Draw the given circle in the scene view.
- static void DrawDebugSphere (Vector3 center, float radius, Color color, float duration=0.0f, bool depth↵ Test=true, int pointCount=32)

  Draw the given sphere in the scene view.
- static void DebugBreakAfterFrames (this MonoBehaviour cmp, int frames)

  Pause the editor after a given number of frames, using a Coroutine and Debug.Break().
- static bool CheckPlaying (bool propertySet=false, [CallerMemberName] string callerName="")

  Checks whether in play mode (including standalone), and prints an error if it is.

6.6.1 Detailed Description

Contains various static methods relating to debugging and diagnostics.

6.6.2 Member Function Documentation

6.6.2.1 CheckPlaying()

```
static bool Infohazard.Core.DebugUtility.CheckPlaying (
        bool propertySet = false,
        [CallerMemberName] string callerName = "" )  [static]
```

Checks whether in play mode (including standalone), and prints an error if it is.

Used to ensure certain properties are not edited while playing.

Parameters

propertySet	Whether the caller is a property set accessor (changes error log).
callerName	Set automatically, do not supply a value for this parameter.

Returns

True if in play mode.

6.6.2.2 DebugBreakAfterFrames() `static void Infohazard.Core.DebugUtility.DebugBreakAfterFrames (`

```
            this MonoBehaviour cmp,
            int frames )  [static]
```

Pause the editor after a given number of frames, using a Coroutine and Debug.Break().

Only works in play mode. Will not cause errors if used in a standalone build, but will do unnecessary work.

Parameters

cmp	Component to attach the Coroutine to.
frames	Number of frames to wait before pausing.

6.6.2.3 DrawDebugBounds() `static void Infohazard.Core.DebugUtility.DrawDebugBounds (`

```
            Bounds bounds,
            Color color,
            float duration = 0.0f,
            bool depthTest = true )  [static]
```

Draw the given Bounds in the scene view.

Parameters

bounds	Bounds to draw.
color	Color to use.
duration	Time, in seconds, to draw the lines for.
depthTest	Whether to depth dest the drawn lines.

6.6.2.4 DrawDebugCircle() `static void Infohazard.Core.DebugUtility.DrawDebugCircle (`

```
            Vector3 point,
            Vector3 normal,
            float radius,
            Color color,
            float duration = 0.0f,
            bool depthTest = true,
            int pointCount = 32 )  [static]
```

Draw the given circle in the scene view.

Parameters

point	Center of the circle.
normal	Normal that is perpendicular to the circle.
radius	Radius of the circle.
color	Color to use.
duration	Time, in seconds, to draw the circle for.
depthTest	Whether to depth dest the drawn circle.
pointCount	How many points the circle will consist of.

6.6.2.5 DrawDebugSphere()

```
static void Infohazard.Core.DebugUtility.DrawDebugSphere (
Vector3 center,
float radius,
Color color,
float duration = 0.0f,
bool depthTest = true,
int pointCount = 32 )  [static]
```

Draw the given sphere in the scene view.

Parameters

center	Center of the sphere.
radius	Radius of the sphere.
color	Color to use.
duration	Time, in seconds, to draw the sphere for.
depthTest	Whether to depth dest the drawn sphere.
pointCount	How many points each circle will consist of.

The documentation for this class was generated from the following file:

- Runtime/Utility/DebugUtility.cs

6.7 Infohazard.Core.DefaultPoolHandler Class Reference

An IPoolHandler that is passed a prefab directly.

Public Member Functions

- DefaultPoolHandler (Spawnable prefab, Transform transform)

  Construct DefaultPoolHandler for a given prefab, attached to a given Transform.
- override string ToString ()
- virtual Spawnable Spawn ()

  Spawn an object from the pool, using an inactive instance from the pool if possible (otherwise, a new instance is created via Instantiate).
- virtual void Despawn (Spawnable instance)

  Despawn an object, deactivating it and releasing it back to the pool.
- virtual void Retain ()

  Add an additional user to the pool, ensuring pooled instances will remain available.
- virtual void Release ()

  Remove a user from the pool, clearing inactive instances if user count reaches zero.

Protected Member Functions

- virtual void OnGet (Spawnable obj)

  Override to perform custom logic when an object is being spawned from the pool.
- virtual void OnRelease (Spawnable obj)

  Override to perform custom logic when an object is being despawned and released back to the pool.
- virtual Spawnable Instantiate ()

  Instantiate an object to add to the pool. Override to perform custom instantiation logic.
- virtual void Destroy (Spawnable obj)

  Destroy an object after it is removed from the pool. Override to perform custom destruction logic.
- void CheckClear ()

  Clear the pool if ShouldClear returns true.
- virtual bool ShouldClear ()

  Should the pool be cleared?
- virtual void Clear ()

  Clear all the inactive instances from the pool to free up memory.

Properties

- Spawnable Prefab `[get, protected set]`

  The loaded prefab to be spawned by the handler.
- Pool< Spawnable > Pool `[get]`

  The pool that will spawn the prefab.
- Transform Transform `[get]`

  The transform that inactive instances in the pool are parented to.
- virtual bool ShouldPool `[get]`

  Whether to use pooling - by default this returns Spawnable.Pooled.
- int RetainCount `[get, private set]`

6.7.1 Detailed Description

An IPoolHandler that is passed a prefab directly.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DefaultPoolHandler() `Infohazard.Core.DefaultPoolHandler.DefaultPoolHandler ( Spawnable prefab, Transform transform )`

Construct DefaultPoolHandler for a given prefab, attached to a given Transform.

Parameters

prefab	Prefab that this handler will spawn.
transform	Transform to attach inactive instances to.

6.7.3 Member Function Documentation

6.7.3.1 CheckClear() `void Infohazard.Core.DefaultPoolHandler.CheckClear ( ) [protected]`

Clear the pool if ShouldClear returns true.

6.7.3.2 Clear() `virtual void Infohazard.Core.DefaultPoolHandler.Clear ( ) [protected], [virtual]`

Clear all the inactive instances from the pool to free up memory.

6.7.3.3 Despawn() `virtual void Infohazard.Core.DefaultPoolHandler.Despawn ( Spawnable instance ) [virtual]`

Despawn an object, deactivating it and releasing it back to the pool.

Parameters

instance	The object to be despawned.

6.7.3.4 Destroy() `virtual void Infohazard.Core.DefaultPoolHandler.Destroy ( Spawnable obj ) [protected], [virtual]`

Destroy an object after it is removed from the pool. Override to perform custom destruction logic.

This method is only used when an object is no longer needed and is permanently destroyed. To implement logic that happens every time an object is despawned, see OnRelease.

Parameters

obj	The object to be destroyed.

6.7.3.5 Instantiate() `virtual Spawnable Infohazard.Core.DefaultPoolHandler.Instantiate ( ) [protected], [virtual]`

Instantiate an object to add to the pool. Override to perform custom instantiation logic.

This method is only used when a new object is needed and the pool is empty. To implement logic that happens every time an object is spawned, see OnGet. This method makes the spawned object inactive. If you implement your own instantiate logic instead of calling the base method, you should also make the object inactive.

Returns

The spawned object.

6.7.3.6 OnGet() `virtual void Infohazard.Core.DefaultPoolHandler.OnGet ( Spawnable obj ) [protected], [virtual]`

Override to perform custom logic when an object is being spawned from the pool.

Invoked before the object is made active. This method is called every time a pooled object becomes active (including right after it is instantiated). To implement custom creation logic that only happens once per object, see Instantiate.

Parameters

obj	The object being spawned.

6.7.3.7 OnRelease() `virtual void Infohazard.Core.DefaultPoolHandler.OnRelease ( Spawnable obj ) [protected], [virtual]`

Override to perform custom logic when an object is being despawned and released back to the pool.

Invoked before the object is made inactive. This method is called every time a pooled object becomes inactive (included right before it is destroyed). To implement cleanup creation logic that only happens once per object, see Destroy(Spawnable).

Parameters

obj	The object being spawned.

6.7.3.8 Release() `virtual void Infohazard.Core.DefaultPoolHandler.Release ( ) [virtual]`

Remove a user from the pool, clearing inactive instances if user count reaches zero.

When removing the last user, all inactive instances will be destroyed, and any additional calls to Spawn will need to create a new object via Instantiate.

6.7.3.9 Retain() `virtual void Infohazard.Core.DefaultPoolHandler.Retain ( ) [virtual]`

Add an additional user to the pool, ensuring pooled instances will remain available.

6.7.3.10 ShouldClear() `virtual bool Infohazard.Core.DefaultPoolHandler.ShouldClear ( ) [protected], [virtual]`

Should the pool be cleared?

The base implementation returns true if RetainCount == 0.

Returns

True if the pool should be cleared.

6.7.3.11 Spawn() `virtual Spawnable Infohazard.Core.DefaultPoolHandler.Spawn ( ) [virtual]`

Spawn an object from the pool, using an inactive instance from the pool if possible (otherwise, a new instance is created via Instantiate).

Returns

The spawned object.

6.7.3.12 ToString() `override string Infohazard.Core.DefaultPoolHandler.ToString ( )`

6.7.4 Property Documentation

6.7.4.1 Pool `Pool<Spawnable> Infohazard.Core.DefaultPoolHandler.Pool [get], [protected]`

The pool that will spawn the prefab.

6.7.4.2 Prefab `Spawnable Infohazard.Core.DefaultPoolHandler.Prefab [get], [protected set]`

The loaded prefab to be spawned by the handler.

6.7.4.3 RetainCount `int Infohazard.Core.DefaultPoolHandler.RetainCount [get], [private set]`

6.7.4.4 ShouldPool `virtual bool Infohazard.Core.DefaultPoolHandler.ShouldPool [get]`

Whether to use pooling - by default this returns Spawnable.Pooled.

6.7.4.5 Transform `Transform Infohazard.Core.DefaultPoolHandler.Transform [get]`

The transform that inactive instances in the pool are parented to.

The documentation for this class was generated from the following file:

- Runtime/Pooling/DefaultPoolHandler.cs

6.8 Infohazard.Core.DrawSingleChildPropertyAttribute Class Reference

Attribute draws only a single child property of a property.

Public Member Functions

- DrawSingleChildPropertyAttribute (string propertyName)

  DrawSingleChildPropertyAttribute that will draw only the specified property.

Properties

- string PropertyName `[get]`

  The child property to draw.

6.8.1 Detailed Description

Attribute draws only a single child property of a property.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DrawSingleChildPropertyAttribute() `Infohazard.Core.DrawSingleChildPropertyAttribute.↩`

```
DrawSingleChildPropertyAttribute (
            string propertyName )
```

DrawSingleChildPropertyAttribute that will draw only the specified property.

Parameters

propertyName	The child property to draw.

6.8.3 Property Documentation

6.8.3.1 PropertyName `string Infohazard.Core.DrawSingleChildPropertyAttribute.PropertyName [get]`

The child property to draw.

The documentation for this class was generated from the following file:

- Runtime/Attributes/DrawSingleChildPropertyAttribute.cs

6.9 Infohazard.Core.EditNameOnlyAttribute Class Reference

Attribute that draws only the name of an Object reference field.

6.9.1 Detailed Description

Attribute that draws only the name of an Object reference field.

The documentation for this class was generated from the following file:

- Runtime/Attributes/EditNameOnlyAttribute.cs

6.10 Infohazard.Core.EnumerableUtility Class Reference

Contains various static methods for working with sequences, extending the functionality of LINQ.

Public Member Functions

- delegate bool SelectWhereDelegate< in T1, T2 > (T1 input, out T2 output)

  Delegate for functions that perform both a select/map and where/filter operation.

Static Public Member Functions

- static IEnumerable< T2 > SelectWhere< T1, T2 > (this IEnumerable< T1 > input, SelectWhereDelegate< T1, T2 > selectionDelegate)

  Perform select/map and where/filter operations on a sequence with a single function.
- static T2 FirstOrDefaultWhere< T1, T2 > (this IEnumerable< T1 > input, SelectWhereDelegate< T1, T2 > selectionDelegate)

  Perform select/map and where/filter operations on a sequence with a single function, and returns the result of the select operation for the first passing element.
- static int IndexOf< T > (this IReadOnlyList< T > list, in T item)

  IndexOf operation for an IReadOnlyList, which is not included in .NET.

6.10.1 Detailed Description

Contains various static methods for working with sequences, extending the functionality of LINQ.

6.10.2 Member Function Documentation

6.10.2.1 FirstOrDefaultWhere< T1, T2 >()

```
static T2 Infohazard.Core.EnumerableUtility.FirstOr↩
DefaultWhere< T1, T2 > (
            this IEnumerable< T1 > input,
            SelectWhereDelegate< T1, T2 > selectionDelegate )  [static]
```

Perform select/map and where/filter operations on a sequence with a single function, and returns the result of the select operation for the first passing element.

Parameters

input	The input sequence.
selectionDelegate	The delegate to use.

Template Parameters

T1	Input type of the delegate.
T2	Output type of the select operation.

Returns

The result of the select operation for the first passing element.

6.10.2.2 IndexOf< T >()

```
static int Infohazard.Core.EnumerableUtility.IndexOf< T > (
            this IReadOnlyList< T > list,
            in T item )  [static]
```

IndexOf operation for an IReadOnlyList, which is not included in .NET.

This cannot take into account custom equality comparers.

Parameters

list	List to search.
item	Item to search for.

Template Parameters

T	Type of items in the list.

Returns

The index of the item, or -1 if not found.

6.10.2.3 SelectWhere< T1, T2 >()

```
static IEnumerable< T2 > Infohazard.Core.EnumerableUtility.↩
SelectWhere< T1, T2 > (
            this IEnumerable< T1 > input,
            SelectWhereDelegate< T1, T2 > selectionDelegate ) [static]
```

Perform select/map and where/filter operations on a sequence with a single function.

Parameters

input	The input sequence.
selectionDelegate	The delegate to use.

Template Parameters

T1	Input type of the delegate.
T2	Output type of the select operation.

Returns

The resulting sequence.

6.10.2.4 SelectWhereDelegate< in T1, T2 >()

```
delegate bool Infohazard.Core.EnumerableUtility.↩
SelectWhereDelegate< in T1, T2 > (
            T1 input,
            out T2 output )
```

Delegate for functions that perform both a select/map and where/filter operation.

Template Parameters

T1	Input type of the function.
T2	Output type of the select operation.

The documentation for this class was generated from the following file:

- Runtime/Utility/EnumerableUtility.cs

6.11 Infohazard.Core.ExpandableAttribute Class Reference

Attribute that enables editing properties of a referenced object.

Public Member Functions

- ExpandableAttribute (bool alwaysExpanded=false, string savePath=null, bool showNewButton=true, bool showChildTypes=false, params Type[ ] requiredInterfaces)

  Construct a new ExpandableAttribute.

Properties

- bool AlwaysExpanded `[get]`

  Whether the attribute is always expanded.
- bool ShowNewButton `[get]`

  Whether to show a "New" button to create new instances.
- string SavePath `[get]`

  The default path to save newly created ScriptableObjects at.
- bool ShowChildTypes `[get]`

  Whether to show a dropdown of all types extending the type of the variable.
- IReadOnlyList< Type > RequiredInterfaces `[get]`

  Interfaces that must be implemented by assigned objects.
- bool OnlyShowMainLine `[get, set]`

  If true, don't show the child properties even if the property is expanded. You can use this to implement your own inspector logic for the child properties.

6.11.1 Detailed Description

Attribute that enables editing properties of a referenced object.

Normally, to edit a referenced object, you'd have to change the inspector context. With an ExpandableAttribute, you can just expand the property and edit the referenced object inline. If the type of the reference field is a Scriptable↩ Object, the ExpandableAttribute also allows creating new instances in the inspector.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ExpandableAttribute() `Infohazard.Core.ExpandableAttribute.ExpandableAttribute (`

```
bool alwaysExpanded = false,
string savePath = null,
bool showNewButton = true,
bool showChildTypes = false,
params Type[] requiredInterfaces )
```

Construct a new ExpandableAttribute.

Parameters

alwaysExpanded	Whether the attribute is always expanded.
savePath	Whether to show a "New" button to create new instances.
showNewButton	The default path to save newly created ScriptableObjects at.
showChildTypes	Whether to show a dropdown of all types extending the type of the variable.
requiredInterfaces	Interfaces that must be implemented by assigned objects.

6.11.3 Property Documentation

6.11.3.1 AlwaysExpanded `bool Infohazard.Core.ExpandableAttribute.AlwaysExpanded [get]`

Whether the attribute is always expanded.

If false, will show the expander.

6.11.3.2 OnlyShowMainLine `bool Infohazard.Core.ExpandableAttribute.OnlyShowMainLine [get], [set]`

If true, don't show the child properties even if the property is expanded. You can use this to implement your own inspector logic for the child properties.

6.11.3.3 RequiredInterfaces `IReadOnlyList<Type> Infohazard.Core.ExpandableAttribute.Required↩ Interfaces [get]`

Interfaces that must be implemented by assigned objects.

6.11.3.4 SavePath `string Infohazard.Core.ExpandableAttribute.SavePath [get]`

The default path to save newly created ScriptableObjects at.

If unset, this will use the same path as the containing object if it is an asset.

6.11.3.5 ShowChildTypes `bool Infohazard.Core.ExpandableAttribute.ShowChildTypes [get]`

Whether to show a dropdown of all types extending the type of the variable.

6.11.3.6 ShowNewButton `bool Infohazard.Core.ExpandableAttribute.ShowNewButton [get]`

Whether to show a "New" button to create new instances.

This only works if the type of the field is a ScriptableObject.

The documentation for this class was generated from the following file:

- Runtime/Attributes/ExpandableAttribute.cs

6.12 Infohazard.Core.GameObjectUtility Class Reference

Contains utility methods for dealing with GameObjects and Transforms.

Static Public Member Functions

- static void GetCapsuleInfo (float radius, float height, Vector3 center, int direction, Matrix4x4 transform, out Vector3 point1, out Vector3 point2, out float worldRadius, out float worldHeight)

  Converts capsule info in transform/height/radius form to two-point form for use with Physics.CapsuleCast.
- static void GetCapsuleInfo (this CharacterController capsule, out Vector3 point1, out Vector3 point2, out float worldRadius, out float worldHeight)

  Converts capsule info in a CharacterController to two-point form for use with Physics.CapsuleCast.
- static void GetCapsuleInfo (this CapsuleCollider capsule, out Vector3 point1, out Vector3 point2, out float worldRadius, out float worldHeight)

  Converts capsule info in a CapsuleCollider to two-point form for use with Physics.CapsuleCast.
- static bool ColliderCast (this Collider collider, float padding, Vector3 direction, out RaycastHit hit, float maxDistance=float.PositiveInfinity, int layerMask=Physics.DefaultRaycastLayers, QueryTriggerInteraction triggerInteraction=QueryTriggerInteraction.UseGlobal)

  Perform a physics cast depending on the type of collider, using its parameters. Only BoxCollider, SphereCollider, and CapsuleCollider are supported. For simplicity, the scale of the transform is assumed to be uniform.
- static int ColliderCastNonAlloc (this Collider collider, float padding, Vector3 direction, RaycastHit[ ] hits, float maxDistance=float.PositiveInfinity, int layerMask=Physics.DefaultRaycastLayers, QueryTriggerInteraction triggerInteraction=QueryTriggerInteraction.UseGlobal)

  Perform a physics cast depending on the type of collider, using its parameters. Only BoxCollider, SphereCollider, and CapsuleCollider are supported. For simplicity, the scale of the transform is assumed to be uniform.
- static void SetParentAndReset (this Transform transform, Transform parent)

  Set the parent of the given transform, and reset it's local position, rotation, and scale.
- static void Initialize (this Transform transform, Transform parent, Vector3? position=null, Quaternion? rotation=null, Vector3? scale=null, bool inWorldSpace=false, in Scene? scene=null)

  Initialize the transform with the given parent, position, rotation, and scale.
- static void Initialize (this Transform transform, in SpawnParams spawnParams=default)

  Initialize the transform with the given spawn params.
- static void Initialize (this Transform transform, Vector3? position=null, Quaternion? rotation=null, Vector3? scale=null)

  Set's the transform's position, and rotation, and scale (if they are specified).
- static void DestroyChildren (this Transform transform)

  Destroy all of the child GameObjects of a Transform at the end of this frame.
- static void DestroyChildrenImmediate (this Transform transform)

  Destroy all of the child GameObjects of a Transform immediately.
- static void DespawnChildren (this Transform transform)

  Despawn all of the child GameObjects of a Transform.
- static bool TryGetComponentInParent< T > (this GameObject obj, out T result)

  Like GetComponentInParent, but more convenient if using in conditionals and also using the component value.
- static bool TryGetComponentInChildren< T > (this GameObject obj, out T result, bool includeInactive=false)

  Like GetComponentInChildren, but more convenient if using in conditionals and also using the component value.
- static bool TryGetComponentInParent< T > (this Component cmp, out T result)

  Like GetComponentInParent, but more convenient if using in if statements and also using the component value.
- static bool TryGetComponentInChildren< T > (this Component cmp, out T result)

  Like GetComponentInChildren, but more convenient if using in if statements and also using the component value.
- static string GetRelativeTransformPath (this Transform from, Transform to)

  Get the path from one transform to another (object names separated by slashes). Names are URL encoded in case they contain slashes.

- static Transform GetTransformAtRelativePath (this Transform from, string path)

  Parses a slash-separated Transform path from a parent object to find a child. The names are expected to be URL encoded.
- static bool IsPathUnique (this Transform from, Transform to)

  Check whether the path from one transform to another (the one returned by GetRelativeTransformPath) is unique. If the path is not unique, using GetTransformAtRelativePath will not necessarily return the correct transform.
- static bool IsNameUniqueInSiblings (this Transform transform)

  Return whether the name of an object is unique within its siblings.
- static void SetLayerRecursively (this GameObject obj, int layer)

  Sets the layer of a GameObject and all of its children.
- static void SetLinearVelocity (this Rigidbody rb, Vector3 velocity)

  Utility method to accommodate Unity 6 velocity name change.
- static Vector3 GetLinearVelocity (this Rigidbody rb)

  Utility method to accommodate Unity 6 velocity name change.
- static void AddLinearVelocity (this Rigidbody rb, Vector3 velocity)

  Utility method to accommodate Unity 6 velocity name change.
- static void MoveTowardsLinearVelocity (this Rigidbody rb, Vector3 targetVelocity, float maxDeltaV)

  Utility method to accommodate Unity 6 velocity name change.

6.12.1 Detailed Description

Contains utility methods for dealing with GameObjects and Transforms.

6.12.2 Member Function Documentation

6.12.2.1 AddLinearVelocity()

```
static void Infohazard.Core.GameObjectUtility.AddLinearVelocity (
        this Rigidbody rb,
        Vector3 velocity ) [static]
```

Utility method to accommodate Unity 6 velocity name change.

6.12.2.2 ColliderCast()

```
static bool Infohazard.Core.GameObjectUtility.ColliderCast (
        this Collider collider,
        float padding,
        Vector3 direction,
        out RaycastHit hit,
        float maxDistance = float.PositiveInfinity,
        int layerMask = Physics.DefaultRaycastLayers,
        QueryTriggerInteraction triggerInteraction = QueryTriggerInteraction.UseGlobal )
[static]
```

Perform a physics cast depending on the type of collider, using its parameters. Only BoxCollider, SphereCollider, and CapsuleCollider are supported. For simplicity, the scale of the transform is assumed to be uniform.

Parameters

collider	The BoxCollider, SphereCollider, or CapsuleCollider to cast from.
padding	A padding to reduce the collider's extents. Given in world units.
direction	Direction to cast in.
hit	The hit information.
maxDistance	The maximum distance to cast. Default is infinity.
layerMask	The layer mask to cast against. Default is default raycast layers.
triggerInteraction	Whether to include triggers. Default is use global settings.

Returns

Whether the cast hit something.

6.12.2.3 ColliderCastNonAlloc() `static int Infohazard.Core.GameObjectUtility.ColliderCastNon↩ Alloc (`

```
            this Collider collider,
            float padding,
            Vector3 direction,
            RaycastHit[] hits,
            float maxDistance = float.PositiveInfinity,
            int layerMask = Physics.DefaultRaycastLayers,
            QueryTriggerInteraction triggerInteraction = QueryTriggerInteraction.UseGlobal )
[static]
```

Perform a physics cast depending on the type of collider, using its parameters. Only BoxCollider, SphereCollider, and CapsuleCollider are supported. For simplicity, the scale of the transform is assumed to be uniform.

Parameters

collider	The BoxCollider, SphereCollider, or CapsuleCollider to cast from.
padding	A padding to reduce the collider's extents. Given in world units.
direction	Direction to cast in.
hits	The hit information.
maxDistance	The maximum distance to cast. Default is infinity.
layerMask	The layer mask to cast against. Default is default raycast layers.
triggerInteraction	Whether to include triggers. Default is use global settings.

Returns

The number of valid hits.

6.12.2.4 DespawnChildren() `static void Infohazard.Core.GameObjectUtility.DespawnChildren (`

```
            this Transform transform )  [static]
```

Despawn all of the child GameObjects of a Transform.

Parameters

transform	Transform to despawn children of.

6.12.2.5 DestroyChildren()

```
static void Infohazard.Core.GameObjectUtility.DestroyChildren (
        this Transform transform )  [static]
```

Destroy all of the child GameObjects of a Transform at the end of this frame.

Parameters

transform	Transform to destroy children of.

6.12.2.6 DestroyChildrenImmediate()

```
static void Infohazard.Core.GameObjectUtility.Destroy↵
ChildrenImmediate (
        this Transform transform )  [static]
```

Destroy all of the child GameObjects of a Transform immediately.

Parameters

transform	Transform to destroy children of.

6.12.2.7 GetCapsuleInfo() [1/3]

```
static void Infohazard.Core.GameObjectUtility.GetCapsuleInfo (
        float radius,
        float height,
        Vector3 center,
        int direction,
        Matrix4x4 transform,
        out Vector3 point1,
        out Vector3 point2,
        out float worldRadius,
        out float worldHeight )  [static]
```

Converts capsule info in transform/height/radius form to two-point form for use with Physics.CapsuleCast.

Also tells you the radius and height of the capsule in world space.

Parameters

radius	Radius of the capsule in local space.
height	Height of the capsule in local space.
center	Center of the capsule in local space.
direction	On which axis the capsule extends (0 = x, 1 = y, 2 = z).

Parameters

transform	Transform that the capsule is parented to.
point1	The first point of the capsule in world space.
point2	The second point of the capsule in world space.
worldRadius	The radius of the capsule in world space.
worldHeight	The height of the capsule in world space.

6.12.2.8 GetCapsuleInfo() `[2/3]`

```
static void Infohazard.Core.GameObjectUtility.GetCapsuleInfo (
        this CapsuleCollider capsule,
        out Vector3 point1,
        out Vector3 point2,
        out float worldRadius,
        out float worldHeight )  [static]
```

Converts capsule info in a CapsuleCollider to two-point form for use with Physics.CapsuleCast.

Also tells you the radius and height of the capsule in world space.

Parameters

capsule	The CapsuleCollider to read.
point1	The first point of the capsule in world space.
point2	The second point of the capsule in world space.
worldRadius	The radius of the capsule in world space.
worldHeight	The height of the capsule in world space.

6.12.2.9 GetCapsuleInfo() `[3/3]`

```
static void Infohazard.Core.GameObjectUtility.GetCapsuleInfo (
        this CharacterController capsule,
        out Vector3 point1,
        out Vector3 point2,
        out float worldRadius,
        out float worldHeight )  [static]
```

Converts capsule info in a CharacterController to two-point form for use with Physics.CapsuleCast.

Also tells you the radius and height of the capsule in world space.

Parameters

capsule	The CharacterController to read.
point1	The first point of the capsule in world space.
point2	The second point of the capsule in world space.
worldRadius	The radius of the capsule in world space.
worldHeight	The height of the capsule in world space.

6.12.2.10 GetLinearVelocity() `static Vector3 Infohazard.Core.GameObjectUtility.GetLinearVelocity (`

```
            this Rigidbody rb )  [static]
```

Utility method to accommodate Unity 6 velocity name change.

6.12.2.11 GetRelativeTransformPath() `static string Infohazard.Core.GameObjectUtility.Get↩ RelativeTransformPath (`

```
            this Transform from,
            Transform to )  [static]
```

Get the path from one transform to another (object names separated by slashes). Names are URL encoded in case they contain slashes.

The parameter to must be a direct descendent of from, or an error is logged. The returned path contains the name of to but not from. This path can be turned back to an object reference using GetTransformAtRelativePath.

Parameters

from	The parent Transform to get the path from.
to	The Transform to get the path to.

Returns

The path relative transform path separated by slashes.

6.12.2.12 GetTransformAtRelativePath() `static Transform Infohazard.Core.GameObjectUtility.Get↩ TransformAtRelativePath (`

```
            this Transform from,
            string path )  [static]
```

Parses a slash-separated Transform path from a parent object to find a child. The names are expected to be URL encoded.

This can be used to turn a path created by GetRelativeTransformPath back to an object reference.

Parameters

from	The parent Transform to search from.
path	The slash-separated path to search for.

Returns

The found child Transform, or null if not found.

6.12.2.13 Initialize() [1/3]

```
static void Infohazard.Core.GameObjectUtility.Initialize (
            this Transform transform,
            in SpawnParams spawnParams = default )  [static]
```

Initialize the transform with the given spawn params.

Parameters

transform	The transform to initialize.
spawnParams	The spawn parameters.

6.12.2.14 Initialize() [2/3]

```
static void Infohazard.Core.GameObjectUtility.Initialize (
            this Transform transform,
            Transform parent,
            Vector3? position = null,
            Quaternion? rotation = null,
            Vector3? scale = null,
            bool inWorldSpace = false,
            in Scene? scene = null )  [static]
```

Initialize the transform with the given parent, position, rotation, and scale.

Parameters

transform	The transform to initialize.
parent	The parent to attach to.
position	The position (if null, do not set).
rotation	The rotation (if null, do not set).
scale	The scale (if null, do not set).
inWorldSpace	Whether the given position, rotation, and scale should be considered global.
scene	An optional scene to move the object to.

6.12.2.15 Initialize() [3/3]

```
static void Infohazard.Core.GameObjectUtility.Initialize (
            this Transform transform,
            Vector3? position = null,
            Quaternion? rotation = null,
            Vector3? scale = null )  [static]
```

Set's the transform's position, and rotation, and scale (if they are specified).

Parameters

transform	The transform to initialize.
position	The position (if null, do not set).
rotation	The rotation (if null, do not set).
scale	The scale (if null, do not set).

6.12.2.16 IsNameUniqueInSiblings()

```
static bool Infohazard.Core.GameObjectUtility.IsNameUnique↩
InSiblings (
            this Transform transform )  [static]
```

Return whether the name of an object is unique within its siblings.

Parameters

transform	The object to check.

Returns

Whether the name is unique.

6.12.2.17 IsPathUnique()

```
static bool Infohazard.Core.GameObjectUtility.IsPathUnique (
            this Transform from,
            Transform to )  [static]
```

Check whether the path from one transform to another (the one returned by GetRelativeTransformPath) is unique. If the path is not unique, using GetTransformAtRelativePath will not necessarily return the correct transform.

The names of the objects do not need to be globally unique. Rather, they must be unique within all of their siblings according to IsNameUniqueInSiblings.

Parameters

from	The transform to check the path from.
to	The transform to check the path to.

Returns

Whether the path is unique.

6.12.2.18 MoveTowardsLinearVelocity()

```
static void Infohazard.Core.GameObjectUtility.Move↩
TowardsLinearVelocity (
            this Rigidbody rb,
            Vector3 targetVelocity,
            float maxDeltaV )  [static]
```

Utility method to accommodate Unity 6 velocity name change.

6.12.2.19 SetLayerRecursively() `static void Infohazard.Core.GameObjectUtility.SetLayerRecursively (`

```
            this GameObject obj,
            int layer ) [static]
```

Sets the layer of a GameObject and all of its children.

Parameters

obj	The GameObject to set the layer on.
layer	The layer index to set.

6.12.2.20 SetLinearVelocity() `static void Infohazard.Core.GameObjectUtility.SetLinearVelocity (`

```
            this Rigidbody rb,
            Vector3 velocity ) [static]
```

Utility method to accommodate Unity 6 velocity name change.

6.12.2.21 SetParentAndReset() `static void Infohazard.Core.GameObjectUtility.SetParentAndReset (`

```
            this Transform transform,
            Transform parent ) [static]
```

Set the parent of the given transform, and reset it's local position, rotation, and scale.

Parameters

transform	The transform to reset.
parent	The transform to parent it to (can be null).

6.12.2.22 TryGetComponentInChildren< T >() [1/2] `static bool Infohazard.Core.GameObject↩ Utility.TryGetComponentInChildren< T > (`

```
            this Component cmp,
            out T result ) [static]
```

Like GetComponentInChildren, but more convenient if using in if statements and also using the component value.

6.12.2.23 TryGetComponentInChildren< T >() [2/2] `static bool Infohazard.Core.GameObject↩ Utility.TryGetComponentInChildren< T > (`

```
            this GameObject obj,
            out T result,
            bool includeInactive = false ) [static]
```

Like GetComponentInChildren, but more convenient if using in conditionals and also using the component value.

6.12.2.24 TryGetComponentInParent< T >() [1/2]

```
static bool Infohazard.Core.GameObjectUtility.TryGetComponentInParent< T > (
            this Component cmp,
            out T result ) [static]
```

Like GetComponentInParent, but more convenient if using in if statements and also using the component value.

6.12.2.25 TryGetComponentInParent< T >() [2/2]

```
static bool Infohazard.Core.GameObjectUtility.TryGetComponentInParent< T > (
            this GameObject obj,
            out T result ) [static]
```

Like GetComponentInParent, but more convenient if using in conditionals and also using the component value.

Parameters

obj	The object to search from.
result	The found component or null.

Template Parameters

T	The type of component to search for.

Returns

Whether a component of the given type was found.

The documentation for this class was generated from the following file:

- Runtime/Utility/GameObjectUtility.cs

6.13 Infohazard.Core.HelpBoxAttribute Class Reference

Used to add a detailed help box that can be toggled in the inspector. This can be used to provide more information than a simple tooltip.

6.13.1 Detailed Description

Used to add a detailed help box that can be toggled in the inspector. This can be used to provide more information than a simple tooltip.

This attribute does not contain a custom drawer, in order to allow compatibility with other custom drawers. Instead, a custom editor script must be created to handle the display of the help box.

The documentation for this class was generated from the following file:

- Runtime/Attributes/HelpBoxAttribute.cs

6.14 Infohazard.Core.Editor.IObjectReferenceFieldAssignmentValidator Interface Reference

Used to validate and modify values that a user is attempting to assign to an object reference field.

Public Member Functions

- Object ValidateObjectFieldAssignment (Object[ ] references, Type objType, SerializedProperty property, int options)

  Validate the input objects for assignment to a field of the given type, and return a valid one.

6.14.1 Detailed Description

Used to validate and modify values that a user is attempting to assign to an object reference field.

6.14.2 Member Function Documentation

6.14.2.1 ValidateObjectFieldAssignment()

```
Object Infohazard.Core.Editor.IObjectReferenceField↩
AssignmentValidator.ValidateObjectFieldAssignment (
            Object[ ] references,
            Type objType,
            SerializedProperty property,
            int options )
```

Validate the input objects for assignment to a field of the given type, and return a valid one.

Parameters

references	All objects attempting to be assigned.
objType	The type of the field.
property	The property that references the field.
options	Additional options (used internally by Unity).

Returns

The object to assign to the field, or null.

Implemented in Infohazard.Core.Editor.ObjectReferenceFieldAssignmentValidator.

The documentation for this interface was generated from the following file:

- Editor/Utility/IObjectReferenceFieldAssignmentValidator.cs

6.15 Infohazard.Core.IPersistedInstance Interface Reference

This is a hack so that PoolManager can send messages to PersistedGameObjects.

Public Member Functions

- void SetupDynamicInstance (ulong persistedInstanceID)

  Initialize the current object as a new persisted instance with the given ID.
- void RegisterDestroyed ()

  Remove the current object from persistence.

6.15.1 Detailed Description

This is a hack so that PoolManager can send messages to PersistedGameObjects.

There is likely no need to use this interface yourself.

6.15.2 Member Function Documentation

6.15.2.1 RegisterDestroyed() `void Infohazard.Core.IPersistedInstance.RegisterDestroyed ( )`

Remove the current object from persistence.

6.15.2.2 SetupDynamicInstance() `void Infohazard.Core.IPersistedInstance.SetupDynamicInstance ( ulong persistedInstanceID )`

Initialize the current object as a new persisted instance with the given ID.

Parameters

persistedInstanceID	The instance ID to use.

The documentation for this interface was generated from the following file:

- Runtime/Pooling/IPersistedInstance.cs

6.16 Infohazard.Core.ListQueue< T > Class Template Reference

A FIFO data structure similar to a Queue, except that it implements all List operations.

Public Member Functions

- ListQueue (int initialCapacity=32)

  Construct a new ListQueue with the given capacity.
- ListQueue (IEnumerable< T > enumerable)

Construct a new ListQueue containing all the elements of the given sequence.

- void Enqueue (T item)

  Add an item to the front of the queue.
- void EnsureCapacity (int capacity)

  Ensures that the capacity of the queue is at least the given value, and grows if not.
- T Peek ()

  Returns the element at the front of the queue without removing it.
- bool TryPeek (out T item)

  Get the element at the front of the queue if it is not empty, and return whether this was successful.
- T Dequeue ()

  Removes and returns the element at the front of the queue.
- bool TryDequeue (out T item)

  Get the element at the front of the queue if it is not empty, remove it, and return whether this was successful.
- IEnumerator< T > GetEnumerator ()
- void Add (T item)
- void Clear ()
- bool Contains (T item)
- void CopyTo (T[ ] array, int arrayIndex)
- bool Remove (T item)
- int IndexOf (T item)
- void Insert (int index, T item)
- void RemoveAt (int index)
- void RemoveRange (int index, int count)

  Removes a range of items from the queue.

Properties

- int Count `[get, private set]`
- int Capacity `[get]`

  Current capacity, which will be automatically expanded when needed.
- bool IsReadOnly `[get]`
- T this[int index] `[get, set]`

6.16.1 Detailed Description

A FIFO data structure similar to a Queue, except that it implements all List operations.

This enables much greater flexibility than the builtin .NET Queue class. Unlike a List, it has O(1) performance for both Enqueue and Dequeue operations (assuming there is enough room).

Template Parameters

T	Type of elements in the structure.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 ListQueue() [1/2] Infohazard.Core.ListQueue< T >.ListQueue (int *initialCapacity* = 32)

Construct a new ListQueue with the given capacity.

Parameters

initialCapacity	Initial capacity, which will be expanded as needed.

6.16.2.2 ListQueue() [2/2] Infohazard.Core.ListQueue< T >.ListQueue (IEnumerable< T > *enumerable*)

Construct a new ListQueue containing all the elements of the given sequence.

Parameters

enumerable	Sequence to initialize the queue.

6.16.3 Member Function Documentation

6.16.3.1 Add() void Infohazard.Core.ListQueue< T >.Add (T *item*)

6.16.3.2 Clear() void Infohazard.Core.ListQueue< T >.Clear ( )

6.16.3.3 Contains() bool Infohazard.Core.ListQueue< T >.Contains (T *item*)

6.16.3.4 CopyTo() void Infohazard.Core.ListQueue< T >.CopyTo (T[ ] *array*, int *arrayIndex*)

6.16.3.5 Dequeue() T Infohazard.Core.ListQueue< T >.Dequeue ( )

Removes and returns the element at the front of the queue.

Returns

The item at the front of the queue.

Exceptions

InvalidOperationException	If the queue is empty.

6.16.3.6 Enqueue()

```
void Infohazard.Core.ListQueue< T >.Enqueue (
        T item )
```

Add an item to the front of the queue.

The capacity of the queue will be grown if needed.

Parameters

item	The item to add.

6.16.3.7 EnsureCapacity()

```
void Infohazard.Core.ListQueue< T >.EnsureCapacity (
        int capacity )
```

Ensures that the capacity of the queue is at least the given value, and grows if not.

Parameters

capacity	The capacity to ensure.

6.16.3.8 GetEnumerator()

```
IEnumerator< T > Infohazard.Core.ListQueue< T >.GetEnumerator ( )
```

6.16.3.9 IndexOf()

```
int Infohazard.Core.ListQueue< T >.IndexOf (
        T item )
```

6.16.3.10 Insert()

```
void Infohazard.Core.ListQueue< T >.Insert (
        int index,
        T item )
```

6.16.3.11 Peek()

```
T Infohazard.Core.ListQueue< T >.Peek ( )
```

Returns the element at the front of the queue without removing it.

Returns

The item at the front of the queue.

Exceptions

InvalidOperationException	If the queue is empty.

6.16.3.12 Remove()

```
bool Infohazard.Core.ListQueue< T >.Remove (
        T item )
```

6.16.3.13 RemoveAt()

```
void Infohazard.Core.ListQueue< T >.RemoveAt (
        int index )
```

6.16.3.14 RemoveRange()

```
void Infohazard.Core.ListQueue< T >.RemoveRange (
        int index,
        int count )
```

Removes a range of items from the queue.

Parameters

index	The first index to remove.
count	The number of items to remove.

6.16.3.15 TryDequeue()

```
bool Infohazard.Core.ListQueue< T >.TryDequeue (
        out T item )
```

Get the element at the front of the queue if it is not empty, remove it, and return whether this was successful.

Parameters

item	The item at the front of the queue.

Returns

Whether an item was available to dequeue.

6.16.3.16 TryPeek()

```
bool Infohazard.Core.ListQueue< T >.TryPeek (
        out T item )
```

Get the element at the front of the queue if it is not empty, and return whether this was successful.

Parameters

item	The item at the front of the queue.

Returns

Whether an item was available to peek.

6.16.4 Property Documentation

6.16.4.1 Capacity `int Infohazard.Core.ListQueue< T >.Capacity [get]`

Current capacity, which will be automatically expanded when needed.

Expanding capacity is an O(n) operation, so it should be avoided when possible.

6.16.4.2 Count `int Infohazard.Core.ListQueue< T >.Count [get], [private set]`

6.16.4.3 IsReadOnly `bool Infohazard.Core.ListQueue< T >.IsReadOnly [get]`

6.16.4.4 this[int index] `T Infohazard.Core.ListQueue< T >.this[int index] [get], [set]`

The documentation for this class was generated from the following file:

- Runtime/DataStructures/ListQueue.cs

6.17 Infohazard.Core.MathUtility Class Reference

Contains utility methods for working with mathematical types and solving math equations.

Static Public Member Functions

- static float RoundToNearest (float value, float factor)

  Round a value to the nearest multiple of a given factor.
- static float SignZero (float value)

  Same as Mathf.Sign, except that if the input is zero, it returns zero.
- static float NormalizeAngle (float angle)

  Normalize an angle to a value between 0 and 360.
- static Vector3 NormalizeAngles (Vector3 angles)

  Normalize a set of euler angles to values between 0 and 360.
- static float NormalizeInnerAngle (float angle)

  Normalize an angle to a value between -180 and 180.
- static Vector3 NormalizeInnerAngles (Vector3 angles)

  Normalize a set of euler angles to values between -180 and 180.
- static float ClampInnerAngle (float angle, float min, float max)

  Normalize an angle to a value between -180 and 180, then clamp it in the given range.
- static Vector3 ClampInnerAngles (Vector3 angles, Vector3 min, Vector3 max)

  Normalize a set of euler angles to values between -180 and 180, then clamp them in the given ranges.
- static Vector3 Multiply (Vector3 left, Vector3 right)

  Multiply the components of left by the components of right.
- static Vector3 Divide (Vector3 left, Vector3 right)

  Divide the components of left by the components of right.
- static Vector3 Reciprocal (Vector3 vector)

  Take the reciprocal of each component of a vector.
- static Vector3 Divide (float left, Vector3 right)

  Divide a float by each component of a vector.
- static Vector3 RoundToNearest (Vector3 vector, float factor)

  Round a each component of a vector to the nearest multiple of a given factor.
- static bool GetNearestPointOnLines (Ray line1, Ray line2, out float t1, out float t2)

  Find the point along each line where the lines come closest to each other.
- static float GetNearestPointOnLine (Ray line, Vector3 p)

  Get the point on a line where it is nearest to a position.
- static bool GetNearestPointOnSegment (Vector3 v1, Vector3 v2, Vector3 point, out Vector3 pointOnSegment)

  Find the point on a bounded line segment where it is nearest to a position, and return whether that point is in the segment's bounds.
- static Vector3 GetNearestPointOnTriangleIncludingBounds (Vector3 v1, Vector3 v2, Vector3 v3, Vector3 point)

  Find the point on a triangle (including its bounds) where it is nearest to a position.
- static bool GetNearestPointOnTriangle (Vector3 v1, Vector3 v2, Vector3 v3, Vector3 point, out Vector3 point↩ OnTriangle)

  Find the point on a triangle where it is nearest to a position, and return whether that point is in the triangle's bounds.
- static bool IsPointInsideBound (Vector3 v1, Vector3 v2, Vector3 normal, Vector3 point)

  Returns true if a given point is on the inner side (defined by a given normal) of a segment.
- static bool DoesSegmentIntersectTriangle (Vector3 v1, Vector3 v2, Vector3 v3, Vector3 s1, Vector3 s2, out float t)

  Raycast a line segment against a triangle, and return whether they intersect.
- static Vector3 WorldToCanvasPoint (this Camera camera, Canvas canvas, Vector3 point)

  Projects a point in the world onto a canvas in camera or overlay space.
- static Vector3 GetPerpendicularVector (this Vector3 vector)

  Returns a vector that is perpendicular to the given vector.
- static int Dot (Vector3Int v1, Vector3Int v2)

  Dot product of two int vectors.

- static Vector3 WithX (this Vector3 v, float x)

  Replace the X component of a vector.
- static Vector3 WithY (this Vector3 v, float y)

  Replace the Y component of a vector.
- static Vector3 WithZ (this Vector3 v, float z)

  Replace the Z component of a vector.
- static Vector2 WithX (this Vector2 v, float x)

  Replace the X component of a vector.
- static Vector2 WithY (this Vector2 v, float y)

  Replace the Y component of a vector.
- static Vector3 WithZ (this Vector2 v, float z)

  Convert a Vector2 to a Vector3 with the given Z.
- static Vector3 AsXY (this Vector2 v)

  Get a Vector3 with the components (x, y, 0).
- static Vector3 AsYX (this Vector2 v)

  Get a Vector3 with the components (y, x, 0).
- static Vector3 AsXZ (this Vector2 v)

  Get a Vector3 with the components (x, 0, y).
- static Vector3 AsZX (this Vector2 v)

  Get a Vector3 with the components (y, 0, x).
- static Vector3 AsYZ (this Vector2 v)

  Get a Vector3 with the components (0, x, y).
- static Vector3 AsZY (this Vector2 v)

  Get a Vector3 with the components (0, y, x).
- static Vector2 ToXY (this Vector3 v)

  Get a Vector2 with the components (x, y).
- static Vector2 ToYX (this Vector3 v)

  Get a Vector2 with the components (y, x).
- static Vector2 ToXZ (this Vector3 v)

  Get a Vector2 with the components (x, z).
- static Vector2 ToZX (this Vector3 v)

  Get a Vector2 with the components (z, x).
- static Vector2 ToYZ (this Vector3 v)

  Get a Vector2 with the components (y, z).
- static Vector2 ToZY (this Vector3 v)

  Get a Vector2 with the components (z, y).
- static Vector4 ToV4Pos (this Vector3 vector)

  Get a Vector4 with the components (x, y, z, 1).
- static Vector4 ToV4 (this Vector3 vector)

  Get a Vector4 with the components (x, y, z, 0).
- static Quaternion XYRotation (Vector3 right, Vector3 upHint)

  Get a quaternion based on a right vector and approximate up vector.
- static Quaternion YXRotation (Vector3 up, Vector3 rightHint)

  Get a quaternion based on a up vector and approximate right vector.
- static Quaternion XZRotation (Vector3 right, Vector3 forwardHint)

  Get a quaternion based on a right vector and approximate forward vector.
- static Quaternion ZXRotation (Vector3 forward, Vector3 rightHint)

  Get a quaternion based on a forward vector and approximate right vector.
- static Quaternion YZRotation (Vector3 up, Vector3 forwardHint)

  Get a quaternion based on a up vector and approximate forward vector.
- static Quaternion ZYRotation (Vector3 forward, Vector3 upHint)

Get a quaternion based on a forward vector and approximate up vector.
- static void GetCorners (this Bounds bounds, Vector3[ ] corners)

  Get the eight corners of a bounding box and save them in the given array.
- static bool BoundsToScreenRect (Transform transform, Bounds bounds, Func< Vector3, Vector3 > world↩ToScreen, out Rect rect)

  Get a screen rect that encapsulates the given bounds.
- static void SplitHorizontal (Rect rect, float gap, out Rect out1, out Rect out2, float div=0.5f)

  Split a rect into two halves horizontally, with given gap between the halves.
- static void SplitHorizontal (Rect rect, float gap, out Rect out1, out Rect out2, out Rect out3, float div1=1.0f/3.0f, float div2=2.0f/3.0f)

  Split a rect into three thirds horizontally, with given gap between the thirds.

Static Public Attributes

- static Complex

  Evaluate all cubic roots of this Complex.
- static Complex r1

  Solve a quadratic equation (find x such that the result is zero) in the form ax^2 + bx + c = 0.
- static readonly Vector3[ ] BoundsCornerArray = new Vector3[8]

  A static array that can be used to store the output of GetCorners, as long as the values are copied from the array right away.

6.17.1 Detailed Description

Contains utility methods for working with mathematical types and solving math equations.

6.17.2 Member Function Documentation

6.17.2.1 AsXY() `static Vector3 Infohazard.Core.MathUtility.AsXY ( this Vector2 v ) [static]`

Get a Vector3 with the components (x, y, 0).

6.17.2.2 AsXZ() `static Vector3 Infohazard.Core.MathUtility.AsXZ ( this Vector2 v ) [static]`

Get a Vector3 with the components (x, 0, y).

6.17.2.3 AsYX() `static Vector3 Infohazard.Core.MathUtility.AsYX ( this Vector2 v ) [static]`

Get a Vector3 with the components (y, x, 0).

6.17.2.4 AsYZ()

```
static Vector3 Infohazard.Core.MathUtility.AsYZ (
        this Vector2 v )  [static]
```

Get a Vector3 with the components (0, x, y).

6.17.2.5 AsZX()

```
static Vector3 Infohazard.Core.MathUtility.AsZX (
        this Vector2 v )  [static]
```

Get a Vector3 with the components (y, 0, x).

6.17.2.6 AsZY()

```
static Vector3 Infohazard.Core.MathUtility.AsZY (
        this Vector2 v )  [static]
```

Get a Vector3 with the components (0, y, x).

6.17.2.7 BoundsToScreenRect()

```
static bool Infohazard.Core.MathUtility.BoundsToScreenRect (
        Transform transform,
        Bounds bounds,
        Func< Vector3, Vector3 > worldToScreen,
        out Rect rect )  [static]
```

Get a screen rect that encapsulates the given bounds.

Parameters

transform	Parent that the bounds are attached to (can be null).
bounds	The input bounds.
worldToScreen	A function that converts world points to screen points, such as Camera.WorldToScreenPoint.
rect	A screen rect that encapsulates the bounds.

Returns

Whether a screen rect could be calculated (false if completely off screen).

6.17.2.8 ClampInnerAngle()

```
static float Infohazard.Core.MathUtility.ClampInnerAngle (
        float angle,
        float min,
        float max )  [static]
```

Normalize an angle to a value between -180 and 180, then clamp it in the given range.

Parameters

angle	Input angle.
min	Min clamp value (applied after normalize).
max	Max Clamp value (applied after normalize).

Returns

Angle between min and max.

6.17.2.9 ClampInnerAngles()

```
static Vector3 Infohazard.Core.MathUtility.ClampInnerAngles (
            Vector3 angles,
            Vector3 min,
            Vector3 max ) [static]
```

Normalize a set of euler angles to values between -180 and 180, then clamp them in the given ranges.

Parameters

angles	Input angles.
min	Min clamp values (applied after normalize).
max	Max Clamp values (applied after normalize).

Returns

Angles between min and max.

6.17.2.10 Divide() [1/2]

```
static Vector3 Infohazard.Core.MathUtility.Divide (
            float left,
            Vector3 right ) [static]
```

Divide a float by each component of a vector.

6.17.2.11 Divide() [2/2]

```
static Vector3 Infohazard.Core.MathUtility.Divide (
            Vector3 left,
            Vector3 right ) [static]
```

Divide the components of left by the components of right.

6.17.2.12 DoesSegmentIntersectTriangle()

```
static bool Infohazard.Core.MathUtility.DoesSegment↩
IntersectTriangle (
            Vector3 v1,
            Vector3 v2,
            Vector3 v3,
            Vector3 s1,
            Vector3 s2,
            out float t )  [static]
```

Raycast a line segment against a triangle, and return whether they intersect.

Parameters

v1	The first triangle point.
v2	The second triangle point.
v3	The third triangle point.
s1	The start of the segment.
s2	The end of the segment.
t	The point along the input segment where it intersects the triangle, or -1.

Returns

Whether the segment intersects the triangle.

6.17.2.13 Dot()

```
static int Infohazard.Core.MathUtility.Dot (
            Vector3Int v1,
            Vector3Int v2 )  [static]
```

Dot product of two int vectors.

6.17.2.14 GetCorners()

```
static void Infohazard.Core.MathUtility.GetCorners (
            this Bounds bounds,
            Vector3[] corners )  [static]
```

Get the eight corners of a bounding box and save them in the given array.

You can use BoundsCornerArray to avoid allocating here.

Parameters

bounds	The input bounds.
corners	Array to save the values in.

6.17.2.15 GetNearestPointOnLine()

```
static float Infohazard.Core.MathUtility.GetNearestPointOn↩
Line (
            Ray line,
            Vector3 p )  [static]
```

Get the point on a line where it is nearest to a position.

Parameters

line	The input line.
p	The input position.

Returns

THe point along the line where it is nearest to the position.

6.17.2.16 GetNearestPointOnLines()

```
static bool Infohazard.Core.MathUtility.GetNearestPointOn↩
Lines (
            Ray line1,
            Ray line2,
            out float t1,
            out float t2 )  [static]
```

Find the point along each line where the lines come closest to each other.

If the lines are parallel, then return false.

Parameters

line1	The first line.
line2	The second line.
t1	The point along the first line where they are closest to intersecting.
t2	The point along the second line where they are closest to intersecting.

Returns

False if the lines are parallel, true otherwise.

6.17.2.17 GetNearestPointOnSegment()

```
static bool Infohazard.Core.MathUtility.GetNearestPoint↩
OnSegment (
            Vector3 v1,
            Vector3 v2,
            Vector3 point,
            out Vector3 pointOnSegment )  [static]
```

Find the point on a bounded line segment where it is nearest to a position, and return whether that point is in the segment's bounds.

Does not return points on the ends of the segment. If the nearest point on the segment's line is outside the segment, will fail and not return a valid point.

Parameters

v1	The start of the segment.
v2	The end of the segment.
point	The point to search for.
pointOnSegment	The point on the segment closest to the input point.

Returns

Whether the nearest point is within the segment's bounds.

6.17.2.18 GetNearestPointOnTriangle()

```
static bool Infohazard.Core.MathUtility.GetNearestPoint↵
OnTriangle (
            Vector3 v1,
            Vector3 v2,
            Vector3 v3,
            Vector3 point,
            out Vector3 pointOnTriangle ) [static]
```

Find the point on a triangle where it is nearest to a position, and return whether that point is in the triangle's bounds.

Does not return points on the edge of the triangle. If the nearest point on the triangle's plane is outside the triangle, will fail and not return a valid point.

Parameters

v1	The first triangle point.
v2	The second triangle point.
v3	The third triangle point.
point	The point to search for.
pointOnTriangle	The point on the triangle closest to the input point.

Returns

Whether the nearest point is within the triangle's bounds.

6.17.2.19 GetNearestPointOnTriangleIncludingBounds()

```
static Vector3 Infohazard.Core.Math↵
Utility.GetNearestPointOnTriangleIncludingBounds (
            Vector3 v1,
            Vector3 v2,
            Vector3 v3,
            Vector3 point ) [static]
```

Find the point on a triangle (including its bounds) where it is nearest to a position.

If nearest point is on the triangle's bounds, that point will be returned, unlike GetNearestPointOnTriangle.

Parameters

v1	The first triangle point.
v2	The second triangle point.
v3	The third triangle point.
point	The point to search for.

Returns

The nearest point on the triangle to the given point.

6.17.2.20 GetPerpendicularVector()

```
static Vector3 Infohazard.Core.MathUtility.GetPerpendicular↩
Vector (
            this Vector3 vector )  [static]
```

Returns a vector that is perpendicular to the given vector.

Parameters

vector	Input vector.

Returns

A perpendicular vector.

6.17.2.21 IsPointInsideBound()

```
static bool Infohazard.Core.MathUtility.IsPointInsideBound (
            Vector3 v1,
            Vector3 v2,
            Vector3 normal,
            Vector3 point )  [static]
```

Returns true if a given point is on the inner side (defined by a given normal) of a segment.

Parameters

v1	The start of the segment.
v2	The end of the segment.
normal	The normal, defining which side is inside.
point	The point to search for.

Returns

Whether the point is on the inner side.

6.17.2.22 Multiply() `static Vector3 Infohazard.Core.MathUtility.Multiply ( Vector3 left, Vector3 right ) [static]`

Multiply the components of left by the components of right.

6.17.2.23 NormalizeAngle() `static float Infohazard.Core.MathUtility.NormalizeAngle ( float angle ) [static]`

Normalize an angle to a value between 0 and 360.

Parameters

angle	Input angle.

Returns

Angle between 0 and 360.

6.17.2.24 NormalizeAngles() `static Vector3 Infohazard.Core.MathUtility.NormalizeAngles ( Vector3 angles ) [static]`

Normalize a set of euler angles to values between 0 and 360.

Parameters

angles	Input angles.

Returns

Angles between 0 and 360.

6.17.2.25 NormalizeInnerAngle() `static float Infohazard.Core.MathUtility.NormalizeInnerAngle ( float angle ) [static]`

Normalize an angle to a value between -180 and 180.

Parameters

angle	Input angle.

Returns

Angle between -180 and 180.

6.17.2.26 NormalizeInnerAngles()

```
static Vector3 Infohazard.Core.MathUtility.NormalizeInner←↩
Angles (
          Vector3 angles )  [static]
```

Normalize a set of euler angles to values between -180 and 180.

Parameters

angles	Input angles.

Returns

Angles between -180 and 180.

6.17.2.27 Reciprocal()

```
static Vector3 Infohazard.Core.MathUtility.Reciprocal (
          Vector3 vector )  [static]
```

Take the reciprocal of each component of a vector.

6.17.2.28 RoundToNearest() [1/2]

```
static float Infohazard.Core.MathUtility.RoundToNearest (
          float value,
          float factor )  [static]
```

Round a value to the nearest multiple of a given factor.

Parameters

value	Input value.
factor	Value to round to a multiple of.

Returns

Rounded value.

6.17.2.29 RoundToNearest() [2/2]

```
static Vector3 Infohazard.Core.MathUtility.RoundToNearest (
          Vector3 vector,
          float factor )  [static]
```

Round a each component of a vector to the nearest multiple of a given factor.

Parameters

vector	Input values.
factor	Value to round to a multiple of.

Returns

Rounded values.

6.17.2.30 SignZero() `static float Infohazard.Core.MathUtility.SignZero ( float value ) [static]`

Same as Mathf.Sign, except that if the input is zero, it returns zero.

Parameters

value	A number to get the sign of.

Returns

1 if the number is positive, -1 if the number is negative, 0 if the number is 0.

6.17.2.31 SplitHorizontal() [1/2]

```
static void Infohazard.Core.MathUtility.SplitHorizontal (
        Rect rect,
        float gap,
        out Rect out1,
        out Rect out2,
        float div = 0.5f ) [static]
```

Split a rect into two halves horizontally, with given gap between the halves.

Parameters

rect	Rect to split.
gap	Gap between the split halves.
out1	Output rect 1.
out2	Output rect 2.
div	The ratio of the total space taken up by the left rect.

6.17.2.32 SplitHorizontal() [2/2]

```
static void Infohazard.Core.MathUtility.SplitHorizontal (
        Rect rect,
             float gap,
             out Rect out1,
             out Rect out2,
             out Rect out3,
             float div1 = 1.0f / 3.0f,
             float div2 = 2.0f / 3.0f )  [static]
```

Split a rect into three thirds horizontally, with given gap between the thirds.

Parameters

rect	Rect to split.
gap	Gap between the split halves.
out1	Output rect 1.
out2	Output rect 2.
out3	Output rect 3.
div1	The ratio of the total space taken up by the left rect.
div2	The ratio of the total space taken up by the left and center rect.

6.17.2.33 ToV4() `static Vector4 Infohazard.Core.MathUtility.ToV4 (`

```
             this Vector3 vector )  [static]
```

Get a Vector4 with the components (x, y, z, 0).

6.17.2.34 ToV4Pos() `static Vector4 Infohazard.Core.MathUtility.ToV4Pos (`

```
             this Vector3 vector )  [static]
```

Get a Vector4 with the components (x, y, z, 1).

6.17.2.35 ToXY() `static Vector2 Infohazard.Core.MathUtility.ToXY (`

```
             this Vector3 v )  [static]
```

Get a Vector2 with the components (x, y).

6.17.2.36 ToXZ() `static Vector2 Infohazard.Core.MathUtility.ToXZ (`

```
             this Vector3 v )  [static]
```

Get a Vector2 with the components (x, z).

6.17.2.37 ToYX() `static Vector2 Infohazard.Core.MathUtility.ToYX ( this Vector3 v ) [static]`

Get a Vector2 with the components (y, x).

6.17.2.38 ToYZ() `static Vector2 Infohazard.Core.MathUtility.ToYZ ( this Vector3 v ) [static]`

Get a Vector2 with the components (y, z).

6.17.2.39 ToZX() `static Vector2 Infohazard.Core.MathUtility.ToZX ( this Vector3 v ) [static]`

Get a Vector2 with the components (z, x).

6.17.2.40 ToZY() `static Vector2 Infohazard.Core.MathUtility.ToZY ( this Vector3 v ) [static]`

Get a Vector2 with the components (z, y).

6.17.2.41 WithX() [1/2] `static Vector2 Infohazard.Core.MathUtility.WithX ( this Vector2 v, float x ) [static]`

Replace the X component of a vector.

6.17.2.42 WithX() [2/2] `static Vector3 Infohazard.Core.MathUtility.WithX ( this Vector3 v, float x ) [static]`

Replace the X component of a vector.

6.17.2.43 WithY() [1/2] `static Vector2 Infohazard.Core.MathUtility.WithY ( this Vector2 v, float y ) [static]`

Replace the Y component of a vector.

6.17.2.44 WithY() [2/2]

```
static Vector3 Infohazard.Core.MathUtility.WithY (
    this Vector3 v,
    float y ) [static]
```

Replace the Y component of a vector.

6.17.2.45 WithZ() [1/2]

```
static Vector3 Infohazard.Core.MathUtility.WithZ (
    this Vector2 v,
    float z ) [static]
```

Convert a Vector2 to a Vector3 with the given Z.

6.17.2.46 WithZ() [2/2]

```
static Vector3 Infohazard.Core.MathUtility.WithZ (
    this Vector3 v,
    float z ) [static]
```

Replace the Z component of a vector.

6.17.2.47 WorldToCanvasPoint()

```
static Vector3 Infohazard.Core.MathUtility.WorldToCanvasPoint (
    this Camera camera,
    Canvas canvas,
    Vector3 point ) [static]
```

Projects a point in the world onto a canvas in camera or overlay space.

Similar to Camera.WorldToScreenPoint, but scaled to the size of the canvas and its viewport. Logs an error if the canvas is in world space, as that is not supported.

Parameters

camera	The camera to use for reference.
canvas	The canvas to use for reference.
point	The world point to find on the canvas.

Returns

The point on the canvas, usable as an anchoredPosition.

6.17.2.48 XYRotation()

```
static Quaternion Infohazard.Core.MathUtility.XYRotation (
    Vector3 right,
    Vector3 upHint ) [static]
```

Get a quaternion based on a right vector and approximate up vector.

6.17.2.49 XZRotation() `static Quaternion Infohazard.Core.MathUtility.XZRotation (`

```
Vector3 right,
Vector3 forwardHint )  [static]
```

Get a quaternion based on a right vector and approximate forward vector.

6.17.2.50 YXRotation() `static Quaternion Infohazard.Core.MathUtility.YXRotation (`

```
Vector3 up,
Vector3 rightHint )  [static]
```

Get a quaternion based on a up vector and approximate right vector.

6.17.2.51 YZRotation() `static Quaternion Infohazard.Core.MathUtility.YZRotation (`

```
Vector3 up,
Vector3 forwardHint )  [static]
```

Get a quaternion based on a up vector and approximate forward vector.

6.17.2.52 ZXRotation() `static Quaternion Infohazard.Core.MathUtility.ZXRotation (`

```
Vector3 forward,
Vector3 rightHint )  [static]
```

Get a quaternion based on a forward vector and approximate right vector.

6.17.2.53 ZYRotation() `static Quaternion Infohazard.Core.MathUtility.ZYRotation (`

```
Vector3 forward,
Vector3 upHint )  [static]
```

Get a quaternion based on a forward vector and approximate up vector.

6.17.3 Member Data Documentation

6.17.3.1 BoundsCornerArray `readonly Vector3 [] Infohazard.Core.MathUtility.BoundsCornerArray = new Vector3[8]  [static]`

A static array that can be used to store the output of GetCorners, as long as the values are copied from the array right away.

6.17.3.2 Complex `static Infohazard.Core.MathUtility.Complex  [static]`

Evaluate all cubic roots of this Complex.

Parameters

complex	The number to get the cube roots of.

Returns

All three complex cube roots.

6.17.3.3 r1 `static Complex Infohazard.Core.MathUtility.r1 [static]`

Solve a quadratic equation (find x such that the result is zero) in the form ax^2 + bx + c = 0.

Solve a quartic equation (find x such that the result is zero) of the form ax^4 + bx^3 + cx^2 + dx + e = 0.

Solve a cubic equation (find x such that the result is zero) in the form ax^3 + bx^2 + cx + d = 0.

Parameters

a	The coefficient for the x^2 term.
b	The coefficient for the x term.
c	The constant term.

Returns

The two roots of the quadratic equation, which may be complex.

Parameters

a	The coefficient for the x^3 term.
b	The coefficient for the x^2 term.
c	The coefficient for the x term.
d	The constant term.

Returns

The three roots of the cubic, which may be complex.

Parameters

a	The coefficient for the x^4 term.
b	The coefficient for the x^3 term.
c	The coefficient for the x^2 term.
d	The coefficient for the x term.
e	The constant term.

Returns

The four roots of the quartic, which may be complex.

The documentation for this class was generated from the following file:

- Runtime/Utility/MathUtility.cs

6.18 Infohazard.Core.MustImplementAttribute Class Reference

Applied to object reference fields to ensure that an assigned Object must implement one or more interfaces.

Public Member Functions

- MustImplementAttribute (params Type[ ] types)
  Construct a new MustImplementAttribute.

Properties

- IReadOnlyList< Type > RequiredTypes `[get]`
  Interfaces that must be implemented by the assigned Object.

6.18.1 Detailed Description

Applied to object reference fields to ensure that an assigned Object must implement one or more interfaces.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 MustImplementAttribute() `Infohazard.Core.MustImplementAttribute.MustImplementAttribute (`

```
            params Type[] types )
```

Construct a new MustImplementAttribute.

Parameters

types	Interfaces that must be implemented by the assigned Object.

6.18.3 Property Documentation

6.18.3.1 RequiredTypes `IReadOnlyList<Type> Infohazard.Core.MustImplementAttribute.Required←↩ Types [get]`

Interfaces that must be implemented by the assigned Object.

The documentation for this class was generated from the following file:

- Runtime/Attributes/MustImplementAttribute.cs

6.19 Infohazard.Core.Editor.ObjectReferenceFieldAssignmentValidator Class Reference

IObjectReferenceFieldAssignmentValidator that uses Unity validation internally, and allows adding additional validation with a simplified interface.

Public Member Functions

- ObjectReferenceFieldAssignmentValidator (Func< Object, Object > validator=null)

  Construct with a given validation function.
- virtual Object ValidateObjectFieldAssignment (Object[ ] references, Type objType, SerializedProperty property, int options)

  Validate the input objects for assignment to a field of the given type, and return a valid one.

Parameters

references	*All objects attempting to be assigned.*
objType	*The type of the field.*
property	*The property that references the field.*
options	*Additional options (used internally by Unity).*

Returns

The object to assign to the field, or null.

6.19.1 Detailed Description

IObjectReferenceFieldAssignmentValidator that uses Unity validation internally, and allows adding additional validation with a simplified interface.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 ObjectReferenceFieldAssignmentValidator() `Infohazard.Core.Editor.ObjectReferenceField←↩ AssignmentValidator.ObjectReferenceFieldAssignmentValidator (`

```
            Func< Object, Object > validator = null )
```

Construct with a given validation function.

Parameters

validator	Simplified validation function that validates a single object.

6.19.3 Member Function Documentation

6.19.3.1 ValidateObjectFieldAssignment()

```
virtual Object Infohazard.Core.Editor.ObjectReference←
FieldAssignmentValidator.ValidateObjectFieldAssignment (
            Object[] references,
            Type objType,
            SerializedProperty property,
            int options ) [virtual]
```

Validate the input objects for assignment to a field of the given type, and return a valid one.

Parameters

references	All objects attempting to be assigned.
objType	The type of the field.
property	The property that references the field.
options	Additional options (used internally by Unity).

Returns

The object to assign to the field, or null.

Implements Infohazard.Core.Editor.IObjectReferenceFieldAssignmentValidator.

The documentation for this class was generated from the following file:

- Editor/Utility/ObjectReferenceFieldAssignmentValidator.cs

6.20 Infohazard.Core.Editor.ObjectValidators Class Reference

Provides methods for validating objects.

Static Public Member Functions

- static Func< Object, Object > MustImplement (IReadOnlyList< Type > interfaces)
  Ensures an object (or one of its components, if it's a GameObject) implements a list of interfaces.

6.20.1 Detailed Description

Provides methods for validating objects.

6.20.2 Member Function Documentation

6.20.2.1 MustImplement()

```
static Func< Object, Object > Infohazard.Core.Editor.ObjectValidators.↩
MustImplement (
            IReadOnlyList< Type > interfaces )  [static]
```

Ensures an object (or one of its components, if it's a GameObject) implements a list of interfaces.

If the object is a GameObject and one of its components implements all interfaces, that component will be returned.

Parameters

interfaces	List of interfaces that must be implemented.

Returns

The object or one of its components that implements all interfaces, or null.

The documentation for this class was generated from the following file:

- Editor/Utility/ObjectValidators.cs

6.21 Infohazard.Core.PassiveTimer Struct Reference

A lightweight timer that does not need to be updated each frame.

Public Types

- enum TimeMode

  The various modes available for timers.

Public Member Functions

- PassiveTimer (float interval, TimeMode mode=TimeMode.Scaled, bool initialize=true)

  Construct a PassiveTimer with the given interval, which will be both the initial and repeat interval.
- PassiveTimer (float initialInterval, float interval, TimeMode mode=TimeMode.Scaled, bool initialize=true)

  Construct a PassiveTimer with the given interval.
- void Initialize ()

  Initialize the timer.
- bool TryConsume ()

  If the current interval has ended, reset the interval and return true, else return false without reset.
- void StartInterval ()

  Restart the interval, so that the timer starts counting down from its repeat interval.
- void EndInterval ()

  End the interval, so the timer is in the expired state.

Properties

- float InitialInterval `[get, set]`

  Initial interval to set the timer for in seconds.
- float Interval `[get, set]`

  The repeat interval for the in seconds.
- TimeMode Mode `[get, set]`

  What value for time that the timer uses (scaled, unscaled, or realtime).
- bool IsIntervalEnded `[get]`

  Whether the timer is in the expired state, meaning the current interval has elapsed.
- float IntervalStartTime `[get, set]`

  The start time for the current interval.
- float IntervalEndTime `[get]`

  The time at which the current interval will end (or has ended).
- float TimeSinceIntervalEnded `[get]`

  Time that has passed since the current interval ended (or, if not ended, a negative value).
- bool IsInitialized `[get, private set]`

  Whether the timer is initialized.
- bool HasIntervalStarted `[get, private set]`

  Whether an interval has started yet.
- float TimeUntilIntervalEnd `[get, set]`

  The time in seconds until the current interval ends.
- float RatioUntilIntervalEnd `[get]`

  A ratio going from one at interval start to zero at interval end.
- float TimeSinceIntervalStart `[get, set]`

  The time in seconds since the current interval started.
- float RatioSinceIntervalStart `[get]`

  A ratio going from zero at interval start to one at interval end.
- bool DidIntervalEndThisFrame `[get]`

  Whether the current interval ended during the current frame.
- float CurrentTimeWithoutPause `[get]`

  The current time read from Unity, based on the Mode.
- float CurrentTime `[get]`

  The current time read from Unity, taking into account time that the PassiveTimer has spent paused.
- float DeltaTime `[get]`

  The delta time read from Unity, based on the Mode.
- float PausedTime `[get, private set]`

  The time that the PassiveTimer has spent in a paused state.
- bool IsPaused `[get, set]`

  Get or set whether the PassiveTimer is paused.

6.21.1 Detailed Description

A lightweight timer that does not need to be updated each frame.

Can be serialized directly in the inspector or created in code. If it is assigned in the inspector, you must call Initialize in Start/Awake/OnEnable/OnSpawned.

A PassiveTimer can be in one of four states:

- It has not yet been initialized (uninitialized state).
- An interval is active and counting down (counting state).
- The timer is paused (paused state).
- An interval has expired (expired state).

6.21.2 Member Enumeration Documentation

6.21.2.1 TimeMode `enum Infohazard.Core.PassiveTimer.TimeMode`

The various modes available for timers.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 PassiveTimer() [1/2]

```
Infohazard.Core.PassiveTimer.PassiveTimer (
        float interval,
        TimeMode mode = TimeMode.Scaled,
        bool initialize = true )
```

Construct a PassiveTimer with the given interval, which will be both the initial and repeat interval.

Parameters

interval	The initial and repeat interval.
mode	Time mode to use.
initialize	Whether to initialize the timer and start counting immediately.

6.21.3.2 PassiveTimer() [2/2]

```
Infohazard.Core.PassiveTimer.PassiveTimer (
        float initialInterval,
        float interval,
        TimeMode mode = TimeMode.Scaled,
        bool initialize = true )
```

Construct a PassiveTimer with the given interval.

Parameters

initialInterval	The initial interval.
interval	The repeat interval.
mode	Time mode to use.
initialize	Whether to initialize the timer and start counting immediately.

6.21.4 Member Function Documentation

6.21.4.1 EndInterval() `void Infohazard.Core.PassiveTimer.EndInterval ( )`

End the interval, so the timer is in the expired state.

6.21.4.2 Initialize() `void Infohazard.Core.PassiveTimer.Initialize ( )`

Initialize the timer.

You must call this when your script initializes if the timer was assigned in the inspector.

6.21.4.3 StartInterval() `void Infohazard.Core.PassiveTimer.StartInterval ( )`

Restart the interval, so that the timer starts counting down from its repeat interval.

6.21.4.4 TryConsume() `bool Infohazard.Core.PassiveTimer.TryConsume ( )`

If the current interval has ended, reset the interval and return true, else return false without reset.

This is useful to create ability cooldowns or weapon fire rates. See the following example:

```
if (AbilityButtonPressed() && AbilityTimer.TryConsume()) {
    UseAbility();
}
```

Returns

Whether the interval was ended and has been reset.

6.21.5 Property Documentation

6.21.5.1 CurrentTime `float Infohazard.Core.PassiveTimer.CurrentTime [get]`

The current time read from Unity, taking into account time that the PassiveTimer has spent paused.

6.21.5.2 CurrentTimeWithoutPause `float Infohazard.Core.PassiveTimer.CurrentTimeWithoutPause [get]`

The current time read from Unity, based on the Mode.

Exceptions

ArgumentOutOfRangeException	If Mode is invalid.

6.21.5.3 DeltaTime `float Infohazard.Core.PassiveTimer.DeltaTime [get]`

The delta time read from Unity, based on the Mode.

Exceptions

ArgumentOutOfRangeException	If Mode is invalid or realtime.

6.21.5.4 DidIntervalEndThisFrame `bool Infohazard.Core.PassiveTimer.DidIntervalEndThisFrame [get]`

Whether the current interval ended during the current frame.

This can be used to create actions that happen only once, the moment a timer expires.

6.21.5.5 HasIntervalStarted `bool Infohazard.Core.PassiveTimer.HasIntervalStarted [get], [private set]`

Whether an interval has started yet.

6.21.5.6 InitialInterval `float Infohazard.Core.PassiveTimer.InitialInterval [get], [set]`

Initial interval to set the timer for in seconds.

This interval begins when Initialize is called, or when the timer is created from a non-default constructor.

6.21.5.7 Interval `float Infohazard.Core.PassiveTimer.Interval [get], [set]`

The repeat interval for the in seconds.

This interval begins when StartInterval or TryConsume is used.

6.21.5.8 IntervalEndTime `float Infohazard.Core.PassiveTimer.IntervalEndTime [get]`

The time at which the current interval will end (or has ended).

6.21.5.9 IntervalStartTime `float Infohazard.Core.PassiveTimer.IntervalStartTime [get], [set]`

The start time for the current interval.

6.21.5.10 IsInitialized `bool Infohazard.Core.PassiveTimer.IsInitialized [get], [private set]`

Whether the timer is initialized.

6.21.5.11 IsIntervalEnded `bool Infohazard.Core.PassiveTimer.IsIntervalEnded [get]`

Whether the timer is in the expired state, meaning the current interval has elapsed.

6.21.5.12 IsPaused `bool Infohazard.Core.PassiveTimer.IsPaused [get], [set]`

Get or set whether the PassiveTimer is paused.

It is not necessary to pause timers to account for the game pausing, as long as they are using realtime. This allows an individual timer to be paused separately from the rest of the game.

6.21.5.13 Mode `TimeMode Infohazard.Core.PassiveTimer.Mode [get], [set]`

What value for time that the timer uses (scaled, unscaled, or realtime).

6.21.5.14 PausedTime `float Infohazard.Core.PassiveTimer.PausedTime [get], [private set]`

The time that the PassiveTimer has spent in a paused state.

6.21.5.15 RatioSinceIntervalStart `float Infohazard.Core.PassiveTimer.RatioSinceIntervalStart [get]`

A ratio going from zero at interval start to one at interval end.

6.21.5.16 RatioUntillIntervalEnd `float Infohazard.Core.PassiveTimer.RatioUntilIntervalEnd [get]`

A ratio going from one at interval start to zero at interval end.

6.21.5.17 TimeSinceIntervalEnded `float Infohazard.Core.PassiveTimer.TimeSinceIntervalEnded [get]`

Time that has passed since the current interval ended (or, if not ended, a negative value).

6.21.5.18 TimeSinceIntervalStart `float Infohazard.Core.PassiveTimer.TimeSinceIntervalStart [get], [set]`

The time in seconds since the current interval started.

6.21.5.19 TimeUntilIntervalEnd `float Infohazard.Core.PassiveTimer.TimeUntilIntervalEnd [get], [set]`

The time in seconds until the current interval ends.

The documentation for this struct was generated from the following file:

- Runtime/Timing/PassiveTimer.cs

6.22 Infohazard.Core.Pause Class Reference

Manages pausing and unpausing of the game.

Properties

- static float TimeScale `[get, set]`

  Get or set the non-paused timescale. Only affects current Time.timeScale if not paused.
- static bool Paused `[get, set]`

  Controls paused state of the game.

Events

- static Action GamePaused

  Invoked when the game pauses.
- static Action GameResumed

  Invoked when the game un-pauses.

6.22.1 Detailed Description

Manages pausing and unpausing of the game.

Any actions that should only happen when the game is not paused should check Pause.paused. Can be used statically if pause is controlled elsewhere, or placed as a component to pause the game from a UnityEvent. The game will automatically unpause when a new scene is loaded.

6.22.2 Property Documentation

6.22.2.1 Paused `bool Infohazard.Core.Pause.Paused [static], [get], [set]`

Controls paused state of the game.

This cannot completely prevent game actions from happening, but it does sets Time.timeScale to 0 so that Physics and animation will stop.

6.22.2.2 TimeScale `float Infohazard.Core.Pause.TimeScale [static], [get], [set]`

Get or set the non-paused timescale. Only affects current Time.timeScale if not paused.

6.22.3 Event Documentation

6.22.3.1 GamePaused `Action Infohazard.Core.Pause.GamePaused [static]`

Invoked when the game pauses.

6.22.3.2 GameResumed `Action Infohazard.Core.Pause.GameResumed [static]`

Invoked when the game un-pauses.

The documentation for this class was generated from the following file:

- Runtime/Timing/Pause.cs

6.23 Infohazard.Core.Pool< T > Class Template Reference

Provides a simple pool with an interface similar to the official Unity pool added in 2021.

Public Member Functions

- Pool (Func< T > createFunc, Action< T > getAction=null, Action< T > releaseAction=null, Action< T > destroyAction=null, int maxCount=0)

  Create a new ObjectPool.
- T Get ()

  Retrieve an item from the pool, creating a new one if necessary.
- void Release (T item)

  Return an object to the pool, destroying it if over max count.
- void Clear ()

  Destroy all objects in the pool.
- void Remove (T item)

  Remove an item from the pool without cleaning it up.
- void Dispose ()

  Destroy all objects in the pool.

Properties

- Func< T > CreateFunc `[get, set]`

  Function invoked to create an instance, which must not be null.
- Action< T > GetAction `[get, set]`

  Callback invoked when an object is retrieved from the pool.
- Action< T > ReleaseAction `[get, set]`

  Callback invoked when an object is returned to the pool.
- Action< T > DestroyAction `[get, set]`

  Callback invoked when an object is destroyed in the pool.
- int MaxCount = 0 `[get, set]`

  Max objects in pool, or 0 for no limit.

6.23.1 Detailed Description

Provides a simple pool with an interface similar to the official Unity pool added in 2021.

Template Parameters

T	Type of pooled object.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 Pool()

```
Infohazard.Core.Pool< T >.Pool (
        Func< T > createFunc,
        Action< T > getAction = null,
        Action< T > releaseAction = null,
        Action< T > destroyAction = null,
        int maxCount = 0 )
```

Create a new ObjectPool.

Parameters

createFunc	Function invoked to create an instance, which must not be null.
getAction	Callback invoked when an object is retrieved from the pool.
releaseAction	Callback invoked when an object is returned to the pool.
destroyAction	Callback invoked when an object is destroyed in the pool.
maxCount	Max objects in pool, or 0 for no limit.

6.23.3 Member Function Documentation

6.23.3.1 Clear() `void Infohazard.Core.Pool< T >.Clear ( )`

Destroy all objects in the pool.

6.23.3.2 Dispose() `void Infohazard.Core.Pool< T >.Dispose ( )`

Destroy all objects in the pool.

6.23.3.3 Get() `T Infohazard.Core.Pool< T >.Get ( )`

Retrieve an item from the pool, creating a new one if necessary.

Returns

The retrieved object.

6.23.3.4 Release() `void Infohazard.Core.Pool< T >.Release ( T item )`

Return an object to the pool, destroying it if over max count.

Parameters

item	The object to release.

6.23.3.5 Remove() `void Infohazard.Core.Pool< T >.Remove ( T item )`

Remove an item from the pool without cleaning it up.

Parameters

item	Item to remove.

6.23.4 Property Documentation

6.23.4.1 CreateFunc `Func<T> Infohazard.Core.Pool< T >.CreateFunc [get], [set]`

Function invoked to create an instance, which must not be null.

6.23.4.2 DestroyAction `Action<T> Infohazard.Core.Pool< T >.DestroyAction [get], [set]`

Callback invoked when an object is destroyed in the pool.

6.23.4.3 GetAction `Action<T> Infohazard.Core.Pool< T >.GetAction [get], [set]`

Callback invoked when an object is retrieved from the pool.

6.23.4.4 MaxCount `int Infohazard.Core.Pool< T >.MaxCount = 0 [get], [set]`

Max objects in pool, or 0 for no limit.

6.23.4.5 ReleaseAction `Action<T> Infohazard.Core.Pool< T >.ReleaseAction [get], [set]`

Callback invoked when an object is returned to the pool.

The documentation for this class was generated from the following file:

- Runtime/Pooling/Pool.cs

6.24 Infohazard.Core.PooledParticleEffect Class Reference

A component that can be attached to ParticleSystem GameObjects to make them work correctly with pooling.

6.24.1 Detailed Description

A component that can be attached to ParticleSystem GameObjects to make them work correctly with pooling.

This script enables ParticleSystems to reset and play when they are spawned using the pooling system, and to optionally despawn themselves when they complete. This script must be placed on a GameObject with a Particle↩ System component, and there must be a Spawnable component in this object or a parent. In order for despawning to work, the ParticleSystem must have its Stop Action set to Callback. If there are multiple ParticleSystems in a prefab, only the root one should have _despawnOnDone set to true.

The documentation for this class was generated from the following file:

- Runtime/Pooling/PooledParticleEffect.cs

6.25 Infohazard.Core.PooledTrail Class Reference

A component that can be attached to TrailRenderer GameObjects to make them work correctly with pooling.

6.25.1 Detailed Description

A component that can be attached to TrailRenderer GameObjects to make them work correctly with pooling.

This script enables TrailRenderers to reset when they are spawned using the pooling system. This script must be placed on a GameObject with a TrailRenderer component.

The documentation for this class was generated from the following file:

- Runtime/Pooling/PooledTrail.cs

6.26 Infohazard.Core.PoolManager Class Reference

The singleton manager class that handles object pooling.

Public Member Functions

- Spawnable SpawnPrefab (Spawnable prefab, in SpawnParams spawnParams=default)

  Spawn an instance of the specified prefab, creating a DefaultPoolHandler if one does not already exist.
- Spawnable SpawnFromKey (object key, in SpawnParams spawnParams=default)

  Spawn an instance with the specified key. If no IPoolHandler is registered for key, an error is logged and null is returned.
- bool HasPoolHandler (object key)

  Return whether a IPoolHandler has been registered with the given key.
- bool TryGetPoolHandler (object key, out IPoolHandler handler)

  Get the IPoolHandler for the given key, if one is registered (return false otherwise).
- void AddPoolHandler (object key, IPoolHandler handler)

  Register a new IPoolHandler for the given key.
- void DespawnInstance (Spawnable instance)

  Despawn the given instance, returning it back to its pool if it has one.
- void ClearInactiveObjects ()

  Destroy and cleanup any inactive instances for DefaultPoolHandlers created by the PoolManager.

Properties

- static PoolManager Instance `[get]`

  The singleton PoolManager instance.
- Transform PoolTransform `[get]`

  Transform that inactive instances will be parented to.

6.26.1 Detailed Description

The singleton manager class that handles object pooling.

There should only ever be on PoolManager at a time, but this can be created either manually or automatically when needed. You can have one PoolManager per scene, or have a shared one across all scenes. The only time you'll typically need to interact with the PoolManager is to call ClearInactiveObjects when loading a new level, if the previously pooled objects are no longer necessary. All normal spawning and despawning of instances should be done through the Spawnable class.

6.26.2 Member Function Documentation

6.26.2.1 AddPoolHandler()

```
void Infohazard.Core.PoolManager.AddPoolHandler (
            object key,
            IPoolHandler handler )
```

Register a new IPoolHandler for the given key.

The registered IPoolHandler will NOT be retained by the PoolManager.

Parameters

key	Key used to spawn objects using handler.
handler	IPoolHandler used to spawn objects for key.

6.26.2.2 ClearInactiveObjects()

```
void Infohazard.Core.PoolManager.ClearInactiveObjects ( )
```

Destroy and cleanup any inactive instances for DefaultPoolHandlers created by the PoolManager.

IPoolHandlers created manually and added using AddPoolHandler will NOT be affected. This is meant to be called when loading a new level, to ensure pooled instances to not stay around forever.

6.26.2.3 DespawnInstance()

```
void Infohazard.Core.PoolManager.DespawnInstance (
            Spawnable instance )
```

Despawn the given instance, returning it back to its pool if it has one.

Parameters

instance	

6.26.2.4 HasPoolHandler()

```
bool Infohazard.Core.PoolManager.HasPoolHandler (
            object key )
```

Return whether a IPoolHandler has been registered with the given key.

Parameters

key	Key of the IPoolHandler, such as the prefab itself.

Returns

Whether there is an IPoolHandler for key.

6.26.2.5 SpawnFromKey()

```
Spawnable Infohazard.Core.PoolManager.SpawnFromKey (
        object key,
        in SpawnParams spawnParams = default )
```

Spawn an instance with the specified key. If no IPoolHandler is registered for key, an error is logged and null is returned.

The key may be a prefab that was previously registered using SpawnPrefab, or another object for which an IPool↩ Handler was manually registered using AddPoolHandler.

Parameters

key	Key of the IPoolHandler, such as the prefab itself.
spawnParams	Additional spawn info.

Returns

The spawned instance, or null if no handler found.

6.26.2.6 SpawnPrefab()

```
Spawnable Infohazard.Core.PoolManager.SpawnPrefab (
        Spawnable prefab,
        in SpawnParams spawnParams = default )
```

Spawn an instance of the specified prefab, creating a DefaultPoolHandler if one does not already exist.

If a new DefaultPoolHandler is created, it will be retained by the PoolManager.

Parameters

prefab	The prefab to spawn.
spawnParams	Additional spawn information.

Returns

The spawned instance.

6.26.2.7 TryGetPoolHandler() `bool Infohazard.Core.PoolManager.TryGetPoolHandler ( object key, out IPoolHandler handler )`

Get the IPoolHandler for the given key, if one is registered (return false otherwise).

Parameters

key	Key of the IPoolHandler, such as the prefab itself.
handler	The registered IPoolHandler for key, or null if none.

Returns

Whether there is an IPoolHandler for key.

6.26.3 Property Documentation

6.26.3.1 Instance `PoolManager Infohazard.Core.PoolManager.Instance [static], [get]`

The singleton PoolManager instance.

If this property is accessed when there is no active instance, one will be created automatically.

6.26.3.2 PoolTransform `Transform Infohazard.Core.PoolManager.PoolTransform [get]`

Transform that inactive instances will be parented to.

The documentation for this class was generated from the following file:

- Runtime/Pooling/PoolManager.cs

6.27 Infohazard.Core.ProgressBar Class Reference

Used to create health bars and other types of progress bars without using a Slider.

Properties

- float FillAmount `[get, set]`

  By what value to fill the bar.

6.27.1 Detailed Description

Used to create health bars and other types of progress bars without using a Slider.

It supports images that fill the bar using either the “filled” image type or by manipulating the RectTransform anchors.

6.27.2 Property Documentation

6.27.2.1 FillAmount `float Infohazard.Core.ProgressBar.FillAmount [get], [set]`

By what value to fill the bar.

A value of zero means empty, one means full.

The documentation for this class was generated from the following file:

- Runtime/Misc/ProgressBar.cs

6.28 Infohazard.Core.SpawnRefBase< T >.PropNames Class Reference

This is used to refer to the names of private fields in this class from a custom Editor.

6.28.1 Detailed Description

This is used to refer to the names of private fields in this class from a custom Editor.

The documentation for this class was generated from the following file:

- Runtime/Pooling/SpawnRef.cs

6.29 Infohazard.Core.RandomUtility Class Reference

Contains extensions to builtin randomization functionality.

Static Public Member Functions

- static long NextLong (this System.Random random, long min, long max)

  Generate a random long between min and max.
- static ulong NextUlong (this System.Random random)

  Generate a random ulong.

6.29.1 Detailed Description

Contains extensions to builtin randomization functionality.

6.29.2 Member Function Documentation

6.29.2.1 NextLong()

```
static long Infohazard.Core.RandomUtility.NextLong (
        this System.Random random,
        long min,
        long max )  [static]
```

Generate a random long between min and max.

6.29.2.2 NextUlong()

```
static ulong Infohazard.Core.RandomUtility.NextUlong (
        this System.Random random )  [static]
```

Generate a random ulong.

The documentation for this class was generated from the following file:

- Runtime/Utility/RandomUtility.cs

6.30 Infohazard.Core.SceneControl Class Reference

Provides some methods to navigate to scenes.

Public Member Functions

- void QuitButton ()

  Non-static equivalent of Quit.
- void ReloadScene ()

  Reload the current scene.
- void LoadScene (string sceneName)

  Load a scene with a given name.

Static Public Member Functions

- static void Quit ()

  If in the editor, exit play mode. Otherwise, close the application.

6.30.1 Detailed Description

Provides some methods to navigate to scenes.

Also provides a static method to quit the game that works in a standalone build as well as in the editor. This script is useful if you're building a super quick main menu (such as in the last half hour of a game jam) and need to hook up your buttons as fast as possible.

6.30.2 Member Function Documentation

6.30.2.1 LoadScene() `void Infohazard.Core.SceneControl.LoadScene ( string sceneName )`

Load a scene with a given name.

Scene will be loaded as a single (not additively).

Parameters

sceneName	The scene name to load.

6.30.2.2 Quit() `static void Infohazard.Core.SceneControl.Quit ( ) [static]`

If in the editor, exit play mode. Otherwise, close the application.

6.30.2.3 QuitButton() `void Infohazard.Core.SceneControl.QuitButton ( )`

Non-static equivalent of Quit.

6.30.2.4 ReloadScene() `void Infohazard.Core.SceneControl.ReloadScene ( )`

Reload the current scene.

Current scene is determined by `SceneManager.GetActiveScene()`, and is loaded as a single scene. This method is not very helpful if your game has multiple scenes open at a time.

The documentation for this class was generated from the following file:

- Runtime/Misc/SceneControl.cs

6.31 Infohazard.Core.Singleton< T > Class Template Reference

Base class that makes it easier to write scripts that always have exactly one instance.

Properties

- static T Instance `[get]`

  Get the singleton instance of the script.

6.31.1 Detailed Description

Base class that makes it easier to write scripts that always have exactly one instance.

You can inherit from this script in managers or other scripts. You must still place the script on a GameObject in the scene. A static Instance accessor is automatically provided, which will do a lazy search for the correct instance the first time it is used, or if the previous instance was destroyed. After that it will just return a cached instance.

Template Parameters

T	Pass the name of the inheriting class here to set the type of Instance.

Type Constraints

T* : *SingletonBase

6.31.2 Property Documentation

6.31.2.1 Instance `T Infohazard.Core.Singleton< T >.Instance [static], [get]`

Get the singleton instance of the script.

If it hasn't been found yet (or the old instance was destroyed), will do a search using `Object.Find{T}`. Otherwise it returns the cached instance. However, it will not create a new instance for you.

The documentation for this class was generated from the following file:

- Runtime/Misc/Singleton.cs

6.32 Infohazard.Core.SingletonAsset< T > Class Template Reference

Base class that makes it easier to write ScriptableObjects that always have exactly one instance in your project.

Properties

- static T Instance `[get]`

  Get the singleton instance of the script.

6.32.1 Detailed Description

Base class that makes it easier to write ScriptableObjects that always have exactly one instance in your project.

Similar to Singleton, but for ScriptableObjects. You specify a path in your subclass where the instance should live (this must be under a Resources folder) and the editor will automatically handle loading and even creating this asset for you when needed.

Template Parameters

T	Pass the name of the inheriting class here to set the type of Instance.

Type Constraints

T* : *SingletonAssetBase

6.32.2 Property Documentation

6.32.2.1 Instance `T Infohazard.Core.SingletonAsset< T >.Instance [static], [get]`

Get the singleton instance of the script.

If it hasn't been loaded yet, will try to load from SingletonAssetBase.ResourcePath. If in the editor and the asset doesn't exist, it will be created at SingletonAssetBase.ResourceFolderPath/SingletonAssetBase.ResourcePath. Otherwise, the cached instance (or null) will be returned.

The documentation for this class was generated from the following file:

- Runtime/Misc/SingletonAsset.cs

6.33 Infohazard.Core.SingletonAssetBase Class Reference

Base class of SingletonAsset<T>. For internal use only.

Properties

- abstract string ResourceFolderPath `[get]`

    Return the path at which the Resources folder containing this asset lives.
- abstract string ResourcePath `[get]`

    Returns the path of this asset relative to its Resources folder.

6.33.1 Detailed Description

Base class of SingletonAsset<T>. For internal use only.

6.33.2 Property Documentation

6.33.2.1 ResourceFolderPath `abstract string Infohazard.Core.SingletonAssetBase.Resource↩ FolderPath [get]`

Return the path at which the Resources folder containing this asset lives.

```
public override string ResourceFolderPath => "Infohazard.Core.Data/Resources";
```

6.33.2.2 ResourcePath `abstract string Infohazard.Core.SingletonAssetBase.ResourcePath [get]`

Returns the path of this asset relative to its Resources folder.

The documentation for this class was generated from the following file:

- Runtime/Misc/SingletonAsset.cs

6.34 Infohazard.Core.Spawnable Class Reference

Attach this component to a prefab to enable it to use the pooling system.

Public Member Functions

- void DespawnSelf ()
  Despawn this instance and return it to the PoolManager.

Static Public Member Functions

- static Spawnable Spawn (Spawnable prefab, Vector3? position=null, Quaternion? rotation=null, Transform parent=null, bool inWorldSpace=false, ulong persistedInstanceID=0, Scene? scene=null)
  Spawn a new pooled instance with the given properties.
- static Spawnable Spawn (Spawnable prefab, in SpawnParams spawnParams=default)
  Spawn a new pooled instance with the given properties.
- static void Despawn (Spawnable instance, float inSeconds=0.0f)
  Despawn a pooled instance, optionally after some time has passed.
- static GameObject Spawn (GameObject prefab, Vector3? position=null, Quaternion? rotation=null, Transform parent=null, bool inWorldSpace=false, ulong persistedInstanceID=0, Scene? scene=null)
  Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.
- static GameObject Spawn (GameObject prefab, in SpawnParams spawnParams=default)
  Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.
- static void Despawn (GameObject instance, float inSeconds=0.0f)
  Despawn an instance, optionally after some time has passed, using the pooling system if the prefab has a Spawnable script.
- static T Spawn< T > (T prefab, Vector3? position=null, Quaternion? rotation=null, Transform parent=null, bool inWorldSpace=false, ulong persistedInstanceID=0, Scene? scene=null)
  Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.
- static T Spawn< T > (T prefab, in SpawnParams spawnParams=default)
  Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.

Properties

- bool Pooled `[get]`

  Whether this object should be pooled.
- bool IsSpawned `[get, private set]`

  Whether or not this object is an active, spawned instance.
- bool IsDespawned `[get, private set]`

  Whether or not this object is currently an inactive instance.
- IPoolHandler PoolHandler `[get, set]`

  IPoolHandler which was used to spawn the object.

Events

- Action< Spawnable > Spawned

  Invoked when the Spawnable is spawned.
- Action< Spawnable > Despawned

  Invoke when the Spawnable is despawned.
- Action< Spawnable > Destroyed

  Invoked when the Spawnable is destroyed (not just despawned).

6.34.1 Detailed Description

Attach this component to a prefab to enable it to use the pooling system.

The static methods in this class are also the main way to spawn objects in a way that's compatible with the pooling system. When a Spawnable object is spawned, it will broadcast the `OnSpawned` message to its children. When it is despawned, it will broadcast the `OnDespawned` method.

6.34.2 Member Function Documentation

6.34.2.1 Despawn() `[1/2]` `static void Infohazard.Core.Spawnable.Despawn (`

```
GameObject instance,
float inSeconds = 0.0f )  [static]
```

Despawn an instance, optionally after some time has passed, using the pooling system if the prefab has a Spawnable script.

Parameters

instance	The instance to despawn.
inSeconds	The time to wait before despawning. If zero, despawn is synchronous.

6.34.2.2 Despawn() `[2/2]` `static void Infohazard.Core.Spawnable.Despawn (`

```
            Spawnable instance,
            float inSeconds = 0.0f )  [static]
```

Despawn a pooled instance, optionally after some time has passed.

Parameters

instance	The instance to despawn.
inSeconds	The time to wait before despawning. If zero, despawn is synchronous.

6.34.2.3 DespawnSelf() `void Infohazard.Core.Spawnable.DespawnSelf ( )`

Despawn this instance and return it to the PoolManager.

6.34.2.4 Spawn() `[1/4]` `static GameObject Infohazard.Core.Spawnable.Spawn (`

```
            GameObject prefab,
            in SpawnParams spawnParams = default )  [static]
```

Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.

Parameters

prefab	The prefab to spawn.
spawnParams	Spawn properties.

Returns

The spawned instance.

6.34.2.5 Spawn() `[2/4]` `static GameObject Infohazard.Core.Spawnable.Spawn (`

```
            GameObject prefab,
            Vector3?  position = null,
            Quaternion?  rotation = null,
            Transform parent = null,
            bool inWorldSpace = false,
            ulong persistedInstanceID = 0,
            Scene?  scene = null )  [static]
```

Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.

Parameters

prefab	The prefab to spawn.
position	The position to spawn at.

Parameters

rotation	The rotation to spawn at.
parent	The parent to spawn under.
inWorldSpace	If true, position/rotation are global, else they are local to parent.
persistedInstanceID	Existing persisted instance ID to assign.
scene	The scene to spawn in.

Returns

The spawned instance.

6.34.2.6 Spawn() [3/4]

```
static Spawnable Infohazard.Core.Spawnable.Spawn (
        Spawnable prefab,
        in SpawnParams spawnParams = default )  [static]
```

Spawn a new pooled instance with the given properties.

Parameters

prefab	The prefab to spawn.
spawnParams	Spawn properties.

Returns

The spawned instance.

6.34.2.7 Spawn() [4/4]

```
static Spawnable Infohazard.Core.Spawnable.Spawn (
        Spawnable prefab,
        Vector3? position = null,
        Quaternion? rotation = null,
        Transform parent = null,
        bool inWorldSpace = false,
        ulong persistedInstanceID = 0,
        Scene? scene = null )  [static]
```

Spawn a new pooled instance with the given properties.

Parameters

prefab	The prefab to spawn.
position	The position to spawn at.
rotation	The rotation to spawn at.
parent	The parent to spawn under.
inWorldSpace	If true, position/rotation are global, else they are local to parent.
persistedInstanceID	Existing persisted instance ID to assign.
scene	The scene to spawn in.

Returns

The spawned instance.

6.34.2.8 Spawn< T >() [1/2]

```
static T Infohazard.Core.Spawnable.Spawn< T > (
        T prefab,
        in SpawnParams spawnParams = default ) [static]
```

Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.

Parameters

prefab	The prefab to spawn.
spawnParams	Spawn properties.

Returns

The spawned instance.

Type Constraints

T* : *Component

6.34.2.9 Spawn< T >() [2/2]

```
static T Infohazard.Core.Spawnable.Spawn< T > (
        T prefab,
        Vector3? position = null,
        Quaternion? rotation = null,
        Transform parent = null,
        bool inWorldSpace = false,
        ulong persistedInstanceID = 0,
        Scene? scene = null ) [static]
```

Spawn a new instance with the given properties, using the pooling system if the prefab has a Spawnable script.

Parameters

prefab	The prefab to spawn.
position	The position to spawn at.
rotation	The rotation to spawn at.
parent	The parent to spawn under.
inWorldSpace	If true, position/rotation are global, else they are local to parent.
persistedInstanceID	Existing persisted instance ID to assign.
scene	The scene to spawn in.

Returns

The spawned instance.

Type Constraints

T* : *Component

6.34.3 Property Documentation

6.34.3.1 IsDespawned `bool Infohazard.Core.Spawnable.IsDespawned [get], [private set]`

Whether or not this object is currently an inactive instance.

6.34.3.2 IsSpawned `bool Infohazard.Core.Spawnable.IsSpawned [get], [private set]`

Whether or not this object is an active, spawned instance.

6.34.3.3 Pooled `bool Infohazard.Core.Spawnable.Pooled [get]`

Whether this object should be pooled.

6.34.3.4 PoolHandler `IPoolHandler Infohazard.Core.Spawnable.PoolHandler [get], [set], [package]`

IPoolHandler which was used to spawn the object.

6.34.4 Event Documentation

6.34.4.1 Despawned `Action<Spawnable> Infohazard.Core.Spawnable.Despawned`

Invoke when the Spawnable is despawned.

6.34.4.2 Destroyed `Action<Spawnable> Infohazard.Core.Spawnable.Destroyed`

Invoked when the Spawnable is destroyed (not just despawned).

6.34.4.3 Spawned `Action<Spawnable> Infohazard.Core.Spawnable.Spawned`

Invoked when the Spawnable is spawned.

The documentation for this class was generated from the following file:

- Runtime/Pooling/Spawnable.cs

6.35 Infohazard.Core.SpawnParams Struct Reference

Used to pass spawn parameters to various object creation/initialization methods.

Static Public Member Functions

- static SpawnParams At (Transform transform, bool parented=false, bool includeScene=false)

  Spawn at a given transform (copy position and rotation).

Public Attributes

- Vector3? Position

  Position to spawn at (if null, do not set position).
- Quaternion? Rotation

  Position to spawn at (if null, do not set rotation).
- Vector3? Scale

  Scale to spawn at (if null, do not set the scale).
- Transform Parent

  Parent to attach the object to. If null, no parent.
- bool InWorldSpace

  If true, given position/rotation/scale are considered world space. If false, they are considered in the space of the parent.
- ulong PersistedInstanceID

  Instance ID to pass to a IPersistedInstance script.
- Scene? Scene

  Scene to spawn in, if Parent is null.

Static Public Attributes

- static readonly SpawnParams Default = new SpawnParams()

  Default spawn params (no transform, parent, scene, or instance ID).

6.35.1 Detailed Description

Used to pass spawn parameters to various object creation/initialization methods.

6.35.2 Member Function Documentation

6.35.2.1 At()

```
static SpawnParams Infohazard.Core.SpawnParams.At (
        Transform transform,
        bool parented = false,
        bool includeScene = false )  [static]
```

Spawn at a given transform (copy position and rotation).

Parameters

transform	The transform to spawn at.
parented	If true, parent spawned object to given transform.
includeScene	If true and parented is false, move to given transform's scene.

Returns

Resulting SpawnParams.

6.35.3 Member Data Documentation

6.35.3.1 Default `readonly SpawnParams Infohazard.Core.SpawnParams.Default = new SpawnParams() [static]`

Default spawn params (no transform, parent, scene, or instance ID).

6.35.3.2 InWorldSpace `bool Infohazard.Core.SpawnParams.InWorldSpace`

If true, given position/rotation/scale are considered world space. If false, they are considered in the space of the parent.

6.35.3.3 Parent `Transform Infohazard.Core.SpawnParams.Parent`

Parent to attach the object to. If null, no parent.

6.35.3.4 PersistedInstanceID `ulong Infohazard.Core.SpawnParams.PersistedInstanceID`

Instance ID to pass to a IPersistedInstance script.

6.35.3.5 Position `Vector3? Infohazard.Core.SpawnParams.Position`

Position to spawn at (if null, do not set position).

6.35.3.6 Rotation `Quaternion? Infohazard.Core.SpawnParams.Rotation`

Position to spawn at (if null, do not set rotation).

6.35.3.7 Scale `Vector3? Infohazard.Core.SpawnParams.Scale`

Scale to spawn at (if null, do not set the scale).

6.35.3.8 Scene `Scene? Infohazard.Core.SpawnParams.Scene`

Scene to spawn in, if Parent is null.

The documentation for this struct was generated from the following file:

- Runtime/Pooling/SpawnParams.cs

6.36 Infohazard.Core.SpawnRef< T > Class Template Reference

SpawnRef for spawning a GameObject directly.

Public Member Functions

- SpawnRef ()
  Default constructor (needed for Unity serialization).
- SpawnRef (GameObject prefab)
  Construct with a given prefab.
- SpawnRef ()
  Default constructor (needed for Unity serialization).
- SpawnRef (T prefab)
  Construct with a given prefab component.

Protected Member Functions

- override void GetSpawnableAndGameObject (T obj, out Spawnable spawnable, out GameObject game↩Object)
  Override to return the associated Spawnable script and GameObject for given object.

Additional Inherited Members

6.36.1 Detailed Description

SpawnRef for spawning a GameObject directly.

SpawnRef for spawning a GameObject via one of its components.

Type Constraints

T* : *Component

6.36.2 Constructor & Destructor Documentation

6.36.2.1 SpawnRef() [1/4] Infohazard.Core.SpawnRef< T >.SpawnRef ( )

Default constructor (needed for Unity serialization).

6.36.2.2 SpawnRef() [2/4] Infohazard.Core.SpawnRef< T >.SpawnRef (

```
GameObject prefab )
```

Construct with a given prefab.

Parameters

prefab	The prefab to be spawned.

inheritdoc/>

6.36.2.3 SpawnRef() [3/4] Infohazard.Core.SpawnRef< T >.SpawnRef ( )

Default constructor (needed for Unity serialization).

6.36.2.4 SpawnRef() [4/4] Infohazard.Core.SpawnRef< T >.SpawnRef (

```
T prefab )
```

Construct with a given prefab component.

Parameters

prefab	The prefab to be spawned.

inheritdoc/>

6.36.3 Member Function Documentation

6.36.3.1 GetSpawnableAndGameObject() override void Infohazard.Core.SpawnRef< T >.Get↩ SpawnableAndGameObject (

```
T obj,
out Spawnable spawnable,
out GameObject gameObject ) [protected], [virtual]
```

Override to return the associated Spawnable script and GameObject for given object.

Parameters

obj	The attached object.
spawnable	The Spawnable script for obj.
gameObject	The GameObject for obj.

Implements Infohazard.Core.SpawnRefBase< T >.

The documentation for this class was generated from the following file:

- Runtime/Pooling/SpawnRef.cs

6.37 Infohazard.Core.SpawnRefBase< T > Class Template Reference

Only used internally.

Classes

- class PropNames

  This is used to refer to the names of private fields in this class from a custom Editor.

Public Member Functions

- SpawnRefBase ()

  Default constructor (needed for Unity serialization).
- SpawnRefBase (T prefab)

  Construct with a given prefab.
- virtual void Retain ()

  Add a user to the SpawnRef, creating the DefaultPoolHandler if necessary.
- virtual void Release ()

  Remove a user from the SpawnRef, in turn releasing the IPoolHandler.
- T Spawn (in SpawnParams spawnParams=default)

  Spawn an instance of Prefab. The SpawnRef MUST be retained.

Protected Member Functions

- abstract void GetSpawnableAndGameObject (T obj, out Spawnable spawnable, out GameObject game↩ Object)

  Override to return the associated Spawnable script and GameObject for given object.

Properties

- bool IsValid `[get]`

  Whether there is a valid prefab is attached.
- T Prefab `[get]`

  The prefab to be spawned.

6.37.1 Detailed Description

Only used internally.

Used as a serializable utility for referencing a prefab, managing its DefaultPoolHandler, and retaining/releasing that pool handler as necessary.

Template Parameters

T	The type of object to be referenced.

Type Constraints

T* : *Object

6.37.2 Constructor & Destructor Documentation

6.37.2.1 SpawnRefBase() [1/2] `Infohazard.Core.SpawnRefBase< T >.SpawnRefBase ( )`

Default constructor (needed for Unity serialization).

6.37.2.2 SpawnRefBase() [2/2] `Infohazard.Core.SpawnRefBase< T >.SpawnRefBase (`

```
T prefab )
```

Construct with a given prefab.

Parameters

prefab	The prefab to be spawned.

6.37.3 Member Function Documentation

6.37.3.1 GetSpawnableAndGameObject() `abstract void Infohazard.Core.SpawnRefBase< T >.Get↩ SpawnableAndGameObject (`

```
T obj,
out Spawnable spawnable,
out GameObject gameObject )  [protected], [pure virtual]
```

Override to return the associated Spawnable script and GameObject for given object.

Parameters

obj	The attached object.
spawnable	The Spawnable script for obj.
gameObject	The GameObject for obj.

Implemented in Infohazard.Core.SpawnRef< T >.

6.37.3.2 Release() `virtual void Infohazard.Core.SpawnRefBase< T >.Release ( ) [virtual]`

Remove a user from the SpawnRef, in turn releasing the IPoolHandler.

6.37.3.3 Retain() `virtual void Infohazard.Core.SpawnRefBase< T >.Retain ( ) [virtual]`

Add a user to the SpawnRef, creating the DefaultPoolHandler if necessary.

The IPoolHandler for the object will be retained.

6.37.3.4 Spawn() `T Infohazard.Core.SpawnRefBase< T >.Spawn ( in SpawnParams spawnParams = default )`

Spawn an instance of Prefab. The SpawnRef MUST be retained.

Parameters

spawnParams	Additional spawn info.

Returns

The spawned object.

6.37.4 Property Documentation

6.37.4.1 IsValid `bool Infohazard.Core.SpawnRefBase< T >.IsValid [get]`

Whether there is a valid prefab is attached.

6.37.4.2 Prefab `T Infohazard.Core.SpawnRefBase< T >.Prefab [get]`

The prefab to be spawned.

The documentation for this class was generated from the following file:

- Runtime/Pooling/SpawnRef.cs

6.38 Infohazard.Core.StringUtility Class Reference

Contains string processing utilities.

Static Public Member Functions

- static string SplitCamelCase (this string str, bool capitalizeFirst=false)

    Splits a camel-case string into words separated by spaces.

6.38.1 Detailed Description

Contains string processing utilities.

6.38.2 Member Function Documentation

6.38.2.1 SplitCamelCase()

```
static string Infohazard.Core.StringUtility.SplitCamelCase (
        this string str,
        bool capitalizeFirst = false )  [static]
```

Splits a camel-case string into words separated by spaces.

Multiple consecutive capitals are considered the same word.

Parameters

str	The string to split.
capitalizeFirst	Whether to capitalize the first letter.

Returns

The split string.

The documentation for this class was generated from the following file:

- Runtime/Utility/StringUtility.cs

6.39 Infohazard.Core.Tag Class Reference

Provides string constants for builtin Unity tags.

Static Public Attributes

- const string Untagged = "Untagged"

    The string `"Untagged"`*.*
- const string Respawn = "Respawn"

    The string `"Respawn"`*.*
- const string Finish = "Finish"

    The string `"Finish"`*.*

- const string EditorOnly = "EditorOnly"

    The string `"EditorOnly"`*.*
- const string MainCamera = "MainCamera"

    The string `"MainCamera"`*.*
- const string Player = "Player"

    The string `"Player"`*.*
- const string GameController = "GameController"

    The string `"GameController"`*.*
- static readonly string[ ] DefaultTags

    Array of default tags provided by Unity.
- static string[ ] GameOverrideTags = null

    Set by the generated `GameTag` *script.*

Properties

- static string[ ] Tags `[get]`

    Array of all default and custom tags in the project.

6.39.1 Detailed Description

Provides string constants for builtin Unity tags.

To extend with custom tags, see `GameTag`, which you can generate using the command Infohazard > Generate > Update GameTag.cs.

6.39.2 Member Data Documentation

6.39.2.1 DefaultTags `readonly string [] Infohazard.Core.Tag.DefaultTags [static]`

Initial value:
```
= {
            "Untagged", "Respawn", "Finish", "EditorOnly", "MainCamera", "Player", "GameController",
        }
```

Array of default tags provided by Unity.

6.39.2.2 EditorOnly `const string Infohazard.Core.Tag.EditorOnly = "EditorOnly" [static]`

The string `"EditorOnly"`.

6.39.2.3 Finish `const string Infohazard.Core.Tag.Finish = "Finish" [static]`

The string `"Finish"`.

6.39.2.4 GameController `const string Infohazard.Core.Tag.GameController = "GameController" [static]`

The string "`GameController`".

6.39.2.5 GameOverrideTags `string [] Infohazard.Core.Tag.GameOverrideTags = null [static]`

Set by the generated `GameTag` script.

6.39.2.6 MainCamera `const string Infohazard.Core.Tag.MainCamera = "MainCamera" [static]`

The string "`MainCamera`".

6.39.2.7 Player `const string Infohazard.Core.Tag.Player = "Player" [static]`

The string "`Player`".

6.39.2.8 Respawn `const string Infohazard.Core.Tag.Respawn = "Respawn" [static]`

The string "`Respawn`".

6.39.2.9 Untagged `const string Infohazard.Core.Tag.Untagged = "Untagged" [static]`

The string "`Untagged`".

6.39.3 Property Documentation

6.39.3.1 Tags `string [] Infohazard.Core.Tag.Tags [static], [get]`

Array of all default and custom tags in the project.

The documentation for this class was generated from the following file:

- Runtime/Misc/Tag.cs

6.40 Infohazard.Core.TagMask Struct Reference

Used to select tags in the inspector, including the ability to select multiple tags.

Public Member Functions

- TagMask (long value)

    Initialize a new TagMask with the given value.
- override string ToString ()
- bool Equals (TagMask other)
- override bool Equals (object obj)
- override int GetHashCode ()

Static Public Member Functions

- static implicit operator long (TagMask mask)

    Convert a TagMask to a long.
- static implicit operator TagMask (long mask)

    Convert a long to a TagMask.
- static TagMask operator& (in TagMask lhs, in TagMask rhs)

    Apply bitwise AND operator to two TagMasks.
- static TagMask operator& (TagMask lhs, long rhs)

    Apply bitwise AND operator to a TagMask and a long.
- static long operator& (long lhs, TagMask rhs)

    Apply bitwise AND operator to a long and a TagMask.
- static TagMask operator| (TagMask lhs, TagMask rhs)

    Apply bitwise OR operator to a TagMask and a TagMask.
- static TagMask operator| (TagMask lhs, long rhs)

    Apply bitwise OR operator to a TagMask and a long.
- static long operator| (long lhs, TagMask rhs)

    Apply bitwise OR operator to a long and a TagMask.
- static TagMask operator^ (TagMask lhs, TagMask rhs)

    Apply bitwise XOR operator to a TagMask and a TagMask.
- static TagMask operator^ (TagMask lhs, long rhs)

    Apply bitwise XOR operator to a TagMask and a long.
- static long operator^ (long lhs, TagMask rhs)

    Apply bitwise XOR operator to a long and a TagMask.
- static TagMask operator~ (TagMask mask)

    Apply bitwise NOT operator to a TagMask.
- static int NameToTag (string name)

    Gets the index of a given tag in the Tag.Tags array.
- static string TagToName (int tag)

    Gets the tag name at the given index in the Tag.Tags array.
- static long GetMask (params string[ ] names)

    Get a mask value that contains all the given tag names.
- static long GetMask (string name)

    Get a mask value that contains the given tag name.

Static Public Attributes

- const long UntaggedMask = 1 << 0

    Mask value for the Untagged tag.
- const long RespawnMask = 1 << 1

    Mask value for the Respawn tag.
- const long FinishMask = 1 << 2

    Mask value for the Finish tag.
- const long EditorOnlyMask = 1 << 3

    Mask value for the EditorOnly tag.
- const long MainCameraMask = 1 << 4

    Mask value for the MainCamera tag.
- const long PlayerMask = 1 << 5

    Mask value for the Player tag.
- const long GameControllerMask = 1 << 6

    Mask value for the GameController tag.

Properties

- long Value `[get, set]`

    The value of the mask as a 64-bit integer.

6.40.1 Detailed Description

Used to select tags in the inspector, including the ability to select multiple tags.

Works similar to LayerMask. If you have a custom `GameTag` script generated, your custom tags will be available here too. You can find code constants for those tags in `GameTagMask`. Like LayerMask, TagMask is implicitly convertable to and from an integer value (long in this case).

6.40.2 Constructor & Destructor Documentation

6.40.2.1 TagMask() `Infohazard.Core.TagMask.TagMask ( long value )`

Initialize a new TagMask with the given value.

Parameters

value	The value to initialize with, representing which tags are "on".

6.40.3 Member Function Documentation

6.40.3.1 Equals() [1/2]

```
override bool Infohazard.Core.TagMask.Equals (
        object obj )
```

6.40.3.2 Equals() [2/2]

```
bool Infohazard.Core.TagMask.Equals (
        TagMask other )
```

6.40.3.3 GetHashCode()

```
override int Infohazard.Core.TagMask.GetHashCode ( )
```

6.40.3.4 GetMask() [1/2]

```
static long Infohazard.Core.TagMask.GetMask (
        params string[] names ) [static]
```

Get a mask value that contains all the given tag names.

Parameters

names	Names of tags to include in the mask.

Returns

The created mask.

6.40.3.5 GetMask() [2/2]

```
static long Infohazard.Core.TagMask.GetMask (
        string name ) [static]
```

Get a mask value that contains the given tag name.

Parameters

name	Name of tag to include in the mask.

Returns

The created mask.

6.40.3.6 NameToTag()

```
static int Infohazard.Core.TagMask.NameToTag (
        string name ) [static]
```

Gets the index of a given tag in the Tag.Tags array.

Parameters

name	Tag name.

Returns

The index of the tag or -1 if it doesn't exist.

6.40.3.7 operator long()

```
static implicit Infohazard.Core.TagMask.operator long (
        TagMask mask ) [static]
```

Convert a TagMask to a long.

Parameters

mask	TagMask to convert.

Returns

The mask's value.

6.40.3.8 operator TagMask()

```
static implicit Infohazard.Core.TagMask.operator TagMask (
        long mask ) [static]
```

Convert a long to a TagMask.

Parameters

mask	The mask value.

Returns

The created TagMask.

6.40.3.9 operator&() [1/3]

```
static TagMask Infohazard.Core.TagMask.operator& (
        in TagMask lhs,
        in TagMask rhs ) [static]
```

Apply bitwise AND operator to two TagMasks.

6.40.3.10 operator&() [2/3]

```
static long Infohazard.Core.TagMask.operator& (
        long lhs,
        TagMask rhs ) [static]
```

Apply bitwise AND operator to a long and a TagMask.

6.40.3.11 operator&() [3/3]

```
static TagMask Infohazard.Core.TagMask.operator& (
        TagMask lhs,
        long rhs ) [static]
```

Apply bitwise AND operator to a TagMask and a long.

6.40.3.12 operator^() [1/3]

```
static long Infohazard.Core.TagMask.operator^ (
        long lhs,
        TagMask rhs ) [static]
```

Apply bitwise XOR operator to a long and a TagMask.

6.40.3.13 operator^() [2/3]

```
static TagMask Infohazard.Core.TagMask.operator^ (
        TagMask lhs,
        long rhs ) [static]
```

Apply bitwise XOR operator to a TagMask and a long.

6.40.3.14 operator^() [3/3]

```
static TagMask Infohazard.Core.TagMask.operator^ (
        TagMask lhs,
        TagMask rhs ) [static]
```

Apply bitwise XOR operator to a TagMask and a TagMask.

6.40.3.15 operator"|() [1/3]

```
static long Infohazard.Core.TagMask.operator| (
        long lhs,
        TagMask rhs ) [static]
```

Apply bitwise OR operator to a long and a TagMask.

6.40.3.16 operator"\|() [2/3] `static TagMask Infohazard.Core.TagMask.operator| ( TagMask lhs, long rhs ) [static]`

Apply bitwise OR operator to a TagMask and a long.

6.40.3.17 operator"\|() [3/3] `static TagMask Infohazard.Core.TagMask.operator| ( TagMask lhs, TagMask rhs ) [static]`

Apply bitwise OR operator to a TagMask and a TagMask.

6.40.3.18 operator∼() `static TagMask Infohazard.Core.TagMask.operator∼ ( TagMask mask ) [static]`

Apply bitwise NOT operator to a TagMask.

6.40.3.19 TagToName() `static string Infohazard.Core.TagMask.TagToName ( int tag ) [static]`

Gets the tag name at the given index in the Tag.Tags array.

Parameters

tag	Tag index. Must be in range [0, TAG COUNT - 1].

Returns

The tag's name.

6.40.3.20 ToString() `override string Infohazard.Core.TagMask.ToString ( )`

6.40.4 Member Data Documentation

6.40.4.1 EditorOnlyMask `const long Infohazard.Core.TagMask.EditorOnlyMask = 1 << 3 [static]`

Mask value for the EditorOnly tag.

6.40.4.2 FinishMask `const long Infohazard.Core.TagMask.FinishMask = 1 << 2 [static]`

Mask value for the Finish tag.

6.40.4.3 GameControllerMask `const long Infohazard.Core.TagMask.GameControllerMask = 1 << 6 [static]`

Mask value for the GameController tag.

6.40.4.4 MainCameraMask `const long Infohazard.Core.TagMask.MainCameraMask = 1 << 4 [static]`

Mask value for the MainCamera tag.

6.40.4.5 PlayerMask `const long Infohazard.Core.TagMask.PlayerMask = 1 << 5 [static]`

Mask value for the Player tag.

6.40.4.6 RespawnMask `const long Infohazard.Core.TagMask.RespawnMask = 1 << 1 [static]`

Mask value for the Respawn tag.

6.40.4.7 UntaggedMask `const long Infohazard.Core.TagMask.UntaggedMask = 1 << 0 [static]`

Mask value for the Untagged tag.

6.40.5 Property Documentation

6.40.5.1 Value `long Infohazard.Core.TagMask.Value [get], [set]`

The value of the mask as a 64-bit integer.

The documentation for this struct was generated from the following file:

- Runtime/Misc/Tag.cs

6.41 Infohazard.Core.TagMaskUtility Class Reference

Static operations on Tag enum values.

Static Public Member Functions

- static bool CompareTagMask (this GameObject obj, long tag)

  Return true if GameObject's tag matches given any tag in given value.
- static bool CompareTagMask (this Component obj, long tag)

  Return true if Component's tag matches given any tag in given value.
- static void SetTagIndex (this GameObject obj, int tagIndex)

  Set the tag index of a GameObject.
- static int GetTagIndex (this GameObject obj)

  Get the tag index of a GameObject.
- static int GetTagIndex (this Component obj)

  Get the tag index of a Component.
- static long GetTagMask (this GameObject obj)

  Get the tag mask of a GameObject.
- static long GetTagMask (this Component obj)

  Get the tag mask of a Component.

6.41.1 Detailed Description

Static operations on Tag enum values.

6.41.2 Member Function Documentation

6.41.2.1 CompareTagMask() [1/2]

```
static bool Infohazard.Core.TagMaskUtility.CompareTagMask (
        this Component obj,
        long tag ) [static]
```

Return true if Component's tag matches given any tag in given value.

Parameters

obj	The Component to check.
tag	The tag to compare, which may be multiple tags.

Returns

Whether Component matches given tag.

6.41.2.2 CompareTagMask() `[2/2]` `static bool Infohazard.Core.TagMaskUtility.CompareTagMask ( this GameObject obj, long tag ) [static]`

```
static bool Infohazard.Core.TagMaskUtility.CompareTagMask (
        this GameObject obj,
        long tag ) [static]
```

Return true if GameObject's tag matches given any tag in given value.

Parameters

obj	The GameObject to check.
tag	The tag to compare, which may be multiple tags.

Returns

Whether GameObject matches given tag.

6.41.2.3 GetTagIndex() `[1/2]`

```
static int Infohazard.Core.TagMaskUtility.GetTagIndex (
        this Component obj ) [static]
```

Get the tag index of a Component.

Parameters

obj	Object to read.

Returns

The Component's tag index.

6.41.2.4 GetTagIndex() `[2/2]`

```
static int Infohazard.Core.TagMaskUtility.GetTagIndex (
        this GameObject obj ) [static]
```

Get the tag index of a GameObject.

Parameters

obj	Object to read.

Returns

The GameObject's tag index.

6.41.2.5 GetTagMask() [1/2] `static long Infohazard.Core.TagMaskUtility.GetTagMask ( this Component obj ) [static]`

Get the tag mask of a Component.

Parameters

obj	Object to read.

Returns

The Component's tag as a mask.

6.41.2.6 GetTagMask() [2/2]

```
static long Infohazard.Core.TagMaskUtility.GetTagMask (
            this GameObject obj )  [static]
```

Get the tag mask of a GameObject.

Parameters

obj	Object to read.

Returns

The GameObject's tag as a mask.

6.41.2.7 SetTagIndex()

```
static void Infohazard.Core.TagMaskUtility.SetTagIndex (
            this GameObject obj,
            int tagIndex )  [static]
```

Set the tag index of a GameObject.

Parameters

obj	Object to modify.
tagIndex	Tag to set.

The documentation for this class was generated from the following file:

- Runtime/Misc/Tag.cs

6.42 Infohazard.Core.Editor.TagsGenerator Class Reference

Class used to generate the GameTag.cs file to use your custom tags in code.

Static Public Member Functions

- static void Generate ()

  Generate the GameTag file.
- static void Remove ()

  Remove the GameTag file.

6.42.1 Detailed Description

Class used to generate the GameTag.cs file to use your custom tags in code.

To generate this file, use the menu item Infohazard > Generate > Update GameTag.cs.

6.42.2 Member Function Documentation

6.42.2.1 Generate() `static void Infohazard.Core.Editor.TagsGenerator.Generate ( ) [static]`

Generate the GameTag file.

6.42.2.2 Remove() `static void Infohazard.Core.Editor.TagsGenerator.Remove ( ) [static]`

Remove the GameTag file.

The documentation for this class was generated from the following file:

- Editor/Misc/TagsGenerator.cs

6.43 Infohazard.Core.TimeToLive Class Reference

Despawns a GameObject after a set amount of time.

Properties

- float TimeRemaining `[get, set]`

  How much time remains before the GameObject is destroyed.

6.43.1 Detailed Description

Despawns a GameObject after a set amount of time.

Compatible with the pooling system.

6.43.2 Property Documentation

6.43.2.1 TimeRemaining `float Infohazard.Core.TimeToLive.TimeRemaining [get], [set]`

How much time remains before the GameObject is destroyed.

The documentation for this class was generated from the following file:

- Runtime/Timing/TimeToLive.cs

6.44 Infohazard.Core.TriggerVolume.TriggerEvents Class Reference

Class that stores the UnityEvents used by a TriggerVolume.

Properties

- UnityEvent OnTriggerEnter `[get]`

  Invoked when an object matching the tag filter enters the trigger.
- UnityEvent OnTriggerExit `[get]`

  Invoked when an object matching the tag filter exits the trigger.
- UnityEvent OnAllExit `[get]`

  Invoked when the last object matching the tag filter exits the trigger.

6.44.1 Detailed Description

Class that stores the UnityEvents used by a TriggerVolume.

6.44.2 Property Documentation

6.44.2.1 OnAllExit `UnityEvent Infohazard.Core.TriggerVolume.TriggerEvents.OnAllExit [get]`

Invoked when the last object matching the tag filter exits the trigger.

6.44.2.2 OnTriggerEnter `UnityEvent Infohazard.Core.TriggerVolume.TriggerEvents.OnTriggerEnter [get]`

Invoked when an object matching the tag filter enters the trigger.

6.44.2.3 OnTriggerExit `UnityEvent Infohazard.Core.TriggerVolume.TriggerEvents.OnTriggerExit [get]`

Invoked when an object matching the tag filter exits the trigger.

The documentation for this class was generated from the following file:

- Runtime/Misc/TriggerVolume.cs

6.45 Infohazard.Core.TriggerVolume Class Reference

A script that makes it easy to add events to a trigger collider.

Classes

- class TriggerEvents

  Class that stores the UnityEvents used by a TriggerVolume.

Properties

- TriggerEvents Events `[get]`

  UnityEvents that enable you to assign functionality in the editor.
- IReadOnlyCollection< GameObject > Occupants `[get]`

  All objects currently inside the trigger volume.
- IReadOnlyList< Collider > ControlledColliders `[get]`

  List of colliders to enable/disable along with this component.

Events

- Action< GameObject > TriggerEntered

  Invoked when an object matching the tag filter enters the trigger.
- Action< GameObject > TriggerExited

  Invoked when an object matching the tag filter exits the trigger.
- Action< GameObject > AllExited

  Invoked when the last object matching the tag filter exits the trigger.

6.45.1 Detailed Description

A script that makes it easy to add events to a trigger collider.

Provides both UnityEvents (assignable in the inspector) and normal C# events for when an object enters or leaves the trigger, and when all objects have left the trigger. Also provides a tag filter, allowing you to control which types of object can activate it.

6.45.2 Property Documentation

6.45.2.1 ControlledColliders `IReadOnlyList<Collider> Infohazard.Core.TriggerVolume.ControlledColliders [get]`

List of colliders to enable/disable along with this component.

6.45.2.2 Events `TriggerEvents Infohazard.Core.TriggerVolume.Events [get]`

UnityEvents that enable you to assign functionality in the editor.

6.45.2.3 Occupants `IReadOnlyCollection<GameObject> Infohazard.Core.TriggerVolume.Occupants [get]`

All objects currently inside the trigger volume.

6.45.3 Event Documentation

6.45.3.1 AllExited `Action<GameObject> Infohazard.Core.TriggerVolume.AllExited`

Invoked when the last object matching the tag filter exits the trigger.

6.45.3.2 TriggerEntered `Action<GameObject> Infohazard.Core.TriggerVolume.TriggerEntered`

Invoked when an object matching the tag filter enters the trigger.

6.45.3.3 TriggerExited `Action<GameObject> Infohazard.Core.TriggerVolume.TriggerExited`

Invoked when an object matching the tag filter exits the trigger.

The documentation for this class was generated from the following file:

- Runtime/Misc/TriggerVolume.cs

6.46 Infohazard.Core.TypeSelectAttribute Class Reference

Attribute that draws string fields as a dropdown where a Type can be selected.

Public Member Functions

- TypeSelectAttribute (Type baseClass, bool allowAbstract=false, bool allowGeneric=false, bool search=false)

  Construct a new TypeSelectAttribute.

Properties

- Type BaseClass `[get]`

  If set, dropdown will only show types assignable to this type.
- bool AllowAbstract `[get]`

  Whether to show abstract classes.
- bool AllowGeneric `[get]`

  Whether to show generic types.
- bool Search `[get]`

  Whether to show a search bar.

6.46.1 Detailed Description

Attribute that draws string fields as a dropdown where a Type can be selected.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 TypeSelectAttribute()

```
Infohazard.Core.TypeSelectAttribute.TypeSelectAttribute (
        Type baseClass,
        bool allowAbstract = false,
        bool allowGeneric = false,
        bool search = false )
```

Construct a new TypeSelectAttribute.

Parameters

baseClass	If set, dropdown will only show types assignable to this type.
allowAbstract	Whether to show abstract classes.
allowGeneric	Whether to show generic types.
search	Whether to show a search bar./

6.46.3 Property Documentation

6.46.3.1 AllowAbstract

`bool Infohazard.Core.TypeSelectAttribute.AllowAbstract [get]`

Whether to show abstract classes.

6.46.3.2 AllowGeneric `bool Infohazard.Core.TypeSelectAttribute.AllowGeneric [get]`

Whether to show generic types.

6.46.3.3 BaseClass `Type Infohazard.Core.TypeSelectAttribute.BaseClass [get]`

If set, dropdown will only show types assignable to this type.

6.46.3.4 Search `bool Infohazard.Core.TypeSelectAttribute.Search [get]`

Whether to show a search bar.

The documentation for this class was generated from the following file:

- Runtime/Attributes/TypeSelectAttribute.cs

6.47 Infohazard.Core.TypeUtility Class Reference

Contains utilities for working with C# reflection types and getting a type by its name.

Static Public Member Functions

- static Type GetType (string fullName)

  Get a type given its full name (including namespace).
- static string GetDisplayName (this Type type, bool includeNamespace=false, bool capitalizeFirst=false)

  Get a human-readable name of a type (mostly used for generic types). Generally equivalent to the name you use in C# code.

Properties

- static Assembly[ ] AllAssemblies `[get]`

  Returns an array of all loaded assemblies.
- static IEnumerable< Type > AllTypes `[get]`

  Returns an enumeration of all loaded types.

6.47.1 Detailed Description

Contains utilities for working with C# reflection types and getting a type by its name.

6.47.2 Member Function Documentation

6.47.2.1 GetDisplayName()

```
static string Infohazard.Core.TypeUtility.GetDisplayName (
        this Type type,
        bool includeNamespace = false,
        bool capitalizeFirst = false ) [static]
```

Get a human-readable name of a type (mostly used for generic types). Generally equivalent to the name you use in C# code.

Parameters

type	The type.
includeNamespace	Whether to include namespaces in the result.
capitalizeFirst	If true, force the first character to a capital.

Returns

A human-readable name of the type.

6.47.2.2 GetType() `static Type Infohazard.Core.TypeUtility.GetType ( string fullName ) [static]`

Get a type given its full name (including namespace).

Parameters

fullName	Name of the type including namespace.

Returns

The found type, or null.

6.47.3 Property Documentation

6.47.3.1 AllAssemblies `Assembly [] Infohazard.Core.TypeUtility.AllAssemblies [static], [get]`

Returns an array of all loaded assemblies.

6.47.3.2 AllTypes `IEnumerable<Type> Infohazard.Core.TypeUtility.AllTypes [static], [get]`

Returns an enumeration of all loaded types.

The documentation for this class was generated from the following file:

- Runtime/Utility/TypeUtility.cs

6.48 Infohazard.Core.UniqueNamedObject Class Reference

This script is used to assign a unique name to an object, which can then be used to find that object.

Static Public Member Functions

- static bool TryGetObject (string name, out GameObject result)

  Try to get a GameObject with the given unique name, and return whether it was found.
- static bool TryGetObject (UniqueNameListEntry entry, out GameObject result)

  Try to get a GameObject with the given unique name asset, and return whether it was found.

Properties

- UniqueNameListEntry UniqueNameListEntry `[get]`

  Unique name asset for the object.
- string UniqueName `[get, private set]`

  Unique name string for the object.
- static IReadOnlyDictionary< string, UniqueNamedObject > Objects `[get]`

  Dictionary of all active UniqueNamedObjects keyed by their unique names.

Events

- static Action< UniqueNamedObject > ObjectAdded

  Invoked when a new UniqueNamedObject is added to the dictionary.
- static Action< UniqueNamedObject > ObjectRemoved

  Invoked when a UniqueNamedObject is removed from the dictionary.

6.48.1 Detailed Description

This script is used to assign a unique name to an object, which can then be used to find that object.

Unique names can be created under a UniqueNameList. The static methods in this class can be used to quickly find objects by their unique names. Since the unique names are asset references, there is no chance of making typos, and they can even be renamed without breaking references. There is nothing that prevents two objects from sharing the same name, but you will get a log error if they are active at the same time.

6.48.2 Member Function Documentation

6.48.2.1 TryGetObject() [1/2]

```
static bool Infohazard.Core.UniqueNamedObject.TryGetObject (
        string name,
        out GameObject result ) [static]
```

Try to get a GameObject with the given unique name, and return whether it was found.

Parameters

name	The name to search for.
result	The object with that name, or null if not found.

Returns

Whether the object was found.

6.48.2.2 TryGetObject() [2/2] `static bool Infohazard.Core.UniqueNamedObject.TryGetObject ( UniqueNameListEntry entry, out GameObject result ) [static]`

Try to get a GameObject with the given unique name asset, and return whether it was found.

Parameters

entry	The name asset to search for.
result	The object with that name, or null if not found.

Returns

Whether the object was found.

6.48.3 Property Documentation

6.48.3.1 Objects `IReadOnlyDictionary<string, UniqueNamedObject> Infohazard.Core.UniqueNamed←Object.Objects [static], [get]`

Dictionary of all active UniqueNamedObjects keyed by their unique names.

6.48.3.2 UniqueName `string Infohazard.Core.UniqueNamedObject.UniqueName [get], [private set]`

Unique name string for the object.

6.48.3.3 UniqueNameListEntry `UniqueNameListEntry Infohazard.Core.UniqueNamedObject.Unique←NameListEntry [get]`

Unique name asset for the object.

6.48.4 Event Documentation

6.48.4.1 ObjectAdded `Action<UniqueNamedObject> Infohazard.Core.UniqueNamedObject.ObjectAdded [static]`

Invoked when a new UniqueNamedObject is added to the dictionary.

6.48.4.2 ObjectRemoved `Action<UniqueNamedObject> Infohazard.Core.UniqueNamedObject.Object↩ Removed [static]`

Invoked when a UniqueNamedObject is removed from the dictionary.

The documentation for this class was generated from the following file:

- Runtime/UniqueName/UniqueNamedObject.cs

6.49 Infohazard.Core.UniqueNameList Class Reference

A list used to organize unique names used by objects.

Properties

- IReadOnlyList< UniqueNameListEntry > Entries `[get]`

    All unique name assets in this list.

6.49.1 Detailed Description

A list used to organize unique names used by objects.

You can have one or many UniqueNameLists in your project, it is totally up to you. When selecting a unique name for an object, you will have the option to create a new one in any UniqueNameList.

6.49.2 Property Documentation

6.49.2.1 Entries `IReadOnlyList<UniqueNameListEntry> Infohazard.Core.UniqueNameList.Entries [get]`

All unique name assets in this list.

The documentation for this class was generated from the following file:

- Runtime/UniqueName/UniqueNameList.cs

6.50 Infohazard.Core.UniqueNameListEntry Class Reference

A unique name asset, usable by a UniqueNamedObject.

6.50.1 Detailed Description

A unique name asset, usable by a UniqueNamedObject.

The asset's name is the unique name that will be referenced. UniqueNameListEntries should be created via a UniqueNameList.

The documentation for this class was generated from the following file:

- Runtime/UniqueName/UniqueNameListEntry.cs