

# Namespace Cobilas.GodotEngine.Utility

## Classes

### [Coroutine](#)

This class represents a corrotine process.

### [CoroutineManager](#)

This class is responsible for managing all coroutines.

### [DebugLog](#)

Static class to print messages to the console.

### [GDDirectory](#)

Represents a directory file.

### [GDFeature](#)

This class contains some Features pre-defined by the engine.

### [GDFile](#)

This class is a representation of a file.

### [GDFileBase](#)

This is a base class for other classes that represent files or directory files.

### [GDIONull](#)

This class is a representation of a null file.

### [Gizmos](#)

Gizmos are used to give visual debugging or setup aids in the Scene view.

### [NullNode](#)

A null representation of the Godot.Node class.

### [Randomico](#)

The class allows the creation of pseudo random numbers.

### [Screen](#)

Gets or changes game screen information.

## Structs

### [CustomResolutionList](#)

Stores custom resolutions.

## [DisplayInfo](#)

Contains information from a specific screen.

## [FixedRunTimeSecond](#)

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [PhysicsProcess\(float\)](#).

## [LastFixedRunTimeSecond](#)

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [PhysicsProcess\(float\)](#).

This class allows the coroutine to be called after the methods of updating the current scene.

## [LastRunTimeSecond](#)

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [Process\(float\)](#).

This class allows the coroutine to be called after the methods of updating the current scene.

## [Resolution](#)

Stores information about a screen resolution.

## [RunTimeSecond](#)

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [Process\(float\)](#).

# Interfaces

## [IYieldCoroutine](#)

A base interface for all Yield class.

## [IYieldFixedUpdate](#)

Yield Class to be excited in the [PhysicsProcess\(float\)](#)

## [IYieldUpdate](#)

Yield Class to be excited in the [Process\(float\)](#)

## [IYieldVolatile](#)

The `IyieldVolatile` interface allows the `Yield` class to change the type of process.

This interface allows you to change the type of update if the object will use the [Process\(float\)](#) or [PhysicsProcess\(float\)](#) [Coroutine](#) process.

## Enums

### [GDFileAttributes](#)

Represents the file attributes.

### [ScreenMode](#)

Represents screen modes.

# Class Coroutine

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class represents a corrotine process.

```
public sealed class Coroutine : IEnumerable, IDisposable
```

## Inheritance

[object](#) ← Coroutine

## Implements

[IEnumerable](#), [IDisposable](#)

## Inherited Members

[object.ToString\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#)

## Constructors

### Coroutine(IEnumerator?, string?)

Creates a new instance of this object.

```
public Coroutine(IEnumerator? enumerator, string? iD)
```

## Parameters

enumerator [IEnumerator](#)

The [IEnumerator](#) object that will be used.

iD [string](#)

The ID of this [Coroutine](#) object.

## Exceptions

## [ArgumentNullException](#)

It occurs when the enumerator parameter is null.

## [ArgumentNullException](#)

It occurs when the ID parameter is null.

# Properties

## ID

The process id.

```
public string ID { get; }
```

### Property Value

[string](#)

Returns a [string](#) with process ID.

## IsCancellationRequested

Gets whether cancellation has been requested for this [CancellationTokenSource](#).

```
public bool IsCancellationRequested { get; }
```

### Property Value

[bool](#)

[true](#) if cancellation has been requested for this [CancellationTokenSource](#); otherwise, [false](#).

## IsRunning

Indicates if the process is running.

```
public bool IsRunning { get; }
```

## Property Value

[bool](#)

Returns `true` if the process is ongoing.

Returns `false` when the process is completed.

## Methods

### Cancel()

Emits the cancellation signal for the [Coroutine](#) process.

```
public void Cancel()
```

### Exceptions

[ObjectDisposedException](#)

This [CancellationTokenSource](#) has been disposed.

[AggregateException](#)

An aggregate exception containing all the exceptions thrown by the registered callbacks on the associated [CancellationToken](#).

### CancelAfter(int)

Emits a delayed cancellation signal for the [Coroutine](#) process.

```
public void CancelAfter(int millisecondsDelay)
```

### Parameters

`millisecondsDelay` [int](#)

Defines the delay in milliseconds where the cancellation signal will be issued.

## Exceptions

### [ObjectDisposedException](#)

The exception thrown when this [CancellationTokenSource](#) has been disposed.

### [ArgumentOutOfRangeException](#)

The exception that is thrown when `delay` is less than -1 or greater than Int32.MaxValue.

## CancelAfter(TimeSpan)

Emits a delayed cancellation signal for the [Coroutine](#) process.

```
public void CancelAfter(TimeSpan delay)
```

## Parameters

### `delay` [TimeSpan](#)

Defines the delay where the cancellation signal will be issued.

## Exceptions

### [ObjectDisposedException](#)

The exception thrown when this [CancellationTokenSource](#) has been disposed.

### [ArgumentOutOfRangeException](#)

The exception that is thrown when `delay` is less than -1 or greater than Int32.MaxValue.

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## `~Coroutine()`

The destructor is responsible for discarding unmanaged resources.

```
protected ~Coroutine()
```

# Class CoroutineManager

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class is responsible for managing all coroutines.

```
[RunTimeInitializationClass("CoroutineManager", Priority.StartBefore, 0, false)]
public class CoroutineManager : Node, IDisposable
```

## Inheritance

[Object](#) ← Object ← Node ← CoroutineManager

## Implements

[IDisposable](#)

## Inherited Members

Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,  
Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOslmeUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrBeNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , Node.\_UnhandledInput(InputEvent) ,  
Node.\_UnhandledKeyInput(InputEventKey) , [Node.AddChildBelowNode\(Node, Node, bool\)](#) ,  
[Node.SetName\(string\)](#) , Node.GetName() , [Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) ,  
Node.GetChildCount() , Node.GetChildren() , [Node.GetChild\(int\)](#) , Node.HasNode(NodePath) ,  
Node.GetNode(NodePath) , Node.GetNodeOrNull(NodePath) , Node.GetParent() ,  
[Node.FindNode\(string, bool, bool\)](#) , [Node.FindParent\(string\)](#) ,  
Node.HasNodeAndResource(NodePath) , Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() ,  
Node.IsAParentOf(Node) , Node.IsGreaterThanOrEqual(Node) , Node.GetPath() , Node.GetPathTo(Node) ,

[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDeltaTime() , Node.IsPhysicsProcessing() , Node.GetProcessDeltaTime() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostInitialize ,  
Object.NotificationPreDelete , Object.IsValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,  
[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,

[Object.GetMetaList\(\)](#) , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Cally\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
[Object.GetSignalList\(\)](#) , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#).

# Methods

## GenID()

Generates an ID to be used in a [Coroutine](#).

```
public static string GenID()
```

Returns

[string](#)

Returns in [string](#) form the ID generated.

## StartCoroutine(IEnumerator?)

Starts a collating process from an [IEnumerator](#).

```
public static Coroutine StartCoroutine(IEnumerator? enumerator)
```

Parameters

enumerator [IEnumerator](#)

The [IEnumerator](#) that will be used to start the [Coroutine](#).

Returns

[Coroutine](#)

Returns the [Coroutine](#) process that was started.

Exceptions

[ArgumentNullException](#)

When the object is null.

## StopAllCoroutines()

Ends all open Coroutines.

```
public static void StopAllCoroutines()
```

## StopCoroutine(Coroutine?)

Ends all open Coroutines.

```
public static void StopCoroutine(Coroutine? Coroutine)
```

Parameters

## [Coroutine](#) [Coroutine](#)

The [Coroutine](#) that will be closed.

## \_PhysicsProcess(float)

Called during the physics processing step of the main loop. Physics processing means that the frame rate is synced to the physics, i.e. the `delta` variable should be constant. `delta` is in seconds.

It is only called if physics processing is enabled, which is done automatically if this method is overridden, and can be toggled with [SetPhysicsProcess\(bool\)](#).

Corresponds to the Godot.Node.NotificationPhysicsProcess notification in [\\_Notification\(int\)](#).

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

```
public override void _PhysicsProcess(float delta)
```

## Parameters

`delta` [float](#)

## \_Process(float)

Called during the processing step of the main loop. Processing happens at every frame and as fast as possible, so the `delta` time since the previous frame is not constant. `delta` is in seconds.

It is only called if processing is enabled, which is done automatically if this method is overridden, and can be toggled with [SetProcess\(bool\)](#).

Corresponds to the Godot.Node.NotificationProcess notification in [\\_Notification\(int\)](#).

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

```
public override void _Process(float delta)
```

## Parameters

`delta` [float](#)

## \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their Godot.Node.\_Ready() callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the Godot.Node.NotificationReady notification in [Notification\(int\)](#). See also the [onready](#) keyword for variables.

Usually used for initialization. For even earlier initialization, [\\_init\(\)](#) may be used. See also Godot.Node.\_EnterTree().

Note: Godot.Node.\_Ready() may be called only once for each node. After removing a node from the scene tree and adding it again, [\\_ready](#) will not be called a second time. This can be bypassed by requesting another call with Godot.Node.RequestReady(), which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Struct CustomResolutionList

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Stores custom resolutions.

```
[Serializable]
public readonly struct CustomResolutionList : IEnumerable<Resolution>, IEnumerable
```

Implements

[IEnumerable](#)<[Resolution](#)>, [IEnumerable](#)

Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Remarks

Stores all custom resolutions defined in the application to be used.

## Constructors

### CustomResolutionList(in int, in Resolution[])

Starts a new instance of the object.

```
public CustomResolutionList(in int hash, in Resolution[] list)
```

Parameters

hash [int](#)

list [Resolution](#)[]

### CustomResolutionList(in int, in IEnumerable<Resolution>)

Starts a new instance of the object.

```
public CustomResolutionList(in int hash, in IEnumerable<Resolution> list)
```

## Parameters

hash [int](#)

list [IEnumerable](#)<[Resolution](#)>

## Methods

### Deserialize(Stream?)

Deserializes a list of objects from a specified file.

```
public static CustomResolutionList[] Deserialize(Stream? stream)
```

## Parameters

stream [Stream](#)

The file where the list is.

## Returns

[CustomResolutionList](#)[]

Returns a custom resolution list.

## Exceptions

[ArgumentNullException](#)

An exception will be thrown if the parameters are null.

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumarator<Resolution> GetEnumarator()
```

Returns

[IEnumarator](#)<Resolution>

An enumerator that can be used to iterate through the collection.

## Serialize(in CustomResolutionList[], Stream?)

Serializes a list of objects to a specified file.

```
public static void Serialize(in CustomResolutionList[] list, Stream? stream)
```

Parameters

**list** [CustomResolutionList](#)[]

The list to be serialized.

**stream** [Stream](#)

The file that will receive the list.

Exceptions

[ArgumentNullException](#)

An exception will be thrown if the parameters are null.

## Operators

### explicit operator int(CustomResolutionList)

Explicit conversion operator.([CustomResolutionList](#) to [int](#))

```
public static explicit operator int(CustomResolutionList R)
```

## Parameters

R [CustonResolutionList](#)

Object to be converted.

## Returns

[int](#) ↗

# Class DebugLog

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Static class to print messages to the console.

```
public static class DebugLog
```

## Inheritance

[object](#) ← DebugLog

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Fields

### PrintTrace

Allows you to print the complete log trace.

```
public static bool PrintTrace
```

## Field Value

[bool](#)

## Methods

### CloseStream()

Allows the termination of a log stream.

```
public static void CloseStream()
```

## Exceptions

### [InvalidOperationException](#)

Occurs when the flow is already closed.

## ErroLog(params object[])

Allows you to print an error log to the console.

```
public static void ErroLog(params object[] message)
```

### Parameters

#### `message object[]`

The message that will be printed to the console.

## ExceptionLog(Exception)

Allows you to print an exception log to the console.

```
public static void ExceptionLog(Exception ex)
```

### Parameters

#### `ex Exception`

The exception that will be printed to the console.

## Log(params object[])

Allows you to print a log to the console.

```
public static void Log(params object[] message)
```

### Parameters

`message object[]`

The message that will be printed to the console.

## LogToStream(string?, bool)

Allows you to redirect logs to a log file.

```
public static void LogToStream(string? filePath, bool clear = false)
```

### Parameters

`filePath string`

The path of the log file.

`clear bool`

`true` to have the log file cleared.

### Exceptions

[ArgumentNullException](#)

When the FilePath parameter is null.

[InvalidOperationException](#)

Occurs when the flow is still open.

# Struct DisplayInfo

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Contains information from a specific screen.

```
public readonly struct DisplayInfo : IEquatable<DisplayInfo>
```

Implements

[IEquatable](#)<[DisplayInfo](#)>

Inherited Members

[ValueType.ToString\(\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.GetType\(\)](#)

## Constructors

**DisplayInfo(in int, in DisplayDevice?)**

Starts a new instance of the object.

```
public DisplayInfo(in int index, in DisplayDevice? device)
```

Parameters

**index** [int](#)

The monitor index.

**device** [DisplayDevice](#)

The OpenTK.DisplayDevice that contains the screen information.

## Properties

**CurrentResolution**

Current screen resolution.

```
public Resolution CurrentResolution { get; }
```

Property Value

#### [Resolution](#)

Returns the current resolution of this OpenTK.DisplayDevice.

## CustomResolutions

The list of custom resolutions.

```
public CustomResolutionList CustomResolutions { get; }
```

Property Value

#### [CustomResolutionList](#)

Returns a list of custom resolutions from [DisplayInfo](#).

## Index

Screen index

```
public int Index { get; }
```

Property Value

#### [int](#)

Returns the screen index.

## None

Empty display.

```
public static DisplayInfo None { get; }
```

## Property Value

### [DisplayInfo](#)

Returns an empty instance of [DisplayInfo](#).

## Resolutions

Screen resolutions.

```
public Resolution[] Resolutions { get; }
```

## Property Value

### [Resolution\[\]](#)

Returns all resolutions supported by the display.

## Methods

### AddCustomResolution([in Resolution](#), [in DisplayInfo](#))

Adds a custom resolution to [DisplayInfo](#).

```
public static DisplayInfo AddCustomResolution(in Resolution resolution, in  
DisplayInfo display)
```

## Parameters

### [resolution](#) [Resolution](#)

The resolution to be added.

### [display](#) [DisplayInfo](#)

The target [DisplayInfo](#).

Returns

## [DisplayInfo](#)

Returns a new, modified instance of [DisplayInfo](#).

## ChangeCurrentResolution(*in Resolution*, *in DisplayInfo*)

Allows you to change the current resolution of this display.

```
public static DisplayInfo ChangeCurrentResolution(in Resolution resolution, in  
DisplayInfo display)
```

Parameters

### *resolution* [Resolution](#)

The new resolution.

### *display* [DisplayInfo](#)

The target [DisplayInfo](#).

Returns

## [DisplayInfo](#)

Returns a new, modified instance of [DisplayInfo](#).

## Contains(*in Resolution*, *in bool*)

Allows you to check whether a certain resolution exists on this display.

```
public bool Contains(in Resolution resolution, in bool includeCustomResolution = false)
```

Parameters

### *resolution* [Resolution](#)

Target resolution.

## includeCustomResolution [bool](#)

Tells the method whether to compare with the custom resolution list.

Returns

### [bool](#)

If the resolution exists, it will return [true](#).

## Equals(DisplayInfo)

Indicates whether the current object is equal to another object of the same type.

### `public bool Equals(DisplayInfo other)`

Parameters

#### [other](#) [DisplayInfo](#)

An object to compare with this object.

Returns

### [bool](#)

[true](#) if the current object is equal to the [other](#) parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

### `public override bool Equals(object obj)`

Parameters

#### [obj](#) [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if [obj](#) and this instance are the same type and represent the same value; otherwise, [false](#).

## GetHash([in DisplayInfo](#))

Generates a hash from [DisplayInfo](#).

```
public static int GetHash(in DisplayInfo display\)
```

Parameters

[display](#) [DisplayInfo](#)

The object that will be used.

Returns

[int](#)

Returns a hash generated from the [DisplayInfo](#) index and resolution list.

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

# Operators

## operator ==(DisplayInfo, DisplayInfo)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(DisplayInfo A, DisplayInfo B)
```

### Parameters

A [DisplayInfo](#)

Object to be compared.

B [DisplayInfo](#)

Object of comparison.

### Returns

[bool](#) ↗

Returns the result of the comparison.

## operator !=(DisplayInfo, DisplayInfo)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(DisplayInfo A, DisplayInfo B)
```

### Parameters

A [DisplayInfo](#)

Object to be compared.

B [DisplayInfo](#)

Object of comparison.

### Returns

[bool](#) ↗

Returns the result of the comparison.

# Struct FixedRunTimeSecond

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [PhysicsProcess\(float\)](#).

```
public readonly struct FixedRunTimeSecond : IYieldFixedUpdate, IYieldCoroutine
```

Implements

[IYieldFixedUpdate](#), [IYieldCoroutine](#)

Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### FixedRunTimeSecond(double)

Creates a new instance of this object.

```
public FixedRunTimeSecond(double second)
```

Parameters

second [double](#)

# Class GDDirectory

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Represents a directory file.

```
public sealed class GDDirectory : GDFileBase, IDisposable
```

## Inheritance

[object](#) ← [GDFileBase](#) ← GDDirectory

## Implements

[IDisposable](#)

## Inherited Members

[GDFileBase.Null](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

# Properties

## Attribute

The type of the item.

```
public override GDFileAttributes Attribute { get; protected set; }
```

## Property Value

[GDFileAttributes](#)

Returns the type of attribute the item has.

**Null**: Indicates that the item is a representation of a null file.

**File**: Indicates that the item is a representation of a file.

**Directory**: Indicates that the item is a representation of a directory file.

## Count

Subdirectory count.

```
public int Count { get; }
```

### Property Value

[int](#)

Returns the number of subdirectories present in this directory.

## Name

Item name.

```
public override string Name { get; }
```

### Property Value

[string](#)

Returns the name of the item.

## NameWithoutExtension

The name of the item without extension.

```
public override string NameWithoutExtension { get; }
```

### Property Value

[string](#)

Returns the item name without its extension.

## Parent

The item's parent [GDFFileBase](#).

```
public override GDFFileBase Parent { get; protected set; }
```

## Property Value

[GDFFileBase](#)

Returns the parent GDFFileBase that the item is affiliated with.

## Path

The path of the item.

```
public override string Path { get; protected set; }
```

## Property Value

[string](#)

Returns the full or relative path of the item.

## Methods

### CreateDirectory(string?)

Allows you to create a new directory.

```
public bool CreateDirectory(string? directoryName)
```

## Parameters

`directoryName` [string](#)

The name of the new directory.

## Returns

[bool](#)

Returns [true](#) if the directory was created successfully.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when [string](#) object is null.

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public override void Dispose()
```

## ~GDDirectory()

The finalizer allows the disposal of unmanageable code.

```
protected ~GDDirectory()
```

## GetDirectories()

Get all directories.

```
public GDDirectory[] GetDirectories()
```

Returns

[GDDirectory](#)[]

If no directory is found an empty list will be returned.

## Remarks

Subdirectories of directories already obtained will not be included.

## GetDirectory(string?, bool)

Gets a specific directory.

```
public GDDirectory? GetDirectory(string? relativePath, bool isSubdirectory = false)
```

Parameters

**relativePath** [string](#)

The name of the directory to search.

**isSubdirectory** [bool](#)

Allows the method to search subdirectories.

Returns

[GDDirectory](#)

Exceptions

[ArgumentNullException](#)

The exception is thrown when [string](#) object is null.

## GetFile(string?, bool)

Gets a specific file.

```
public GDFile GetFile(string? name, bool isSubdirectory = false)
```

Parameters

**name** [string](#)

The name of the file to search for.

**isSubdirectory** [bool](#)

Allows the method to search subdirectories.

Returns

## [GDFile](#)

If the directory is not found, a [GDDirectory](#) marked as [Null](#) will be returned.

Exceptions

## [ArgumentNullException](#)

The exception is thrown when [string](#) object is null.

## GetFiles(bool)

Gets multiple files in the same directory.

```
public GDFile[] GetFiles(bool isSubdirectory = false)
```

Parameters

### [isSubdirectory](#) [bool](#)

Allows the method to search subdirectories.

Returns

## [GDFile\[\]](#)

If no files are found, an empty list will be returned.

## GetGDDirectory()

Opens an existing directory of the filesystem.

```
public static GDDirectory GetGDDirectory()
```

Returns

## [GDDirectory](#)

## GetGDDirectory(string?)

Opens an existing directory of the filesystem.

```
public static GDDirectory GetGDDirectory(string? path)
```

Parameters

`path` [string](#)

The path argument can be within the project tree (`res://folder`), the user directory (`user://folder`) or an absolute path of the user filesystem (e.g. `/tmp/folder` or `C:\tmp\folder`).

Returns

[GDDirectory](#)

## RemoveDirectory(string?)

Allows you to remove a specific directory.

```
public bool RemoveDirectory(string? directoryName)
```

Parameters

`directoryName` [string](#)

The directory to be removed.

Returns

[bool](#)

Returns `true` if the directory was deleted successfully.

Exceptions

[ArgumentNullException](#)

The exception is thrown when `string` object is null.

## RemoveFile(string?)

Allows the removal of a specific file.

```
public bool RemoveFile(string? fileName)
```

Parameters

[fileName](#) [string](#)

The name of the file to be deleted.

Returns

[bool](#)

Returns [true](#) if the file was deleted successfully.

Exceptions

[ArgumentNullException](#)

The exception is thrown when [string](#) object is null.

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

# Class GDFeature

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class contains some Features pre-defined by the engine.

```
public static class GDFeature
```

## Inheritance

[object](#) ← GDFeature

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Remarks

Features are an alternative to the .Net preprocessing definitions as Godot 3.5's GDScript does not support preprocessing definitions.

## Properties

### HasARM32

Running on a 32-bit ARM build.

```
public static bool HasARM32 { get; }
```

### Property Value

[bool](#)

### HasARM64

Running on a 64-bit ARM build.

```
public static bool HasARM64 { get; }
```

Property Value

[bool](#) ↗

## HasAndroid

Running on Android.

```
public static bool HasAndroid { get; }
```

Property Value

[bool](#) ↗

## HasDebug

Running on a debug build (including the editor).

```
public static bool HasDebug { get; }
```

Property Value

[bool](#) ↗

## HasETC1

Textures using ETC1 compression are supported.

```
public static bool HasETC1 { get; }
```

Property Value

[bool](#) ↗

## HasETC2

Textures using ETC2 compression are supported.

```
public static bool HasETC2 { get; }
```

Property Value

[bool](#) ↗

## HasEditor

Running on an editor build.

```
public static bool HasEditor { get; }
```

Property Value

[bool](#) ↗

## HasHTML5

Running on HTML5.

```
public static bool HasHTML5 { get; }
```

Property Value

[bool](#) ↗

## HasIOS

Running on iOS.

```
public static bool HasIOS { get; }
```

Property Value

[bool](#) ↗

## HasJavaScript

JavaScript singleton is available.

```
public static bool HasJavaScript { get; }
```

Property Value

[bool](#) ↗

## HasMobile

Host OS is a mobile platform.

```
public static bool HasMobile { get; }
```

Property Value

[bool](#) ↗

## HasOSX

Running on macOS.

```
public static bool HasOSX { get; }
```

Property Value

[bool](#) ↗

## HasPC

Host OS is a PC platform (desktop/laptop).

```
public static bool HasPC { get; }
```

Property Value

[bool](#) ↗

## HasPVRTC

Textures using PVRTC compression are supported.

```
public static bool HasPVRTC { get; }
```

Property Value

[bool](#) ↗

## HasRelease

Running on a release build.

```
public static bool HasRelease { get; }
```

Property Value

[bool](#) ↗

## HasS3TC

Textures using S3TC (DXT/BC) compression are supported.

```
public static bool HasS3TC { get; }
```

Property Value

[bool](#)

## HasServer

Running on the headless server platform.

```
public static bool HasServer { get; }
```

Property Value

[bool](#)

## HasStandalone

Running on a non-editor build.

```
public static bool HasStandalone { get; }
```

Property Value

[bool](#)

## HasUWP

Running on UWP.

```
public static bool HasUWP { get; }
```

Property Value

[bool](#)

## HasWeb

Host OS is a Web browser.

```
public static bool HasWeb { get; }
```

Property Value

[bool](#) ↗

## HasWindows

Running on Windows.

```
public static bool HasWindows { get; }
```

Property Value

[bool](#) ↗

## HasX11

Running on X11 (Linux/BSD desktop).

```
public static bool HasX11 { get; }
```

Property Value

[bool](#) ↗

## HasX32

Running on a 32-bit build (any architecture).

```
public static bool HasX32 { get; }
```

Property Value

[bool](#) ↗

## HasX64

Running on a 64-bit build (any architecture).

```
public static bool HasX64 { get; }
```

Property Value

[bool](#) ↗

## HasX86\_32

Running on a 32-bit x86 build.

```
public static bool HasX86_32 { get; }
```

Property Value

[bool](#) ↗

## HasX86\_64

Running on a 64-bit x86 build.

```
public static bool HasX86_64 { get; }
```

Property Value

[bool](#) ↗

## Methods

### HasFeature(string?)

Can be used to check whether you're currently running a debug build, on a certain platform or arch, etc.  
Refer to the [Feature Tags](#) ↗ documentation for more details.

```
public static bool HasFeature(string? tagName)
```

## Parameters

tagName [string](#)

Tag names are case-sensitive.

## Returns

[bool](#)

]Returns [true](#) if the feature for the given feature tag is supported in the currently running instance, depending on the platform, build etc.

# Class GDFile

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class is a representation of a file.

```
public class GDFile : GDFileBase, IDisposable
```

## Inheritance

[object](#) ← [GDFileBase](#) ← GDFile

## Implements

[IDisposable](#)

## Inherited Members

[GDFileBase.Null](#) , [object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

# Properties

## Attribute

The type of the item.

```
public override GDFileAttributes Attribute { get; protected set; }
```

## Property Value

[GDFileAttributes](#)

Returns the type of attribute the item has.

**Null**: Indicates that the item is a representation of a null file.

**File**: Indicates that the item is a representation of a file.

**Directory**: Indicates that the item is a representation of a directory file.

## Name

Item name.

```
public override string Name { get; }
```

### Property Value

[string](#)

Returns the name of the item.

## NameWithoutExtension

The name of the item without extension.

```
public override string NameWithoutExtension { get; }
```

### Property Value

[string](#)

Returns the item name without its extension.

## Parent

The item's parent [GDFFileBase](#).

```
public override GDFFileBase Parent { get; protected set; }
```

### Property Value

[GDFFileBase](#)

Returns the parent GDFFileBase that the item is affiliated with.

## Path

The path of the item.

```
public override string Path { get; protected set; }
```

Property Value

[string](#)

Returns the full or relative path of the item.

## Methods

**Dispose()**

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public override void Dispose()
```

**Dispose(bool)**

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
protected virtual void Dispose(bool disposing)
```

Parameters

**disposing** [bool](#)

When the value is true the object discards the manageable resources.

**~GDFile()**

The finalizer allows the disposal of unmanageable code.

```
protected ~GDFile()
```

## Load()

Loads a resource at the given [path](#), caching the result for further access.

```
public Resource Load()
```

Returns

Resource

Returns an empty resource if no Godot.ResourceFormatLoader could handle the file.

Remarks

The registered Godot.ResourceFormatLoader are queried sequentially to find the first one which can handle the file's extension, and then attempt loading. If loading fails, the remaining ResourceFormatLoaders are also attempted.

## Load(string, bool)

Loads a resource at the given [path](#), caching the result for further access.

```
public Resource Load(string typeHint = "", bool noCache = false)
```

Parameters

[typeHint](#) [string](#)

An optional [type\\_hint](#) can be used to further specify the Godot.Resource type that should be handled by the Godot.ResourceFormatLoader. Anything that inherits from Godot.Resource can be used as a type hint, for example Godot.Image.

[noCache](#) [bool](#)

If [no\\_cache](#) is [true](#), the resource cache will be bypassed and the resource will be loaded anew. Otherwise, the cached resource will be returned if it exists.

Returns

Resource

Returns an empty resource if no Godot.ResourceFormatLoader could handle the file.

## Remarks

The registered Godot.ResourceFormatLoader are queried sequentially to find the first one which can handle the file's extension, and then attempt loading. If loading fails, the remaining ResourceFormatLoaders are also attempted.

## Load<T>()

Loads a resource at the given [path](#), caching the result for further access.

```
public T Load<T>() where T : class
```

## Returns

T

Returns an empty resource if no Godot.ResourceFormatLoader could handle the file.

## Type Parameters

T

The type to cast to. Should be a descendant of Godot.Resource.

## Remarks

The registered Godot.ResourceFormatLoader are queried sequentially to find the first one which can handle the file's extension, and then attempt loading. If loading fails, the remaining ResourceFormatLoaders are also attempted.

## Exceptions

[InvalidCastException](#)

Thrown when the given the loaded resource can't be casted to the given type T.

## Load<T>(string, bool)

Loads a resource at the given `path`, caching the result for further access.

```
public T Load<T>(string typeHint = "", bool noCache = false) where T : class
```

## Parameters

`typeHint` [string](#)

An optional `type_hint` can be used to further specify the Godot.Resource type that should be handled by the Godot.ResourceFormatLoader. Anything that inherits from Godot.Resource can be used as a type hint, for example Godot.Image.

`noCache` [bool](#)

If `no_cache` is `true`, the resource cache will be bypassed and the resource will be loaded anew. Otherwise, the cached resource will be returned if it exists.

## Returns

`T`

Returns an empty resource if no Godot.ResourceFormatLoader could handle the file.

## Type Parameters

`T`

The type to cast to. Should be a descendant of Godot.Resource.

## Remarks

The registered Godot.ResourceFormatLoader are queried sequentially to find the first one which can handle the file's extension, and then attempt loading. If loading fails, the remaining ResourceFormatLoaders are also attempted.

## Exceptions

[InvalidCastException](#)

Thrown when the given the loaded resource can't be casted to the given type `T`.

## Read()

Allows reading of the file.

```
public string Read()
```

Returns

string

Returns the list of bytes read from the file converted to a string using the UTF-8 decoder.

## Write(byte[]?)

Allows you to write data into the file.

```
public void Write(byte[]? buffer)
```

Parameters

buffer byte[][]

The list of bytes that will be written into the file.

# Enum GDFileAttributes

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Represents the file attributes.

```
public enum GDFileAttributes : byte
```

## Fields

**Directory** = 0

Indicates that it is a directory file.

**File** = 1

Indicates that it is a file.

**Null** = 3

Indicates that it is a null item(Exe: [GDIONull](#)).

# Class GDFileBase

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This is a base class for other classes that represent files or directory files.

```
public abstract class GDFileBase : IDisposable
```

Inheritance

[object](#) ← GDFileBase

Implements

[IDisposable](#)

Derived

[GDDirectory](#), [GDFile](#), [GDIONull](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Properties

### Attribute

The type of the item.

```
public abstract GDFileAttributes Attribute { get; protected set; }
```

### Property Value

[GDFileAttributes](#)

Returns the type of attribute the item has.

**Null**: Indicates that the item is a representation of a null file.

**File**: Indicates that the item is a representation of a file.

**Directory**: Indicates that the item is a representation of a directory file.

## Name

Item name.

```
public abstract string Name { get; }
```

## Property Value

[string](#)

Returns the name of the item.

## NameWithoutExtension

The name of the item without extension.

```
public abstract string NameWithoutExtension { get; }
```

## Property Value

[string](#)

Returns the item name without its extension.

## Null

A null item.

```
public static GDFFileBase Null { get; }
```

## Property Value

[GDFFileBase](#)

Returns a representation of a null item.

## Parent

The item's parent [GDFFileBase](#).

```
public abstract GDFFileBase Parent { get; protected set; }
```

### Property Value

[GDFFileBase](#)

Returns the parent GDFFileBase that the item is affiliated with.

## Path

The path of the item.

```
public abstract string Path { get; protected set; }
```

### Property Value

[string](#) ↗

Returns the full or relative path of the item.

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public abstract void Dispose()
```

# Class GDIONull

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class is a representation of a null file.

```
public sealed class GDIONull : GDFFileBase, IDisposable
```

## Inheritance

[object](#) ← [GDFFileBase](#) ← GDIONull

## Implements

[IDisposable](#)

## Inherited Members

[GDFFileBase.Null](#) , [object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

# Properties

## Attribute

The type of the item.

```
public override GDFFileAttributes Attribute { get; protected set; }
```

## Property Value

[GDFFileAttributes](#)

Returns the type of attribute the item has.

**Null**: Indicates that the item is a representation of a null file.

**File**: Indicates that the item is a representation of a file.

**Directory**: Indicates that the item is a representation of a directory file.

## DirectoryNull

Represents a null directory file.

```
public static GDDirectory DirectoryNull { get; }
```

### Property Value

[GDDirectory](#)

Returns a representation of a null directory file.

## FileNull

Represents a null file.

```
public static GDFile FileNull { get; }
```

### Property Value

[GDFile](#)

Returns a representation of a null file.

## Name

Item name.

```
public override string Name { get; }
```

### Property Value

[string](#)

Returns the name of the item.

## NameWithoutExtension

The name of the item without extension.

```
public override string NameWithoutExtension { get; }
```

## Property Value

[string ↗](#)

Returns the item name without its extension.

## Parent

The item's parent [GDFFileBase](#).

```
public override GDFFileBase Parent { get; protected set; }
```

## Property Value

[GDFFileBase](#)

Returns the parent GDFFileBase that the item is affiliated with.

## Path

The path of the item.

```
public override string Path { get; protected set; }
```

## Property Value

[string ↗](#)

Returns the full or relative path of the item.

## Methods

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public override void Dispose()
```

# Class Gizmos

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Gizmos are used to give visual debugging or setup aids in the Scene view.

```
[RunTimeInitializationClass("Gizmos", Priority.StartLater, 0, false)]
public class Gizmos : CanvasLayer, IDisposable
```

## Inheritance

[Object](#) ← Object ← Node ← CanvasLayer ← Gizmos

## Implements

[IDisposable](#)

## Inherited Members

[CanvasLayer.SetLayer\(int\)](#) , [CanvasLayer.GetLayer\(\)](#) , [CanvasLayer.SetVisible\(bool\)](#) ,  
[CanvasLayer.IsVisible\(\)](#) , [CanvasLayer.Show\(\)](#) , [CanvasLayer.Hide\(\)](#) ,  
[CanvasLayer.SetTransform\(Transform2D\)](#) , [CanvasLayer.GetTransform\(\)](#) , [CanvasLayer.GetFinalTransform\(\)](#) ,  
[CanvasLayer.SetOffset\(Vector2\)](#) , [CanvasLayer.GetOffset\(\)](#) , [CanvasLayer.SetRotation\(float\)](#) ,  
[CanvasLayer.GetRotation\(\)](#) , [CanvasLayer.SetRotationDegrees\(float\)](#) ,  
[CanvasLayer.GetRotationDegrees\(\)](#) , [CanvasLayer.setScale\(Vector2\)](#) , [CanvasLayer.GetScale\(\)](#) ,  
[CanvasLayer.SetFollowViewport\(bool\)](#) , [CanvasLayer.IsFollowingViewport\(\)](#) ,  
[CanvasLayer.SetFollowViewportScale\(float\)](#) , [CanvasLayer.GetFollowViewportScale\(\)](#) ,  
[CanvasLayer.SetCustomViewport\(Node\)](#) , [CanvasLayer.GetCustomViewport\(\)](#) , [CanvasLayer.GetCanvas\(\)](#) ,  
[CanvasLayer.Layer](#) , [CanvasLayer.Visible](#) , [CanvasLayer.Offset](#) , [CanvasLayer.RotationDegrees](#) ,  
[CanvasLayer.Rotation](#) , [CanvasLayer.Scale](#) , [CanvasLayer.Transform](#) , [CanvasLayer.CustomViewport](#) ,  
[CanvasLayer.FollowViewportEnable](#) , [CanvasLayer.FollowViewportScale](#) , [Node.NotificationEnterTree](#) ,  
[Node.NotificationExitTree](#) , [Node.NotificationMovedInParent](#) , [Node.NotificationReady](#) ,  
[Node.NotificationPaused](#) , [Node.NotificationUnpaused](#) , [Node.NotificationPhysicsProcess](#) ,  
[Node.NotificationProcess](#) , [Node.NotificationParented](#) , [Node.NotificationUnparented](#) ,  
[Node.NotificationInstanced](#) , [Node.NotificationDragBegin](#) , [Node.NotificationDragEnd](#) ,  
[Node.NotificationPathChanged](#) , [Node.NotificationInternalProcess](#) ,  
[Node.NotificationInternalPhysicsProcess](#) , [Node.NotificationPostEnterTree](#) ,  
[Node.NotificationResetPhysicsInterpolation](#) , [Node.NotificationWmMouseEnter](#) ,  
[Node.NotificationWmMouseExit](#) , [Node.NotificationWmFocusIn](#) , [Node.NotificationWmFocusOut](#) ,  
[Node.NotificationWmQuitRequest](#) , [Node.NotificationWmGoBackRequest](#) ,  
[Node.NotificationWmUnfocusRequest](#) , [Node.NotificationOsMemoryWarning](#) ,  
[Node.NotificationTranslationChanged](#) , [Node.NotificationWmAbout](#) , [Node.NotificationCrash](#) ,

Node.NotificationOsImeUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrBeNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , [Node\\_PhysicsProcess\(float\)](#) ,  
Node.\_UnhandledInput(InputEvent) , Node.\_UnhandledKeyInput(InputEventKey) ,  
[Node.AddChildBelowNode\(Node, Node, bool\)](#) , [Node.SetName\(string\)](#) , Node.GetName() ,  
[Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) , Node.GetChildCount() , Node.GetChildren() ,  
[Node.GetChild\(int\)](#) , Node.HasNode(NodePath) , Node.GetNode(NodePath) ,  
Node.GetNodeOrNull(NodePath) , Node.GetParent() , [Node.FindNode\(string, bool, bool\)](#) ,  
[Node.FindParent\(string\)](#) , Node.HasNodeAndResource(NodePath) ,  
Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() , Node.IsAParentOf(Node) ,  
Node.IsGreaterThan(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDeltaTime() , Node.IsPhysicsProcessing() , Node.GetProcessDeltaTime() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetScenelInstanceLoadPlaceholder\(bool\)](#) , Node.GetScenelInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,

[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostinitialize ,  
Object.NotificationPredelete , Object.IsValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,  
[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Callv\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#)

# Properties

## Color

Sets the Color of the gizmos that are drawn next.

```
public static Color Color { get; set; }
```

## Property Value

Color

Returns or sets the color of the next gizmo.

# Methods

## DrawArc(Vector2, float, float, float, int)

Draws a unfilled arc between the given angles. The larger the value of `point_count`, the smoother the curve. See also [DrawCircle\(Vector2, float, Color\)](#).

Note: Line drawing is not accelerated by batching if `antialiased` is `true`.

Note: Due to how it works, built-in antialiasing will not look correct for translucent lines and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedRegularPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawArc(Vector2 center, float radius, float startAngle, float endAngle,  
int pointCount)
```

## Parameters

`center` Vector2

The central position of the arc.

`radius` [float](#)

The radius of the arc.

### **startAngle** [float](#)

The angle at which the arc will be drawn.

### **endAngle** [float](#)

The angle where the arc will finish being drawn.

### **pointCount** [int](#)

The number of stitches used in the arc.

This parameter will define the visual aspect of the bow, whether it will have a smoother or more serrated curve.

## DrawArc(Vector2, float, float, float, int, float)

Draws a unfilled arc between the given angles. The larger the value of [point\\_count](#), the smoother the curve. See also [DrawCircle\(Vector2, float, Color\)](#).

Note: Line drawing is not accelerated by batching if [antialiased](#) is true.

Note: Due to how it works, built-in antialiasing will not look correct for translucent lines and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedRegularPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawArc(Vector2 center, float radius, float startAngle, float endAngle, int pointCount, float width)
```

## Parameters

### **center** Vector2

The central position of the arc.

### **radius** [float](#)

The radius of the arc.

### **startAngle** [float](#)

The angle at which the arc will be drawn.

### **endAngle** [float](#)

The angle where the arc will finish being drawn.

### **pointCount** [int](#)

The number of stitches used in the arc.

This parameter will define the visual aspect of the bow, whether it will have a smoother or more serrated curve.

### **width** [float](#)

The thickness of the line.

## DrawCircle(Vector2, float)

Draws a colored, filled circle. See also [DrawArc\(Vector2, float, float, float, int, Color, float, bool\)](#), [DrawPolyline\(Vector2\[\], Color, float, bool\)](#) and [DrawPolygon\(Vector2\[\], Color\[\], Vector2\[\], Texture, Texture, bool\)](#).

Note: Built-in antialiasing is not provided for [DrawCircle\(Vector2, float, Color\)](#). As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedRegularPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing.

```
public static void DrawCircle(Vector2 position, float radius)
```

## Parameters

### **position** Vector2

The central position of the circle.

### **radius** [float](#)

The radius of the circle.

## DrawLine(Vector2, Vector2)

Draws a line from a 2D point to another, with a given color and width. It can be optionally antialiased. See also [DrawMultiline\(Vector2\[\], Color, float, bool\)](#) and [DrawPolyline\(Vector2\[\], Color, float, bool\)](#).

Note: Line drawing is not accelerated by batching if `antialiased` is `true`.

Note: Due to how it works, built-in antialiasing will not look correct for translucent lines and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an `AntialiasedLine2D` node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawLine(Vector2 start, Vector2 end)
```

## Parameters

**start** `Vector2`

The beginning of the line.

**end** `Vector2`

The end of the line.

## DrawLine(Vector2, Vector2, float)

Draws a line from a 2D point to another, with a given color and width. It can be optionally antialiased. See also [DrawMultiline\(Vector2\[\], Color, float, bool\)](#) and [DrawPolyline\(Vector2\[\], Color, float, bool\)](#).

Note: Line drawing is not accelerated by batching if `antialiased` is `true`.

Note: Due to how it works, built-in antialiasing will not look correct for translucent lines and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an `AntialiasedLine2D` node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawLine(Vector2 start, Vector2 end, float width)
```

## Parameters

**start** `Vector2`

The beginning of the line.

**end** `Vector2`

The end of the line.

**width** [float](#)

The thickness of the line.

## DrawMesh(Mesh, Texture, Texture?, Transform2D?)

Draws a Godot.Mesh in 2D, using the provided texture. See [Godot.MeshInstance2D](#) for related documentation.

```
public static void DrawMesh(Mesh mesh, Texture texture, Texture? normalMap = null,  
Transform2D? transform = null)
```

### Parameters

**mesh** Mesh

The mesh that will be used to shape the design.

**texture** Texture

The texture that will be used to draw on the mesh.

**normalMap** Texture

The texture normal map.

**transform** Transform2D?

If the parameter is null, then the default value is `Transform2D.Identity`

## DrawMultiline(Vector2[])

Draws multiple disconnected lines with a uniform [color](#). When drawing large amounts of lines, this is faster than using individual [DrawLine\(Vector2, Vector2, Color, float, bool\)](#) calls. To draw interconnected lines, use [DrawPolyline\(Vector2\[\], Color, float, bool\)](#) instead.

Note: [width](#) and [antialiased](#) are currently not implemented and have no effect. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedLine2D node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawMultiline(Vector2[] points)
```

## Parameters

**points** Vector2[]

The points that form the line.

## DrawMultiline(Vector2[], float)

Draws multiple disconnected lines with a uniform **color**. When drawing large amounts of lines, this is faster than using individual [DrawLine\(Vector2, Vector2, Color, float, bool\)](#) calls. To draw interconnected lines, use [DrawPolyline\(Vector2\[\], Color, float, bool\)](#) instead.

Note: **width** and **antialiased** are currently not implemented and have no effect. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedLine2D node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawMultiline(Vector2[] points, float width)
```

## Parameters

**points** Vector2[]

The points that form the line.

**width** [float](#)

The thickness of the line.

## DrawMultiline(List<Vector2>)

Draws multiple disconnected lines with a uniform **color**. When drawing large amounts of lines, this is faster than using individual [DrawLine\(Vector2, Vector2, Color, float, bool\)](#) calls. To draw interconnected lines, use [DrawPolyline\(Vector2\[\], Color, float, bool\)](#) instead.

Note: **width** and **antialiased** are currently not implemented and have no effect. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedLine2D node. That node relies on a texture

with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawMultiline(List<Vector2> points)
```

## Parameters

**points** [List](#)<Vector2>

The points that form the line.

## DrawMultiline(List<Vector2>, float)

Draws multiple disconnected lines with a uniform [color](#). When drawing large amounts of lines, this is faster than using individual [DrawLine\(Vector2, Vector2, Color, float, bool\)](#) calls. To draw interconnected lines, use [DrawPolyline\(Vector2\[\], Color, float, bool\)](#) instead.

Note: [width](#) and [antialiased](#) are currently not implemented and have no effect. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedLine2D node. That node relies on a texture with custom mipmaps to perform antialiasing. 2D batching is also still supported with those antialiased lines.

```
public static void DrawMultiline(List<Vector2> points, float width)
```

## Parameters

**points** [List](#)<Vector2>

The points that form the line.

**width** [float](#)

The thickness of the line.

## DrawRect(Rect2)

Draws a solid body rectangle.

Note: Due to how it works, built-in antialiasing will not look correct for translucent polygons and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an

AntialiasedPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing.

```
public static void DrawRect(Rect2 rect)
```

## Parameters

**rect** Rect2

The dimensions of the rectangle.

## DrawTexture(Texture, Vector2, Texture?)

Draws a texture at a given position.

```
public static void DrawTexture(Texture texture, Vector2 position, Texture? normalMap = null)
```

## Parameters

**texture** Texture

**position** Vector2

**normalMap** Texture

## DrawTextureRect(Texture, Rect2, bool, bool, Texture?)

Draws a textured rectangle at a given position, optionally modulated by a color. If **transpose** is **true**, the texture will have its X and Y coordinates swapped.

```
public static void DrawTextureRect(Texture texture, Rect2 rect, bool tile, bool transpose = false, Texture? normalMap = null)
```

## Parameters

**texture** Texture

**rect** Rect2

**tile** bool

`transpose` [bool](#)

`normalMap` [Texture](#)

## DrawWireRect(Rect2)

Draw a rectangle of a skeletonized body.

Note: Due to how it works, built-in antialiasing will not look correct for translucent polygons and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing.

```
public static void DrawWireRect(Rect2 rect)
```

### Parameters

`rect` [Rect2](#)

The dimensions of the rectangle.

## DrawWireRect(Rect2, float)

Draw a rectangle of a skeletonized body.

Note: Due to how it works, built-in antialiasing will not look correct for translucent polygons and may not work on certain platforms. As a workaround, install the [Antialiased Line2D](#) add-on then create an AntialiasedPolygon2D node. That node relies on a texture with custom mipmaps to perform antialiasing.

```
public static void DrawWireRect(Rect2 rect, float width)
```

### Parameters

`rect` [Rect2](#)

The dimensions of the rectangle.

`width` [float](#)

The thickness of the line.

## \_Process(float)

Called during the processing step of the main loop. Processing happens at every frame and as fast as possible, so the `delta` time since the previous frame is not constant. `delta` is in seconds.

It is only called if processing is enabled, which is done automatically if this method is overridden, and can be toggled with [SetProcess\(bool\)](#).

Corresponds to the Godot.Node.NotificationProcess notification in [Notification\(int\)](#).

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

```
public override void _Process(float delta)
```

## Parameters

`delta` [float](#)

## \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their Godot.Node.\_Ready() callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the Godot.Node.NotificationReady notification in [Notification\(int\)](#). See also the `onready` keyword for variables.

Usually used for initialization. For even earlier initialization, may be used. See also Godot.Node.\_EnterTree().

Note: Godot.Node.\_Ready() may be called only once for each node. After removing a node from the scene tree and adding it again, `_ready` will not be called a second time. This can be bypassed by requesting another call with Godot.Node.RequestReady(), which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Interface IYieldCoroutine

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

A base interface for all Yield class.

```
public interface IYieldCoroutine
```

## Properties

### Delay

The delay of the Yield class.

```
TimeSpan Delay { get; }
```

Property Value

[TimeSpan](#)

Returns a delay value in the form of [TimeSpan](#).

### IsLastCoroutine

Indicates that it is the last to be executed.

```
bool IsLastCoroutine { get; }
```

Property Value

[bool](#)

Returns **true** when class Yield is marked to run last.

# Interface IYieldFixedUpdate

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Yield Class to be excited in the [PhysicsProcess\(float\)](#)

```
public interface IYieldFixedUpdate : IYieldCoroutine
```

## Inherited Members

[IYieldCoroutine.Delay](#) , [IYieldCoroutine.IsLastCoroutine](#)

# Interface IYieldUpdate

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Yield Class to be excited in the [Process\(float\)](#)

```
public interface IYieldUpdate : IYieldCoroutine
```

## Inherited Members

[IYieldCoroutine.Delay](#) , [IYieldCoroutine.IsLastCoroutine](#)

# Interface IYieldVolatile

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

The IYieldVolatile interface allows the Yield class to change the type of process.

This interface allows you to change the type of update if the object will use the [Process\(float\)](#) or [PhysicsProcess\(float\)](#) [Coroutine](#) process.

```
public interface IYieldVolatile : IYieldCoroutine
```

## Inherited Members

[IYieldCoroutine.Delay](#) , [IYieldCoroutine.IsLastCoroutine](#)

## Properties

### IsPhysicsProcess

Indicates which process of updating the [Coroutine](#) is using.

```
bool IsPhysicsProcess { get; }
```

#### Property Value

[bool](#)

**true** when the corrotine is using [PhysicsProcess\(float\)](#).

**false** when the corrotine is using [Process\(float\)](#).

# Struct LastFixedRunTimeSecond

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [PhysicsProcess\(float\)](#).

This class allows the coroutine to be called after the methods of updating the current scene.

```
public readonly struct LastFixedRunTimeSecond : IYieldFixedUpdate, IYieldCoroutine
```

## Implements

[IYieldFixedUpdate](#), [IYieldCoroutine](#)

## Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### LastFixedRunTimeSecond(double)

Creates a new instance of this object.

```
public LastFixedRunTimeSecond(double second)
```

## Parameters

**second** [double](#)

# Struct LastRunTimeSecond

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [Process\(float\)](#).

This class allows the coroutine to be called after the methods of updating the current scene.

```
public readonly struct LastRunTimeSecond : IYieldUpdate, IYieldCoroutine
```

Implements

[IYieldUpdate](#), [IYieldCoroutine](#)

Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### LastRunTimeSecond(double)

Creates a new instance of this object.

```
public LastRunTimeSecond(double second)
```

Parameters

**second** [double](#)

# Class NullNode

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

A null representation of the Godot.Node class.

```
public class NullNode : Node, IDisposable
```

## Inheritance

[object](#) ← Object ← Node ← NullNode

## Implements

[IDisposable](#)

## Inherited Members

Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,  
Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOsIMEUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , [Node.PhysicsProcess\(float\)](#) ,  
[Node.Process\(float\)](#) , Node.\_Ready() , Node.\_UnhandledInput(InputEvent) ,  
Node.\_UnhandledKeyInput(InputEventKey) , [Node.AddChildBelowNode\(Node, Node, bool\)](#) ,  
[NodeSetName\(string\)](#) , Node.GetName() , [Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) ,  
Node.GetChildCount() , Node.GetChildren() , [Node.GetChild\(int\)](#) , Node.HasNode(NodePath) ,  
Node.GetNode(NodePath) , Node.GetNodeOrNull(NodePath) , Node.GetParent() ,  
[Node.FindNode\(string, bool, bool\)](#) , [Node.FindParent\(string\)](#) ,  
Node.HasNodeAndResource(NodePath) , Node.GetNodeAndResource(NodePath) , Node.IsInsideTree()

Node.IsAParentOf(Node) , Node.IsGreaterThanOrEqual(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDeltaTime() , Node.IsPhysicsProcessing() , Node.GetProcessDeltaTime() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostInitialize ,  
Object.NotificationPreDelete , Object.IsInstanceValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,

[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Callv\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#).

# Properties

## Null

Null Godot.Node object.

```
public static NullNode Null { get; }
```

## Property Value

[NullNode](#)

This property will return a null representation of Godot.Node.

# Class Randomico

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

The class allows the creation of pseudo random numbers.

```
public static class Randomico
```

## Inheritance

[object](#) ← Randomico

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

# Properties

## BooleanRandom

Less than `0.5f` is false, greater than `0.5f` is true. ([Randomico.value > 0.5f](#))

```
public static bool BooleanRandom { get; }
```

## Property Value

[bool](#)

Returns a [bool](#) value in a pseudo-random manner.

## value

Returns a random number between 0.0 [inclusive] and 1.0 [inclusive] (Read Only).

```
public static double value { get; }
```

## Property Value

### [double](#)

Returns a pseudo-random floating-point number between **0.0** and **1.0**.

## Methods

### ByteList(byte[])

Fills the elements of a specified array of bytes with random numbers.

```
public static void ByteList(byte[] buffer)
```

#### Parameters

##### **buffer** [byte](#)[]

An array of bytes to contain random numbers.

#### Exceptions

##### [ArgumentNullException](#)

buffer is null.

### ByteRange()

Return a random integer number between min [0] and max [255] (ReadOnly).

```
public static byte ByteRange()
```

#### Returns

### [byte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ByteRange(byte)

Return a random integer number between min [0] and max [exclusive] (ReadOnly).

```
public static byte ByteRange(byte max)
```

### Parameters

max [byte](#)

Defines the maximum range of the pseudo-random number.

### Returns

[byte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ByteRange(byte, byte)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static byte ByteRange(byte min, byte max)
```

### Parameters

min [byte](#)

Sets the minimum range of the pseudo-random number.

max [byte](#)

Defines the maximum range of the pseudo-random number.

### Returns

[byte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## DecimalRange()

Return a random float number between min [-79228162514264337593543950335M] and max [79228162514264337593543950335M] (ReadOnly).

```
public static decimal DecimalRange()
```

Returns

[decimal](#)

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## DecimalRange(decimal)

Return a random float number between min [-79228162514264337593543950335M] and max [exclusive] (ReadOnly).

```
public static decimal DecimalRange(decimal max)
```

Parameters

[max](#) [decimal](#)

Defines the maximum range of the pseudo-random number.

Returns

[decimal](#)

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## DecimalRange(decimal, decimal)

Return a random float number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static decimal DecimalRange(decimal min, decimal max)
```

## Parameters

`min decimal` ↗

Sets the minimum range of the pseudo-random number.

`max decimal` ↗

Defines the maximum range of the pseudo-random number.

## Returns

`decimal` ↗

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## DoubleRange()

Return a random float number between min [-1.7976931348623157E+308] and max [1.7976931348623157E+308] (ReadOnly).

```
public static double DoubleRange()
```

## Returns

`double` ↗

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## DoubleRange(double)

Return a random float number between min [-1.7976931348623157E+308] and max [exclusive] (ReadOnly).

```
public static double DoubleRange(double max)
```

## Parameters

`max double` ↗

Defines the maximum range of the pseudo-random number.

Returns

double ↗

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## DoubleRange(double, double)

Return a random float number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static double DoubleRange(double min, double max)
```

Parameters

min double ↗

Sets the minimum range of the pseudo-random number.

max double ↗

Defines the maximum range of the pseudo-random number.

Returns

double ↗

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## FloatRange()

Return a random float number between min [-3.4028235E+38F] and max [3.4028235E+38F] (ReadOnly).

```
public static float FloatRange()
```

Returns

float ↗

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## FloatRange(float)

Return a random float number between min [-3.4028235E+38F] and max [exclusive] (ReadOnly).

```
public static float FloatRange(float max)
```

Parameters

max [float](#)

Defines the maximum range of the pseudo-random number.

Returns

[float](#)

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## FloatRange(float, float)

Return a random float number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static float FloatRange(float min, float max)
```

Parameters

min [float](#)

Sets the minimum range of the pseudo-random number.

max [float](#)

Defines the maximum range of the pseudo-random number.

Returns

[float](#)

Returns a pseudo-random floating-point number according to the range defined in the parameters.

## InitSeed(in int)

Starts a new seed in the pseudo-random number generator.

```
public static void InitSeed(in int seed)
```

Parameters

seed [int](#)

A number used to calculate a starting value for the pseudo-random number sequence.

If a negative number is specified, the absolute value of the number is used.

## IntRange()

Return a random integer number between min [-2147483648] and max [2147483647] (ReadOnly).

```
public static int IntRange()
```

Returns

[int](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## IntRange(int)

Return a random integer number between min [-2147483648] and max [exclusive] (ReadOnly).

```
public static int IntRange(int max)
```

Parameters

max [int](#)

Defines the maximum range of the pseudo-random number.

Returns

[int↗](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## IntRange(int, int)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static int IntRange(int min, int max)
```

Parameters

[min](#) [int↗](#)

Sets the minimum range of the pseudo-random number.

[max](#) [int↗](#)

Defines the maximum range of the pseudo-random number.

Returns

[int↗](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## LongRange()

Return a random integer number between min [-9223372036854775808] and max [9223372036854775807] (ReadOnly).

```
public static long LongRange()
```

Returns

## [long](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### LongRange(long)

Return a random integer number between min [-9223372036854775808] and max [exclusive] (ReadOnly).

```
public static long LongRange(long max)
```

#### Parameters

max [long](#)

Defines the maximum range of the pseudo-random number.

#### Returns

[long](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### LongRange(long, long)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static long LongRange(long min, long max)
```

#### Parameters

min [long](#)

Sets the minimum range of the pseudo-random number.

max [long](#)

Defines the maximum range of the pseudo-random number.

#### Returns

## [long](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## SByteRange()

Return a random integer number between min [-128] and max [127] (ReadOnly).

```
public static sbyte SByteRange()
```

Returns

### [sbyte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## SByteRange(sbyte)

Return a random integer number between min [-128] and max [exclusive] (ReadOnly).

```
public static sbyte SByteRange(sbyte max)
```

Parameters

### [max](#) [sbyte](#)

Defines the maximum range of the pseudo-random number.

Returns

### [sbyte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## SByteRange(sbyte, sbyte)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static sbyte SByteRange(sbyte min, sbyte max)
```

## Parameters

**min** [sbyte](#)

Sets the minimum range of the pseudo-random number.

**max** [sbyte](#)

Defines the maximum range of the pseudo-random number.

## Returns

[sbyte](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ShortRange()

Return a random integer number between min [-32768] and max [32767] (ReadOnly).

```
public static short ShortRange()
```

## Returns

[short](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ShortRange(short)

Return a random integer number between min [-32768] and max [exclusive] (ReadOnly).

```
public static short ShortRange(short max)
```

## Parameters

**max** [short](#)

Defines the maximum range of the pseudo-random number.

Returns

[short](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ShortRange(short, short)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static short ShortRange(short min, short max)
```

Parameters

**min** [short](#)

Sets the minimum range of the pseudo-random number.

**max** [short](#)

Defines the maximum range of the pseudo-random number.

Returns

[short](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

## ULongRange()

Return a random integer number between min [0] and max [18446744073709551615] (ReadOnly).

```
public static ulong ULongRange()
```

Returns

## [ulong](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### ULongRange(ulong)

Return a random integer number between min [0] and max [exclusive] (ReadOnly).

```
public static ulong ULongRange(ulong max)
```

Parameters

**max** [ulong](#)

Defines the maximum range of the pseudo-random number.

Returns

## [ulong](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### ULongRange(ulong, ulong)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static ulong ULongRange(ulong min, ulong max)
```

Parameters

**min** [ulong](#)

Sets the minimum range of the pseudo-random number.

**max** [ulong](#)

Defines the maximum range of the pseudo-random number.

Returns

## [ulong](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### UShortRange()

Return a random integer number between min [0] and max [65535] (ReadOnly).

```
public static ushort UShortRange()
```

Returns

## [ushort](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### UShortRange(ushort)

Return a random integer number between min [0] and max [exclusive] (ReadOnly).

```
public static ushort UShortRange(ushort max)
```

Parameters

## [max](#) [ushort](#)

Defines the maximum range of the pseudo-random number.

Returns

## [ushort](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

### UShortRange(ushort, ushort)

Return a random integer number between min [inclusive] and max [exclusive] (ReadOnly).

```
public static ushort UShortRange(ushort min, ushort max)
```

## Parameters

**min** [ushort](#)

Sets the minimum range of the pseudo-random number.

**max** [ushort](#)

Defines the maximum range of the pseudo-random number.

## Returns

[ushort](#)

Returns a pseudo-random number of integer type according to the range defined in the parameters.

# Struct Resolution

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Stores information about a screen resolution.

```
[Serializable]
public readonly struct Resolution : IEquatable<Resolution>, IEquatable<Vector2D>,
IEquatable<Vector2DInt>, IEquatable<Vector2>, IEquatable<int>
```

Implements

[IEquatable](#)<[Resolution](#)>, [IEquatable](#)<[Vector2D](#)>, [IEquatable](#)<[Vector2DInt](#)>,  
[IEquatable](#)<[Vector2](#)>, [IEquatable](#)<[int](#)>

Inherited Members

[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Constructors

### Resolution(in Vector2D, in int)

Starts a new instance of the object.

```
public Resolution(in Vector2D resolution, in int frequency)
```

Parameters

**resolution** [Vector2D](#)

Screen resolution.

**frequency** [int](#)

Screen frequency.

### Resolution(in Vector2DInt, in int)

Starts a new instance of the object.

```
public Resolution(in Vector2DInt resolution, in int frequency)
```

Parameters

**resolution** [Vector2DInt](#)

Screen resolution.

**frequency** [int](#)

Screen frequency.

## Resolution(in Vector2, in int)

Starts a new instance of the object.

```
public Resolution(in Vector2 resolution, in int frequency)
```

Parameters

**resolution** [Vector2](#)

Screen resolution.

**frequency** [int](#)

Screen frequency.

## Resolution(in int, in int, in int)

Starts a new instance of the object.

```
public Resolution(in int width, in int height, in int frequency)
```

Parameters

**width** [int](#)

The width of the screen.

**height** [int](#)

The height of the screen.

**frequency** [int](#)

Screen frequency.

## Resolution(in float, in float, in int)

Starts a new instance of the object.

```
public Resolution(in float width, in float height, in int frequency)
```

### Parameters

**width** [float](#)

The width of the screen.

**height** [float](#)

The height of the screen.

**frequency** [int](#)

Screen frequency.

## Properties

### Frequency

Screen frequency.

```
public int Frequency { get; }
```

### Property Value

[int ↗](#)

Returns the frequency of this resolution.

## Height

The height of the screen.

```
public float Height { get; }
```

### Property Value

[float ↗](#)

Returns the width of this resolution.

## Width

The width of the screen.

```
public float Width { get; }
```

### Property Value

[float ↗](#)

Returns the height of this resolution.

## Methods

### Equals(Vector2D)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Vector2D other)
```

### Parameters

`other` [Vector2D](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(Vector2DInt)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Vector2DInt other)
```

Parameters

`other` [Vector2DInt](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(Resolution)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Resolution other)
```

Parameters

`other` [Resolution](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(Vector2)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(Vector2 other)
```

Parameters

**other** Vector2

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(int)

Indicates whether the current object is equal to another object of the same type.

```
public bool Equals(int other)
```

Parameters

**other** [int](#)

An object to compare with this object.

Returns

[bool](#)

`true` if the current object is equal to the `other` parameter; otherwise, `false`.

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

### Parameters

`obj` [object](#)

The object to compare with the current instance.

### Returns

[bool](#)

`true` if `obj` and this instance are the same type and represent the same value; otherwise, `false`.

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

### Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

## Operators

### operator ==(Resolution, Resolution)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Resolution A, Resolution B)
```

Parameters

A [Resolution](#)

Object to be compared.

B [Resolution](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

### explicit operator Vector2D(Resolution)

Explicit conversion operator.([Resolution](#) to [Vector2D](#))

```
public static explicit operator Vector2D(Resolution R)
```

Parameters

R [Resolution](#)

Object to be converted.

Returns

[Vector2D](#)

## explicit operator Vector2DInt(Resolution)

Explicit conversion operator.([Resolution](#) to [Vector2DInt](#))

```
public static explicit operator Vector2DInt(Resolution R)
```

Parameters

R [Resolution](#)

Object to be converted.

Returns

[Vector2DInt](#)

## explicit operator Vector2(Resolution)

Explicit conversion operator.([Resolution](#) to Godot.Vector2)

```
public static explicit operator Vector2(Resolution R)
```

Parameters

R [Resolution](#)

Object to be converted.

Returns

Vector2

## explicit operator int(Resolution)

Explicit conversion operator.([Resolution](#) to [int](#))

```
public static explicit operator int(Resolution R)
```

### Parameters

R [Resolution](#)

Object to be converted.

### Returns

[int](#)

## operator !=(Resolution, Resolution)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Resolution A, Resolution B)
```

### Parameters

A [Resolution](#)

Object to be compared.

B [Resolution](#)

Object of comparison.

### Returns

[bool](#)

Returns the result of the comparison.

# Struct RunTimeSecond

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

This class represents a delay in seconds to methods that return [IEnumerator](#) and use the keyword Yield.

This class is performed in the [Process\(float\)](#).

```
public readonly struct RunTimeSecond : IYieldUpdate, IYieldCoroutine
```

Implements

[IYieldUpdate](#), [IYieldCoroutine](#)

Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### RunTimeSecond(double)

Creates a new instance of this object.

```
public RunTimeSecond(double second)
```

Parameters

second [double](#)

# Class Screen

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Gets or changes game screen information.

```
public static class Screen
```

## Inheritance

[object](#) ← Screen

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Properties

### CurrentDisplay

The current screen.

```
public static DisplayInfo CurrentDisplay { get; }
```

#### Property Value

[DisplayInfo](#)

Returns a [DisplayInfo](#) with the information of the current screen.

### CurrentResolution

The current resolution of the game screen.

```
public static Resolution CurrentResolution { get; }
```

## Property Value

### Resolution

Returns the current resolution of the game screen as Vector2D.

## DisplayCount

Number of screens detected.

```
public static int DisplayCount { get; }
```

## Property Value

### int

Returns the number of screens detected when starting the application.

## Displays

Gets all detected screens.

```
public static DisplayInfo[] Displays { get; }
```

## Property Value

### DisplayInfo[]

Returns all screens that were detected.

## Remarks

This property will only return all screens that were detected from the start of the application, if another screen is connected during the execution of the application it will not be detected.

## Mode

Represents current game screen mode.

```
public static ScreenMode Mode { get; set; }
```

## Property Value

### [ScreenMode](#)

The current game screen mode.

## OrphanList

This property contains all custom resolution lists that are not tied to a [DisplayInfo](#).

```
public static CustomResolutionList[] OrphanList { get; }
```

## Property Value

### [CustomResolutionList\[\]](#)

Returns lists of custom resolutions that are not tied to a [DisplayInfo](#).

## Resolutions

Represents the game screen resolutions.

```
public static Resolution[] Resolutions { get; }
```

## Property Value

### [Resolution\[\]](#)

Returns all stored resolutions.

## ScreenRefreshRate

The current frequency of the game screen.

```
public static float ScreenRefreshRate { get; }
```

## Property Value

[float ↗](#)

Returns the current game screen frequency as a floating point.

## Methods

### AddResolution(in float, in float)

Add a custom resolution.

```
public static void AddResolution(in float width, in float height)
```

#### Parameters

[width float ↗](#)

The width of the screen.

[height float ↗](#)

The height of the screen.

### AddResolution(in float, in float, in int)

Add a custom resolution.

```
public static void AddResolution(in float width, in float height, in int refreshRate)
```

#### Parameters

[width float ↗](#)

The width of the screen.

`height` [float](#)

The height of the screen.

`refreshRate` [int](#)

The refresh rate of the monitor.

## SetCurrentDisplay([in int](#))

Defines which screen will be used.

```
public static void SetCurrentDisplay(in int index)
```

### Parameters

`index` [int](#)

The target index of the screen.

## SetResolution([in Resolution](#))

sets the current screen resolution.

```
public static void SetResolution(in Resolution resolution)
```

### Parameters

`resolution` [Resolution](#)

The new screen resolution.

## SetResolution([in Resolution](#), [in ScreenMode](#))

sets the current screen resolution.

```
public static void SetResolution(in Resolution resolution, in ScreenMode mode)
```

## Parameters

### **resolution** [Resolution](#)

The new screen resolution.

### **mode** [ScreenMode](#)

Screen display mode.

## SetResolution(in Vector2)

sets the current screen resolution.

```
public static void SetResolution(in Vector2 size)
```

## Parameters

### **size** Vector2

The size of the screen.

## SetResolution(in Vector2, in ScreenMode)

sets the current screen resolution.

```
public static void SetResolution(in Vector2 size, in ScreenMode mode)
```

## Parameters

### **size** Vector2

The size of the screen.

### **mode** [ScreenMode](#)

Screen display mode.

## SetResolution(in Vector2, in int)

sets the current screen resolution.

```
public static void SetResolution(in Vector2 size, in int refreshRate)
```

### Parameters

**size** Vector2

The size of the screen.

**refreshRate** [int](#)

The refresh rate of the monitor.

## SetResolution(in Vector2, in int, in ScreenMode)

sets the current screen resolution.

```
public static void SetResolution(in Vector2 size, in int refreshRate, in ScreenMode mode)
```

### Parameters

**size** Vector2

The size of the screen.

**refreshRate** [int](#)

The refresh rate of the monitor.

**mode** [ScreenMode](#)

Screen display mode.

### Exceptions

[ArgumentException](#)

description

## SetResolution(in float, in float)

sets the current screen resolution.

```
public static void SetResolution(in float width, in float height)
```

### Parameters

**width** [float](#)

The width of the screen.

**height** [float](#)

The height of the screen.

## SetResolution(in float, in float, in ScreenMode)

sets the current screen resolution.

```
public static void SetResolution(in float width, in float height, in ScreenMode mode)
```

### Parameters

**width** [float](#)

The width of the screen.

**height** [float](#)

The height of the screen.

**mode** [ScreenMode](#)

Screen display mode.

## SetResolution(in float, in float, in int)

sets the current screen resolution.

```
public static void SetResolution(in float width, in float height, in int refreshRate)
```

## Parameters

**width** [float](#)

The width of the screen.

**height** [float](#)

The height of the screen.

**refreshRate** [int](#)

The refresh rate of the monitor.

## SetResolution(**in float**, **in float**, **in int**, **in ScreenMode**)

sets the current screen resolution.

```
public static void SetResolution(in float width, in float height, in int refreshRate, in  
ScreenMode mode)
```

## Parameters

**width** [float](#)

The width of the screen.

**height** [float](#)

The height of the screen.

**refreshRate** [int](#)

The refresh rate of the monitor.

**mode** [ScreenMode](#)

Screen display mode.

# Enum ScreenMode

Namespace: [Cobilas.GodotEngine.Utility](#)

Assembly: com.cobilas.godot.utility.dll

Represents screen modes.

```
public enum ScreenMode : byte
```

## Fields

**Borderless = 1**

This mode will maintain a borderless, non-resizable window.

**Fullscreen = 2**

This mode will make the screen exclusively for the application.

**Resizable = 0**

This mode enables the screen in resizable windowed mode.

# Namespace Cobilas.GodotEngine.Utility.Editor.Serialization

## Classes

### [BuildSerialization](#)

Class allows to build a serialization list of properties of a node class.

### [HidePropertyAttribute](#)

The attribute allows you to hide and save the value of a field or property in the editor.

### [MemberItem](#)

Represents a property or field.

### [NoSerializedProperty](#)

Represents a property that does not have a [PropertyCustom](#).

### [PrimitiveTypeCustom](#)

Allows customization of a primitive type in the inspector.

### [PropertyCustom](#)

Base class for custom properties.

### [PropertyCustomAttribute](#)

This attribute indicates to [BuildSerialization](#) that the [PropertyCustom](#) class belongs to a certain type.

### [PropertyItem](#)

The class stores the information for drawing in the editor.

### [SONull](#)

SerializedObjectNull; Represents a null [SerializedObject](#).

### [SPCNull](#)

Null representation of [PropertyCustom](#).

### [SerializationCache](#)

Class to handle property caching.

### [SerializeFieldAttribute](#)

Base attribute for field and property serialization attributes.

### [SerializedNode](#)

The class is a serialization representation of a node.

## [SerializedObject](#)

Base class for serialization properties.

## [SerializedProperty](#)

Property serialization class in inspector.

## [ShowPropertyAttribute](#)

The attribute allows you to show a field or property in the editor.

# Interfaces

## [ISerializedPropertyManipulation](#)

The interface allows property manipulation.

# Class BuildSerialization

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Class allows to build a serialization list of properties of a node class.

```
public static class BuildSerialization
```

## Inheritance

[object](#) ← BuildSerialization

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

# Methods

## Build(Node)

The method constructs a serialization list of properties for the node.

```
public static SerializedNode? Build(Node node)
```

## Parameters

**node** Node

The node to use.

## Returns

[SerializedNode](#)

Returns a serialization representation of the node.

## IsPropertyCustom(Type)

Checks if the type has a [PropertyCustom](#).

```
public static bool IsPropertyCustom(Type type)
```

### Parameters

**type** [Type](#)

The type to check.

### Returns

[bool](#)

Returns **true** when the type has a [PropertyCustom](#).

# Class HidePropertyAttribute

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

The attribute allows you to hide and save the value of a field or property in the editor.

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field, AllowMultiple = false,
Inherited = false)]
public class HidePropertyAttribute : SerializeFieldAttribute, _Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← [SerializeFieldAttribute](#) ← [HidePropertyAttribute](#)

## Implements

[Attribute](#)

## Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,  
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,  
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,

[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

## Constructors

### HidePropertyAttribute()

Creates a new instance of this object.

```
public HidePropertyAttribute()
```

### HidePropertyAttribute(bool)

Creates a new instance of this object.

```
public HidePropertyAttribute(bool saveInCache)
```

## Parameters

saveInCache [bool](#)

Allows fields and properties that are not normally serialized.

## Properties

### Flags

The flags that will be used to serialize fields and properties.

```
public override PropertyUsageFlags Flags { get; protected set; }
```

## Property Value

PropertyUsageFlags

Returns the flags that will be used to serialize fields and properties.

## SaveInCache

Indicates whether fields and properties are cached.

```
public override bool SaveInCache { get; protected set; }
```

Property Value

[bool](#)

**true** if fields and properties are cached.

# Interface ISerializedPropertyManipulation

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

The interface allows property manipulation.

```
public interface ISerializedPropertyManipulation
```

## Methods

### Get(string?)

The method allows you to get the value of the property.

```
object? Get(string? propertyName)
```

Parameters

propertyName [string](#)

The name of the property.

Returns

[object](#)

Returns the value of the property.

### GetPropertyList()

The method allows you to create a property serialization list.

```
PropertyItem[] GetPropertyList()
```

Returns

## [PropertyItem\[\]](#)

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
bool Set(string? propertyName, object? value)
```

Parameters

propertyName [string](#)

The name of the property.

value [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns [true](#) when the property is changed.

# Class MemberItem

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Represents a property or field.

```
public sealed class MemberItem : INullObject
```

## Inheritance

[object](#) ← MemberItem

## Implements

[INullObject](#)

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

# Properties

## IsHide

Check if the member is hidden from the editor.

```
public bool IsHide { get; }
```

## Property Value

[bool](#)

returns [true](#) when the member is hidden from the editor.

## IsRead

Check if it is reading.

```
public bool IsRead { get; }
```

## Property Value

[bool](#) ↗

Returns **true** if it is read.

## IsSaveCache

Checks if the member value is cacheable.

```
public bool IsSaveCache { get; }
```

## Property Value

[bool](#) ↗

Returns **true** when the member value is cacheable.

## IsStruct

Checks if the member is a **struct**.

```
public bool IsStruct { get; }
```

## Property Value

[bool](#) ↗

Returns **true** when the member is a **struct**.

## IsWrite

Check if it is written.

```
public bool IsWrite { get; }
```

## Property Value

[bool](#)

Returns `true` if written.

## Menber

The field or property.

```
public MemberInfo? Menber { get; set; }
```

## Property Value

[MemberInfo](#)

Receives the field or property.

## Name

The name of the member.

```
public string Name { get; }
```

## Property Value

[string](#)

Returns the name of the member.

## Null

Null representation of [MemberItem](#).

```
public static MemberItem Null { get; }
```

## Property Value

### [MemberItem](#)

Returns a null representation of [MemberItem](#).

## Parent

The parent object of the member.

```
public object? Parent { get; set; }
```

## Property Value

### [object](#)

Receives the parent object of the member.

## TypeMember

The type of member.

```
public Type TypeMember { get; }
```

## Property Value

### [Type](#)

Returns the type of the member.

## Value

The value of the member.

```
public object? Value { get; set; }
```

## Property Value

### object ↗

Returns the member value.

# Class NoSerializedProperty

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Represents a property that does not have a [PropertyCustom](#).

```
public class NoSerializedProperty : SerializedObject, ISerializedPropertyManipulation,
IEnumerable<SerializedObject>, IEnumerable
```

## Inheritance

[object](#) ← [SerializedObject](#) ← NoSerializedProperty

## Implements

[ISerializedPropertyManipulation](#), [IEnumerable](#)<[SerializedObject](#)>, [IEnumerable](#)

## Inherited Members

[SerializedObject.GetPath\(SerializedObject\)](#), [object.ToString\(\)](#), [object.Equals\(object\)](#),  
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

## Constructors

NoSerializedProperty(string, SerializedObject, string)

Creates a new instance of this object.

```
public NoSerializedProperty(string name, SerializedObject parent, string rootNodeId)
```

## Parameters

name [string](#)

parent [SerializedObject](#)

rootNodeId [string](#)

# Properties

## Member

The custom class member.

```
public override MemberItem Member { get; set; }
```

## Property Value

### [MemberItem](#)

Returns the custom class member.

## Name

The name of the object.

```
public override string Name { get; protected set; }
```

## Property Value

### [string](#)

Returns the name of the object.

## Parent

The parent object of the SerializedObject.

```
public override SerializedObject Parent { get; protected set; }
```

## Property Value

### [SerializedObject](#)

Returns the parent object of the SerializedObject.

# PropertyPath

The custom property path.

```
public override string PropertyPath { get; }
```

## Property Value

[string](#)

Returns the path of the custom property.

## RootNodeId

The id of the node object.

```
public override string RootNodeId { get; protected set; }
```

## Property Value

[string](#)

Returns a [string](#) with the id of the node object.

# Methods

## Add(SerializedObject)

Allows you to add a new [SerializedObject](#) object.

```
public void Add(SerializedObject obj)
```

## Parameters

**obj** [SerializedObject](#)

The [SerializedObject](#) object to be added.

## Add(IEnumerable<SerializedObject>)

Allows you to add a new [SerializedObject](#) object.

```
public void Add(IEnumerable<SerializedObject> objs)
```

Parameters

**objs** [IEnumerable](#)<[SerializedObject](#)>

The list of [SerializedObject](#) objects to add.

## Get(string?)

The method allows you to get the value of the property.

```
public override object? Get(string? propertyName)
```

Parameters

**propertyName** [string](#)

The name of the property.

Returns

[object](#)

Returns the value of the property.

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<SerializedObject> GetEnumerator()
```

Returns

## [IEnumerator](#) <SerializedObject>

An enumerator that can be used to iterate through the collection.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public override PropertyItem[] GetPropertyList()
```

Returns

[PropertyItem](#)[]

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public override bool Set(string? propertyName, object? value)
```

Parameters

propertyName [string](#)

The name of the property.

value [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns [true](#) when the property is changed.

# Class PrimitiveTypeCustom

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Allows customization of a primitive type in the inspector.

```
public sealed class PrimitiveTypeCustom : PropertyCustom, ISerializedPropertyManipulation
```

## Inheritance

[object](#) ← [PropertyCustom](#) ← PrimitiveTypeCustom

## Implements

[ISerializedPropertyManipulation](#)

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

# Properties

## IsHide

Check if the member is hidden from the editor.

```
public override bool IsHide { get; }
```

## Property Value

[bool](#)

returns [true](#) when the member is hidden from the editor.

## Member

The property to be manipulated.

```
public override MemberItem Member { get; set; }
```

## Property Value

### [MemberItem](#)

Defines a new member.

## PropertyPath

The path of the property.

```
public override string PropertyPath { get; set; }
```

## Property Value

### [string](#)

Sets the path of the property.

## Methods

### CacheValueToObject(string?, string?)

Converts the value obtained from the cache to a specific value.

```
public override object? CacheValueToObject(string? propertyName, string? value)
```

## Parameters

### [propertyName](#) [string](#)

The name of the property.

### [value](#) [string](#)

The value to be converted.

Returns

## object ↗

Returns the already converted value.

## Get(string?)

The method allows you to get the value of the property.

```
public override object? Get(string? propertyName)
```

Parameters

### propertyName string ↗

The name of the property.

Returns

## object ↗

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public override PropertyItem[] GetPropertyList()
```

Returns

## PropertyItem[]

Returns a property serialization list.

## IsPrimitiveType(Type)

```
public static bool IsPrimitiveType(Type type)
```

Parameters

**type** [Type](#)

Returns

[bool](#)

## Set(string?, object?)

The method allows you to set the property value.

```
public override bool Set(string? propertyName, object? value)
```

Parameters

**propertyName** [string](#)

The name of the property.

**value** [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns **true** when the property is changed.

# Class PropertyCustom

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Base class for custom properties.

```
public abstract class PropertyCustom : ISerializedPropertyManipulation
```

## Inheritance

[object](#) ← PropertyCustom

## Implements

[ISerializedPropertyManipulation](#)

## Derived

[PrimitiveTypeCustom](#), [SPCNull](#)

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

# Properties

## IsHide

Check if the member is hidden from the editor.

```
public abstract bool IsHide { get; }
```

## Property Value

[bool](#)

returns [true](#) when the member is hidden from the editor.

## Member

The property to be manipulated.

```
public abstract MemberItem Member { get; set; }
```

### Property Value

#### [MemberItem](#)

Defines a new member.

### PropertyPath

The path of the property.

```
public abstract string PropertyPath { get; set; }
```

### Property Value

#### [string](#)

Sets the path of the property.

## Methods

### CacheValueToObject(string?, string?)

Converts the value obtained from the cache to a specific value.

```
public abstract object? CacheValueToObject(string? propertyName, string? value)
```

### Parameters

#### [propertyName](#) [string](#)

The name of the property.

**value** [string](#)

The value to be converted.

Returns

[object](#)

Returns the already converted value.

## Get(string?)

The method allows you to get the value of the property.

```
public abstract object? Get(string? propertyName)
```

Parameters

**propertyName** [string](#)

The name of the property.

Returns

[object](#)

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public abstract PropertyItem[] GetPropertyList()
```

Returns

[PropertyItem](#)[]

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public abstract bool Set(string? propertyName, object? value)
```

### Parameters

**propertyName** [string](#)

The name of the property.

**value** [object](#)

The value that will be set in the property.

### Returns

[bool](#)

Returns [true](#) when the property is changed.

# Class PropertyCustomAttribute

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

This attribute indicates to [BuildSerialization](#) that the [PropertyCustom](#) class belongs to a certain type.

```
[AttributeUsage(AttributeTargets.Class, Inherited = false, AllowMultiple = true)]
public sealed class PropertyCustomAttribute : Attribute, _Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← [PropertyCustomAttribute](#)

## Implements

[Attribute](#)

## Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,  
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,  
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,  
[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

# Constructors

## PropertyCustomAttribute(Type)

Creates a new instance of this object.

```
public PropertyCustomAttribute(Type target)
```

Parameters

target [Type](#)

# Properties

## Target

The target type to be customized.

```
public Type Target { get; }
```

Property Value

[Type](#)

Returns the target type.

# Class PropertyItem

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

The class stores the information for drawing in the editor.

```
public sealed class PropertyItem : IDisposable
```

## Inheritance

[object](#) ← PropertyItem

## Implements

[IDisposable](#)

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

## Constructors

PropertyItem(string, Type, PropertyHint, string, PropertyUsageFlags)

Creates a new instance of this object.

```
public PropertyItem(string name, Variant.Type type, PropertyHint hint = PropertyHint.None,  
string hintString = "", PropertyUsageFlags usage = (PropertyUsageFlags)8195)
```

## Parameters

**name** [string](#)

**type** Variant.Type

**hint** PropertyHint

**hintString** [string](#)

**usage** PropertyUsageFlags

## Methods

### Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

### ToDictionary()

Converts [PropertyItem](#) to Godot.Collections.Dictionary.

```
public Dictionary ToDictionary()
```

Returns

Dictionary

Returns [PropertyItem](#) converted to Godot.Collections.Dictionary.

# Class SONull

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

SerializedObjectNull; Represents a null [SerializedObject](#).

```
public sealed class SONull : SerializedObject, ISerializedPropertyManipulation, INullObject
```

## Inheritance

[object](#) ↴ ← [SerializedObject](#) ← SONull

## Implements

[ISerializedPropertyManipulation](#), [INullObject](#)

## Inherited Members

[SerializedObject.GetPath\(SerializedObject\)](#), [object.ToString\(\)](#) ↴ , [object.Equals\(object\)](#) ↴ ,  
[object.Equals\(object, object\)](#) ↴ , [object.ReferenceEquals\(object, object\)](#) ↴ , [object.GetHashCode\(\)](#) ↴ ,  
[object.GetType\(\)](#) ↴

## Constructors

### SONull(string, SerializedObject, string)

Creates a new instance of this object.

```
public SONull(string name, SerializedObject parent, string rootNodeId)
```

## Parameters

**name** [string](#) ↴

**parent** [SerializedObject](#)

**rootNodeId** [string](#) ↴

## Properties

## Member

The custom class member.

```
public override MemberItem Member { get; set; }
```

### Property Value

#### [MemberItem](#)

Returns the custom class member.

## Name

The name of the object.

```
public override string Name { get; protected set; }
```

### Property Value

#### [string](#)

Returns the name of the object.

## Null

Represents a null representation of [SerializedObject](#).

```
public static SONull Null { get; }
```

### Property Value

#### [SONull](#)

Returns a null representation of [SerializedObject](#).

## Parent

The parent object of the SerializedObject.

```
public override SerializedObject Parent { get; protected set; }
```

## PropertyValue

### [SerializedObject](#)

Returns the parent object of the SerializedObject.

## PropertyPath

The custom property path.

```
public override string PropertyPath { get; }
```

## PropertyValue

### [string](#)

Returns the path of the custom property.

## RootNodeId

The id of the node object.

```
public override string RootNodeId { get; protected set; }
```

## PropertyValue

### [string](#)

Returns a [string](#) with the id of the node object.

## Methods

## Get(string?)

The method allows you to get the value of the property.

```
public override object? Get(string? propertyName)
```

Parameters

propertyName [string](#)

The name of the property.

Returns

[object](#)

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public override PropertyItem[] GetPropertyList()
```

Returns

[PropertyItem](#)[]

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public override bool Set(string? propertyName, object? value)
```

Parameters

`propertyName` [string](#)

The name of the property.

`value` [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns `true` when the property is changed.

# Class SPCNull

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Null representation of [PropertyCustom](#).

```
public sealed class SPCNull : PropertyCustom, ISerializedPropertyManipulation, INullObject
```

## Inheritance

[object](#) ← [PropertyCustom](#) ← SPCNull

## Implements

[ISerializedPropertyManipulation](#), [INullObject](#)

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#)

# Properties

## IsHide

Check if the member is hidden from the editor.

```
public override bool IsHide { get; }
```

## Property Value

[bool](#)

returns [true](#) when the member is hidden from the editor.

## Member

The property to be manipulated.

```
public override MemberItem Member { get; set; }
```

## Property Value

### [MemberItem](#)

Defines a new member.

## Null

Null representation of [PropertyCustom](#).

```
public static SPCNull Null { get; }
```

## Property Value

### [SPCNull](#)

Returns a null representation of [PropertyCustom](#).

## PropertyPath

The path of the property.

```
public override string PropertyPath { get; set; }
```

## Property Value

### [string](#)

Sets the path of the property.

## Methods

### CacheValueToObject(string?, string?)

Converts the value obtained from the cache to a specific value.

```
public override object? CacheValueToObject(string? propertyName, string? value)
```

## Parameters

**propertyName** [string](#)

The name of the property.

**value** [string](#)

The value to be converted.

## Returns

[object](#)

Returns the already converted value.

## Get(string?)

The method allows you to get the value of the property.

```
public override object? Get(string? propertyName)
```

## Parameters

**propertyName** [string](#)

The name of the property.

## Returns

[object](#)

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public override PropertyItem[] GetPropertyList()
```

Returns

[PropertyItem\[\]](#)

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public override bool Set(string? propertyName, object? value)
```

Parameters

propertyName [string](#)

The name of the property.

value [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns [true](#) when the property is changed.

# Class SerializationCache

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Class to handle property caching.

```
public static class SerializationCache
```

## Inheritance

[object](#) ← SerializationCache

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Methods

### GetValueInCache(string?, string?, out string)

Gets the value of the property that is cached.

```
public static bool GetValueInCache(string? id, string? propertyName, out string value)
```

#### Parameters

**id** [string](#)

The ID of the cache

**propertyName** [string](#)

The name of the property.

**value** [string](#)

The value of the property that is cached.

Returns

[bool](#)

Returns **true** when the value is retrieved from the cache.

## SetValueInCache(string?, string?, object?)

Caches the property value.

```
public static bool SetValueInCache(string? id, string? propertyName, object? value)
```

Parameters

**id** [string](#)

The ID of the cache

**propertyName** [string](#)

The name of the property.

**value** [object](#)

The property value that will be cached.

Returns

[bool](#)

Returns **true** when the property value is cached.

# Class SerializeFieldAttribute

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Base attribute for field and property serialization attributes.

```
public abstract class SerializeFieldAttribute : Attribute, _Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← SerializeFieldAttribute

## Implements

[Attribute](#)

## Derived

[HidePropertyAttribute](#), [ShowPropertyAttribute](#)

## Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,  
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,  
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,

[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

## Constructors

### SerializeFieldAttribute(PropertyUsageFlags, bool)

Creates a new instance of this object.

```
protected SerializeFieldAttribute(PropertyUsageFlags flags, bool saveInCache)
```

#### Parameters

**flags** PropertyUsageFlags

The flags that will be used to serialize fields and properties.

**saveInCache** [bool](#)

Allows fields and properties that are not normally serialized.

## Properties

### Flags

The flags that will be used to serialize fields and properties.

```
public abstract PropertyUsageFlags Flags { get; protected set; }
```

### Property Value

PropertyUsageFlags

Returns the flags that will be used to serialize fields and properties.

### SaveInCache

Indicates whether fields and properties are cached.

```
public abstract bool SaveInCache { get; protected set; }
```

## Property Value

bool ↗

**true** if fields and properties are cached.

# Class SerializedNode

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

The class is a serialization representation of a node.

```
public class SerializedNode : ISerializedPropertyManipulation
```

## Inheritance

[object](#) ← SerializedNode

## Implements

[ISerializedPropertyManipulation](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

## Constructors

### SerializedNode(string)

Creates a new instance of this object.

```
public SerializedNode(string id)
```

## Parameters

**id** [string](#)

## Properties

### Id

The id of the node object.

```
public string Id { get; }
```

## Property Value

[string ↗](#)

Returns the id of the node object.

## Methods

### Add(SerializedObject)

Allows you to add a new [SerializedObject](#) object.

```
public void Add(SerializedObject obj)
```

#### Parameters

**obj** [SerializedObject](#)

The [SerializedObject](#) object to be added.

### Add(IEnumerable<SerializedObject>)

Allows you to add a new [SerializedObject](#) object.

```
public void Add(IEnumerable<SerializedObject> objs)
```

#### Parameters

**objs** [IEnumerable ↗ <SerializedObject>](#)

The list of [SerializedObject](#) objects to add.

### Get(string?)

The method allows you to get the value of the property.

```
public object? Get(string? propertyName)
```

## Parameters

**propertyName** [string](#)

The name of the property.

## Returns

[object](#)

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public PropertyItem[] GetPropertyList()
```

## Returns

[PropertyItem](#)[]

Returns a property serialization list.

## GetPropertyList(PropertyItem[])

The method converts a list of [PropertyItem](#) to an Godot.Collections.Array

```
public static Array GetPropertyList(PropertyItem[] list)
```

## Parameters

**list** [PropertyItem](#)[]

The list to be converted.

Returns

Array

Returns an Godot.Collections.Array containing the serialization information for the properties.

## GetPropertyList(SerializedNode)

The method converts a list of [PropertyItem](#) to an Godot.Collections.Array

```
public static Array GetPropertyList(SerializedNode node)
```

Parameters

node [SerializedNode](#)

The [SerializedNode](#) that will get the list of [PropertyItem](#) to be converted.

Returns

Array

Returns an Godot.Collections.Array containing the serialization information for the properties.

## Set(string?, object?)

The method allows you to set the property value.

```
public bool Set(string? propertyName, object? value)
```

Parameters

propertyName [string](#)

The name of the property.

value [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns **true** when the property is changed.

## ToString()

Returns a string that represents the current object.

`public override string ToString()`

Returns

[string](#)

A string that represents the current object.

# Class SerializedObject

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Base class for serialization properties.

```
public abstract class SerializedObject : ISerializedPropertyManipulation
```

Inheritance

[object](#) ← SerializedObject

Implements

[ISerializedPropertyManipulation](#)

Derived

[NoSerializedProperty](#), [SONull](#), [SerializedProperty](#)

Inherited Members

[object.ToString\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),  
[object.MemberwiseClone\(\)](#)

## Constructors

### SerializedObject(string, SerializedObject, string)

Creates a new instance of this object.

```
protected SerializedObject(string name, SerializedObject parent, string rootNodeId)
```

Parameters

name [string](#)

parent [SerializedObject](#)

rootNodeId [string](#)

# Properties

## Member

The custom class member.

```
public abstract MemberItem Member { get; set; }
```

## Property Value

### [MemberItem](#)

Returns the custom class member.

## Name

The name of the object.

```
public abstract string Name { get; protected set; }
```

## Property Value

### [string](#)

Returns the name of the object.

## Parent

The parent object of the SerializedObject.

```
public abstract SerializedObject Parent { get; protected set; }
```

## Property Value

### [SerializedObject](#)

Returns the parent object of the SerializedObject.

# PropertyPath

The custom property path.

```
public abstract string PropertyPath { get; }
```

## Property Value

[string](#)

Returns the path of the custom property.

## RootNodeId

The id of the node object.

```
public abstract string RootNodeId { get; protected set; }
```

## Property Value

[string](#)

Returns a [string](#) with the id of the node object.

# Methods

## Get(string?)

The method allows you to get the value of the property.

```
public abstract object? Get(string? propertyName)
```

## Parameters

propertyName [string](#)

The name of the property.

Returns

## object ↗

Returns the value of the property.

## GetPath(SerializedObject)

Allows you to get the path of the [SerializedObject](#).

```
public static string GetPath(SerializedObject obj)
```

Parameters

### obj [SerializedObject](#)

The [SerializedObject](#) that will be used to generate the path.

Returns

## string ↗

Returns a [string](#) containing the path of the [SerializedObject](#).

## GetPropertyList()

The method allows you to create a property serialization list.

```
public abstract PropertyItem[] GetPropertyList()
```

Returns

## [PropertyItem](#)[]

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public abstract bool Set(string? propertyName, object? value)
```

## Parameters

**propertyName** [string](#)

The name of the property.

**value** [object](#)

The value that will be set in the property.

## Returns

[bool](#)

Returns **true** when the property is changed.

# Class SerializedProperty

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

Property serialization class in inspector.

```
public class SerializedProperty : SerializedObject, ISerializedPropertyManipulation
```

## Inheritance

[object](#) ← [SerializedObject](#) ← SerializedProperty

## Implements

[ISerializedPropertyManipulation](#)

## Inherited Members

[SerializedObject.GetPath\(SerializedObject\)](#) , [object.ToString\(\)](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

## Constructors

### SerializedProperty(string, SerializedObject, string)

Creates a new instance of this object.

```
public SerializedProperty(string name, SerializedObject parent, string rootNodeId)
```

## Parameters

name [string](#)

parent [SerializedObject](#)

rootNodeId [string](#)

## Properties

# Custom

Property customization object.

```
public PropertyCustom Custom { get; set; }
```

## Property Value

### [PropertyCustom](#)

Allows you to define a property customization object.

## Member

The custom class member.

```
public override MemberItem Member { get; set; }
```

## Property Value

### [MemberItem](#)

Returns the custom class member.

## Name

The name of the object.

```
public override string Name { get; protected set; }
```

## Property Value

### [string](#) ↗

Returns the name of the object.

## Parent

The parent object of the SerializedObject.

```
public override SerializedObject Parent { get; protected set; }
```

## PropertyValue

### [SerializedObject](#)

Returns the parent object of the SerializedObject.

## PropertyPath

The custom property path.

```
public override string PropertyPath { get; }
```

## PropertyValue

### [string](#)

Returns the path of the custom property.

## RootNodeId

The id of the node object.

```
public override string RootNodeId { get; protected set; }
```

## PropertyValue

### [string](#)

Returns a [string](#) with the id of the node object.

## Methods

## Get(string?)

The method allows you to get the value of the property.

```
public override object? Get(string? propertyName)
```

Parameters

propertyName [string](#)

The name of the property.

Returns

[object](#)

Returns the value of the property.

## GetPropertyList()

The method allows you to create a property serialization list.

```
public override PropertyItem[] GetPropertyList()
```

Returns

[PropertyItem](#)[]

Returns a property serialization list.

## Set(string?, object?)

The method allows you to set the property value.

```
public override bool Set(string? propertyName, object? value)
```

Parameters

**propertyName** [string](#)

The name of the property.

**value** [object](#)

The value that will be set in the property.

Returns

[bool](#)

Returns [true](#) when the property is changed.

# Class ShowPropertyAttribute

Namespace: [Cobilas.GodotEngine.Utility.EditorSerialization](#)

Assembly: com.cobilas.godot.utility.dll

The attribute allows you to show a field or property in the editor.

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field, AllowMultiple = false,
Inherited = false)]
public class ShowPropertyAttribute : SerializeFieldAttribute, _Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← [SerializeFieldAttribute](#) ← ShowPropertyAttribute

## Implements

[Attribute](#)

## Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,  
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,  
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,

[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

## Constructors

### ShowPropertyAttribute()

Creates a new instance of this object.

```
public ShowPropertyAttribute()
```

### ShowPropertyAttribute(bool)

Creates a new instance of this object.

```
public ShowPropertyAttribute(bool saveInCache)
```

## Parameters

### saveInCache [bool](#)

Allows fields and properties that are not normally serialized.

## Properties

### Flags

The flags that will be used to serialize fields and properties.

```
public override PropertyUsageFlags Flags { get; protected set; }
```

## Property Value

### PropertyUsageFlags

Returns the flags that will be used to serialize fields and properties.

## SaveInCache

Indicates whether fields and properties are cached.

```
public override bool SaveInCache { get; protected set; }
```

Property Value

[bool](#)

**true** if fields and properties are cached.

# Namespace Cobilas.GodotEngine.Utility.Input Classes

## [InputKeyBoard](#)

This class has methods and properties that get information from keyboard and mouse inputs.

## Structs

### [PeripheralItem](#)

Responsible for maintaining the status information of a peripheral.

## Enums

### [KeyCode](#)

Represents a list of peripheral input key codes.

### [KeyStatus](#)

represents the state of a key.

### [MouseButton](#)

Represents mouse triggers.

# Class InputKeyBoard

Namespace: [Cobilas.GodotEngine.Utility.Input](#)

Assembly: com.cobilas.godot.utility.dll

This class has methods and properties that get information from keyboard and mouse inputs.

```
[RunTimeInitializationClass("InputKeyBoard", Priority.StartBefore, 0, false)]
public class InputKeyBoard : Node, IDisposable
```

## Inheritance

[Object](#) ← Object ← Node ← InputKeyBoard

## Implements

[IDisposable](#)

## Inherited Members

Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,  
Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOslmeUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrBeNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , [Node.PhysicsProcess\(float\)](#) , [Node.Process\(float\)](#) ,  
Node.\_UnhandledInput(InputEvent) , Node.\_UnhandledKeyInput(InputEventKey) ,  
[Node.AddChildBelowNode\(Node, Node, bool\)](#) , [Node.SetName\(string\)](#) , Node.GetName() ,  
[Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) , Node.GetChildCount() , Node.GetChildren() ,  
[Node.GetChild\(int\)](#) , Node.HasNode(NodePath) , Node.GetNode(NodePath) ,  
Node.GetNodeOrNull(NodePath) , Node.GetParent() , [Node.FindNode\(string, bool, bool\)](#) ,  
[Node.FindParent\(string\)](#) , Node.HasNodeAndResource(NodePath) ,  
Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() , Node.IsAParentOf(Node) ,

Node.IsGreater Than(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDelta Time() , Node.IsPhysicsProcessing() , Node.GetProcessDelta Time() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostInitialize ,  
Object.NotificationPreDelete , Object.IsInstanceValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,

[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Callv\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#).

# Properties

## DeltaScroll

Indicates a change in mouse scroll.

```
public static float DeltaScroll { get; }
```

## Property Value

[float](#)

Returns a floating-point value when the mouse scroll is changed.

## DoubleClick

Detects a double mouse click.

```
public static bool DoubleClick { get; }
```

### Property Value

[bool](#)

Returns true when a double mouse click is detected.

## MouseGlobalPosition

The current global mouse position.

```
public static Vector2D MouseGlobalPosition { get; }
```

### Property Value

[Vector2D](#)

Returns the mouse position based on a root Godot.Viewport.

## MouseIndex

The mouse trigger index.

```
public static int MouseIndex { get; }
```

### Property Value

[int](#)

Returns an [int](#) containing the index of the mouse trigger.

# MousePosition

The current mouse position.

```
public static Vector2D MousePosition { get; }
```

Property Value

[Vector2D](#)

Returns the mouse position based on the defined Godot.Viewport.

## Methods

### GetKeyDown(KeyCode)

Determines whether the key was pressed.

```
public static bool GetKeyDown(KeyCode key)
```

Parameters

**key** [KeyCode](#)

The target key to be verified.

Returns

[bool](#)

Returns [true](#) if the key was pressed.

Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyDown(KeyCode.A))
```

```
        GD.Print("This instruction will only be called each time the key is pressed.");
    }
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyDown(MouseButton)

Determines whether the key was pressed.

```
public static bool GetKeyDown(MouseButton key)
```

## Parameters

**key** [MouseButton](#)

The target key to be verified.

## Returns

[bool](#) ↗

Returns **true** if the key was pressed.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyDown(KeyCode.A))
        GD.Print("This instruction will only be called each time the key is pressed.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyDown(ButtonList)

Determines whether the key was pressed.

```
public static bool GetKeyDown(ButtonList key)
```

### Parameters

**key** ButtonList

The target key to be verified.

### Returns

[bool](#)

Returns **true** if the key was pressed.

### Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyDown(KeyCode.A))
        GD.Print("This instruction will only be called each time the key is pressed.");
}
```

### Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyDown(KeyList)

Determines whether the key was pressed.

```
public static bool GetKeyDown(KeyList key)
```

### Parameters

## key [KeyList](#)

The target key to be verified.

Returns

## [bool](#) ↗

Returns [true](#) if the key was pressed.

Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyDown(KeyCode.A))
        GD.Print("This instruction will only be called each time the key is pressed.");
}
```

Remarks

The method has overloads that allow the use of the [Godot.KeyList](#), [MouseButton](#) and [Godot.ButtonList](#) enumerators.

## GetKeyPress(KeyCode)

Determines whether the key is being pressed.

```
public static bool GetKeyPress(KeyCode key)
```

Parameters

## key [KeyCode](#)

The target key to be verified.

Returns

## [bool](#) ↗

Returns `true` if the key is being pressed.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyBoard.GetKeyPress(KeyCode.A))
        GD.Print("The instruction will be called constantly as long as the key
is pressed.");
}
```

## Remarks

The method has overloads that allow the use of the `Godot.KeyList`, [MouseButton](#) and `Godot.ButtonList` enumerators.

## GetKeyPress(MouseButton)

Determines whether the key is being pressed.

```
public static bool GetKeyPress(MouseButton key)
```

## Parameters

`key` [MouseButton](#)

The target key to be verified.

## Returns

[bool](#) ↗

Returns `true` if the key is being pressed.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;
```

```
public override void _Process(float delta) {
    if (InputKeyboard.GetKeyPress(KeyCode.A))
        GD.Print("The instruction will be called constantly as long as the key
is pressed.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyPress(ButtonList)

Determines whether the key is being pressed.

```
public static bool GetKeyPress(ButtonList key)
```

### Parameters

**key** ButtonList

The target key to be verified.

### Returns

[bool](#)

Returns **true** if the key is being pressed.

### Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyPress(KeyCode.A))
        GD.Print("The instruction will be called constantly as long as the key
is pressed.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyPress(KeyList)

Determines whether the key is being pressed.

```
public static bool GetKeyPress(KeyList key)
```

### Parameters

**key** KeyList

The target key to be verified.

### Returns

[bool](#) ↗

Returns **true** if the key is being pressed.

### Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyPress(KeyCode.A))
        GD.Print("The instruction will be called constantly as long as the key
is pressed.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyUp(KeyCode)

Determines whether the key has been released.

```
public static bool GetKeyUp(KeyCode key)
```

## Parameters

**key** [KeyCode](#)

The target key to be verified.

## Returns

[bool](#) ↗

Returns **true** if the key was released.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyUp(KeyCode.A))
        GD.Print("This instruction will only be called whenever the key is released.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyUp(MouseButton)

Determines whether the key has been released.

```
public static bool GetKeyUp(MouseButton key)
```

## Parameters

**key** [MouseButton](#)

The target key to be verified.

Returns

[bool](#)

Returns `true` if the key was released.

Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyBoard.GetKeyUp(KeyCode.A))
        GD.Print("This instruction will only be called whenever the key is released.");
}
```

Remarks

The method has overloads that allow the use of the `Godot.KeyList`, [MouseButton](#) and `Godot.ButtonList` enumerators.

## GetKeyUp(ButtonList)

Determines whether the key has been released.

```
public static bool GetKeyUp(ButtonList key)
```

Parameters

**key** ButtonList

The target key to be verified.

Returns

[bool](#)

Returns `true` if the key was released.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyUp(KeyCode.A))
        GD.Print("This instruction will only be called whenever the key is released.");
}
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetKeyUp(KeyList)

Determines whether the key has been released.

```
public static bool GetKeyUp(KeyList key)
```

## Parameters

**key** KeyList

The target key to be verified.

## Returns

[bool](#)

Returns **true** if the key was released.

## Examples

```
using Godot;
using Cobilas.GodotEngine.Utility.Input;

public override void _Process(float delta) {
    if (InputKeyboard.GetKeyUp(KeyCode.A))
```

```
        GD.Print("This instruction will only be called whenever the key is released.");
    }
```

## Remarks

The method has overloads that allow the use of the Godot.KeyList, [MouseButton](#) and Godot.ButtonList enumerators.

## GetMouseDown(MouseButton)

Determines whether the mouse trigger was pressed.

```
public static bool GetMouseDown(MouseButton button)
```

### Parameters

**button** [MouseButton](#)

The target mouse trigger to be checked.

### Returns

[bool](#) ↗

Returns **true** if the mouse trigger was pressed.

## GetMouseDown(int)

Determines whether the mouse trigger was pressed.

```
public static bool GetMouseDown(int buttonIndex)
```

### Parameters

**buttonIndex** [int](#) ↗

The target index to be checked.

### Returns

[bool](#)

Returns `true` if the mouse trigger was pressed.

## GetMousePress(MouseButton)

Determines whether the mouse trigger is being pressed.

```
public static bool GetMousePress(MouseButton button)
```

Parameters

`button` [MouseButton](#)

The target mouse trigger to be checked.

Returns

[bool](#)

Returns `true` if the mouse trigger is being pressed.

## GetMousePress(int)

Determines whether the mouse trigger is being pressed.

```
public static bool GetMousePress(int buttonIndex)
```

Parameters

`buttonIndex` [int](#)

The target index to be checked.

Returns

[bool](#)

Returns `true` if the mouse trigger is being pressed.

## GetMouseUp(MouseButton)

Determines whether the mouse trigger has been drop by the user.

```
public static bool GetMouseUp(MouseButton button)
```

Parameters

button [MouseButton](#)

The target mouse trigger to be checked.

Returns

[bool](#)

Returns [true](#) if the mouse trigger was released by the user.

## GetMouseUp(int)

Determines whether the mouse trigger has been drop by the user.

```
public static bool GetMouseUp(int buttonIndex)
```

Parameters

buttonIndex [int](#)

The target index to be checked.

Returns

[bool](#)

Returns [true](#) if the mouse trigger was released by the user.

## \_Input(InputEvent)

Called when there is an input event. The input event propagates up through the node tree until a node consumes it.

It is only called if input processing is enabled, which is done automatically if this method is overridden, and can be toggled with [SetProcessInput\(bool\)](#).

To consume the input event and stop it propagating further to other nodes, Godot.SceneTree.SetInputAsHandled() can be called.

For gameplay input, Godot.Node.\_UnhandledInput(Godot.InputEvent) and Godot.Node.\_UnhandledKeyInput(Godot.InputEventKey) are usually a better fit as they allow the GUI to intercept the events first.

Note: This method is only called if the node is present in the scene tree (i.e. if it's not an orphan).

```
public override void _Input(InputEvent @event)
```

## Parameters

**event** InputEvent

## \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their Godot.Node.\_Ready() callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the Godot.Node.NotificationReady notification in [Notification\(int\)](#). See also the **onready** keyword for variables.

Usually used for initialization. For even earlier initialization, **ready** may be used. See also Godot.Node.\_EnterTree().

Note: Godot.Node.\_Ready() may be called only once for each node. After removing a node from the scene tree and adding it again, **\_ready** will not be called a second time. This can be bypassed by requesting another call with Godot.Node.RequestReady(), which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Enum KeyCode

Namespace: [Cobilas.GodotEngine.Utility.Input](#)

Assembly: com.cobilas.godot.utility.dll

Represents a list of peripheral input key codes.

```
public enum KeyCode : ulong
```

## Fields

**A** = 65

A key.

**Aacute** = 193

Á key.

**Acircumflex** = 194

Â key.

**Acute** = 180

‘ key.

**Adiaeresis** = 196

Ã key.

**Ae** = 198

Æ key.

**Agrave** = 192

À key.

**Alt** = 16777240

Alt key.

**Ampersand** = 38

& key.

**Apostrophe** = 39

' key.

**Aring** = 197

Å key.

**Asciicircum** = 94

^ key.

**Asciitilde** = 126

~ key.

**Asterisk** = 42

\* key.

**At** = 64

@ key.

**Atilde** = 195

Ã key.

**B** = 66

B key.

**Back** = 16777280

Media back key. Not to be confused with the Back button on an Android device.

**Backslash** = 92

\ key.

**Backspace** = 16777220

Backspace key.

**Backtab** = 16777219

Shift+Tab key.

**Bar** = 124

| key.

**Bassboost** = 16777287

Bass Boost key.

**Bassdown** = 16777289

Bass down key.

**Bassup** = 16777288

Bass up key.

**Braceleft** = 123

{ key.

**Braceright** = 125

} key.

**Bracketleft** = 91

[ key.

**Bracketright** = 93

] key.

**Brokenbar** = 166

\ key.

**C** = 67

C key.

**Capslock** = 16777241

Caps Lock key.

**Ccedilla** = 199

Ç key.

Cedilla = 184

ÿ key.

Cent = 162

¢ key.

Clear = 16777228

Clear key.

Colon = 58

: key.

Comma = 44

, key.

Control = 16777238

Control key.

Copyright = 169

© key.

Currency = 164

¤ key.

D = 68

D key.

Degree = 176

° key.

Delete = 16777224

Delete key.

Diaeresis = 168

.. key.

**DirectionL** = 16777266

Left Direction key.

**DirectionR** = 16777267

Right Direction key.

**Division** = 247

÷ key.

**Dollar** = 36

\$ key.

**Down** = 16777234

Down arrow key.

**E** = 69

E key.

**Eacute** = 201

É key.

**Ecircumflex** = 202

Ê key.

**Ediaeresis** = 203

Ë key.

**Egrave** = 200

È key.

**End** = 16777230

End key.

**Enter** = 16777221

**Return** key (on the main keyboard).

**Equal** = 61

= key.

**Escape** = 16777217

Escape key.

**Eth** = 208

Ð key.

**Exclam** = 33

! key.

**Exclamdown** = 161

¡ key.

**F** = 70

F key.

**F1** = 16777244

F1 key.

**F10** = 16777253

F10 key.

**F11** = 16777254

F11 key.

**F12** = 16777255

F12 key.

**F13** = 16777256

F13 key.

**F14** = 16777257

F14 key.

F15 = 16777258

F15 key.

F16 = 16777259

F16 key.

F2 = 16777245

F2 key.

F3 = 16777246

F3 key.

F4 = 16777247

F4 key.

F5 = 16777248

F5 key.

F6 = 16777249

F6 key.

F7 = 16777250

F7 key.

F8 = 16777251

F8 key.

F9 = 16777252

F9 key.

Favorites = 16777298

Favorites key.

Forward = 16777281

Media forward key.

**G** = 71

G key.

**Greater** = 62

> key.

**Guillemotleft** = 171

« key.

**Guillemotright** = 187

» key.

**H** = 72

H key.

**Help** = 16777265

Help key.

**Home** = 16777229

F16 key.

**Homepage** = 16777297

Home page key.

**HyperL** = 16777263

Left Hyper key.

**HyperR** = 16777264

Right Hyper key.

**Hyphen** = 173

Soft hyphen key.

**I** = 73

I key.

Iacute = 205

í key.

Icircumflex = 206

î key.

Idiaeresis = 207

ï key.

Igrave = 204

ì key.

Insert = 16777223

Insert key.

J = 74

J key.

K = 75

K key.

Key0 = 48

Number 0.

Key1 = 49

Number 1.

Key2 = 50

Number 2.

Key3 = 51

Number 3.

Key4 = 52

Number 4.

**Key5 = 53**

Number 5.

**Key6 = 54**

Number 6.

**Key7 = 55**

Number 7.

**Key8 = 56**

Number 8.

**Key9 = 57**

Number 9.

**Kp0 = 16777350**

Number 0 on the numeric keypad.

**Kp1 = 16777351**

Number 1 on the numeric keypad.

**Kp2 = 16777352**

Number 2 on the numeric keypad.

**Kp3 = 16777353**

Number 3 on the numeric keypad.

**Kp4 = 16777354**

Number 4 on the numeric keypad.

**Kp5 = 16777355**

Number 5 on the numeric keypad.

**Kp6 = 16777356**

Number 6 on the numeric keypad.

**Kp7 = 16777357**

Number 7 on the numeric keypad.

**Kp8 = 16777358**

Number 8 on the numeric keypad.

**Kp9 = 16777359**

Number 9 on the numeric keypad.

**KpAdd = 16777349**

Add (+) key on the numeric keypad.

**KpDivide = 16777346**

Divide (/) key on the numeric keypad.

**KpEnter = 16777222**

Enter key on the numeric keypad.

**KpMultiply = 16777345**

Multiply (\*) key on the numeric keypad.

**KpPeriod = 16777348**

Period (.) key on the numeric keypad.

**KpSubtract = 16777347**

Subtract (-) key on the numeric keypad.

**L = 76**

L key.

**Launch0 = 16777304**

Launch Shortcut 0 key.

**Launch1 = 16777305**

Launch Shortcut 1 key.

**Launch2 = 16777306**

Launch Shortcut 2 key.

**Launch3 = 16777307**

Launch Shortcut 3 key.

**Launch4 = 16777308**

Launch Shortcut 4 key.

**Launch5 = 16777309**

Launch Shortcut 5 key.

**Launch6 = 16777310**

Launch Shortcut 6 key.

**Launch7 = 16777311**

Launch Shortcut 7 key.

**Launch8 = 16777312**

Launch Shortcut 8 key.

**Launch9 = 16777313**

Launch Shortcut 9 key.

**Launcha = 16777314**

Launch Shortcut A key.

**Launchb = 16777315**

Launch Shortcut B key.

**Launchc = 16777316**

Launch Shortcut C key.

**Launchd = 16777317**

Launch Shortcut D key.

**Launche** = 16777318

Launch Shortcut E key.

**Launchf** = 16777319

Launch Shortcut F key.

**Launchmail** = 16777302

Launch Mail key.

**Launchmedia** = 16777303

Launch Media key.

**Left** = 16777231

Left arrow key.

**Less** = 60

< key.

**M** = 77

M key.

**Macron** = 175

– key.

**Masculine** = 186

° key.

**Medianext** = 16777295

Next song key.

**Mediaplay** = 16777292

Media play key.

**Mediaprevious** = 16777294

Previous song key.

**Mediarecord** = 16777296

Media record key.

**Mediastop** = 16777293

Media stop key.

**Menu** = 16777262

Context menu key.

**Meta** = 16777239

Meta key.

**Minus** = 45

- key.

**MouseLeft** = 2

Left mouse button.

**MouseMiddle** = 3

Middle mouse button.

**MouseRight** = 1

Right mouse button.

**MouseWheelDown** = 5

Mouse wheel down.

**MouseWheelLeft** = 6

Mouse wheel left button (only present on some mice).

**MouseWheelRight** = 7

Mouse wheel right button (only present on some mice).

**MouseWheelUp** = 4

Mouse wheel up.

**MouseXB1 = 8**

Extra mouse button 1 (only present on some mice).

**MouseXB2 = 9**

Extra mouse button 2 (only present on some mice).

**MouseXB3 = 10**

Extra mouse button 3 (only present on some mice).

**MouseXB4 = 11**

Extra mouse button 4 (only present on some mice).

**MouseXB5 = 12**

Extra mouse button 5 (only present on some mice).

**MouseXB6 = 13**

Extra mouse button 6 (only present on some mice).

**Mu = 181**

$\mu$  key.

**Multiply = 215**

$\times$  key.

**N = 78**

N key.

**Nobreakspace = 160**

Non-breakable space key.

**None = 0**

None key.

**Notsign = 172**

¬ key.

Ntilde = 209

Ñ key.

Numbersign = 35

# key.

Numlock = 16777242

Num Lock key.

0 = 79

O key.

Oacute = 211

Ó key.

Ocircumflex = 212

Ô key.

Odiaeresis = 214

Ö key.

Ograve = 210

Ô key.

Onehalf = 189

½ key.

Onequarter = 188

¼ key.

Onesuperior = 185

¹ key.

Ooblique = 216

$\emptyset$  key.

**Openurl** = 16777301

Open URL / Launch Browser key.

**Ordfeminine** = 170

$^{\text{a}}$  key.

**Otilde** = 213

$\tilde{\text{O}}$  key.

**P** = 80

P key.

**Pagedown** = 16777236

Page Down key.

**Pageup** = 16777235

Page Up key.

**Paragraph** = 182

$\text{¶}$  key.

**Parenleft** = 40

( key.

**Parenright** = 41

) key.

**Pause** = 16777225

Pause key.

**Percent** = 37

% key.

**Period** = 46

. key.

**Periodcentered** = 183

. key.

**Plus** = 43

+ key.

**Plusminus** = 177

± key.

**Print** = 16777226

Print Screen key.

**Q** = 81

Q key.

**Question** = 63

? key.

**Questiondown** = 191

¿ key.

**Quotedbl** = 34

" key.

**Quoteleft** = 96

` key.

**R** = 82

R key.

**Refresh** = 16777283

Media refresh key.

**Registered** = 174

<sup>®</sup> key.

**Right** = 16777233

Right arrow key.

**S** = 83

S key.

**Scrolllock** = 16777243

Scroll Lock key.

**Search** = 16777299

Search key.

**Section** = 167

§ key.

**Semicolon** = 59

; key.

**Shift** = 16777237

Shift key.

**Slash** = 47

/ key.

**Space** = 32

Space key.

**Ssharp** = 223

ß key.

**Standby** = 16777300

Standby key.

**Sterling** = 163

£ key.

**Stop** = 16777282

Media stop key.

**SuperL** = 16777260

Left Super key (Windows key).

**SuperR** = 16777261

Right Super key (Windows key).

**Sysreq** = 16777227

System Request key.

**T** = 84

T key.

**Tab** = 16777218

Tab key.

**Thorn** = 222

Þ key.

**Threequarters** = 190

$\frac{3}{4}$  key.

**Threesuperior** = 179

$\mathring{\text{3}}$  key.

**Trebledown** = 16777291

Treble down key.

**Trebleup** = 16777290

Treble up key.

**Twosuperior** = 178

<sup>2</sup> key.

**U** = 85

U key.

**Uacute** = 218

Ú key.

**Ucircumflex** = 219

Û key.

**Udiaeresis** = 220

Ü key.

**Ugrave** = 217

Ù key.

**Underscore** = 95

\_ key.

**Unknown** = 33554431

Unknown key.

**Up** = 16777232

Up arrow key.

**V** = 86

V key.

**Volumedown** = 16777284

Volume down key.

**Volumemute** = 16777285

Mute volume key.

**Volumeup** = 16777286

Volume up key.

W = 87

W key.

X = 88

X key.

Y = 89

Y key.

Yacute = 221

Ý key.

Ydiaeresis = 255

ÿ key.

Yen = 165

¥ key.

Z = 90

Z key.

# Enum KeyStatus

Namespace: [Cobilas.GodotEngine.Utility.Input](#)

Assembly: com.cobilas.godot.utility.dll

represents the state of a key.

```
[Flags]  
public enum KeyStatus : byte
```

## Fields

**Down** = 16

Occurs when the key has been pressed.

**None** = 0

No status detected.

**Press** = 4

Occurs when the key is being pressed.

**Up** = 2

Occurs when the key has been released.

# Enum MouseButton

Namespace: [Cobilas.GodotEngine.Utility.Input](#)

Assembly: com.cobilas.godot.utility.dll

Represents mouse triggers.

```
public enum MouseButton : byte
```

## Fields

**MouseLeft** = 1

Left mouse button.

**MouseMiddle** = 3

Middle mouse button.

**MouseRight** = 2

Right mouse button.

**MouseWheelDown** = 5

Mouse wheel down.

**MouseWheelLeft** = 6

Mouse wheel left button (only present on some mice).

**MouseWheelRight** = 7

Mouse wheel right button (only present on some mice).

**MouseWheelUp** = 4

Mouse wheel up.

**MouseXB1** = 8

Extra mouse button 1 (only present on some mice).

**MouseXB2** = 9

Extra mouse button 2 (only present on some mice).

**MouseXB3 = 10**

Extra mouse button 3 (only present on some mice).

**MouseXB4 = 11**

Extra mouse button 4 (only present on some mice).

**MouseXB5 = 12**

Extra mouse button 5 (only present on some mice).

**MouseXB6 = 13**

Extra mouse button 6 (only present on some mice).

**Unknown = 0**

Unidentified trigger.

# Struct PeripheralItem

Namespace: [Cobilas.GodotEngine.Utility.Input](#)

Assembly: com.cobilas.godot.utility.dll

Responsible for maintaining the status information of a peripheral.

```
public struct PeripheralItem : IEquatable<PeripheralItem>, IEquatable<KeyCode>,
IEquatable<KeyList>, IEquatable<MouseButton>, IEquatable<KeyStatus>, IEquatable<ulong>
```

## Implements

[IEquatable](#)<[PeripheralItem](#)>, [IEquatable](#)<[KeyCode](#)>, [IEquatable](#)<[KeyList](#)>,  
[IEquatable](#)<[MouseButton](#)>, [IEquatable](#)<[KeyStatus](#)>, [IEquatable](#)<[ulong](#)>

## Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### PeripheralItem(KeyCode)

Starts a new instance of the object.

```
public PeripheralItem(KeyCode keyCode)
```

#### Parameters

keyCode [KeyCode](#)

### PeripheralItem(KeyCode, KeyStatus)

Starts a new instance of the object.

```
public PeripheralItem(KeyCode keyCode, KeyStatus keyStatus)
```

#### Parameters

[keyCode](#) [KeyCode](#)

[keyStatus](#) [KeyStatus](#)

## Properties

### Empty

Represents an empty PeripheralItem.

```
public static PeripheralItem Empty { get; }
```

### Property Value

[PeripheralItem](#)

Returns an empty representation of PeripheralItem.

### IsMouseButton

Indicates whether it is a mouse trigger.

```
public readonly bool IsMouseButton { get; }
```

### Property Value

[bool](#) ↗

Returns [true](#) when the object is a mouse trigger.

### KeyCode

Represents a keyboard or mouse input.

```
public readonly KeyCode KeyCode { get; }
```

### Property Value

## [KeyCode](#)

Returns a representation of a peripheral input.

## ScanCode

The ScanCode is a representation of the [KeyCode](#).

```
public readonly ulong ScanCode { get; }
```

### Property Value

[ulong](#)

Returns a code representing a [KeyCode](#) key.

## Status

Represents the status of a peripheral input.

```
public KeyStatus Status { readonly get; set; }
```

### Property Value

[KeyStatus](#)

Allows you to change the status of the object.

## Methods

### Equals(KeyCode)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(KeyCode other)
```

### Parameters

`other` [KeyCode](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(KeyStatus)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(KeyStatus other)
```

Parameters

`other` [KeyStatus](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(MouseButton)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(MouseButton other)
```

Parameters

`other` [MouseButton](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(PeripheralItem)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(PeripheralItem other)
```

Parameters

**other** [PeripheralItem](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(KeyList)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(KeyList other)
```

Parameters

**other** KeyList

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

### Parameters

`obj` [object](#)

The object to compare with the current instance.

### Returns

[bool](#)

[true](#) if `obj` and this instance are the same type and represent the same value; otherwise, [false](#).

## Equals(ulong)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(ulong other)
```

### Parameters

`other` [ulong](#)

An object to compare with this object.

### Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## Operators

### operator ==(PeripheralItem, KeyCode)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, KeyCode B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [KeyCode](#)

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator ==(PeripheralItem, KeyStatus)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, KeyStatus B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [KeyStatus](#)

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator ==(PeripheralItem, MouseButton)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, MouseButton B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [MouseButton](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator ==(PeripheralItem, PeripheralItem)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, PeripheralItem B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [PeripheralItem](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator ==(PeripheralItem, KeyList)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, KeyList B)
```

## Parameters

### A [PeripheralItem](#)

Object to be compared.

### B [KeyList](#)

Object of comparison.

## Returns

### [bool](#) ↗

Returns the result of the comparison.

## operator ==(PeripheralItem, ulong)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(PeripheralItem A, ulong B)
```

## Parameters

### A [PeripheralItem](#)

Object to be compared.

### B [ulong](#) ↗

Object of comparison.

## Returns

### [bool](#) ↗

Returns the result of the comparison.

## explicit operator KeyCode(PeripheralItem)

Explicit conversion operator.([PeripheralItem](#) to [KeyCode](#))

```
public static explicit operator KeyCode(PeripheralItem A)
```

### Parameters

A [PeripheralItem](#)

Object to be converted.

### Returns

[KeyCode](#)

## explicit operator KeyStatus(PeripheralItem)

Explicit conversion operator.([PeripheralItem](#) to [KeyStatus](#))

```
public static explicit operator KeyStatus(PeripheralItem A)
```

### Parameters

A [PeripheralItem](#)

Object to be converted.

### Returns

[KeyStatus](#)

## explicit operator MouseButton(PeripheralItem)

Explicit conversion operator.([PeripheralItem](#) to [MouseButton](#))

```
public static explicit operator MouseButton(PeripheralItem A)
```

Parameters

A [PeripheralItem](#)

Object to be converted.

Returns

[MouseButton](#)

## explicit operator KeyList(PeripheralItem)

Explicit conversion operator.([PeripheralItem](#) to Godot.KeyList)

```
public static explicit operator KeyList(PeripheralItem A)
```

Parameters

A [PeripheralItem](#)

Object to be converted.

Returns

KeyList

## explicit operator ulong(PeripheralItem)

Explicit conversion operator.([PeripheralItem](#) to [ulong](#))

```
public static explicit operator ulong(PeripheralItem A)
```

Parameters

A [PeripheralItem](#)

Object to be converted.

Returns

[ulong](#)

## operator !=(PeripheralItem, KeyCode)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, KeyCode B)
```

### Parameters

A [PeripheralItem](#)

Object to be compared.

B [KeyCode](#)

Object of comparison.

### Returns

[bool](#)

Returns the result of the comparison.

## operator !=(PeripheralItem, KeyStatus)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, KeyStatus B)
```

### Parameters

A [PeripheralItem](#)

Object to be compared.

B [KeyStatus](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator !=(PeripheralItem, MouseButton)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, MouseButton B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [MouseButton](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator !=(PeripheralItem, PeripheralItem)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, PeripheralItem B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

## B [PeripheralItem](#)

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator !=(PeripheralItem, KeyList)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, KeyList B)
```

Parameters

### A [PeripheralItem](#)

Object to be compared.

### B KeyList

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator !=(PeripheralItem, ulong)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(PeripheralItem A, ulong B)
```

Parameters

A [PeripheralItem](#)

Object to be compared.

B [ulong](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

# Namespace Cobilas.GodotEngine.Utility.Numerics

## Structs

### [Quaternion](#)

Quaternions are used to represent rotations.

### [Vector2D](#)

Represents a two-dimensional vector

### [Vector2DInt](#)

Representation of a two-dimensional vector using integers.

### [Vector3D](#)

Represents a three-dimensional vector.

### [Vector3DInt](#)

Representation of a three-dimensional vector using integers.

### [Vector4D](#)

Represents a four-dimensional vector.(Four-axis vector)

### [VectorEqualityComparer](#)

Defines methods to support comparing vectors for equality.

## Interfaces

### [IIntVector](#)

Standardization interface for vectors.

### [IVector](#)

Standardization interface for vectors.

### [IVectorGeneric<TVector>](#)

Standardization interface for vectors.

# Interface IIntVector

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Standardization interface for vectors.

```
public interface IIntVector : IFormattable
```

## Inherited Members

[IFormattable.ToString\(string, IFormatProvider\)](#) ↗

## Properties

### AxisCount

Number of axles.

```
int AxisCount { get; }
```

### Property Value

[int](#) ↗

Returns the number of axes a vector has.

### this[int]

Allows you to access the axes of a vector through an index.

```
int this[int index] { get; set; }
```

### Parameters

[index](#) [int](#) ↗

The axis index.

## Property Value

[int](#) ↗

Sets the value of an axis by specifying its index.

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector](#).x to [IVector](#).y.

```
float aspect { get; }
```

## Property Value

[float](#) ↗

The [IVector](#).x component divided by the [IVector](#).y component.

## magnitude

Returns the length (magnitude) of this vector.

```
float magnitude { get; }
```

## Property Value

[float](#) ↗

The length of this vector.

### See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
int sqrMagnitude { get; }
```

## Property Value

[int](#)

The squared length of this vector.

## Methods

### ToString(string)

Formats the value of the current instance using the specified format.

```
string ToString(string format)
```

#### Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

#### Returns

[string](#)

The value of the current instance in the specified format.

# Interface IVector

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Standardization interface for vectors.

```
public interface IVector : IFormattable
```

## Inherited Members

[IFormattable.ToString\(string, IFormatProvider\)](#) ↗

## Properties

### AxisCount

Number of axles.

```
int AxisCount { get; }
```

### Property Value

[int](#) ↗

Returns the number of axes a vector has.

### this[int]

Allows you to access the axes of a vector through an index.

```
float this[int index] { get; set; }
```

### Parameters

[index](#) [int](#) ↗

The axis index.

## Property Value

[float](#) ↗

Sets the value of an axis by specifying its index.

## Normalized

Returns the vector scaled to unit length. Equivalent to `v / v.Length()`.

```
IVector Normalized { get; }
```

## Property Value

[IVector](#)

A normalized version of the vector.

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector.x](#) to [IVector.y](#).

```
float aspect { get; }
```

## Property Value

[float](#) ↗

The [IVector.x](#) component divided by the [IVector.y](#) component.

## ceil

Returns a new vector with all components rounded up (towards positive infinity).

```
IVector ceil { get; }
```

## Property Value

## [IVector](#)

A vector with [Ceil\(float\)](#) called on each component.

## floor

Returns a new vector with all components rounded down (towards negative infinity).

```
IVector floor { get; }
```

## Property Value

### [IVector](#)

A vector with [Floor\(float\)](#) called on each component.

## magnitude

Returns the length (magnitude) of this vector.

```
float magnitude { get; }
```

## Property Value

### [float](#)

The length of this vector.

## See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
float sqrMagnitude { get; }
```

## Property Value

[float](#)

The squared length of this vector.

## Methods

### Round()

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

`IVector Round()`

#### Returns

[IVector](#)

The rounded vector.

### ToString(string)

Formats the value of the current instance using the specified format.

`string ToString(string format)`

#### Parameters

`format string`

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

#### Returns

[string](#)

The value of the current instance in the specified format.



# Interface IVectorGeneric<TVector>

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Standardization interface for vectors.

```
public interface IVectorGeneric<TVector> : IEquatable<TVector>, IVector, IFormattable where
    TVector : IVector
```

## Type Parameters

TVector

### Inherited Members

[IEquatable<TVector>.Equals\(TVector\)](#) , [IVector.magnitude](#) , [IVector.sqrMagnitude](#) , [IVector.aspect](#) ,  
[IVector.AxisCount](#) , [IVector.this\[int\]](#) , [IVector.ToString\(string\)](#) ,  
[IFormattable.ToString\(string, IFormatProvider\)](#)

## Properties

### Normalized

Returns the vector scaled to unit length. Equivalent to `v / v.Length()`.

```
TVector Normalized { get; }
```

### Property Value

TVector

A normalized version of the vector.

### ceil

Returns a new vector with all components rounded up (towards positive infinity).

```
TVector ceil { get; }
```

## Property Value

TVector

A vector with [Ceil\(float\)](#) called on each component.

## floor

Returns a new vector with all components rounded down (towards negative infinity).

```
TVector floor { get; }
```

## Property Value

TVector

A vector with [Floor\(float\)](#) called on each component.

# Methods

## Round()

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
TVector Round()
```

## Returns

TVector

The rounded vector.

# Struct Quaternion

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Quaternions are used to represent rotations.

```
[Serializable]
public struct Quaternion : IEquatable<Quaternion>, IFormattable
```

Implements

[IEquatable](#)<[Quaternion](#)>, [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### Quaternion(Quaternion)

Starts a new instance of the object.

```
public Quaternion(Quaternion vector)
```

Parameters

vector [Quaternion](#)

### Quaternion(Vector4D)

Starts a new instance of the object.

```
public Quaternion(Vector4D vector)
```

Parameters

`vector` [Vector4D](#)

## Quaternion(float, float)

Starts a new instance of the object.

```
public Quaternion(float x, float y)
```

Parameters

`x` [float](#)

`y` [float](#)

## Quaternion(float, float, float)

Starts a new instance of the object.

```
public Quaternion(float x, float y, float z)
```

Parameters

`x` [float](#)

`y` [float](#)

`z` [float](#)

## Quaternion(float, float, float, float)

Starts a new instance of the object.

```
public Quaternion(float x, float y, float z, float w)
```

Parameters

`x` [float](#)

y [float](#)

z [float](#)

w [float](#)

## Fields

### Deg2Rad

Radians-to-degrees conversion constant (Read Only).

```
public const double Deg2Rad = 0.017453292519943295
```

### Field Value

[double](#)

### KEpsilon

A small value used to compare quaternions for equality.

```
public const float KEpsilon = 1E-06
```

### Field Value

[float](#)

### Remarks

The **kEpsilon** constant is used to determine if two quaternions are nearly equal, accounting for floating-point precision errors.

### Rad2Deg

Degrees-to-radians conversion constant (Read Only).

```
public const double Rad2Deg = 57.29577951308232
```

## Field Value

[double](#) ↗

## W

W component of the Quaternion. Do not directly modify quaternions.

```
[ShowProperty(true)]  
public float w
```

## Field Value

[float](#) ↗

## X

X component of the Quaternion. Don't modify this directly unless you know quaternions inside out.

```
[ShowProperty(true)]  
public float x
```

## Field Value

[float](#) ↗

## y

Y component of the Quaternion. Don't modify this directly unless you know quaternions inside out.

```
[ShowProperty(true)]  
public float y
```

## Field Value

[float](#)

### Z

Z component of the Quaternion. Don't modify this directly unless you know quaternions inside out.

```
[ShowProperty(true)]  
public float z
```

## Field Value

[float](#)

# Properties

## Euler

Returns or sets the euler angle representation of the rotation.

```
public readonly Vector3D Euler { get; }
```

## Property Value

[Vector3D](#)

## Identity

The identity rotation (Read Only).

```
public static Quaternion Identity { get; }
```

## Property Value

[Quaternion](#)

# Normalized

Returns this quaternion with a magnitude of 1 (Read Only).

```
public readonly Quaternion Normalized { get; }
```

Property Value

[Quaternion](#)

## Methods

### Angle(Quaternion, Quaternion)

Returns the angle in degrees between two rotations a and b.

```
public static float Angle(Quaternion a, Quaternion b)
```

Parameters

a [Quaternion](#)

Object to be compared.

b [Quaternion](#)

Object of comparison.

Returns

[float](#)

The floating point angle.

### Dot(Quaternion, Quaternion)

The dot product between two rotations.

```
public static float Dot(Quaternion a, Quaternion b)
```

## Parameters

a [Quaternion](#)

Object to be compared.

b [Quaternion](#)

Object of comparison.

## Returns

[float](#)

Returns the dot product of two quaternions.

## Equals(Quaternion)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(Quaternion other)
```

## Parameters

other [Quaternion](#)

An object to compare with this object.

## Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

## Parameters

**obj** [object](#)

The object to compare with the current instance.

## Returns

[bool](#)

[true](#) if **obj** and this instance are the same type and represent the same value; otherwise, [false](#).

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

## Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Normalize(Quaternion)

Converts this quaternion to one with the same orientation but with a magnitude of 1.

```
public static Quaternion Normalize(Quaternion q)
```

## Parameters

**q** [Quaternion](#)

The Quaternion that will be normalized.

Returns

## [Quaternion](#)

Returns an already normalized Quaternion.

## ToEuler(Quaternion)

Convert quaternion to euler-angle.

```
public static Vector3D ToEuler(Quaternion quaternion)
```

Parameters

### `quaternion` [Quaternion](#)

The quaternion that will be converted.

Returns

## [Vector3D](#)

The result of the conversion from quaternion to euler.

## ToQuaternion(Vector3D)

Convert euler-angle to quaternion.

```
public static Quaternion ToQuaternion(Vector3D vector)
```

Parameters

### `vector` [Vector3D](#)

Euler-angle that will be converted.

Returns

## [Quaternion](#)

Return result of euler to quaternion conversion.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

## Parameters

### format [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

### formatProvider [IFormatProvider](#)

The provider to use to format the value.-or- A null reference ([Nothing](#) in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

## Returns

### [string](#)

The value of the current instance in the specified format.

## Operators

### operator ==(Quaternion, Quaternion)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Quaternion lhs, Quaternion rhs)
```

## Parameters

### [lhs](#) [Quaternion](#)

Object to be compared.

### [rhs](#) [Quaternion](#)

Object of comparison.

## Returns

[bool](#)

Returns the result of the comparison.

## implicit operator Vector4D(Quaternion)

Implicit conversion operator.([Quaternion](#) to [Vector4D](#))

```
public static implicit operator Vector4D(Quaternion v)
```

Parameters

v [Quaternion](#)

Object to be converted.

Returns

[Vector4D](#)

## operator !=(Quaternion, Quaternion)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Quaternion lhs, Quaternion rhs)
```

Parameters

lhs [Quaternion](#)

Object to be compared.

rhs [Quaternion](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator \*(Quaternion, Quaternion)

Multiplication operation between two values.([Quaternion \\* Quaternion](#))

```
public static Quaternion operator *(Quaternion lhs, Quaternion rhs)
```

Parameters

**lhs** [Quaternion](#)

First module.

**rhs** [Quaternion](#)

Second module.

Returns

[Quaternion](#)

The result of the multiplication.

## operator \*(Quaternion, Vector3D)

Multiplication operation between two values.([Quaternion \\* Vector3D](#))

```
public static Vector3D operator *(Quaternion rotation, Vector3D point)
```

Parameters

**rotation** [Quaternion](#)

First module.

**point** [Vector3D](#)

Second module.

Returns

[Vector3D](#)

The result of the multiplication.

# Struct Vector2D

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Represents a two-dimensional vector

```
[Serializable]
public struct Vector2D : IVectorGeneric<Vector2D>, IEquatable<Vector2D>,
IVector, IFormattable
```

Implements

[IVectorGeneric<Vector2D>](#), [IEquatable](#)<Vector2D>, [IVector](#), [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### Vector2D(Vector2D)

Starts a new instance of the object.

```
public Vector2D(Vector2D vector)
```

Parameters

vector [Vector2D](#)

### Vector2D(Vector2)

Starts a new instance of the object.

```
public Vector2D(Vector2 vector)
```

Parameters

`vector` Vector2

## Vector2D(float, float)

Starts a new instance of the object.

```
public Vector2D(float x, float y)
```

Parameters

x [float](#)

y [float](#)

## Fields

X

X component of the vector.

```
[ShowProperty(true)]  
public float x
```

Field Value

[float](#)

y

Y component of the vector.

```
[ShowProperty(true)]  
public float y
```

Field Value

[float](#)

# Properties

## AxisCount

Number of axes.

```
public readonly int AxisCount { get; }
```

## Property Value

[int](#)

Returns the number of axes a vector has.

## Down

Shorthand for writing Vector2(0,1f).

```
public static Vector2D Down { get; }
```

## Property Value

[Vector2D](#)

## this[int]

Allows you to access the axes of a vector through an index.

```
public float this[int index] { readonly get; set; }
```

## Parameters

[index](#) [int](#)

The axis index.

## Property Value

### [float](#)

Sets the value of an axis by specifying its index.

## Left

Shorthand for writing Vector2(-1f,0).

```
public static Vector2D Left { get; }
```

## Property Value

### [Vector2D](#)

## Normalized

Returns the vector scaled to unit length. Equivalent to `v / v.Length()`.

```
public readonly Vector2D Normalized { get; }
```

## Property Value

### [Vector2D](#)

A normalized version of the vector.

## One

Shorthand for writing Vector2(1f,1f).

```
public static Vector2D One { get; }
```

## Property Value

### [Vector2D](#)

## Right

Shorthand for writing Vector2(1f,0).

```
public static Vector2D Right { get; }
```

## Property Value

[Vector2D](#)

## Up

Shorthand for writing Vector2(0,-1f).

```
public static Vector2D Up { get; }
```

## Property Value

[Vector2D](#)

## Zero

Shorthand for writing Vector2(0,0).

```
public static Vector2D Zero { get; }
```

## Property Value

[Vector2D](#)

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector.x](#) to [IVector.y](#).

```
public readonly float aspect { get; }
```

## Property Value

[float](#)

The [IVector](#).x component divided by the [IVector](#).y component.

## ceil

Returns a new vector with all components rounded up (towards positive infinity).

```
public readonly Vector2D ceil { get; }
```

## Property Value

[Vector2D](#)

A vector with [Ceil\(float\)](#) called on each component.

## floor

Returns a new vector with all components rounded down (towards negative infinity).

```
public readonly Vector2D floor { get; }
```

## Property Value

[Vector2D](#)

A vector with [Floor\(float\)](#) called on each component.

## magnitude

Returns the length (magnitude) of this vector.

```
public readonly float magnitude { get; }
```

## Property Value

[float](#)

The length of this vector.

### See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
public readonly float sqrMagnitude { get; }
```

## Property Value

[float](#)

The squared length of this vector.

## Methods

### Abs(in Vector2D)

Returns an absolute value of the vector.

```
public static Vector2D Abs(in Vector2D a)
```

## Parameters

a [Vector2D](#)

The vector to become absolute.

Returns

## [Vector2D](#)

Returns the vector with its absolute value axes.

## Abs(bool, bool)

Returns an absolute value of the vector.

```
public readonly Vector2D Abs(bool absX = true, bool absY = true)
```

Parameters

### `absX` [bool](#)

The X axis becomes absolute.

### `absY` [bool](#)

The Y axis becomes absolute.

Returns

## [Vector2D](#)

Returns the vector with its absolute value axes.

## AngleTo(in Vector2D, in Vector2D)

Returns the angle in degrees between from and to.

```
public static float AngleTo(in Vector2D lhs, in Vector2D rhs)
```

Parameters

### `lhs` [Vector2D](#)

The vector from which the angular difference is measured.

## **rhs** [Vector2D](#)

The vector to which the angular difference is measured.

Returns

### [float](#) ↗

The angle in degrees between the two vectors.

## AngleToPoint([in Vector2D](#), [in Vector2D](#))

Returns the angle between the line connecting the two points and the X axis, in radians.

```
public static float AngleToPoint(in Vector2D lhs, in Vector2D rhs)
```

Parameters

### [lhs](#) [Vector2D](#)

One of the values.

### [rhs](#) [Vector2D](#)

The other value.

Returns

### [float](#) ↗

The angle between the two vectors, in radians.

## Aspect([in Vector2D](#))

Returns the aspect ratio of this vector, the ratio of [IVector](#).x to [IVector](#).y.

```
public static float Aspect(in Vector2D a)
```

Parameters

a [Vector2D](#)

The vector whose aspect will be calculated.

Returns

[float](#)

The [IVector](#).x component divided by the [IVector](#).y component.

## Ceil(in Vector2D)

Returns a new vector with all components rounded up (towards positive infinity).

```
public static Vector2D Ceil(in Vector2D a)
```

Parameters

a [Vector2D](#)

The vector that will be the ceiling.

Returns

[Vector2D](#)

A vector with [Ceil\(float\)](#) called on each component.

## Cross(in Vector2D, in Vector2D)

Cross Product of two vectors.

```
public static float Cross(in Vector2D lhs, in Vector2D rhs)
```

Parameters

lhs [Vector2D](#)

One of the values.

`rhs` [Vector2D](#)

The other value.

Returns

[float](#) ↗

Returns the cross product of vectors.

## Distance([in Vector2D](#), [in Vector2D](#))

Returns the distance between a and b.

```
public static float Distance(in Vector2D a, in Vector2D b)
```

Parameters

`a` [Vector2D](#)

One of the values.

`b` [Vector2D](#)

The other value.

Returns

[float](#) ↗

The distance between two vectors.

## Dot([in Vector2D](#), [in Vector2D](#))

Dot Product of two vectors.

```
public static float Dot(in Vector2D lhs, in Vector2D rhs)
```

Parameters

`lhs` [Vector2D](#)

`rhs` [Vector2D](#)

Returns

[float](#)

returns the result of the dot product.

## Equals(Vector2D)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(Vector2D other)
```

Parameters

`other` [Vector2D](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

Parameters

`obj` [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if [obj](#) and this instance are the same type and represent the same value; otherwise, [false](#).

## Floor(in Vector2D)

Returns a new vector with all components rounded down (towards negative infinity).

```
public static Vector2D Floor(in Vector2D a)
```

Parameters

a [Vector2D](#)

The vector that will be the floor.

Returns

[Vector2D](#)

A vector with [Floor\(float\)](#) called on each component.

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Magnitude(in Vector2D)

Returns the length of this vector

```
public static float Magnitude(in Vector2D a)
```

Parameters

a [Vector2D](#)

One of the values.

Returns

[float](#)

vector length.

## Max(Vector2D, Vector2D)

Returns a vector that is made from the largest components of two vectors.

```
public static Vector2D Max(Vector2D lhs, Vector2D rhs)
```

Parameters

lhs [Vector2D](#)

One of the values.

rhs [Vector2D](#)

The other value.

Returns

[Vector2D](#)

Whichever of the two values is higher.

## Min(Vector2D, Vector2D)

Returns a vector that is made from the smallest components of two vectors.

```
public static Vector2D Min(Vector2D lhs, Vector2D rhs)
```

Parameters

**lhs** [Vector2D](#)

One of the values.

**rhs** [Vector2D](#)

The other value.

Returns

[Vector2D](#)

Whichever of the two values is lower.

## Neg(in Vector2D)

Inverts the values of a vector.

```
public static Vector2D Neg(in Vector2D a)
```

Parameters

**a** [Vector2D](#)

The vector to be inverted.

Returns

[Vector2D](#)

Returns the axes of an inverted vector.

## Neg(bool, bool)

Inverts the values of a vector.

```
public readonly Vector2D Neg(bool negX = true, bool negY = true)
```

Parameters

`negX` [bool](#)

The X axis becomes inverted.

`negY` [bool](#)

The Y axis becomes inverted.

Returns

[Vector2D](#)

Returns the axes of an inverted vector.

## Normalize(in Vector2D)

Makes this vector have a magnitude of 1.

```
public static Vector2D Normalize(in Vector2D a)
```

Parameters

`a` [Vector2D](#)

The vector to be normalized.

Returns

[Vector2D](#)

Returns the already normalized vector.

## Round()

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public readonly Vector2D Round()
```

Returns

[Vector2D](#)

The rounded vector.

## Round(in Vector2D)

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public static Vector2D Round(in Vector2D a)
```

Parameters

a [Vector2D](#)

the vector to be rounded

Returns

[Vector2D](#)

The rounded vector.

## SqrMagnitude(in Vector2D)

Returns the squared length of this vector

```
public static float SqrMagnitude(in Vector2D a)
```

Parameters

a [Vector2D](#)

One of the values.

Returns

[float](#)

square length of the vector.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

Parameters

**format** [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

**formatProvider** [IFormatProvider](#)

The provider to use to format the value.-or- A null reference ([Nothing](#) in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

[string](#)

The value of the current instance in the specified format.

## Operators

### operator +(Vector2D, Vector2D)

Addition operation between two values.([Vector2D](#) + [Vector2D](#))

```
public static Vector2D operator +(Vector2D a, Vector2D b)
```

Parameters

**a** [Vector2D](#)

First module.

**b** [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the addition.

## operator +(Vector2D, Vector2)

Addition operation between two values. ([Vector2D](#) + Godot.Vector2)

```
public static Vector2D operator +(Vector2D a, Vector2 b)
```

Parameters

**a** [Vector2D](#)

First module.

**b** Vector2

Second module.

Returns

[Vector2D](#)

The result of the addition.

## operator +(Vector2, Vector2D)

Addition operation between two values. (Godot.Vector2 + [Vector2D](#))

```
public static Vector2D operator +(Vector2 a, Vector2D b)
```

Parameters

a [Vector2](#)

First module.

b [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the addition.

## operator /(Vector2D, Vector2D)

Division operation between two values.([Vector2D](#) / [Vector2D](#))

```
public static Vector2D operator /(Vector2D a, Vector2D b)
```

Parameters

a [Vector2D](#)

First module.

b [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the division.

## operator /(Vector2D, Vector2)

Division operation between two values.([Vector2D](#) / Godot.Vector2)

```
public static Vector2D operator /(Vector2D a, Vector2 b)
```

## Parameters

a [Vector2D](#)

First module.

b [Vector2](#)

Second module.

## Returns

[Vector2D](#)

The result of the division.

## operator /(Vector2D, float)

Division operation between two values. ([Vector2D](#) / [float](#))

```
public static Vector2D operator /(Vector2D a, float b)
```

## Parameters

a [Vector2D](#)

First module.

b [float](#)

Second module.

## Returns

[Vector2D](#)

The result of the division.

## operator /(Vector2, Vector2D)

Division operation between two values.(Godot.Vector2 / [Vector2D](#))

```
public static Vector2D operator /(Vector2 a, Vector2D b)
```

### Parameters

**a** [Vector2](#)

First module.

**b** [Vector2D](#)

Second module.

### Returns

[Vector2D](#)

The result of the division.

## operator ==(in Vector2D, in Vector2D)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(in Vector2D lhs, in Vector2D rhs)
```

### Parameters

**lhs** [Vector2D](#)

Object to be compared.

**rhs** [Vector2D](#)

Object of comparison.

### Returns

[bool](#) ↗

Returns the result of the comparison.

## implicit operator Vector3D(Vector2D)

Implicit conversion operator.([Vector2D](#) to [Vector3D](#))

```
public static implicit operator Vector3D(Vector2D v)
```

Parameters

v [Vector2D](#)

Object to be converted.

Returns

[Vector3D](#)

## implicit operator Vector4D(Vector2D)

Implicit conversion operator.([Vector2D](#) to [Vector4D](#))

```
public static implicit operator Vector4D(Vector2D v)
```

Parameters

v [Vector2D](#)

Object to be converted.

Returns

[Vector4D](#)

## implicit operator Vector2D(Vector2D)

Implicit conversion operator.([Vector2D](#) to Godot.Vector2)

```
public static implicit operator Vector2(Vector2D v)
```

## Parameters

v [Vector2D](#)

Object to be converted.

## Returns

Vector2

## implicit operator Vector3(Vector2D)

Implicit conversion operator.([Vector2D](#) to Godot.Vector3)

```
public static implicit operator Vector3(Vector2D v)
```

## Parameters

v [Vector2D](#)

Object to be converted.

## Returns

Vector3

## implicit operator Vector2D(Vector2)

Implicit conversion operator.(Godot.Vector2 to [Vector2D](#))

```
public static implicit operator Vector2D(Vector2 v)
```

## Parameters

v Vector2

Object to be converted.

Returns

[Vector2D](#)

## implicit operator Vector2D(Vector3)

Implicit conversion operator.(Godot.Vector3 to [Vector2D](#))

```
public static implicit operator Vector2D(Vector3 v)
```

Parameters

v Vector3

Object to be converted.

Returns

[Vector2D](#)

## operator !=(in Vector2D, in Vector2D)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(in Vector2D lhs, in Vector2D rhs)
```

Parameters

lhs [Vector2D](#)

Object to be compared.

rhs [Vector2D](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator \*(Vector2D, Vector2D)

Multiplication operation between two values.([Vector2D](#) \* [Vector2D](#))

```
public static Vector2D operator *(Vector2D a, Vector2D b)
```

Parameters

a [Vector2D](#)

First module.

b [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the multiplication.

## operator \*(Vector2D, Vector2)

Multiplication operation between two values.([Vector2D](#) \* Godot.Vector2)

```
public static Vector2D operator *(Vector2D a, Vector2 b)
```

Parameters

a [Vector2D](#)

First module.

b Vector2

Second module.

Returns

[Vector2D](#)

The result of the multiplication.

## operator \*(Vector2D, float)

Multiplication operation between two values.([Vector2D \\* float](#))

```
public static Vector2D operator *(Vector2D a, float b)
```

Parameters

a [Vector2D](#)

First module.

b [float](#)

Second module.

Returns

[Vector2D](#)

The result of the multiplication.

## operator \*(Vector2, Vector2D)

Multiplication operation between two values.(Godot.Vector2 \* [Vector2D](#))

```
public static Vector2D operator *(Vector2 a, Vector2D b)
```

Parameters

a Vector2

First module.

b [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the multiplication.

## operator -(Vector2D, Vector2D)

Subtraction operation between two values.([Vector2D](#) - [Vector2D](#))

```
public static Vector2D operator -(Vector2D a, Vector2D b)
```

Parameters

a [Vector2D](#)

First module.

b [Vector2D](#)

Second module.

Returns

[Vector2D](#)

The result of the subtraction.

## operator -(Vector2D, Vector2)

Subtraction operation between two values.([Vector2D](#) - Godot.Vector2)

```
public static Vector2D operator -(Vector2D a, Vector2 b)
```

## Parameters

a [Vector2D](#)

First module.

b Vector2

Second module.

## Returns

[Vector2D](#)

The result of the subtraction.

## operator -(Vector2, Vector2D)

Subtraction operation between two values.(Godot.Vector2 - [Vector2D](#))

```
public static Vector2D operator -(Vector2 a, Vector2D b)
```

## Parameters

a Vector2

First module.

b [Vector2D](#)

Second module.

## Returns

[Vector2D](#)

The result of the subtraction.

## operator -(Vector2D)

The operator allows us to reverse the value.

```
public static Vector2D operator -(Vector2D a)
```

## Parameters

a [Vector2D](#)

Or value that will be inverted.

## Returns

[Vector2D](#)

Returns the result of the inversion.

# Struct Vector2DInt

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Representation of a two-dimensional vector using integers.

```
[Serializable]
public struct Vector2DInt : IIntVector, IFormattable
```

Implements

[IIntVector](#), [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Constructors

### Vector2DInt(in Vector2DInt)

Starts a new instance of the object.

```
public Vector2DInt(in Vector2DInt vector)
```

Parameters

vector [Vector2DInt](#)

### Vector2DInt(in int, in int)

Starts a new instance of the object.

```
public Vector2DInt(in int x, in int y)
```

Parameters

x [int](#)

y [int](#)

## Fields

X

X component of the vector.

```
[ShowProperty(true)]  
public int x
```

Field Value

[int](#)

y

Y component of the vector.

```
[ShowProperty(true)]  
public int y
```

Field Value

[int](#)

## Properties

AxisCount

Number of axles.

```
public readonly int AxisCount { get; }
```

## Property Value

[int ↗](#)

Returns the number of axes a vector has.

## Down

Shorthand for writing Vector2(0,1).

```
public static Vector2DInt Down { get; }
```

## Property Value

[Vector2DInt](#)

## this[int]

Allows you to access the axes of a vector through an index.

```
public int this[int index] { readonly get; set; }
```

## Parameters

[index](#) [int ↗](#)

The axis index.

## Property Value

[int ↗](#)

Sets the value of an axis by specifying its index.

## Left

Shorthand for writing Vector2(-1,0).

```
public static Vector2DInt Left { get; }
```

Property Value

[Vector2DInt](#)

One

Shorthand for writing Vector2(1,1).

```
public static Vector2DInt One { get; }
```

Property Value

[Vector2DInt](#)

Right

Shorthand for writing Vector2(1,0).

```
public static Vector2DInt Right { get; }
```

Property Value

[Vector2DInt](#)

Up

Shorthand for writing Vector2(0,-1).

```
public static Vector2DInt Up { get; }
```

Property Value

[Vector2DInt](#)

## Zero

Shorthand for writing Vector2(0,0).

```
public static Vector2DInt Zero { get; }
```

## Property Value

[Vector2DInt](#)

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector](#).x to [IVector](#).y.

```
public readonly float aspect { get; }
```

## Property Value

[float](#) ↗

The [IVector](#).x component divided by the [IVector](#).y component.

## magnitude

Returns the length (magnitude) of this vector.

```
public readonly float magnitude { get; }
```

## Property Value

[float](#) ↗

The length of this vector.

## See Also

[LengthSquared\(\)](#)

# sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
public readonly int sqrMagnitude { get; }
```

## Property Value

[int](#)

The squared length of this vector.

# Methods

## Abs(in Vector2DInt)

Returns an absolute value of the vector.

```
public static Vector2DInt Abs(in Vector2DInt a)
```

## Parameters

a [Vector2DInt](#)

The vector to become absolute.

## Returns

[Vector2DInt](#)

Returns the vector with its absolute value axes.

## Abs(bool, bool)

Returns an absolute value of the vector.

```
public readonly Vector2DInt Abs(bool absX = true, bool absY = true)
```

## Parameters

**absX** [bool](#)

The X axis becomes absolute.

**absY** [bool](#)

The Y axis becomes absolute.

## Returns

[Vector2DInt](#)

Returns the vector with its absolute value axes.

## Aspect([in Vector2DInt](#))

Returns the aspect ratio of this vector, the ratio of [IVector](#).x to [IVector](#).y.

```
public static float Aspect(in Vector2DInt a)
```

## Parameters

**a** [Vector2DInt](#)

The vector whose aspect will be calculated.

## Returns

[float](#)

The [IVector](#).x component divided by the [IVector](#).y component.

## CeilToInt([in Vector2D](#))

Converts a floating-point vector to an integer vector and applies a Ceiling to each value.

```
public static Vector2DInt CeilToInt(in Vector2D a)
```

## Parameters

a [Vector2D](#)

The vector to be converted is ceiling

## Returns

[Vector2DInt](#)

Returns the converted vector and ceiling

## Distance(in Vector2DInt, in Vector2DInt)

Returns the distance between a and b.

```
public static float Distance(in Vector2DInt a, in Vector2DInt b)
```

## Parameters

a [Vector2DInt](#)

One of the values.

b [Vector2DInt](#)

The other value.

## Returns

[float](#) ↗

The distance between two vectors.

## Equals(Vector2DInt)

```
public readonly bool Equals(Vector2DInt other)
```

## Parameters

other [Vector2DInt](#)

Returns

[bool](#)

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

Parameters

[obj](#) [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if [obj](#) and this instance are the same type and represent the same value; otherwise, [false](#).

## FloorToInt(in Vector2D)

Converts a floating-point vector to an integer vector and applies a Floor to each value.

```
public static Vector2DInt FloorToInt(in Vector2D a)
```

Parameters

[a](#) [Vector2D](#)

The vector to be converted is floor

Returns

[Vector2DInt](#)

Returns the converted vector and floor

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Magnitude(in Vector2DInt)

Returns the length of this vector

```
public static float Magnitude(in Vector2DInt a)
```

Parameters

a [Vector2DInt](#)

One of the values.

Returns

[float](#)

vector length.

## Max(Vector2DInt, Vector2DInt)

Returns a vector that is made from the largest components of two vectors.

```
public static Vector2DInt Max(Vector2DInt lhs, Vector2DInt rhs)
```

## Parameters

**lhs** [Vector2DInt](#)

One of the values.

**rhs** [Vector2DInt](#)

The other value.

## Returns

[Vector2DInt](#)

Whichever of the two values is higher.

## Min(Vector2DInt, Vector2DInt)

Returns a vector that is made from the smallest components of two vectors.

```
public static Vector2DInt Min(Vector2DInt lhs, Vector2DInt rhs)
```

## Parameters

**lhs** [Vector2DInt](#)

One of the values.

**rhs** [Vector2DInt](#)

The other value.

## Returns

[Vector2DInt](#)

Whichever of the two values is lower.

## Neg(in Vector2DInt)

Inverts the values of a vector.

```
public static Vector2DInt Neg(in Vector2DInt a)
```

## Parameters

a [Vector2DInt](#)

The vector to be inverted.

## Returns

[Vector2DInt](#)

Returns the axes of an inverted vector.

## Neg(bool, bool)

Inverts the values of a vector.

```
public readonly Vector2DInt Neg(bool negX = true, bool negY = true)
```

## Parameters

negX [bool](#)

The X axis becomes inverted.

negY [bool](#)

The Y axis becomes inverted.

## Returns

[Vector2DInt](#)

Returns the axes of an inverted vector.

## RoundToInt(in Vector2D)

Converts a floating-point vector to an integer vector by performing a Round for each value.

```
public static Vector2DInt RoundToInt(in Vector2D a)
```

## Parameters

a [Vector2D](#)

The vector that will be converted and rounded

## Returns

[Vector2DInt](#)

Returns the converted and rounded vector

## SqrMagnitude(in Vector2DInt)

Returns the squared length of this vector

```
public static int SqrMagnitude(in Vector2DInt a)
```

## Parameters

a [Vector2DInt](#)

One of the values.

## Returns

[int](#)

square length of the vector.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

[formatProvider](#) [IFormatProvider](#)

The provider to use to format the value.-or- A null reference (**Nothing** in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

Returns

[string](#)

The value of the current instance in the specified format.

## Operators

### operator +(Vector2DInt, Vector2DInt)

Addition operation between two values.([Vector2DInt](#) + [Vector2DInt](#))

```
public static Vector2DInt operator +(Vector2DInt a, Vector2DInt b)
```

Parameters

a [Vector2DInt](#)

First module.

b [Vector2DInt](#)

Second module.

Returns

[Vector2DInt](#)

The result of the addition.

### operator /(Vector2DInt, Vector2DInt)

Division operation between two values.([Vector2DInt](#) / [Vector2DInt](#))

```
public static Vector2DInt operator /(Vector2DInt a, Vector2DInt b)
```

Parameters

a [Vector2DInt](#)

First module.

b [Vector2DInt](#)

Second module.

Returns

[Vector2DInt](#)

The result of the division.

## operator /(Vector2DInt, int)

Division operation between two values. ([Vector2DInt](#) / [int](#))

```
public static Vector2DInt operator /(Vector2DInt a, int b)
```

Parameters

a [Vector2DInt](#)

First module.

b [int](#)

Second module.

Returns

[Vector2DInt](#)

The result of the division.

## operator ==(in Vector2DInt, in Vector2DInt)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(in Vector2DInt lhs, in Vector2DInt rhs)
```

## Parameters

### lhs [Vector2DInt](#)

Object to be compared.

### rhs [Vector2DInt](#)

Object of comparison.

## Returns

### bool ↗

Returns the result of the comparison.

## explicit operator Vector2DInt(Vector2D)

Explicit conversion operator.([Vector2D](#) to [Vector2DInt](#))

```
public static explicit operator Vector2DInt(Vector2D v)
```

## Parameters

### v [Vector2D](#)

Object to be converted.

## Returns

### [Vector2DInt](#)

## explicit operator Vector2DInt(Vector3D)

Explicit conversion operator.([Vector3D](#) to [Vector2DInt](#))

```
public static explicit operator Vector2DInt(Vector3D v)
```

## Parameters

v [Vector3D](#)

Object to be converted.

## Returns

[Vector2DInt](#)

## explicit operator Vector2DInt(Vector4D)

Explicit conversion operator.([Vector4D](#) to [Vector2DInt](#))

```
public static explicit operator Vector2DInt(Vector4D v)
```

## Parameters

v [Vector4D](#)

Object to be converted.

## Returns

[Vector2DInt](#)

## explicit operator Vector2DInt(Vector2)

Explicit conversion operator.(Godot.Vector2 to [Vector2DInt](#))

```
public static explicit operator Vector2DInt(Vector2 v)
```

## Parameters

v [Vector2](#)

Object to be converted.

Returns

[Vector2DInt](#)

## implicit operator Vector2D(Vector2DInt)

Implicit conversion operator.([Vector2DInt](#) to [Vector2D](#))

```
public static implicit operator Vector2D(Vector2DInt v)
```

Parameters

v [Vector2DInt](#)

Object to be converted.

Returns

[Vector2D](#)

## implicit operator Vector3D(Vector2DInt)

Implicit conversion operator.([Vector2DInt](#) to [Vector3D](#))

```
public static implicit operator Vector3D(Vector2DInt v)
```

Parameters

v [Vector2DInt](#)

Object to be converted.

Returns

[Vector3D](#)

## implicit operator Vector3DInt(Vector2DInt)

Implicit conversion operator.([Vector2DInt](#) to [Vector3DInt](#))

```
public static implicit operator Vector3DInt(Vector2DInt v)
```

### Parameters

v [Vector2DInt](#)

Object to be converted.

### Returns

[Vector3DInt](#)

## implicit operator Vector4D(Vector2DInt)

Implicit conversion operator.([Vector2DInt](#) to [Vector4D](#))

```
public static implicit operator Vector4D(Vector2DInt v)
```

### Parameters

v [Vector2DInt](#)

Object to be converted.

### Returns

[Vector4D](#)

## implicit operator Vector2DInt(Vector2DInt)

Implicit conversion operator.([Vector2DInt](#) to Godot.Vector2)

```
public static implicit operator Vector2DInt(Vector2DInt v)
```

## Parameters

v [Vector2DInt](#)

Object to be converted.

## Returns

Vector2

## operator !=(in Vector2DInt, in Vector2DInt)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(in Vector2DInt lhs, in Vector2DInt rhs)
```

## Parameters

lhs [Vector2DInt](#)

Object to be compared.

rhs [Vector2DInt](#)

Object of comparison.

## Returns

[bool](#)

Returns the result of the comparison.

## operator \*(Vector2DInt, Vector2DInt)

Multiplication operation between two values. ([Vector2DInt](#) \* [Vector2DInt](#))

```
public static Vector2DInt operator *(Vector2DInt a, Vector2DInt b)
```

## Parameters

a [Vector2DInt](#)

First module.

b [Vector2DInt](#)

Second module.

Returns

[Vector2DInt](#)

The result of the multiplication.

## operator \*(Vector2DInt, int)

Multiplication operation between two values. ([Vector2DInt](#) \* [int](#))

```
public static Vector2DInt operator *(Vector2DInt a, int b)
```

Parameters

a [Vector2DInt](#)

First module.

b [int](#)

Second module.

Returns

[Vector2DInt](#)

The result of the multiplication.

## operator -(Vector2DInt, Vector2DInt)

Subtraction operation between two values. ([Vector2DInt](#) - [Vector2DInt](#))

```
public static Vector2DInt operator -(Vector2DInt a, Vector2DInt b)
```

## Parameters

a [Vector2DInt](#)

First module.

b [Vector2DInt](#)

Second module.

## Returns

[Vector2DInt](#)

The result of the subtraction.

## operator -(Vector2DInt)

The operator allows us to reverse the value.

```
public static Vector2DInt operator -(Vector2DInt a)
```

## Parameters

a [Vector2DInt](#)

Or value that will be inverted.

## Returns

[Vector2DInt](#)

Returns the result of the inversion.

# Struct Vector3D

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Represents a three-dimensional vector.

```
[Serializable]
public struct Vector3D : IVectorGeneric<Vector3D>, IEquatable<Vector3D>,
IVector, IFormattable
```

Implements

[IVectorGeneric<Vector3D>](#), [IEquatable](#)<Vector3D>, [IVector](#), [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### Vector3D(Vector3D)

Starts a new instance of the object.

```
public Vector3D(Vector3D vector)
```

Parameters

vector [Vector3D](#)

### Vector3D(Vector3)

Starts a new instance of the object.

```
public Vector3D(Vector3 vector)
```

Parameters

`vector` Vector3

## Vector3D(float, float)

Starts a new instance of the object.

```
public Vector3D(float x, float y)
```

Parameters

`x` [float](#)

`y` [float](#)

## Vector3D(float, float, float)

Starts a new instance of the object.

```
public Vector3D(float x, float y, float z)
```

Parameters

`x` [float](#)

`y` [float](#)

`z` [float](#)

## Fields

### X

X component of the vector.

```
[ShowProperty(true)]  
public float x
```

## Field Value

[float](#) ↗

### y

Y component of the vector.

```
[ShowProperty(true)]  
public float y
```

## Field Value

[float](#) ↗

### z

Z component of the vector.

```
[ShowProperty(true)]  
public float z
```

## Field Value

[float](#) ↗

# Properties

## AxisCount

Number of axles.

```
public readonly int AxisCount { get; }
```

## Property Value

[int](#) ↗

Returns the number of axes a vector has.

## Back

Shorthand for writing Vector3(0,0,-1f).

```
public static Vector3D Back { get; }
```

Property Value

[Vector3D](#)

## Down

Shorthand for writing Vector3(0,1f,0).

```
public static Vector3D Down { get; }
```

Property Value

[Vector3D](#)

## Forward

Shorthand for writing Vector3(0,0,1f).

```
public static Vector3D Forward { get; }
```

Property Value

[Vector3D](#)

## this[int]

Allows you to access the axes of a vector through an index.

```
public float this[int index] { readonly get; set; }
```

## Parameters

**index** [int](#)

The axis index.

## Property Value

[float](#)

Sets the value of an axis by specifying its index.

## Left

Shorthand for writing Vector3(-1f,0,0).

```
public static Vector3D Left { get; }
```

## Property Value

[Vector3D](#)

## Normalized

Returns the vector scaled to unit length. Equivalent to `v / v.Length()`.

```
public readonly Vector3D Normalized { get; }
```

## Property Value

[Vector3D](#)

A normalized version of the vector.

## One

Shorthand for writing Vector3(1,1,1).

```
public static Vector3D One { get; }
```

## Property Value

[Vector3D](#)

## Right

Shorthand for writing Vector3(1f,0,0).

```
public static Vector3D Right { get; }
```

## Property Value

[Vector3D](#)

## Up

Shorthand for writing Vector3(0,-1f,0).

```
public static Vector3D Up { get; }
```

## Property Value

[Vector3D](#)

## Zero

Shorthand for writing Vector3(0,0,0).

```
public static Vector3D Zero { get; }
```

Property Value

[Vector3D](#)

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector.x](#) to [IVector.y](#).

```
public readonly float aspect { get; }
```

Property Value

[float](#)

The [IVector.x](#) component divided by the [IVector.y](#) component.

## ceil

Returns a new vector with all components rounded up (towards positive infinity).

```
public readonly Vector3D ceil { get; }
```

Property Value

[Vector3D](#)

A vector with [Ceil\(float\)](#) called on each component.

## floor

Returns a new vector with all components rounded down (towards negative infinity).

```
public readonly Vector3D floor { get; }
```

Property Value

[Vector3D](#)

A vector with [Floor\(float\)](#) called on each component.

## magnitude

Returns the length (magnitude) of this vector.

```
public readonly float magnitude { get; }
```

### Property Value

[float](#)

The length of this vector.

### See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
public readonly float sqrMagnitude { get; }
```

### Property Value

[float](#)

The squared length of this vector.

## Methods

### Abs(in Vector3D)

Returns an absolute value of the vector.

```
public static Vector3D Abs(in Vector3D a)
```

## Parameters

a [Vector3D](#)

The vector to become absolute.

## Returns

[Vector3D](#)

Returns the vector with its absolute value axes.

## Abs(bool, bool, bool)

Returns an absolute value of the vector.

```
public readonly Vector3D Abs(bool absX = true, bool absY = true, bool absZ = true)
```

## Parameters

absX [bool](#)

The X axis becomes absolute.

absY [bool](#)

The Y axis becomes absolute.

absZ [bool](#)

The Z axis becomes absolute.

## Returns

[Vector3D](#)

Returns the vector with its absolute value axes.

## AngleTo(in Vector2D, in Vector2D)

Returns the angle in degrees between from and to.

```
public static float AngleTo(in Vector2D lhs, in Vector2D rhs)
```

## Parameters

**lhs** [Vector2D](#)

The vector from which the angular difference is measured.

**rhs** [Vector2D](#)

The vector to which the angular difference is measured.

## Returns

[float](#) ↗

The angle in degrees between the two vectors.

## Ceil(**in** Vector3D)

Returns a new vector with all components rounded up (towards positive infinity).

```
public static Vector3D Ceil(in Vector3D a)
```

## Parameters

**a** [Vector3D](#)

The vector that will be the ceiling.

## Returns

[Vector3D](#)

A vector with [Ceil\(float\)↗](#) called on each component.

## Cross(**in** Vector3D, **in** Vector3D)

Cross Product of two vectors.

```
public static Vector3D Cross(in Vector3D lhs, in Vector3D rhs)
```

## Parameters

**lhs** [Vector3D](#)

One of the values.

**rhs** [Vector3D](#)

The other value.

## Returns

[Vector3D](#)

Returns the cross product of vectors.

## Distance(**in** Vector3D, **in** Vector3D)

Returns the distance between a and b.

```
public static float Distance(in Vector3D a, in Vector3D b)
```

## Parameters

**a** [Vector3D](#)

One of the values.

**b** [Vector3D](#)

The other value.

## Returns

[float](#) ↗

The distance between two vectors.

## Dot(in Vector3D, in Vector3D)

Dot Product of two vectors.

```
public static float Dot(in Vector3D lhs, in Vector3D rhs)
```

Parameters

`lhs` [Vector3D](#)

`rhs` [Vector3D](#)

Returns

[float](#)

returns the result of the dot product.

## Equals(Vector3D)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(Vector3D other)
```

Parameters

`other` [Vector3D](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

## Parameters

**obj** [object](#)

The object to compare with the current instance.

## Returns

[bool](#)

[true](#) if **obj** and this instance are the same type and represent the same value; otherwise, [false](#).

## Floor(in Vector3D)

Returns a new vector with all components rounded down (towards negative infinity).

```
public static Vector3D Floor(in Vector3D a)
```

## Parameters

**a** [Vector3D](#)

The vector that will be the floor.

## Returns

[Vector3D](#)

A vector with [Floor\(float\)](#) called on each component.

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Magnitude(in Vector3D)

Returns the length of this vector

```
public static float Magnitude(in Vector3D a)
```

Parameters

a [Vector3D](#)

One of the values.

Returns

[float](#)

vector length.

## Max(Vector3D, Vector3D)

Returns a vector that is made from the largest components of two vectors.

```
public static Vector3D Max(Vector3D lhs, Vector3D rhs)
```

Parameters

lhs [Vector3D](#)

One of the values.

rhs [Vector3D](#)

The other value.

Returns

## [Vector3D](#)

Whichever of the two values is higher.

## Min(Vector3D, Vector3D)

Returns a vector that is made from the smallest components of two vectors.

```
public static Vector3D Min(Vector3D lhs, Vector3D rhs)
```

Parameters

### [lhs Vector3D](#)

One of the values.

### [rhs Vector3D](#)

The other value.

Returns

## [Vector3D](#)

Whichever of the two values is lower.

## Neg(in Vector3D)

Inverts the values of a vector.

```
public static Vector3D Neg(in Vector3D a)
```

Parameters

### [a Vector3D](#)

The vector to be inverted.

Returns

## [Vector3D](#)

Returns the axes of an inverted vector.

## Neg(bool, bool, bool)

Inverts the values of a vector.

```
public readonly Vector3D Neg(bool negX = true, bool negY = true, bool negZ = true)
```

Parameters

### **negX** [bool](#)

The X axis becomes inverted.

### **negY** [bool](#)

The Y axis becomes inverted.

### **negZ** [bool](#)

The Z axis becomes inverted.

Returns

## [Vector3D](#)

Returns the axes of an inverted vector.

## Normalize(in Vector3D)

Makes this vector have a magnitude of 1.

```
public static Vector3D Normalize(in Vector3D a)
```

Parameters

a [Vector3D](#)

The vector to be normalized.

Returns

[Vector3D](#)

Returns the already normalized vector.

## Round()

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public readonly Vector3D Round()
```

Returns

[Vector3D](#)

The rounded vector.

## Round(in Vector3D)

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public static Vector3D Round(in Vector3D a)
```

Parameters

a [Vector3D](#)

the vector to be rounded

Returns

[Vector3D](#)

The rounded vector.

## SqrMagnitude(in Vector3D)

Returns the squared length of this vector

```
public static float SqrMagnitude(in Vector3D a)
```

Parameters

a [Vector3D](#)

One of the values.

Returns

[float](#)

square length of the vector.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

## Parameters

**format** [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

## Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

## Parameters

**format** [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

**formatProvider** [IFormatProvider](#)

The provider to use to format the value.-or- A null reference ([Nothing](#) in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

## Returns

[string](#)

The value of the current instance in the specified format.

## Operators

### operator +(Vector3D, Vector3D)

Addition operation between two values.([Vector3D](#) + [Vector3D](#))

```
public static Vector3D operator +(Vector3D a, Vector3D b)
```

## Parameters

a [Vector3D](#)

First module.

b [Vector3D](#)

Second module.

## Returns

[Vector3D](#)

The result of the addition.

## operator +(Vector3D, Vector3)

Addition operation between two values.([Vector3D](#) + Godot.Vector3)

```
public static Vector3D operator +(Vector3D a, Vector3 b)
```

## Parameters

a [Vector3D](#)

First module.

b Vector3

Second module.

## Returns

[Vector3D](#)

The result of the addition.

## operator +(Vector3, Vector3D)

Addition operation between two values.(Godot.Vector3 + [Vector3D](#))

```
public static Vector3D operator +(Vector3 a, Vector3D b)
```

### Parameters

a [Vector3](#)

First module.

b [Vector3D](#)

Second module.

### Returns

[Vector3D](#)

The result of the addition.

## operator /(Vector3D, Vector3D)

Division operation between two values.([Vector3D](#) / [Vector3D](#))

```
public static Vector3D operator /(Vector3D a, Vector3D b)
```

### Parameters

a [Vector3D](#)

First module.

b [Vector3D](#)

Second module.

### Returns

[Vector3D](#)

The result of the division.

## operator /(Vector3D, Vector3)

Division operation between two values.([Vector3D](#) / Godot.Vector3)

```
public static Vector3D operator /(Vector3D a, Vector3 b)
```

### Parameters

a [Vector3D](#)

First module.

b Vector3

Second module.

### Returns

[Vector3D](#)

The result of the division.

## operator /(Vector3D, float)

Division operation between two values.([Vector3D](#) / [float](#))

```
public static Vector3D operator /(Vector3D a, float b)
```

### Parameters

a [Vector3D](#)

First module.

b [float](#)

Second module.

Returns

## [Vector3D](#)

The result of the division.

## operator /(Vector3, Vector3D)

Division operation between two values.(Godot.Vector3 / [Vector3D](#))

```
public static Vector3D operator /(Vector3 a, Vector3D b)
```

Parameters

**a** [Vector3](#)

First module.

**b** [Vector3D](#)

Second module.

Returns

## [Vector3D](#)

The result of the division.

## operator ==(in Vector3D, in Vector3D)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(in Vector3D lhs, in Vector3D rhs)
```

Parameters

**lhs** [Vector3D](#)

Object to be compared.

## [rhs](#) [Vector3D](#)

Object of comparison.

Returns

### [bool](#) ↗

Returns the result of the comparison.

## implicit operator Vector2D(Vector3D)

Implicit conversion operator.([Vector3D](#) to [Vector2D](#))

```
public static implicit operator Vector2D(Vector3D v)
```

Parameters

### v [Vector3D](#)

Object to be converted.

Returns

### [Vector2D](#)

## implicit operator Vector4D(Vector3D)

Implicit conversion operator.([Vector3D](#) to [Vector4D](#))

```
public static implicit operator Vector4D(Vector3D v)
```

Parameters

### v [Vector3D](#)

Object to be converted.

Returns

## implicit operator Vector2(Vector3D)

Implicit conversion operator.([Vector3D](#) to Godot.Vector2)

```
public static implicit operator Vector2(Vector3D v)
```

### Parameters

v [Vector3D](#)

Object to be converted.

### Returns

Vector2

## implicit operator Vector3(Vector3D)

Implicit conversion operator.([Vector3D](#) to Godot.Vector3)

```
public static implicit operator Vector3(Vector3D v)
```

### Parameters

v [Vector3D](#)

Object to be converted.

### Returns

Vector3

## implicit operator Vector3D(Vector2)

Implicit conversion operator.(Godot.Vector2 to [Vector3D](#))

```
public static implicit operator Vector3D(Vector2 v)
```

## Parameters

v Vector2

Object to be converted.

## Returns

[Vector3D](#)

## implicit operator Vector3D(Vector3)

Implicit conversion operator.(Godot.Vector3 to [Vector3D](#))

```
public static implicit operator Vector3D(Vector3 v)
```

## Parameters

v Vector3

Object to be converted.

## Returns

[Vector3D](#)

## operator !=(in Vector3D, in Vector3D)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(in Vector3D lhs, in Vector3D rhs)
```

## Parameters

lhs [Vector3D](#)

Object to be compared.

**rhs** [Vector3D](#)

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator \*(Vector3D, Vector3D)

Multiplication operation between two values. ([Vector3D](#) \* [Vector3D](#))

```
public static Vector3D operator *(Vector3D a, Vector3D b)
```

Parameters

**a** [Vector3D](#)

First module.

**b** [Vector3D](#)

Second module.

Returns

[Vector3D](#)

The result of the multiplication.

## operator \*(Vector3D, Vector3)

Multiplication operation between two values. ([Vector3D](#) \* Godot.Vector3)

```
public static Vector3D operator *(Vector3D a, Vector3 b)
```

Parameters

a [Vector3D](#)

First module.

b Vector3

Second module.

Returns

[Vector3D](#)

The result of the multiplication.

## operator \*(Vector3D, float)

Multiplication operation between two values.([Vector3D](#) \* [float](#))

```
public static Vector3D operator *(Vector3D a, float b)
```

Parameters

a [Vector3D](#)

First module.

b [float](#)

Second module.

Returns

[Vector3D](#)

The result of the multiplication.

## operator \*(Vector3, Vector3D)

Multiplication operation between two values.(Godot.Vector3 \* [Vector3D](#))

```
public static Vector3D operator *(Vector3 a, Vector3D b)
```

## Parameters

a [Vector3](#)

First module.

b [Vector3D](#)

Second module.

## Returns

[Vector3D](#)

The result of the multiplication.

## operator -(Vector3D, Vector3D)

Subtraction operation between two values.([Vector3D](#) - [Vector3D](#))

```
public static Vector3D operator -(Vector3D a, Vector3D b)
```

## Parameters

a [Vector3D](#)

First module.

b [Vector3D](#)

Second module.

## Returns

[Vector3D](#)

The result of the subtraction.

## operator -(Vector3D, Vector3)

Subtraction operation between two values.([Vector3D](#) - Godot.Vector3)

```
public static Vector3D operator -(Vector3D a, Vector3 b)
```

### Parameters

**a** [Vector3D](#)

First module.

**b** Vector3

Second module.

### Returns

[Vector3D](#)

The result of the subtraction.

## operator -(Vector3, Vector3D)

Subtraction operation between two values.(Godot.Vector3 - [Vector3D](#))

```
public static Vector3D operator -(Vector3 a, Vector3D b)
```

### Parameters

**a** Vector3

First module.

**b** [Vector3D](#)

Second module.

### Returns

[Vector3D](#)

The result of the subtraction.

## operator -(Vector3D)

The operator allows us to reverse the value.

```
public static Vector3D operator -(Vector3D a)
```

Parameters

a [Vector3D](#)

Or value that will be invested.

Returns

[Vector3D](#)

Returns the result of the inversion.

# Struct Vector3DInt

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Representation of a three-dimensional vector using integers.

```
[Serializable]
public struct Vector3DInt : IIntVector, IFormattable
```

Implements

[IIntVector](#), [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Constructors

### Vector3DInt(in Vector3DInt)

Starts a new instance of the object.

```
public Vector3DInt(in Vector3DInt vector)
```

Parameters

vector [Vector3DInt](#)

### Vector3DInt(in int, in int)

Starts a new instance of the object.

```
public Vector3DInt(in int x, in int y)
```

Parameters

x [int](#)

y [int](#)

## Vector3DInt(in int, in int, in int)

Starts a new instance of the object.

```
public Vector3DInt(in int x, in int y, in int z)
```

## Parameters

x [int](#)

y [int](#)

z [int](#)

## Fields

### X

X component of the vector.

```
[ShowProperty(true)]  
public int x
```

### Field Value

[int](#)

### y

Y component of the vector.

```
[ShowProperty(true)]  
public int y
```

## Field Value

[int ↗](#)

## Z

Z component of the vector.

```
[ShowProperty(true)]  
public int z
```

## Field Value

[int ↗](#)

# Properties

## AxisCount

Number of axles.

```
public readonly int AxisCount { get; }
```

## Property Value

[int ↗](#)

Returns the number of axes a vector has.

## Back

Shorthand for writing Vector3(0,0,-1).

```
public static Vector3DInt Back { get; }
```

## Property Value

## [Vector3DInt](#)

### Down

Shorthand for writing Vector3(0,1,0).

```
public static Vector3DInt Down { get; }
```

### Property Value

#### [Vector3DInt](#)

### Forward

Shorthand for writing Vector3(0,0,1).

```
public static Vector3DInt Forward { get; }
```

### Property Value

#### [Vector3DInt](#)

### this[int]

Allows you to access the axes of a vector through an index.

```
public int this[int index] { readonly get; set; }
```

### Parameters

#### [index](#) [int](#)

The axis index.

### Property Value

#### [int](#)

Sets the value of an axis by specifying its index.

## Left

Shorthand for writing Vector3(-1,0,0).

```
public static Vector3DInt Left { get; }
```

Property Value

[Vector3DInt](#)

## One

Shorthand for writing Vector3(1,1,1).

```
public static Vector3DInt One { get; }
```

Property Value

[Vector3DInt](#)

## Right

Shorthand for writing Vector3(1,0,0).

```
public static Vector3DInt Right { get; }
```

Property Value

[Vector3DInt](#)

## Up

Shorthand for writing Vector3(0,-1,0).

```
public static Vector3DInt Up { get; }
```

Property Value

[Vector3DInt](#)

## Zero

Shorthand for writing Vector3(0,0,0).

```
public static Vector3DInt Zero { get; }
```

Property Value

[Vector3DInt](#)

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector](#).x to [IVector](#).y.

```
public readonly float aspect { get; }
```

Property Value

[float](#) ↗

The [IVector](#).x component divided by the [IVector](#).y component.

## ceilToInt

```
public readonly Vector3DInt ceilToInt { get; }
```

Property Value

[Vector3DInt](#)

## floorToInt

```
public readonly Vector3DInt floorToInt { get; }
```

Property Value

[Vector3DInt](#)

## magnitude

Returns the length (magnitude) of this vector.

```
public readonly float magnitude { get; }
```

Property Value

[float](#)

The length of this vector.

### See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
public readonly int sqrMagnitude { get; }
```

Property Value

[int](#)

The squared length of this vector.

# Methods

## Abs(in Vector3DInt)

Returns an absolute value of the vector.

```
public static Vector3DInt Abs(in Vector3DInt a)
```

Parameters

a [Vector3DInt](#)

The vector to become absolute.

Returns

[Vector3DInt](#)

Returns the vector with its absolute value axes.

## Abs(bool, bool, bool)

Returns an absolute value of the vector.

```
public readonly Vector3DInt Abs(bool absX = true, bool absY = true, bool absZ = true)
```

Parameters

absX [bool](#)

The X axis becomes absolute.

absY [bool](#)

The Y axis becomes absolute.

absZ [bool](#)

The Z axis becomes absolute.

Returns

## [Vector3DInt](#)

Returns the vector with its absolute value axes.

### CeilToInt(in Vector3D)

Converts a floating-point vector to an integer vector and applies a Ceiling to each value.

```
public static Vector3DInt CeilToInt(in Vector3D a)
```

Parameters

a [Vector3D](#)

The vector to be converted is ceiling

Returns

## [Vector3DInt](#)

Returns the converted vector and ceiling

### Distance(in Vector3DInt, in Vector3DInt)

Returns the distance between a and b.

```
public static float Distance(in Vector3DInt a, in Vector3DInt b)
```

Parameters

a [Vector3DInt](#)

One of the values.

b [Vector3DInt](#)

The other value.

Returns

[float](#)

The distance between two vectors.

## Equals(Vector3DInt)

```
public readonly bool Equals(Vector3DInt other)
```

Parameters

**other** [Vector3DInt](#)

Returns

[bool](#)

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

Parameters

**obj** [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if **obj** and this instance are the same type and represent the same value; otherwise, [false](#).

## FloorToInt(in Vector3D)

Converts a floating-point vector to an integer vector and applies a Floor to each value.

```
public static Vector3DInt FloorToInt(in Vector3D a)
```

## Parameters

a [Vector3D](#)

The vector to be converted is floor

## Returns

[Vector3DInt](#)

Returns the converted vector and floor

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

## Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Magnitude(in Vector3DInt)

Returns the length of this vector

```
public static float Magnitude(in Vector3DInt a)
```

## Parameters

a [Vector3DInt](#)

One of the values.

Returns

[float](#)

vector length.

## Max(Vector3DInt, Vector3DInt)

Returns a vector that is made from the largest components of two vectors.

```
public static Vector3DInt Max(Vector3DInt lhs, Vector3DInt rhs)
```

Parameters

[lhs](#) [Vector3DInt](#)

One of the values.

[rhs](#) [Vector3DInt](#)

The other value.

Returns

[Vector3DInt](#)

Whichever of the two values is higher.

## Min(Vector3DInt, Vector3DInt)

Returns a vector that is made from the smallest components of two vectors.

```
public static Vector3DInt Min(Vector3DInt lhs, Vector3DInt rhs)
```

Parameters

[lhs](#) [Vector3DInt](#)

One of the values.

**rhs** [Vector3DInt](#)

The other value.

Returns

[Vector3DInt](#)

Whichever of the two values is lower.

## Neg(in Vector3DInt)

Inverts the values of a vector.

```
public static Vector3DInt Neg(in Vector3DInt a)
```

Parameters

**a** [Vector3DInt](#)

The vector to be inverted.

Returns

[Vector3DInt](#)

Returns the axes of an inverted vector.

## Neg(bool, bool, bool)

Inverts the values of a vector.

```
public readonly Vector3DInt Neg(bool negX = true, bool negY = true, bool negZ = true)
```

Parameters

**negX** [bool](#)

The X axis becomes inverted.

### negY [bool](#)

The Y axis becomes inverted.

### negZ [bool](#)

The Z axis becomes inverted.

Returns

### [Vector3DInt](#)

Returns the axes of an inverted vector.

## RoundToInt(in Vector3D)

Converts a floating-point vector to an integer vector by performing a Round for each value.

```
public static Vector3DInt RoundToInt(in Vector3D a)
```

Parameters

### a [Vector3D](#)

The vector that will be converted and rounded

Returns

### [Vector3DInt](#)

Returns the converted and rounded vector

## SqrMagnitude(in Vector3DInt)

Returns the squared length of this vector

```
public static int SqrMagnitude(in Vector3DInt a)
```

Parameters

## a [Vector3DInt](#)

One of the values.

Returns

[int](#)

square length of the vector.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

### Parameters

**format** [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

**formatProvider** [IFormatProvider](#)

The provider to use to format the value.-or- A null reference ([Nothing](#) in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

### Returns

[string](#)

The value of the current instance in the specified format.

## Operators

### operator +(Vector3DInt, Vector3DInt)

Addition operation between two values.([Vector3DInt](#) + [Vector3DInt](#))

```
public static Vector3DInt operator +(Vector3DInt a, Vector3DInt b)
```

### Parameters

**a** [Vector3DInt](#)

First module.

b [Vector3DInt](#)

Second module.

Returns

[Vector3DInt](#)

The result of the addition.

## operator /(Vector3DInt, Vector3DInt)

Division operation between two values.([Vector3DInt](#) / [Vector3DInt](#))

```
public static Vector3DInt operator /(Vector3DInt a, Vector3DInt b)
```

Parameters

a [Vector3DInt](#)

First module.

b [Vector3DInt](#)

Second module.

Returns

[Vector3DInt](#)

The result of the division.

## operator /(Vector3DInt, int)

Division operation between two values.([Vector3DInt](#) / [int](#))

```
public static Vector3DInt operator /(Vector3DInt a, int b)
```

Parameters

a [Vector3DInt](#)

First module.

b [int](#)

Second module.

Returns

[Vector3DInt](#)

The result of the division.

## operator ==(in Vector3DInt, in Vector3DInt)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(in Vector3DInt lhs, in Vector3DInt rhs)
```

Parameters

lhs [Vector3DInt](#)

Object to be compared.

rhs [Vector3DInt](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## explicit operator Vector3DInt(Vector2D)

Explicit conversion operator.([Vector2D](#) to [Vector3DInt](#))

```
public static explicit operator Vector3DInt(Vector2D v)
```

## Parameters

v [Vector2D](#)

Object to be converted.

## Returns

[Vector3DInt](#)

## explicit operator Vector3DInt(Vector3D)

Explicit conversion operator.([Vector3D](#) to [Vector3DInt](#))

```
public static explicit operator Vector3DInt(Vector3D v)
```

## Parameters

v [Vector3D](#)

Object to be converted.

## Returns

[Vector3DInt](#)

## explicit operator Vector3DInt(Vector4D)

Explicit conversion operator.([Vector4D](#) to [Vector3DInt](#))

```
public static explicit operator Vector3DInt(Vector4D v)
```

## Parameters

v [Vector4D](#)

Object to be converted.

Returns

[Vector3DInt](#)

## explicit operator Vector3DInt(Vector2)

Explicit conversion operator.(Godot.Vector2 to [Vector3DInt](#))

```
public static explicit operator Vector3DInt(Vector2 v)
```

Parameters

v [Vector2](#)

Object to be converted.

Returns

[Vector3DInt](#)

## implicit operator Vector2D(Vector3DInt)

Implicit conversion operator.([Vector3DInt](#) to [Vector2D](#))

```
public static implicit operator Vector2D(Vector3DInt v)
```

Parameters

v [Vector3DInt](#)

Object to be converted.

Returns

[Vector2D](#)

## implicit operator Vector2DInt(Vector3DInt)

Implicit conversion operator.([Vector3DInt](#) to [Vector2DInt](#))

```
public static implicit operator Vector2DInt(Vector3DInt v)
```

### Parameters

v [Vector3DInt](#)

Object to be converted.

### Returns

[Vector2DInt](#)

## implicit operator Vector3D(Vector3DInt)

Implicit conversion operator.([Vector3DInt](#) to [Vector3D](#))

```
public static implicit operator Vector3D(Vector3DInt v)
```

### Parameters

v [Vector3DInt](#)

Object to be converted.

### Returns

[Vector3D](#)

## implicit operator Vector4D(Vector3DInt)

Implicit conversion operator.([Vector3DInt](#) to [Vector4D](#))

```
public static implicit operator Vector4D(Vector3DInt v)
```

## Parameters

v [Vector3DInt](#)

Object to be converted.

## Returns

[Vector4D](#)

## implicit operator Vector2(Vector3DInt)

Implicit conversion operator.([Vector3DInt](#) to Godot.Vector2)

```
public static implicit operator Vector2(Vector3DInt v)
```

## Parameters

v [Vector3DInt](#)

Object to be converted.

## Returns

Vector2

## operator !=(in Vector3DInt, in Vector3DInt)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(in Vector3DInt lhs, in Vector3DInt rhs)
```

## Parameters

lhs [Vector3DInt](#)

Object to be compared.

rhs [Vector3DInt](#)

Object of comparison.

Returns

[bool](#) ↗

Returns the result of the comparison.

## operator \*(Vector3DInt, Vector3DInt)

Multiplication operation between two values. ([Vector3DInt](#) \* [Vector3DInt](#))

```
public static Vector3DInt operator *(Vector3DInt a, Vector3DInt b)
```

Parameters

a [Vector3DInt](#)

First module.

b [Vector3DInt](#)

Second module.

Returns

[Vector3DInt](#)

The result of the multiplication.

## operator \*(Vector3DInt, int)

Multiplication operation between two values. ([Vector3DInt](#) \* [int](#) ↗)

```
public static Vector3DInt operator *(Vector3DInt a, int b)
```

Parameters

a [Vector3DInt](#)

First module.

b [int](#)

Second module.

Returns

[Vector3DInt](#)

The result of the multiplication.

## operator -(Vector3DInt, Vector3DInt)

Subtraction operation between two values.([Vector3DInt](#) - [Vector3DInt](#))

```
public static Vector3DInt operator -(Vector3DInt a, Vector3DInt b)
```

Parameters

a [Vector3DInt](#)

First module.

b [Vector3DInt](#)

Second module.

Returns

[Vector3DInt](#)

The result of the subtraction.

## operator -(Vector3DInt)

The operator allows us to reverse the value.

```
public static Vector3DInt operator -(Vector3DInt a)
```

## Parameters

a [Vector3DInt](#)

Or value that will be invested.

## Returns

[Vector3DInt](#)

Returns the result of the inversion.

# Struct Vector4D

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Represents a four-dimensional vector.(Four-axis vector)

```
[Serializable]
public struct Vector4D : IVectorGeneric<Vector4D>, IEquatable<Vector4D>,
IVector, IFormattable
```

Implements

[IVectorGeneric<Vector4D>](#), [IEquatable](#)<[Vector4D](#)>, [IVector](#), [IFormattable](#)

Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### Vector4D(Quaternion)

Starts a new instance of the object.

```
public Vector4D(Quaternion vector)
```

Parameters

vector [Quaternion](#)

### Vector4D(Vector4D)

Starts a new instance of the object.

```
public Vector4D(Vector4D vector)
```

Parameters

`vector` [Vector4D](#)

## Vector4D(float, float)

Starts a new instance of the object.

```
public Vector4D(float x, float y)
```

Parameters

`x` [float](#)

`y` [float](#)

## Vector4D(float, float, float)

Starts a new instance of the object.

```
public Vector4D(float x, float y, float z)
```

Parameters

`x` [float](#)

`y` [float](#)

`z` [float](#)

## Vector4D(float, float, float, float)

Starts a new instance of the object.

```
public Vector4D(float x, float y, float z, float w)
```

Parameters

`x` [float](#)

y [float](#)

z [float](#)

w [float](#)

## Fields

W

W component of the vector.

```
[ShowProperty(true)]  
public float w
```

Field Value

[float](#)

X

X component of the vector.

```
[ShowProperty(true)]  
public float x
```

Field Value

[float](#)

y

Y component of the vector.

```
[ShowProperty(true)]  
public float y
```

## Field Value

[float](#) ↗

### Z

Z component of the vector.

```
[ShowProperty(true)]  
public float z
```

## Field Value

[float](#) ↗

## Properties

### AxisCount

Number of axles.

```
public readonly int AxisCount { get; }
```

## Property Value

[int](#) ↗

Returns the number of axes a vector has.

### this[int]

Allows you to access the axes of a vector through an index.

```
public float this[int index] { readonly get; set; }
```

## Parameters

## [index](#) [int](#)

The axis index.

## Property Value

### [float](#)

Sets the value of an axis by specifying its index.

## Normalized

Returns the vector scaled to unit length. Equivalent to `v / v.Length()`.

```
public readonly Vector4D Normalized { get; }
```

## Property Value

### [Vector4D](#)

A normalized version of the vector.

## One

Shorthand for writing `Vector4(1,1,1,1)`.

```
public static Vector4D One { get; }
```

## Property Value

### [Vector4D](#)

## Zero

Shorthand for writing `Vector4(0,0,0,0)`.

```
public static Vector4D Zero { get; }
```

Property Value

[Vector4D](#)

## aspect

Returns the aspect ratio of this vector, the ratio of [IVector.x](#) to [IVector.y](#).

```
public readonly float aspect { get; }
```

Property Value

[float](#)

The [IVector.x](#) component divided by the [IVector.y](#) component.

## ceil

Returns a new vector with all components rounded up (towards positive infinity).

```
public readonly Vector4D ceil { get; }
```

Property Value

[Vector4D](#)

A vector with [Ceil\(float\)](#) called on each component.

## floor

Returns a new vector with all components rounded down (towards negative infinity).

```
public readonly Vector4D floor { get; }
```

Property Value

[Vector4D](#)

A vector with [Floor\(float\)](#) called on each component.

## magnitude

Returns the length (magnitude) of this vector.

```
public readonly float magnitude { get; }
```

### Property Value

[float](#)

The length of this vector.

### See Also

[LengthSquared\(\)](#)

## sqrMagnitude

Returns the squared length (squared magnitude) of this vector. This method runs faster than [magnitude](#), so prefer it if you need to compare vectors or need the squared length for some formula.

```
public readonly float sqrMagnitude { get; }
```

### Property Value

[float](#)

The squared length of this vector.

## Methods

### Abs(in Vector4D)

Returns an absolute value of the vector.

```
public static Vector4D Abs(in Vector4D a)
```

## Parameters

### a [Vector4D](#)

The vector to become absolute.

## Returns

### [Vector4D](#)

Returns the vector with its absolute value axes.

## Abs(bool, bool, bool, bool)

Returns an absolute value of the vector.

```
public readonly Vector4D Abs(bool absX = true, bool absY = true, bool absZ = true, bool absW = true)
```

## Parameters

### absX [bool](#) ↗

The X axis becomes absolute.

### absY [bool](#) ↗

The Y axis becomes absolute.

### absZ [bool](#) ↗

The Z axis becomes absolute.

### absW [bool](#) ↗

The W axis becomes absolute.

## Returns

### [Vector4D](#)

Returns the vector with its absolute value axes.

## Ceil(in Vector4D)

Returns a new vector with all components rounded up (towards positive infinity).

```
public static Vector4D Ceil(in Vector4D a)
```

### Parameters

a [Vector4D](#)

The vector that will be the ceiling.

### Returns

[Vector4D](#)

A vector with [Ceil\(float\)](#) called on each component.

## Distance(in Vector4D, Vector4D)

Returns the distance between a and b.

```
public static float Distance(in Vector4D a, Vector4D b)
```

### Parameters

a [Vector4D](#)

One of the values.

b [Vector4D](#)

The other value.

### Returns

[float](#)

The distance between two vectors.

## Dot(in Vector4D, in Vector4D)

Dot Product of two vectors.

```
public static float Dot(in Vector4D a, in Vector4D b)
```

Parameters

a [Vector4D](#)

One of the values.

b [Vector4D](#)

The other value.

Returns

[float](#)

returns the result of the dot product.

## Equals(Vector4D)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(Vector4D other)
```

Parameters

other [Vector4D](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

Parameters

**obj** [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if **obj** and this instance are the same type and represent the same value; otherwise, [false](#).

## Floor(in Vector4D)

Returns a new vector with all components rounded down (towards negative infinity).

```
public static Vector4D Floor(in Vector4D a)
```

Parameters

**a** [Vector4D](#)

The vector that will be the floor.

Returns

[Vector4D](#)

A vector with [Floor\(float\)](#) called on each component.

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## IsNormalized(in IVector)

Checks if the vector is normalized.

```
public static bool IsNormalized(in IVector a)
```

Parameters

a [IVector](#)

The vector to be checked.

Returns

[bool](#)

Returns [true](#) when vector is normalized.

## Magnitude(in Vector4D)

Returns the length of this vector

```
public static float Magnitude(in Vector4D a)
```

Parameters

a [Vector4D](#)

One of the values.

Returns

[float](#)

vector length.

## Max(in Vector4D, in Vector4D)

Returns a vector that is made from the largest components of two vectors.

```
public static Vector4D Max(in Vector4D lhs, in Vector4D rhs)
```

Parameters

[lhs](#) [Vector4D](#)

One of the values.

[rhs](#) [Vector4D](#)

The other value.

Returns

[Vector4D](#)

Whichever of the two values is higher.

## Min(in Vector4D, in Vector4D)

Returns a vector that is made from the smallest components of two vectors.

```
public static Vector4D Min(in Vector4D lhs, in Vector4D rhs)
```

Parameters

[lhs](#) [Vector4D](#)

One of the values.

**rhs** [Vector4D](#)

The other value.

Returns

[Vector4D](#)

Whichever of the two values is lower.

## Neg(in Vector4D)

Inverts the values of a vector.

```
public static Vector4D Neg(in Vector4D a)
```

Parameters

**a** [Vector4D](#)

The vector to be inverted.

Returns

[Vector4D](#)

Returns the axes of an inverted vector.

## Neg(bool, bool, bool, bool)

Inverts the values of a vector.

```
public readonly Vector4D Neg(bool negX = true, bool negY = true, bool negZ = true, bool negW = true)
```

Parameters

**negX** [bool](#)

The X axis becomes inverted.

**negY** [bool](#)

The Y axis becomes inverted.

**negZ** [bool](#)

The Z axis becomes inverted.

**negW** [bool](#)

The W axis becomes inverted.

Returns

[Vector4D](#)

Returns the axes of an inverted vector.

## Normalize([in Vector4D](#))

Makes this vector have a magnitude of 1.

```
public static Vector4D Normalize(in Vector4D a)
```

Parameters

**a** [Vector4D](#)

The vector to be normalized.

Returns

[Vector4D](#)

Returns the already normalized vector.

## Round()

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public readonly Vector4D Round()
```

Returns

[Vector4D](#)

The rounded vector.

## Round(in Vector4D)

Returns this vector with all components rounded to the nearest integer, with halfway cases rounded towards the nearest multiple of two.

```
public static Vector4D Round(in Vector4D a)
```

Parameters

a [Vector4D](#)

the vector to be rounded

Returns

[Vector4D](#)

The rounded vector.

## SqrMagnitude(in Vector4D)

Returns the squared length of this vector

```
public static float SqrMagnitude(in Vector4D a)
```

Parameters

a [Vector4D](#)

One of the values.

Returns

[float](#)

square length of the vector.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## ToString(string)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format)
```

Parameters

[format](#) [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

Returns

[string](#)

The value of the current instance in the specified format.

## ToString(string, IFormatProvider)

Formats the value of the current instance using the specified format.

```
public readonly string ToString(string format, IFormatProvider formatProvider)
```

## Parameters

**format** [string](#)

The format to use.-or- A null reference ([Nothing](#) in Visual Basic) to use the default format defined for the type of the [IFormattable](#) implementation.

**formatProvider** [IFormatProvider](#)

The provider to use to format the value.-or- A null reference ([Nothing](#) in Visual Basic) to obtain the numeric format information from the current locale setting of the operating system.

## Returns

[string](#)

The value of the current instance in the specified format.

# Operators

## operator +(Vector4D, Vector4D)

Addition operation between two values.([Vector4D](#) + [Vector4D](#))

```
public static Vector4D operator +(Vector4D a, Vector4D b)
```

## Parameters

**a** [Vector4D](#)

First module.

**b** [Vector4D](#)

Second module.

Returns

## [Vector4D](#)

The result of the addition.

## operator /(Vector4D, Vector4D)

Division operation between two values.([Vector4D / Vector4D](#))

```
public static Vector4D operator /(Vector4D a, Vector4D b)
```

Parameters

### a [Vector4D](#)

First module.

### b [Vector4D](#)

Second module.

Returns

## [Vector4D](#)

The result of the division.

## operator /(Vector4D, float)

Division operation between two values.([Vector4D / float](#))

```
public static Vector4D operator /(Vector4D a, float b)
```

Parameters

### a [Vector4D](#)

First module.

b [float](#)

Second module.

Returns

[Vector4D](#)

The result of the division.

## operator /(float, Vector4D)

Division operation between two values.([float](#) / [Vector4D](#))

```
public static Vector4D operator /(float a, Vector4D b)
```

Parameters

a [float](#)

First module.

b [Vector4D](#)

Second module.

Returns

[Vector4D](#)

The result of the division.

## operator ==(Vector4D, Vector4D)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Vector4D A, Vector4D B)
```

Parameters

A [Vector4D](#)

Object to be compared.

B [Vector4D](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## implicit operator Quaternion(Vector4D)

Implicit conversion operator.([Vector4D](#) to [Quaternion](#))

```
public static implicit operator Quaternion(Vector4D v)
```

Parameters

v [Vector4D](#)

Object to be converted.

Returns

[Quaternion](#)

## implicit operator Vector2D(Vector4D)

Implicit conversion operator.([Vector4D](#) to [Vector2D](#))

```
public static implicit operator Vector2D(Vector4D v)
```

Parameters

v [Vector4D](#)

Object to be converted.

Returns

[Vector2D](#)

## implicit operator Vector3D(Vector4D)

Implicit conversion operator.([Vector4D](#) to [Vector3D](#))

```
public static implicit operator Vector3D(Vector4D v)
```

Parameters

v [Vector4D](#)

Object to be converted.

Returns

[Vector3D](#)

## implicit operator Vector2(Vector4D)

Implicit conversion operator.([Vector4D](#) to Godot.Vector2)

```
public static implicit operator Vector2(Vector4D v)
```

Parameters

v [Vector4D](#)

Object to be converted.

Returns

Vector2

## implicit operator Vector3(Vector4D)

Implicit conversion operator.([Vector4D](#) to Godot.Vector3)

```
public static implicit operator Vector3(Vector4D v)
```

### Parameters

v [Vector4D](#)

Object to be converted.

### Returns

Vector3

## implicit operator Vector4D(Vector2)

Implicit conversion operator.(Godot.Vector2 to [Vector4D](#))

```
public static implicit operator Vector4D(Vector2 v)
```

### Parameters

v Vector2

Object to be converted.

### Returns

[Vector4D](#)

## implicit operator Vector4D(Vector3)

Implicit conversion operator.(Godot.Vector3 to [Vector4D](#))

```
public static implicit operator Vector4D(Vector3 v)
```

## Parameters

v Vector3

Object to be converted.

## Returns

[Vector4D](#)

## operator !=(Vector4D, Vector4D)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Vector4D A, Vector4D B)
```

## Parameters

A [Vector4D](#)

Object to be compared.

B [Vector4D](#)

Object of comparison.

## Returns

[bool](#)

Returns the result of the comparison.

## operator \*(Vector4D, Vector4D)

Multiplication operation between two values. ([Vector4D](#) \* [Vector4D](#))

```
public static Vector4D operator *(Vector4D a, Vector4D b)
```

## Parameters

a [Vector4D](#)

First module.

b [Vector4D](#)

Second module.

Returns

[Vector4D](#)

The result of the multiplication.

## operator \*(Vector4D, float)

Multiplication operation between two values. ([Vector4D](#) \* [float](#))

```
public static Vector4D operator *(Vector4D a, float b)
```

Parameters

a [Vector4D](#)

First module.

b [float](#)

Second module.

Returns

[Vector4D](#)

The result of the multiplication.

## operator \*(float, Vector4D)

Multiplication operation between two values. ([float](#) \* [Vector4D](#))

```
public static Vector4D operator *(float a, Vector4D b)
```

## Parameters

a [float](#)

First module.

b [Vector4D](#)

Second module.

## Returns

[Vector4D](#)

The result of the multiplication.

## operator -(Vector4D, Vector4D)

Subtraction operation between two values. ([Vector4D](#) - [Vector4D](#))

```
public static Vector4D operator -(Vector4D a, Vector4D b)
```

## Parameters

a [Vector4D](#)

First module.

b [Vector4D](#)

Second module.

## Returns

[Vector4D](#)

The result of the subtraction.

## operator -(Vector4D)

The operator allows us to reverse the value.

```
public static Vector4D operator -(Vector4D a)
```

### Parameters

a [Vector4D](#)

Or value that will be inverted.

### Returns

[Vector4D](#)

Returns the result of the inversion.

# Struct VectorEqualityComparer

Namespace: [Cobilas.GodotEngine.Utility.Numerics](#)

Assembly: com.cobilas.godot.utility.dll

Defines methods to support comparing vectors for equality.

```
public readonly struct VectorEqualityComparer : IEqualityComparer,
    IEqualityComparer<Vector2D>, IEqualityComparer<Vector3D>, IEqualityComparer<Vector4D>,
    IEqualityComparer<Vector2DInt>, IEqualityComparer<Vector3DInt>
```

## Implements

[IEqualityComparer](#), [IEqualityComparer <Vector2D>](#), [IEqualityComparer <Vector3D>](#),  
[IEqualityComparer <Vector4D>](#), [IEqualityComparer <Vector2DInt>](#),  
[IEqualityComparer <Vector3DInt>](#)

## Inherited Members

[ValueType.Equals\(object\)](#), [ValueType.GetHashCode\(\)](#), [ValueType.ToString\(\)](#),  
[object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Methods

### Equals(Vector2D, Vector2D)

Determines whether the specified objects are equal.

```
public bool Equals(Vector2D x, Vector2D y)
```

#### Parameters

x [Vector2D](#)

The first object of the parameter type to be compared.

y [Vector2D](#)

The second object of the parameter type to be compared.

#### Returns

## [bool](#)

`true` if the specified objects are equal; otherwise, `false`.

## Remarks

Implement this method to provide a custom equality comparison for the specified type.

## Equals(Vector2DInt, Vector2DInt)

Determines whether the specified objects are equal.

```
public bool Equals(Vector2DInt x, Vector2DInt y)
```

## Parameters

`x` [Vector2DInt](#)

The first object of the parameter type to be compared.

`y` [Vector2DInt](#)

The second object of the parameter type to be compared.

## Returns

## [bool](#)

`true` if the specified objects are equal; otherwise, `false`.

## Remarks

Implement this method to provide a custom equality comparison for the specified type.

## Equals(Vector3D, Vector3D)

Determines whether the specified objects are equal.

```
public bool Equals(Vector3D x, Vector3D y)
```

## Parameters

x [Vector3D](#)

The first object of the parameter type to be compared.

y [Vector3D](#)

The second object of the parameter type to be compared.

## Returns

[bool](#)

**true** if the specified objects are equal; otherwise, **false**.

## Remarks

Implement this method to provide a custom equality comparison for the specified type.

## Equals(Vector3DInt, Vector3DInt)

Determines whether the specified objects are equal.

```
public bool Equals(Vector3DInt x, Vector3DInt y)
```

## Parameters

x [Vector3DInt](#)

The first object of the parameter type to be compared.

y [Vector3DInt](#)

The second object of the parameter type to be compared.

## Returns

[bool](#)

**true** if the specified objects are equal; otherwise, **false**.

## Remarks

Implement this method to provide a custom equality comparison for the specified type.

## Equals(Vector4D, Vector4D)

Determines whether the specified objects are equal.

```
public bool Equals(Vector4D x, Vector4D y)
```

### Parameters

x [Vector4D](#)

The first object of the parameter type to be compared.

y [Vector4D](#)

The second object of the parameter type to be compared.

### Returns

[bool](#) ↗

**true** if the specified objects are equal; otherwise, **false**.

## Remarks

Implement this method to provide a custom equality comparison for the specified type.

## Equals(object, object)

Determines whether the specified objects are equal.

```
public bool Equals(object x, object y)
```

### Parameters

x [object](#) ↗

The first object of the parameter type to be compared.

**y** [object](#)

The second object of the parameter type to be compared.

Returns

[bool](#)

**true** if the specified objects are equal; otherwise, **false**.

Remarks

Implement this method to provide a custom equality comparison for the specified type.

## GetHashCode(Vector2D)

Returns a hash code for the specified object.

```
public int GetHashCode(Vector2D obj)
```

Parameters

**obj** [Vector2D](#)

The [object](#) for which a hash code is to be returned.

Returns

[int](#)

A hash code for the specified object.

Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

## GetHashCode(Vector2DInt)

Returns a hash code for the specified object.

```
public int GetHashCode(Vector2DInt obj)
```

Parameters

**obj** [Vector2DInt](#)

The [object](#) for which a hash code is to be returned.

Returns

[int](#)

A hash code for the specified object.

Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

## GetHashCode(Vector3D)

Returns a hash code for the specified object.

```
public int GetHashCode(Vector3D obj)
```

Parameters

**obj** [Vector3D](#)

The [object](#) for which a hash code is to be returned.

Returns

[int](#)

A hash code for the specified object.

Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

## GetHashCode(Vector3DInt)

Returns a hash code for the specified object.

```
public int GetHashCode(Vector3DInt obj)
```

Parameters

**obj** [Vector3DInt](#)

The [object](#) for which a hash code is to be returned.

Returns

[int](#)

A hash code for the specified object.

Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

## GetHashCode(Vector4D)

Returns a hash code for the specified object.

```
public int GetHashCode(Vector4D obj)
```

Parameters

**obj** [Vector4D](#)

The [object](#) for which a hash code is to be returned.

Returns

[int](#)

A hash code for the specified object.

## Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

## GetHashCode(object)

Returns a hash code for the specified object.

```
public int GetHashCode(object obj)
```

## Parameters

[obj](#) [object](#)

The [object](#) for which a hash code is to be returned.

## Returns

[int](#)

A hash code for the specified object.

## Remarks

Implement this method to provide a customized hash code for type, corresponding to the customized equality comparison provided by the Equals method.

# Namespace Cobilas.GodotEngine.Utility.Physics

## Classes

### [Physics2D](#)

This class provides methods for detecting 2D physics bodies.

## Structs

### [Hit2D](#)

Responsible for storing information about a 2D collision.

### [RayHit2D](#)

Responsible for storing information about a 2D collision ray.

# Struct Hit2D

Namespace: [Cobilas.GodotEngine.Utility.Physics](#)

Assembly: com.cobilas.godot.utility.dll

Responsible for storing information about a 2D collision.

```
public struct Hit2D
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Fields

### Collision\_Key

Represents the collider key.

```
public const string Collision_Key = "collider"
```

Field Value

[string](#)

### ID\_Key

Represents the collider\_id key.

```
public const string ID_Key = "collider_id"
```

Field Value

[string](#)

## MetaData\_Key

Represents the metadata key.

```
public const string MetaData_Key = "metadata"
```

Field Value

[string](#)

## RID\_Key

Represents the rid key.

```
public const string RID_Key = "rid"
```

Field Value

[string](#)

## Properties

### Collision

The colliding object.

```
public readonly Node Collision { get; }
```

Property Value

Node

Returns the collider of the object.

### ID

The colliding object's ID.

```
public readonly int ID { get; }
```

## Property Value

[int](#)

Returns the object ID.

## MetaData

The intersecting shape's metadata. This metadata is different from [GetMeta\(string, object\)](#), and is set with [ShapeSetData\(RID, object\)](#).

```
public readonly object MetaData { get; }
```

## Property Value

[object](#)

Returns the object's metadata.

## Name

The name of the object.

```
public readonly string Name { get; }
```

## Property Value

[string](#)

Returns the name of the object.

## RID

The intersecting object's Godot.RID.

```
public readonly RID RID { get; }
```

## Property Value

### RID

Returns the RID of the object.

## Methods

### IsValid(Dictionary?)

Checks if the Godot.Collections.Dictionary type value is valid for conversion.

```
public static bool IsValid(Dictionary? dictionary)
```

#### Parameters

**dictionary** Dictionary

The object to be checked.

#### Returns

bool ↗

Returns **true** when the Godot.Collections.Dictionary type value is valid.

## Operators

### explicit operator Hit2D(Dictionary?)

Explicit conversion operator.(Godot.Collections.Dictionary to [Hit2D](#))

```
public static explicit operator Hit2D(Dictionary? D)
```

#### Parameters

## D Dictionary

Object to be converted.

Returns

[Hit2D](#)

Remarks

The conversion from Godot.Collections.Dictionary to [Hit2D](#) will be valid when Godot.Collections.Dictionary contains the keys "collider\_id", "rid", "metadata" and "collider".

Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Collections.Dictionary object is null.

[InvalidCastException](#)

The exception is called when the Dictionary object is not valid for conversion because it does not contain the keys "collider\_id", "rid", "metadata" and "collider".

# Class Physics2D

Namespace: [Cobilas.GodotEngine.Utility.Physics](#)

Assembly: com.cobilas.godot.utility.dll

This class provides methods for detecting 2D physics bodies.

```
[RunTimeInitializationClass("Physics2D", Priority.StartBefore, 0, false)]
public class Physics2D : Node2D, IDisposable
```

## Inheritance

[Object](#) ← Object ← Node ← CanvasItem ← Node2D ← Physics2D

## Implements

[IDisposable](#)

## Inherited Members

Node2D.SetPosition(Vector2) , [Node2D.SetRotation\(float\)](#) , [Node2D.SetRotationDegrees\(float\)](#) ,  
Node2D.setScale(Vector2) , Node2D.GetPosition() , Node2D.GetRotation() ,  
Node2D.GetRotationDegrees() , Node2D.GetScale() , [Node2D.Rotate\(float\)](#) ,  
[Node2D.MoveLocalX\(float, bool\)](#) , [Node2D.MoveLocalY\(float, bool\)](#) , Node2D.Translate(Vector2) ,  
Node2D.GlobalTranslate(Vector2) , Node2D.ApplyScale(Vector2) , Node2D.SetGlobalPosition(Vector2) ,  
Node2D.GetGlobalPosition() , [Node2D.SetGlobalRotation\(float\)](#) , Node2D.GetGlobalRotation() ,  
[Node2D.SetGlobalRotationDegrees\(float\)](#) , Node2D.GetGlobalRotationDegrees() ,  
Node2D.SetGlobalScale(Vector2) , Node2D.GetGlobalScale() , Node2D.SetTransform(Transform2D) ,  
Node2D.SetGlobalTransform(Transform2D) , Node2D.LookAt(Vector2) , Node2D.GetAngleTo(Vector2) ,  
Node2DToLocal(Vector2) , Node2D.ToGlobal(Vector2) , [Node2D.SetZIndex\(int\)](#) , Node2D.GetZIndex() ,  
[Node2D.SetZAsRelative\(bool\)](#) , Node2D.IsZRelative() , Node2D.GetRelativeTransformToParent(Node) ,  
Node2D.Position , Node2D.Rotation , Node2D.RotationDegrees , Node2D.Scale , Node2D.Transform ,  
Node2D.GlobalPosition , Node2D.GlobalRotation , Node2D.GlobalRotationDegrees ,  
Node2D.GlobalScale , Node2D.GlobalTransform , Node2D.ZIndex , Node2D.ZAsRelative ,  
CanvasItem.NotificationTransformChanged , CanvasItem.NotificationLocalTransformChanged ,  
CanvasItem.NotificationDraw , CanvasItem.NotificationVisibilityChanged ,  
CanvasItem.NotificationEnterCanvas , CanvasItem.NotificationExitCanvas , CanvasItem.\_Draw() ,  
CanvasItem.GetCanvasItem() , [CanvasItem.SetVisible\(bool\)](#) , CanvasItem.IsVisible() ,  
CanvasItem.isVisibleInTree() , CanvasItem.Show() , CanvasItem.Hide() , CanvasItem.Update() ,  
[CanvasItem.SetAsToplevel\(bool\)](#) , CanvasItem.IsSetAsToplevel() , [CanvasItem.SetLightMask\(int\)](#) ,  
CanvasItem.GetLightMask() , CanvasItem.SetModulate(Color) , CanvasItem.GetModulate() ,  
CanvasItem.SetSelfModulate(Color) , CanvasItem.GetSelfModulate() ,  
[CanvasItem.SetDrawBehindParent\(bool\)](#) , CanvasItem.IsDrawBehindParentEnabled() ,

[CanvasItem.DrawLine\(Vector2, Vector2, Color, float, bool\)](#) ,  
[CanvasItem.DrawPolyline\(Vector2\[\], Color, float, bool\)](#) ,  
[CanvasItem.DrawPolylineColors\(Vector2\[\], Color\[\], float, bool\)](#) ,  
[CanvasItem.DrawArc\(Vector2, float, float, float, int, Color, float, bool\)](#) ,  
[CanvasItem.DrawMultiline\(Vector2\[\], Color, float, bool\)](#) ,  
[CanvasItem.DrawMultilineColors\(Vector2\[\], Color\[\], float, bool\)](#) ,  
[CanvasItem.DrawRect\(Rect2, Color, bool, float, bool\)](#) , [CanvasItem.DrawCircle\(Vector2, float, Color\)](#) ,  
CanvasItem.DrawTexture(Texture, Vector2, Color?, Texture) ,  
[CanvasItem.DrawTextureRect\(Texture, Rect2, bool, Color?, bool, Texture\)](#) ,  
[CanvasItem.DrawTextureRectRegion\(Texture, Rect2, Rect2, Color?, bool, Texture, bool\)](#) ,  
CanvasItem.DrawStyleBox(StyleBox, Rect2) ,  
[CanvasItem.DrawPrimitive\(Vector2\[\], Color\[\], Vector2\[\], Texture, float, Texture\)](#) ,  
[CanvasItem.DrawPolygon\(Vector2\[\], Color\[\], Vector2\[\], Texture, Texture, bool\)](#) ,  
[CanvasItem.DrawColoredPolygon\(Vector2\[\], Color, Vector2\[\], Texture, Texture, bool\)](#) ,  
[CanvasItem.DrawString\(Font, Vector2, string, Color?, int\)](#) ,  
[CanvasItem.DrawChar\(Font, Vector2, string, string, Color?\)](#) ,  
CanvasItem.DrawMesh(Mesh, Texture, Texture, Transform2D?, Color?) ,  
CanvasItem.DrawMultimesh(MultiMesh, Texture, Texture) ,  
[CanvasItem.DrawSetTransform\(Vector2, float, Vector2\)](#) ,  
CanvasItem.DrawSetTransformMatrix(Transform2D) , CanvasItem.GetTransform() ,  
CanvasItem.GetGlobalTransform() , CanvasItem.GetGlobalTransformWithCanvas() ,  
CanvasItem.GetViewportTransform() , CanvasItem.GetViewportRect() , CanvasItem.GetCanvasTransform() ,  
CanvasItem.GetLocalMousePosition() , CanvasItem.GetGlobalMousePosition() , CanvasItem.GetCanvas() ,  
CanvasItem.GetWorld2d() , CanvasItem.SetMaterial(Material) , CanvasItem.GetMaterial() ,  
[CanvasItem.SetUseParentMaterial\(bool\)](#) , CanvasItem.GetUseParentMaterial() ,  
[CanvasItem.SetNotifyLocalTransform\(bool\)](#) , CanvasItem.IsLocalTransformNotificationEnabled() ,  
[CanvasItem.SetNotifyTransform\(bool\)](#) , CanvasItem.IsTransformNotificationEnabled() ,  
CanvasItem.ForceUpdateTransform() , CanvasItem.MakeCanvasPositionLocal(Vector2) ,  
CanvasItem.MakeInputLocal(InputEvent) , CanvasItem.Visible , CanvasItem.Modulate ,  
CanvasItem.SelfModulate , CanvasItem.ShowBehindParent , CanvasItem.ShowOnTop ,  
CanvasItem.LightMask , CanvasItem.Material , CanvasItem.UseParentMaterial ,  
Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,

Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOsImeUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , [Node.PhysicsProcess\(float\)](#) ,  
[Node.Process\(float\)](#) , Node.\_UnhandledInput(InputEvent) , Node.\_UnhandledKeyInput(InputEventKey) ,  
[Node.AddChildBelowNode\(Node, Node, bool\)](#) , [Node.SetName\(string\)](#) , Node.GetName() ,  
[Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) , Node.GetChildCount() , Node.GetChildren() ,  
[Node.GetChild\(int\)](#) , Node.HasNode(NodePath) , Node.GetNode(NodePath) ,  
Node.GetNodeOrNull(NodePath) , Node.GetParent() , [Node.FindNode\(string, bool, bool\)](#) ,  
[Node.FindParent\(string\)](#) , Node.HasNodeAndResource(NodePath) ,  
Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() , Node.IsAParentOf(Node) ,  
Node.IsGreater Than(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDelta Time() , Node.IsPhysicsProcessing() , Node.GetProcessDelta Time() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,

[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.RpcId\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostinitialize ,  
Object.NotificationPredelete , Object.IsNullOrInstanceValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,  
[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Callv\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#)

# Methods

## RayCast(Camera2D?, Vector2, Vector2, out RayHit2D)

Projects a ray that intersects the collider of a 2d object.

```
public static bool RayCast(Camera2D? camera, Vector2 from, Vector2 to, out RayHit2D hit)
```

### Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**from** Vector2

The starting point of the collision ray.

**to** Vector2

The endpoint of the collision ray.

**hit** [RayHit2D](#)

The output parameter for the object's collision information.

### Returns

[bool](#) ↗

Returns **true** when any 2d object is detected.

### Exceptions

[ArgumentNullException](#) ↗

The exception is thrown when Godot.Camera2D object is null.

## RayCast(Camera2D?, Vector2, Vector2, CollisionObject2D[]?, out RayHit2D)

Projects a ray that intersects the collider of a 2d object.

```
public static bool RayCast(Camera2D? camera, Vector2 from, Vector2 to, CollisionObject2D[]? exclude, out RayHit2D hit)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**from** Vector2

The starting point of the collision ray.

**to** Vector2

The endpoint of the collision ray.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**hit** [RayHit2D](#)

The output parameter for the object's collision information.

## Returns

[bool](#)

Returns **true** when any 2d object is detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

**RayCast(Camera2D?, Vector2, Vector2, CollisionObject2D[]?, uint, out RayHit2D)**

Projects a ray that intersects the collider of a 2d object.

```
public static bool RayCast(Camera2D? camera, Vector2 from, Vector2 to, CollisionObject2D[]? exclude, uint collisionLayer, out RayHit2D hit)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**from** Vector2

The starting point of the collision ray.

**to** Vector2

The endpoint of the collision ray.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**hit** [RayHit2D](#)

The output parameter for the object's collision information.

## Returns

[bool](#)

Returns **true** when any 2d object is detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCast(Camera2D?, Vector2, Vector2, uint, out RayHit2D)

Projects a ray that intersects the collider of a 2d object.

```
public static bool RayCast(Camera2D? camera, Vector2 from, Vector2 to, uint collisionLayer,  
out RayHit2D hit)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**from** Vector2

The starting point of the collision ray.

**to** Vector2

The endpoint of the collision ray.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**hit** [RayHit2D](#)

The output parameter for the object's collision information.

## Returns

[bool](#)

Returns **true** when any 2d object is detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

**RayCastAllBox(Camera2D?, Vector2, Vector2,  
CollisionObject2D[]?, List<Hit2D>?)**

Creates a 2D box that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllBox(Camera2D? camera, Vector2 mousePosition, Vector2 size,  
CollisionObject2D[]? exclude, List<Hit2D>? list)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**size** Vector2

The size of the 2d collision box.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**list** [List<Hit2D>](#)

Here the objects that were detected by the 2d collision ray will be added.

## Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

[ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

# RayCastAllBox(Camera2D?, Vector2, Vector2, CollisionObject2D[]?, uint, List<Hit2D>?)

Creates a 2D box that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllBox(Camera2D? camera, Vector2 mousePosition, Vector2 size, CollisionObject2D[]? exclude, uint collisionLayer, List<Hit2D>? list)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**size** Vector2

The size of the 2d collision box.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**list** [List](#)<[Hit2D](#)>

Here the objects that were detected by the 2d collision ray will be added.

## Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## [ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

# RayCastAllBox(Camera2D?, Vector2, Vector2, List<Hit2D>?)

Creates a 2D box that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllBox(Camera2D? camera, Vector2 mousePosition, Vector2 size,  
List<Hit2D>? list)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**size** Vector2

The size of the 2d collision box.

**list** [List](#)<[Hit2D](#)>

Here the objects that were detected by the 2d collision ray will be added.

## Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## [ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

# RayCastAllBox(Camera2D?, Vector2, Vector2, uint, List<Hit2D>?)

Creates a 2D box that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllBox(Camera2D? camera, Vector2 mousePosition, Vector2 size, uint collisionLayer, List<Hit2D>? list)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**size** Vector2

The size of the 2d collision box.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**list** [List](#)<[Hit2D](#)>

Here the objects that were detected by the 2d collision ray will be added.

## Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## [ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

# RayCastAllCircle(Camera2D?, Vector2, float, CollisionObject2D[]?, List<Hit2D>?)

Creates a 2D circle that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllCircle(Camera2D? camera, Vector2 mousePosition, float radius,  
CollisionObject2D[]? exclude, List<Hit2D>? list)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**radius** [float](#)

Defines the radius of the 2d collision circle.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**list** [List](#)<[Hit2D](#)>

Here the objects that were detected by the 2d collision ray will be added.

## Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

## Exceptions

## [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## [ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

# RayCastAllCircle(Camera2D?, Vector2, float, CollisionObject2D[]?, uint, List<Hit2D>?)

Creates a 2D circle that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllCircle(Camera2D? camera, Vector2 mousePosition, float radius, CollisionObject2D[ ]? exclude, uint collisionLayer, List<Hit2D>? list)
```

## Parameters

### **camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

### **mousePosition** Vector2

The point from which the ray will be thrown.

### **radius** [float](#)

Defines the radius of the 2d collision circle.

### **exclude** CollisionObject2D[]

The objects that will be excluded from the search.

### **collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

### **list** [List](#)<[Hit2D](#)>

Here the objects that were detected by the 2d collision ray will be added.

## Returns

## [bool](#)

Returns `true` when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when `Godot.Camera2D` object is null.

### [ArgumentNullException](#)

The exception is thrown when `List<T>` object is null.

## RayCastAllCircle(Camera2D?, Vector2, float, List<Hit2D>?)

Creates a 2D circle that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllCircle(Camera2D? camera, Vector2 mousePosition, float radius,  
List<Hit2D>? list)
```

## Parameters

### `camera Camera2D`

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition Vector2`

The point from which the ray will be thrown.

### `radius float`

Defines the radius of the 2d collision circle.

### `list List<Hit2D>`

Here the objects that were detected by the 2d collision ray will be added.

## Returns

## [bool](#)

Returns `true` when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when `Godot.Camera2D` object is null.

### [ArgumentNullException](#)

The exception is thrown when `List<T>` object is null.

## RayCastAllCircle(Camera2D?, Vector2, float, uint, List<Hit2D>?)

Creates a 2D circle that allows you to detect multiple objects in 2D space simultaneously.

```
public static bool RayCastAllCircle(Camera2D? camera, Vector2 mousePosition, float radius,  
uint collisionLayer, List<Hit2D>? list)
```

## Parameters

### `camera Camera2D`

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition Vector2`

The point from which the ray will be thrown.

### `radius float`

Defines the radius of the 2d collision circle.

### `collisionLayer uint`

The collision layers that will be checked by the 2D physical body detection methods.

### `list List<Hit2D>`

Here the objects that were detected by the 2d collision ray will be added.

## Returns

## [bool](#)

Returns `true` when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when `Godot.Camera2D` object is null.

### [ArgumentNullException](#)

The exception is thrown when `List<T>` object is null.

## RayCastBox(Camera2D?, Vector2, Vector2, out Hit2D)

Creates a 2D box that allows you to detect an object in 2D space.

```
public static bool RayCastBox(Camera2D? camera, Vector2 mousePosition, Vector2 size, out  
Hit2D hit)
```

## Parameters

### `camera` Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition` Vector2

The point from which the ray will be thrown.

### `size` Vector2

The size of the 2d collision box.

### `hit` [Hit2D](#)

The output parameter for the object's collision information.

## Returns

## [bool](#)

Returns `true` when any 2d object is detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastBox(Camera2D?, Vector2, Vector2, CollisionObject2D[]?, out Hit2D)

Creates a 2D box that allows you to detect an object in 2D space.

```
public static bool RayCastBox(Camera2D? camera, Vector2 mousePosition, Vector2 size,  
CollisionObject2D[]? exclude, out Hit2D hit)
```

## Parameters

### `camera` Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition` Vector2

The point from which the ray will be thrown.

### `size` Vector2

The size of the 2d collision box.

### `exclude` CollisionObject2D[]

The objects that will be excluded from the search.

### `hit` [Hit2D](#)

The output parameter for the object's collision information.

## Returns

### [bool](#)

Returns `true` when any 2d object is detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastBox(Camera2D?, Vector2, Vector2, CollisionObject2D[]?, uint, out Hit2D)

Creates a 2D box that allows you to detect an object in 2D space.

```
public static bool RayCastBox(Camera2D? camera, Vector2 mousePosition, Vector2 size,
CollisionObject2D[]? exclude, uint collisionLayer, out Hit2D hit)
```

## Parameters

### **camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

### **mousePosition** Vector2

The point from which the ray will be thrown.

### **size** Vector2

The size of the 2d collision box.

### **exclude** CollisionObject2D[]

The objects that will be excluded from the search.

### **collisionLayer** uint

The collision layers that will be checked by the 2D physical body detection methods.

### **hit** Hit2D

The output parameter for the object's collision information.

## Returns

### [bool](#)

Returns `true` when any 2d object is detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastBox(Camera2D?, Vector2, Vector2, uint, out Hit2D)

Creates a 2D box that allows you to detect an object in 2D space.

```
public static bool RayCastBox(Camera2D? camera, Vector2 mousePosition, Vector2 size, uint  
collisionLayer, out Hit2D hit)
```

## Parameters

### `camera` Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition` Vector2

The point from which the ray will be thrown.

### `size` Vector2

The size of the 2d collision box.

### `collisionLayer` [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

### `hit` [Hit2D](#)

The output parameter for the object's collision information.

## Returns

### [bool](#)

Returns `true` when any 2d object is detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastCircle(Camera2D?, Vector2, float, out Hit2D)

Creates a 2D circle that allows you to detect an object in 2D space.

```
public static bool RayCastCircle(Camera2D? camera, Vector2 mousePosition, float radius, out Hit2D hit)
```

### Parameters

#### **camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

#### **mousePosition** Vector2

The point from which the ray will be thrown.

#### **radius** [float](#)

Defines the radius of the 2d collision circle.

#### **hit** [Hit2D](#)

The output parameter for the object's collision information.

### Returns

#### [bool](#)

Returns **true** when any 2d object is detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

# RayCastCircle(Camera2D?, Vector2, float, CollisionObject2D[]?, out Hit2D)

Creates a 2D circle that allows you to detect an object in 2D space.

```
public static bool RayCastCircle(Camera2D? camera, Vector2 mousePosition, float radius,  
CollisionObject2D[]? exclude, out Hit2D hit)
```

## Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**radius** [float](#)

Defines the radius of the 2d collision circle.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

## Returns

[bool](#)

Returns **true** when any 2d object is detected.

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastCircle(Camera2D?, Vector2, float, CollisionObject2D[]?, uint, out Hit2D)

Creates a 2D circle that allows you to detect an object in 2D space.

```
public static bool RayCastCircle(Camera2D? camera, Vector2 mousePosition, float radius,  
CollisionObject2D[]? exclude, uint collisionLayer, out Hit2D hit)
```

### Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**radius** [float](#)

Defines the radius of the 2d collision circle.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

### Returns

[bool](#)

Returns **true** when any 2d object is detected.

### Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastCircle(Camera2D?, Vector2, float, uint, out Hit2D)

Creates a 2D circle that allows you to detect an object in 2D space.

```
public static bool RayCastCircle(Camera2D? camera, Vector2 mousePosition, float radius, uint collisionLayer, out Hit2D hit)
```

### Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**radius** [float](#)

Defines the radius of the 2d collision circle.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

### Returns

[bool](#)

Returns **true** when any 2d object is detected.

### Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastHit(Camera2D?, Vector2, out Hit2D)

Creates a 2D ray that allows you to detect an object in 2D space.

```
public static bool RayCastHit(Camera2D? camera, Vector2 mousePosition, out Hit2D hit)
```

### Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

### Returns

[bool](#)

Returns **true** when any 2d object is detected.

### Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastHit(Camera2D?, Vector2, CollisionObject2D[]?, out Hit2D)

Creates a 2D ray that allows you to detect an object in 2D space.

```
public static bool RayCastHit(Camera2D? camera, Vector2 mousePosition, CollisionObject2D[]? exclude, out Hit2D hit)
```

### Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

Returns

[bool](#)

Returns **true** when any 2d object is detected.

Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastHit(Camera2D?, Vector2, CollisionObject2D[]?, uint, out Hit2D)

Creates a 2D ray that allows you to detect an object in 2D space.

```
public static bool RayCastHit(Camera2D? camera, Vector2 mousePosition, CollisionObject2D[]?
exclude, uint collisionLayer, out Hit2D hit)
```

Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**exclude** CollisionObject2D[]

The objects that will be excluded from the search.

**collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**hit** [Hit2D](#)

The output parameter for the object's collision information.

Returns

[bool](#)

Returns [true](#) when any 2d object is detected.

Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## RayCastHit(Camera2D?, Vector2, uint, out Hit2D)

Creates a 2D ray that allows you to detect an object in 2D space.

```
public static bool RayCastHit(Camera2D? camera, Vector2 mousePosition, uint collisionLayer,  
    out Hit2D hit)
```

Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

## **collisionLayer** [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

## **hit** [Hit2D](#)

The output parameter for the object's collision information.

Returns

## [bool](#)

Returns [true](#) when any 2d object is detected.

Exceptions

## [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

# RayCastHitAll(Camera2D?, Vector2, CollisionObject2D[]?, List<Hit2D>?)

Creates a 2D ray that allows detecting multiple objects in 2D space simultaneously.

```
public static bool RayCastHitAll(Camera2D? camera, Vector2 mousePosition,  
CollisionObject2D[]? exclude, List<Hit2D>? list)
```

Parameters

## **camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

## **mousePosition** Vector2

The point from which the ray will be thrown.

## **exclude** CollisionObject2D[]

The objects that will be excluded from the search.

## **list** [List](#)<Hit2D>

Here the objects that were detected by the 2d collision ray will be added.

Returns

[bool](#)

Returns `true` when multiple 2D objects are detected.

Exceptions

[ArgumentNullException](#)

The exception is thrown when `Godot.Camera2D` object is null.

[ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

## RayCastHitAll(Camera2D?, Vector2, CollisionObject2D[]?, uint, List<Hit2D>?)

Creates a 2D ray that allows detecting multiple objects in 2D space simultaneously.

```
public static bool RayCastHitAll(Camera2D? camera, Vector2 mousePosition,  
CollisionObject2D[]? exclude, uint collisionLayer, List<Hit2D>? list)
```

Parameters

`camera` Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

`mousePosition` Vector2

The point from which the ray will be thrown.

`exclude` CollisionObject2D[]

The objects that will be excluded from the search.

`collisionLayer` [uint](#)

The collision layers that will be checked by the 2D physical body detection methods.

**list** [List](#)<Hit2D>

Here the objects that were detected by the 2d collision ray will be added.

Returns

[bool](#)

Returns **true** when multiple 2D objects are detected.

Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

[ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

## RayCastHitAll(Camera2D?, Vector2, List<Hit2D>?)

Creates a 2D ray that allows detecting multiple objects in 2D space simultaneously.

```
public static bool RayCastHitAll(Camera2D? camera, Vector2 mousePosition, List<Hit2D>? list)
```

Parameters

**camera** Camera2D

Camera that will be used to convert a point on the screen to a point in the 2d world.

**mousePosition** Vector2

The point from which the ray will be thrown.

**list** [List](#)<Hit2D>

Here the objects that were detected by the 2d collision ray will be added.

Returns

[bool](#)

Returns `true` when multiple 2D objects are detected.

## Exceptions

### [ArgumentNullException](#)

The exception is thrown when `Godot.Camera2D` object is null.

### [ArgumentNullException](#)

The exception is thrown when `List<T>` object is null.

## RayCastHitAll(Camera2D?, Vector2, uint, List<Hit2D>?)

Creates a 2D ray that allows detecting multiple objects in 2D space simultaneously.

```
public static bool RayCastHitAll(Camera2D? camera, Vector2 mousePosition, uint  
collisionLayer, List<Hit2D>? list)
```

## Parameters

### `camera Camera2D`

Camera that will be used to convert a point on the screen to a point in the 2d world.

### `mousePosition Vector2`

The point from which the ray will be thrown.

### `collisionLayer uint`

The collision layers that will be checked by the 2D physical body detection methods.

### `list List<Hit2D>`

Here the objects that were detected by the 2d collision ray will be added.

## Returns

### [bool](#)

Returns `true` when multiple 2D objects are detected.

# Exceptions

## [ArgumentNullException](#)

The exception is thrown when Godot.Camera2D object is null.

## [ArgumentNullException](#)

The exception is thrown when [List<T>](#) object is null.

## \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their Godot.Node.\_Ready() callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the Godot.Node.NotificationReady notification in [Notification\(int\)](#). See also the **onready** keyword for variables.

Usually used for initialization. For even earlier initialization, `ready` may be used. See also Godot.Node.\_EnterTree().

Note: Godot.Node.\_Ready() may be called only once for each node. After removing a node from the scene tree and adding it again, **\_ready** will not be called a second time. This can be bypassed by requesting another call with Godot.Node.RequestReady(), which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Struct RayHit2D

Namespace: [Cobilas.GodotEngine.Utility.Physics](#)

Assembly: com.cobilas.godot.utility.dll

Responsible for storing information about a 2D collision ray.

```
public struct RayHit2D
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Fields

### Normal\_Key

Represents normal key.

```
public const string Normal_Key = "normal"
```

Field Value

[string](#)

### Position\_Key

Represents position key.

```
public const string Position_Key = "position"
```

Field Value

[string](#)

# Properties

## Collision

The colliding object.

```
public readonly Node Collision { get; }
```

## Property Value

Node

Returns the collider of the object.

## ID

The colliding object's ID.

```
public readonly int ID { get; }
```

## Property Value

[int](#)

Returns the object ID.

## MetaData

The intersecting shape's metadata. This metadata is different from [GetMeta\(string, object\)](#), and is set with [ShapeSetData\(RID, object\)](#).

```
public readonly object MetaData { get; }
```

## Property Value

[object](#)

Returns the object's metadata.

## Name

The name of the object.

```
public readonly string Name { get; }
```

### Property Value

[string](#)

Returns the name of the object.

## Normal

The object's surface normal at the intersection point.

```
public readonly Vector2 Normal { get; }
```

### Property Value

Vector2

Returns the surface normal of the object at the intersection point.

## Position

The intersection point.

```
public readonly Vector2 Position { get; }
```

### Property Value

Vector2

Returns the intersection point.

## RID

The intersecting object's Godot.RID.

```
public readonly RID RID { get; }
```

Property Value

RID

Returns the RID of the object.

## Methods

### IsValid(Dictionary?)

Checks if the Godot.Collections.Dictionary type value is valid for conversion.

```
public static bool IsValid(Dictionary? dictionary)
```

Parameters

**dictionary** Dictionary

The object to be checked.

Returns

bool ↗

Returns **true** when the Godot.Collections.Dictionary type value is valid.

## Operators

### explicit operator RayHit2D(Dictionary?)

Explicit conversion operator.(Godot.Collections.Dictionary to [RayHit2D](#))

```
public static explicit operator RayHit2D(Dictionary? D)
```

## Parameters

### D Dictionary

Object to be converted.

## Returns

[RayHit2D](#)

## Remarks

The conversion from Godot.Collections.Dictionary to [RayHit2D](#) will be valid when Godot.Collections.Dictionary contains the keys "collider\_id", "rid", "metadata", "normal", "position" and "collider".

## Exceptions

[ArgumentNullException](#)

The exception is thrown when Godot.Collections.Dictionary object is null.

[InvalidCastException](#)

The exception is called when the Dictionary object is not valid for conversion because it does not contain the keys "collider\_id", "rid", "metadata", "normal", "position" and "collider".

# Namespace Cobilas.GodotEngine.Utility.Runtime

## Classes

### [RunTimeInitialization](#)

Responsible for initializing other classes marked with the [RunTimeInitializationClassAttribute](#) attribute.

### [RunTimeInitializationClassAttribute](#)

This attribute marks which classes will be called by [RunTimeInitialization](#).

## Structs

### [PriorityList](#)

Represents a list of [RunTimeInitialization](#) priorities.

### [RunTime](#)

Provides RunTime values and functions.

## Enums

### [ExecutionMode](#)

Represents the editor mode states.

### [Priority](#)

Indicates the boot priority.

# Enum ExecutionMode

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

Represents the editor mode states.

```
public enum ExecutionMode : byte
```

## Fields

**EditorMode = 1**

When the editor is not running the project.

**PlayerMode = 0**

When the editor is running the project.

# Enum Priority

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

Indicates the boot priority.

```
public enum Priority : byte
```

## Fields

**StartBefore** = 0

Starts before everyone else.

**StartLater** = 1

Starts after everyone else.

# Struct PriorityList

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

Represents a list of [RunTimeInitialization](#) priorities.

```
public struct PriorityList : IDisposable
```

Implements

[IDisposable](#)

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Methods

### Add(int, Node)

Adds items to the priority list.

```
public PriorityList Add(int priority, Node node)
```

Parameters

**priority** [int](#)

Object execution priority.

**node** [Node](#)

The object to be added to the list.

Returns

[PriorityList](#)

The method will return a [PriorityList](#) object with its modified priority list.

## Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

## ReorderList()

Sort the priority list according to the priority of the list items.

```
public void ReorderList()
```

## Run(Node)

Execute your priority list.

```
public readonly void Run(Node root)
```

### Parameters

**root** Node

The parent node where nodes will be added to start their priority execution.

# Struct RunTime

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

Provides RunTime values and functions.

```
public readonly struct RunTime
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

## Fields

### DeltaTime

The interval in seconds from the last frame to the current one (Read Only).

```
public const float DeltaTime = 0.33333334
```

Field Value

[float](#)

### FixedDeltaTime

The interval in seconds of in-game time at which physics and other fixed frame rate updates are performed.

```
public const float FixedDeltaTime = 0.02
```

Field Value

[float](#)

# Properties

## ExecutionMode

Allows you to check whether the editor is in [EditorMode](#) or [PlayerMode](#).

```
public static ExecutionMode ExecutionMode { get; }
```

### Property Value

#### [ExecutionMode](#)

Returns the state of the editor.

## FrameCount

The total number of frames since the start of the game (Read Only).

```
public static int FrameCount { get; }
```

### Property Value

#### [int](#)

The total number of frames

## TimeScale

Controls how fast or slow the in-game clock ticks versus the real life one. It defaults to 1.0. A value of 2.0 means the game moves twice as fast as real life, whilst a value of 0.5 means the game moves at half the regular speed. This also affects Godot.Timer and Godot.SceneTreeTimer (see [Godot.SceneTree.CreateTimer\(System.Single, System.Boolean\)](#) for how to control this).

```
public static float TimeScale { get; set; }
```

### Property Value

#### [float](#)



# Class RunTimeInitialization

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

Responsible for initializing other classes marked with the [RunTimeInitializationClassAttribute](#) attribute.

```
public class RunTimeInitialization : Node, IDisposable
```

## Inheritance

[object](#) ← Object ← Node ← RunTimeInitialization

## Implements

[IDisposable](#)

## Inherited Members

Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,  
Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOsIMEUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , [Node.PhysicsProcess\(float\)](#) ,  
[Node.Process\(float\)](#) , Node.\_UnhandledInput(InputEvent) , Node.\_UnhandledKeyInput(InputEventKey) ,  
[Node.AddChildBelowNode\(Node, Node, bool\)](#) , [Node.SetName\(string\)](#) , Node.GetName() ,  
[Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) , Node.GetChildCount() , Node.GetChildren() ,  
[Node.GetChild\(int\)](#) , Node.HasNode(NodePath) , Node.GetNode(NodePath) ,  
Node.GetNodeOrNull(NodePath) , Node.GetParent() , [Node.FindNode\(string, bool, bool\)](#) ,  
[Node.FindParent\(string\)](#) , Node.HasNodeAndResource(NodePath) ,  
Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() , Node.IsAParentOf(Node) ,

Node.IsGreater Than(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDelta Time() , Node.IsPhysicsProcessing() , Node.GetProcessDelta Time() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostInitialize ,  
Object.NotificationPreDelete , Object.IsInstanceValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,

[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Cally\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#).

## Examples

The `RunTimeInitialization` class allows you to automate the `Project>Project Settings>AutoLoad` option. To use the `RunTimeInitialization` class, you must create a class and make it inherit `RunTimeInitialization`.//

```
using Cobilas.GodotEngine.Utility.Runtime;
// The name of the class is up to you.
public class RunTimeProcess : RunTimeInitialization {}
```

And remember to add the class that inherits `RunTimeInitialization` in `Project>Project Settings>AutoLoad` . Remembering that the `RunTimeInitialization` class uses the virtual method `_Ready()` to perform the initialization of other classes. And to initialize other classes along with the

`RunTimeInitialization` class, the class must inherit the `Godot.Node` class or some class that inherits `Godot.Node` and use the `RunTimeInitializationClassAttribute` attribute.

```
using Godot;
using Cobilas.GodotEngine.Utility.Runtime;
[RunTimeInitializationClass]
public class ClassTest : Node {}
```

## Methods

### \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their `Godot.Node._Ready()` callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the `Godot.Node.NotificationReady` notification in [\\_Notification\(int\)](#). See also the `onready` keyword for variables.

Usually used for initialization. For even earlier initialization, may be used. See also `Godot.Node._EnterTree()`.

Note: `Godot.Node._Ready()` may be called only once for each node. After removing a node from the scene tree and adding it again, `_ready` will not be called a second time. This can be bypassed by requesting another call with `Godot.Node.RequestReady()`, which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Class RunTimeInitializationClassAttribute

Namespace: [Cobilas.GodotEngine.Utility.Runtime](#)

Assembly: com.cobilas.godot.utility.dll

This attribute marks which classes will be called by [RunTimeInitialization](#).

```
[AttributeUsage(AttributeTargets.Class, Inherited = false, AllowMultiple = false)]
public sealed class RunTimeInitializationClassAttribute : Attribute, _Attribute
```

## Inheritance

[object](#) ← [Attribute](#) ← RunTimeInitializationClassAttribute

## Implements

[Attribute](#)

## Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,  
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,  
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,  
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,  
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,  
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,  
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,  
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,  
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,  
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,  
[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#)

# Examples

Simple example of class demarcation to be called by [RunTimeInitialization](#).

```
using Godot;
using Cobilas.GodotEngine.Utility.Runtime;
[RunTimeInitializationClass]
public class ClassTest : Node {}
```

# Constructors

## RunTimeInitializationClassAttribute()

Instance the RunTimeInitializationClassAttribute attribute.

```
public RunTimeInitializationClassAttribute()
```

## RunTimeInitializationClassAttribute(string?, Priority, int, bool)

Instance the RunTimeInitializationClassAttribute attribute.

```
public RunTimeInitializationClassAttribute(string? name, Priority bootPriority =
Priority.StartBefore, int subPriority = 0, bool lastBoot = false)
```

### Parameters

**name** [string](#)

The name of the priority.

**bootPriority** [Priority](#)

The type of priority.

**subPriority** [int](#)

The execution priority.

**lastBoot** [bool](#)

Allows the object to be called last in the root object hierarchy.

## Properties

### BootPriority

The type of priority.

```
public Priority BootPriority { get; }
```

#### Property Value

[Priority](#)

Returns the type of priority that will be executed.

### ClassName

The name of the priority.

```
public string ClassName { get; }
```

#### Property Value

[string](#)

Returns the name of the priority that will be executed.

### LastBoot

Allows the object to be called last in the root object hierarchy.

```
public bool LastBoot { get; }
```

#### Property Value

[bool](#)

Returns `true` when marked to be called last in the root object hierarchy.

## SubPriority

The execution priority.

```
public int SubPriority { get; }
```

Property Value

[int↗](#)

Returns the priority execution level.

# Namespace Cobilas.GodotEngine.Utility.Scene

## Classes

### [SceneManager](#)

This class can be used to manage scene switching.

## Structs

### [Scene](#)

Contains the scene information.

# Struct Scene

Namespace: [Cobilas.GodotEngine.Utility.Scene](#)

Assembly: com.cobilas.godot.utility.dll

Contains the scene information.

```
public struct Scene : IEquatable<Scene>, IEquatable<int>, IEquatable<string>
```

Implements

[IEquatable](#)<[Scene](#)>, [IEquatable](#)<[int](#)>, [IEquatable](#)<[string](#)>

Inherited Members

[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetType\(\)](#)

## Constructors

### Scene(string, int, Node)

Starts a new instance of the object.

```
public Scene(string scenePath, int index, Node sceneNode)
```

Parameters

scenePath [string](#)

index [int](#)

sceneNode [Node](#)

## Properties

### Empty

Empty scenario.

```
public static Scene Empty { get; }
```

## Property Value

### [Scene](#)

Returns a representation of an empty scenario.

## Index

The scene index.

```
public readonly int Index { get; }
```

## Property Value

### [int](#)

Returns the scene index.

## Name

The name of the scene file.

```
public readonly string Name { get; }
```

## Property Value

### [string](#)

Returns a string containing the name of the scene file with its extension.

## Exceptions

### [ArgumentException](#)

path contains one or more of the invalid characters defined in [GetInvalidPathChars\(\)](#).

## NameWithoutExtension

The name of the scene file without the extension.

```
public readonly string NameWithoutExtension { get; }
```

### Property Value

[string](#)

Returns a string containing the name of the scene file without its extension.

### Exceptions

[ArgumentException](#)

path contains one or more of the invalid characters defined in [GetInvalidPathChars\(\)](#).

## SceneNode

Stores the root Godot.Node of the scenario.

```
public readonly Node SceneNode { get; }
```

### Property Value

Node

Returns a Godot.Node type object that is the root of the scenario.

## ScenePath

The path of the scene file.

```
public readonly string ScenePath { get; }
```

### Property Value

## [string](#)

Returns the full or relative path of the scene file.

# Methods

## Equals(Scene)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(Scene other)
```

Parameters

**other** [Scene](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the **other** parameter; otherwise, [false](#).

## Equals(int)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(int other)
```

Parameters

**other** [int](#)

An object to compare with this object.

Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override readonly bool Equals(object obj)
```

### Parameters

`obj` [object](#)

The object to compare with the current instance.

### Returns

[bool](#)

[true](#) if `obj` and this instance are the same type and represent the same value; otherwise, [false](#).

## Equals(string)

Indicates whether the current object is equal to another object of the same type.

```
public readonly bool Equals(string other)
```

### Parameters

`other` [string](#)

An object to compare with this object.

### Returns

[bool](#)

[true](#) if the current object is equal to the `other` parameter; otherwise, [false](#).

## GetHashCode()

Returns the hash code for this instance.

```
public override readonly int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## ToString()

Returns the fully qualified type name of this instance.

```
public override readonly string ToString()
```

Returns

[string](#)

The fully qualified type name.

## Operators

### operator ==(Scene, Scene)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Scene left, Scene right)
```

Parameters

[left](#) [Scene](#)

Object to be compared.

## right [Scene](#)

Object of comparison.

Returns

## [bool](#) ↗

Returns the result of the comparison.

## operator ==(Scene, int)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Scene left, int right)
```

Parameters

### left [Scene](#)

Object to be compared.

### right [int](#) ↗

Object of comparison.

Returns

## [bool](#) ↗

Returns the result of the comparison.

## operator ==(Scene, string)

Indicates whether this instance is equal to another instance of the same type.

```
public static bool operator ==(Scene left, string right)
```

Parameters

## **left** [Scene](#)

Object to be compared.

## **right** [string](#) ↗

Object of comparison.

Returns

## [bool](#) ↗

Returns the result of the comparison.

## **explicit operator int(Scene)**

Explicit conversion operator.([Scene](#) to [int](#) ↗)

```
public static explicit operator int(Scene scene)
```

Parameters

## **scene** [Scene](#)

Object to be converted.

Returns

## [int](#) ↗

## **explicit operator string(Scene)**

Explicit conversion operator.([Scene](#) to [string](#) ↗)

```
public static explicit operator string(Scene value)
```

Parameters

## **value** [Scene](#)

Object to be converted.

Returns

[string](#)

## operator !=(Scene, Scene)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Scene left, Scene right)
```

Parameters

**left** [Scene](#)

Object to be compared.

**right** [Scene](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator !=(Scene, int)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Scene left, int right)
```

Parameters

**left** [Scene](#)

Object to be compared.

**right** [int](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

## operator !=(Scene, string)

Indicates whether this instance is different from another instance of the same type.

```
public static bool operator !=(Scene left, string right)
```

Parameters

**left** [Scene](#)

Object to be compared.

**right** [string](#)

Object of comparison.

Returns

[bool](#)

Returns the result of the comparison.

# Class SceneManager

Namespace: [Cobilas.GodotEngine.Utility.Scene](#)

Assembly: com.cobilas.godot.utility.dll

This class can be used to manage scene switching.

```
[RunTimeInitializationClass("SceneManager", Priority.StartBefore, 0, false)]
public class SceneManager : Node, IDisposable
```

## Inheritance

[Object](#) ← Object ← Node ← SceneManager

## Implements

[IDisposable](#)

## Inherited Members

Node.NotificationEnterTree , Node.NotificationExitTree , Node.NotificationMovedInParent ,  
Node.NotificationReady , Node.NotificationPaused , Node.NotificationUnpaused ,  
Node.NotificationPhysicsProcess , Node.NotificationProcess , Node.NotificationParented ,  
Node.NotificationUnparented , Node.NotificationInstanced , Node.NotificationDragBegin ,  
Node.NotificationDragEnd , Node.NotificationPathChanged , Node.NotificationInternalProcess ,  
Node.NotificationInternalPhysicsProcess , Node.NotificationPostEnterTree ,  
Node.NotificationResetPhysicsInterpolation , Node.NotificationWmMouseEnter ,  
Node.NotificationWmMouseExit , Node.NotificationWmFocusIn , Node.NotificationWmFocusOut ,  
Node.NotificationWmQuitRequest , Node.NotificationWmGoBackRequest ,  
Node.NotificationWmUnfocusRequest , Node.NotificationOsMemoryWarning ,  
Node.NotificationTranslationChanged , Node.NotificationWmAbout , Node.NotificationCrash ,  
Node.NotificationOslmeUpdate , Node.NotificationAppResumed , Node.NotificationAppPaused ,  
Node.GetNode<T>(NodePath) , Node.GetNodeOrNull<T>(NodePath) , [Node.GetChild<T>\(int\)](#) ,  
[Node.GetChildOrNull<T>\(int\)](#) , Node.GetOwner<T>() , Node.GetOwnerOrNull<T>() ,  
Node.GetParent<T>() , Node.GetParentOrNull<T>() , Node.\_EnterTree() , Node.\_ExitTree() ,  
Node.\_GetConfigurationWarning() , Node.\_Input(InputEvent) , [Node. PhysicsProcess\(float\)](#) ,  
[Node. Process\(float\)](#) , Node.\_UnhandledInput(InputEvent) , Node.\_UnhandledKeyInput(InputEventKey) ,  
[Node.AddChildBelowNode\(Node, Node, bool\)](#) , [Node.SetName\(string\)](#) , Node.GetName() ,  
[Node.AddChild\(Node, bool\)](#) , Node.RemoveChild(Node) , Node.GetChildCount() , Node.GetChildren() ,  
[Node.GetChild\(int\)](#) , Node.HasNode(NodePath) , Node.GetNode(NodePath) ,  
Node.GetNodeOrNull(NodePath) , Node.GetParent() , [Node.FindNode\(string, bool, bool\)](#) ,  
[Node.FindParent\(string\)](#) , Node.HasNodeAndResource(NodePath) ,  
Node.GetNodeAndResource(NodePath) , Node.IsInsideTree() , Node.IsAParentOf(Node) ,

Node.IsGreater Than(Node) , Node.GetPath() , Node.GetPathTo(Node) ,  
[Node.AddToGroup\(string, bool\)](#) , [Node.RemoveFromGroup\(string\)](#) , [Node.IsInGroup\(string\)](#) ,  
[Node.MoveChild\(Node, int\)](#) , Node.GetGroups() , Node.Raise() , Node.SetOwner(Node) ,  
Node.GetOwner() , Node.RemoveAndSkip() , Node.GetIndex() , Node.PrintTree() , Node.PrintTreePretty() ,  
[Node.SetFilename\(string\)](#) , Node.GetFilename() , [Node.PropagateNotification\(int\)](#) ,  
[Node.PropagateCall\(string, Array, bool\)](#) , [Node.SetPhysicsProcess\(bool\)](#) ,  
Node.GetPhysicsProcessDelta Time() , Node.IsPhysicsProcessing() , Node.GetProcessDelta Time() ,  
[Node.SetProcess\(bool\)](#) , [Node.SetProcessPriority\(int\)](#) , Node.GetProcessPriority() ,  
Node.IsProcessing() , [Node.SetProcessInput\(bool\)](#) , Node.IsProcessingInput() ,  
[Node.SetProcessUnhandledInput\(bool\)](#) , Node.IsProcessingUnhandledInput() ,  
[Node.SetProcessUnhandledKeyInput\(bool\)](#) , Node.IsProcessingUnhandledKeyInput() ,  
Node.SetPauseMode(Node.PauseModeEnum) , Node.GetPauseMode() , Node.CanProcess() ,  
Node.PrintStrayNodes() , NodeGetPositionInParent() , [Node.SetDisplayFolded\(bool\)](#) ,  
Node.IsDisplayedFolded() , [Node.SetProcessInternal\(bool\)](#) , Node.IsProcessingInternal() ,  
[Node.SetPhysicsProcessInternal\(bool\)](#) , Node.IsPhysicsProcessingInternal() ,  
Node.SetPhysicsInterpolationMode(Node.PhysicsInterpolationModeEnum) ,  
Node.GetPhysicsInterpolationMode() , Node.IsPhysicsInterpolated() ,  
Node.IsPhysicsInterpolatedAndEnabled() , Node.ResetPhysicsInterpolation() , Node.GetTree() ,  
Node.CreateTween() , [Node.Duplicate\(int\)](#) , [Node.ReplaceBy\(Node, bool\)](#) ,  
[Node.SetSceneInstanceLoadPlaceholder\(bool\)](#) , Node.GetSceneInstanceLoadPlaceholder() ,  
Node.GetViewport() , Node.QueueFree() , Node.RequestReady() , [Node.SetNetworkMaster\(int, bool\)](#) ,  
Node.GetNetworkMaster() , Node.IsNetworkMaster() , Node.GetMultiplayer() ,  
Node.GetCustomMultiplayer() , Node.SetCustomMultiplayer(MultiplayerAPI) ,  
[Node.RpcConfig\(string, MultiplayerAPI.RPCMode\)](#) ,  
[Node.RsetConfig\(string, MultiplayerAPI.RPCMode\)](#) , [Node.SetUniqueNameInOwner\(bool\)](#) ,  
Node.IsUniqueNameInOwner() , [Node.Rpc\(string, params object\[\]\)](#) ,  
[Node.RpcUnreliable\(string, params object\[\]\)](#) , [Node.Rpcld\(int, string, params object\[\]\)](#) ,  
[Node.RpcUnreliableId\(int, string, params object\[\]\)](#) , [Node.Rset\(string, object\)](#) ,  
[Node.RsetId\(int, string, object\)](#) , [Node.RsetUnreliable\(string, object\)](#) ,  
[Node.RsetUnreliableId\(int, string, object\)](#) , Node.UpdateConfigurationWarning() ,  
Node.EditorDescription , Node.\_ImportPath , Node.PauseMode , Node.PhysicsInterpolationMode ,  
Node.Name , Node.UniqueNameInOwner , Node.Filename , Node.Owner , Node.Multiplayer ,  
Node.CustomMultiplayer , Node.ProcessPriority , Object.NotificationPostInitialize ,  
Object.NotificationPreDelete , Object.IsInstanceValid(Object) , Object.WeakRef(Object) , Object.Dispose() ,  
[Object.Dispose\(bool\)](#) , Object.ToString() , [Object.ToSignal\(Object, string\)](#) , [Object.Get\(string\)](#) ,  
Object.\_GetPropertyList() , [Object.Notification\(int\)](#) , [Object.Set\(string, object\)](#) , Object.Free() ,  
Object.GetClass() , [Object.IsClass\(string\)](#) , [Object.Set\(string, object\)](#) , [Object.Get\(string\)](#) ,  
[Object.SetIndexed\(NodePath, object\)](#) , Object.GetIndexed(NodePath) , Object.GetPropertyList() ,  
Object.GetMethodList() , [Object.Notification\(int, bool\)](#) , Object.GetInstanceId() ,  
Object.SetScript(Reference) , Object.GetScript() , [Object.SetMeta\(string, object\)](#) ,

[Object.RemoveMeta\(string\)](#) , [Object.GetMeta\(string, object\)](#) , [Object.HasMeta\(string\)](#) ,  
Object.GetMetaList() , [Object.AddUserSignal\(string, Array\)](#) , [Object.HasUserSignal\(string\)](#) ,  
[Object.EmitSignal\(string, params object\[\]\)](#) , [Object.Call\(string, params object\[\]\)](#) ,  
[Object.CallDeferred\(string, params object\[\]\)](#) , [Object.SetDeferred\(string, object\)](#) ,  
[Object.Callv\(string, Array\)](#) , [Object.HasMethod\(string\)](#) , [Object.HasSignal\(string\)](#) ,  
Object.GetSignalList() , [Object.GetSignalConnectionList\(string\)](#) , Object.GetIncomingConnections() ,  
[Object.Connect\(string, Object, string, Array, uint\)](#) , [Object.Disconnect\(string, Object, string\)](#) ,  
[Object.IsConnected\(string, Object, string\)](#) , [Object.SetBlockSignals\(bool\)](#) , Object.IsBlockingSignals() ,  
Object.PropertyListChangedNotify() , [Object.SetMessageTranslation\(bool\)](#) ,  
Object.CanTranslateMessages() , [Object.Tr\(string\)](#) , Object.IsQueuedForDeletion() ,  
Object.NativeInstance , Object.DynamicObject , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Extension Methods

[Node GD CB Extension.FindNodeByName\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName\(Node, string, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string\)](#) ,  
[Node GD CB Extension.FindNodeByName<T>\(Node, string, bool\)](#) ,  
[Node GD CB Extension.FindNodes\(Node, Type\)](#) , [Node GD CB Extension.FindNodes\(Node, Type, bool\)](#) ,  
[Node GD CB Extension.FindNodes<T>\(Node\)](#) , [Node GD CB Extension.FindNodes<T>\(Node, bool\)](#) ,  
[Node GD CB Extension.GetNodePosition\(Node\)](#) , [Node GD CB Extension.GetNodeRotation\(Node\)](#) ,  
[Node GD CB Extension.GetNodeScale\(Node\)](#) , [Node GD CB Extension.Print\(Node, params object\[\]\)](#) ,  
[Node GD CB Extension.SetNodePosition\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeRotation\(Node, Vector3D\)](#) ,  
[Node GD CB Extension.SetNodeScale\(Node, Vector3D\)](#).

## Fields

### LoadedScene

This event is called when a new scene is loaded.

```
public static Action<Scene>? LoadedScene
```

### Field Value

[Action<Scene>](#)

## UnloadedScene

This event is called when the current scene is unloaded.

```
public static Action<Scene>? UnloadedScene
```

### Field Value

[Action](#)<[Scene](#)>

## Properties

### BuiltScenes

The compiled scenes.

```
public static Scene[] BuiltScenes { get; }
```

### Property Value

[Scene](#)[]

Returns a list of scenes that were compiled along with the project.

### CurrentScene

The current scene.

```
public static Scene CurrentScene { get; }
```

### Property Value

[Scene](#)

Returns the scene that is currently loaded.

# CurrentSceneNode

The current scene in node form.

```
public static Node? CurrentSceneNode { get; }
```

## Property Value

Node

Returns the currently loaded scene in node form.

## Methods

### DontDestroyOnLoad(Node)

Prevents an object from being destroyed when switching scenes.

```
public static void DontDestroyOnLoad(Node obj)
```

#### Parameters

**obj** Node

The object that will be marked so as not to be destroyed when changing scenes.

### LoadScene(int)

Allows you to load a specific scene.

```
public static bool LoadScene(int index)
```

#### Parameters

**index** [int](#)

The specific scene index.

Returns

[bool](#)

Returns `true` if the scene loaded correctly.

## LoadScene(string)

Allows you to load a specific scene.

```
public static bool LoadScene(string name)
```

Parameters

`name` [string](#)

The specific scene name.

Returns

[bool](#)

Returns `true` if the scene loaded correctly.

## \_Ready()

Called when the node is "ready", i.e. when both the node and its children have entered the scene tree. If the node has children, their Godot.Node.\_Ready() callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the Godot.Node.NotificationReady notification in [Notification\(int\)](#). See also the `onready` keyword for variables.

Usually used for initialization. For even earlier initialization, `Godot.Node._EnterTree()`.

Note: Godot.Node.\_Ready() may be called only once for each node. After removing a node from the scene tree and adding it again, `_ready` will not be called a second time. This can be bypassed by requesting another call with `Godot.Node.RequestReady()`, which may be called anywhere before adding the node again.

```
public override void _Ready()
```

# Namespace Godot

## Classes

### [Camera2D\\_GD\\_CB\\_Extension](#)

Extension for Godot.Camera2D class.

### [NodePath\\_GD\\_CB\\_Extension](#)

Extensions for Godot.NodePath class.

### [Node\\_GD\\_CB\\_Extension](#)

Extension for Godot.Node class.

### [Rect2\\_GD\\_CB\\_Extension](#)

Extension to the Godot.Rect2 struct.

# Class Camera2D\_GD\_CB\_Extension

Namespace: [Godot](#)

Assembly: com.cobilas.godot.utility.dll

Extension for Godot.Camera2D class.

```
public static class Camera2D_GD_CB_Extension
```

## Inheritance

[object](#) ← Camera2D\_GD\_CB\_Extension

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Methods

### ScreenToWorldPoint(Camera2D, Vector2D)

Converts a position on the screen to a position in the 2D world.

```
public static Vector2D ScreenToWorldPoint(this Camera2D C, Vector2D mousePosition)
```

#### Parameters

**C** Camera2D

The 2D camera that will be used for conversion.

**mousePosition** [Vector2D](#)

The position on the screen that will be converted.

#### Returns

[Vector2D](#)

Return a two-dimensional vector with the result of converting the screen position to a position in the 2D world.

## WorldToScreenPoint(Camera2D, Vector2D)

Converts a position in the 2D world to a position on the screen.

```
public static Vector2D WorldToScreenPoint(this Camera2D C, Vector2D position)
```

### Parameters

**C** Camera2D

The 2D camera that will be used for conversion.

**position** [Vector2D](#)

The position of the 2D world that will be converted.

### Returns

[Vector2D](#)

Returns a two-dimensional vector with the result of converting the 2D world position to a screen position.

# Class NodePath\_GD\_CB\_Extension

Namespace: [Godot](#)

Assembly: com.cobilas.godot.utility.dll

Extensions for Godot.NodePath class.

```
public static class NodePath_GD_CB_Extension
```

## Inheritance

[object](#) ← NodePath\_GD\_CB\_Extension

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Methods

### GetNode(NodePath?)

Fetches a node. The NodePath can be either a relative path (from the current node) or an absolute path (in the scene tree) to a node.

```
public static Node GetNode(this NodePath? np)
```

#### Parameters

**np** NodePath

Godot.NodePath from where the node will be obtained.

#### Returns

Node

returns the node that was obtained from the Godot.NodePath.

## GetNode<T>(NodePath?)

Fetches a node. The NodePath can be either a relative path (from the current node) or an absolute path (in the scene tree) to a node.

```
public static T GetNode<T>(this NodePath? np) where T : Node
```

### Parameters

**np** NodePath

Godot.NodePath from where the node will be obtained.

### Returns

T

returns the node that was obtained from the Godot.NodePath.

### Type Parameters

T

## Hash(NodePath?)

Hash the Godot.NodePath and return a 32 bits unsigned integer.

```
public static uint Hash(this NodePath? np)
```

### Parameters

**np** NodePath

Godot.NodePath to be calculated.

### Returns

[uint](#)

The calculated hash of the Godot.NodePath.

## Exceptions

### [ArgumentNullException](#)

Occurs when the np parameter is null!

## StringHash(NodePath?)

Hash the [string](#) and return a 32 bits unsigned integer.

```
public static string StringHash(this NodePath? np)
```

### Parameters

**np** NodePath

Godot.NodePath to be calculated.

### Returns

[string](#)

The calculated hash of the [string](#).

## Exceptions

### [ArgumentNullException](#)

Occurs when the np parameter is null!

# Class Node\_GD\_CB\_Extension

Namespace: [Godot](#)

Assembly: com.cobilas.godot.utility.dll

Extension for Godot.Node class.

```
public static class Node_GD_CB_Extension
```

## Inheritance

[object](#) ← Node\_GD\_CB\_Extension

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Methods

### FindNodeByName(Node, string)

Get a node from name.

By default, the method looks for a node of type node.([Type](#) typeNode = typeof(Godot.Node))

By default, the method searches recursively.([bool](#) recursive = true)

```
public static Node FindNodeByName(this Node N, string name)
```

## Parameters

### N Node

The Godot.Node that will be used.

### name [string](#)

The node name

Returns

Node

## FindNodeByName(Node, string, bool)

Get a node from name.

By default, the method looks for a node of type node.([Type](#) typeNode = typeof(Godot.Node))

```
public static Node FindNodeByName(this Node N, string name, bool recursive)
```

Parameters

**N** Node

The Godot.Node that will be used.

**name** [string](#)

The node name

**recursive** [bool](#)

Also look for your children.

Returns

Node

## FindNodeByName(Node, string, Type, bool)

Get a node from name.

```
public static Node FindNodeByName(this Node N, string name, Type typeNode, bool recursive)
```

Parameters

**N** Node

The Godot.Node that will be used.

**name** [string](#)

The node name

**typeNode** [Type](#)

The type to look for.

**recusive** [bool](#)

Also look for your children.

Returns

Node

The method returns the object based on its name, if not found the method returns [NullNode](#).

## FindNodeByName<T>(Node, string)

Get a node from name.

By default, the method searches recursively. ([bool](#) recusive = true)

```
public static T FindNodeByName<T>(this Node N, string name) where T : Node
```

Parameters

**N** Node

The Godot.Node that will be used.

**name** [string](#)

The node name

Returns

T

## Type Parameters

T

### FindNodeByName<T>(Node, string, bool)

Get a node from name.

```
public static T FindNodeByName<T>(this Node N, string name, bool recursive) where T : Node
```

## Parameters

N Node

The Godot.Node that will be used.

**name** [string](#)

The node name

**recursive** [bool](#)

Also look for your children.

## Returns

T

## Type Parameters

T

The type to look for.

### FindNodes(Node, Type)

Get the nodes from a type.

By default, the method searches recursively.([bool](#) recursive = true)

```
public static Node[] FindNodes(this Node N, Type typeNode)
```

## Parameters

### N Node

The Godot.Node that will be used.

### typeNode [Type](#)

The type to look for.

## Returns

### Node[]

Returns a list of nodes.

## FindNodes(Node, Type, bool)

Get the nodes from a type.

```
public static Node[] FindNodes(this Node N, Type typeNode, bool recursive)
```

## Parameters

### N Node

The Godot.Node that will be used.

### typeNode [Type](#)

The type to look for.

### recursive [bool](#)

Also look for your children.

## Returns

### Node[]

Returns a list of nodes.

## FindNodes<T>(Node)

Get the nodes from a type.

By default, the method searches recursively.([bool ↴](#) recursive = true)

```
public static T[] FindNodes<T>(this Node N) where T : Node
```

Parameters

**N** Node

Returns

T[]

Returns a list of nodes.

Type Parameters

**T**

The type to look for.

## FindNodes<T>(Node, bool)

Get the nodes from a type.

```
public static T[] FindNodes<T>(this Node N, bool recursive) where T : Node
```

Parameters

**N** Node

The Godot.Node that will be used.

**recursive** [bool ↴](#)

Also look for your children.

Returns

T[]

Returns a list of nodes.

Type Parameters

T

The type to look for.

## GetNodePosition(Node)

Obtem a posição atual do objeto Godot.Node.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static Vector3D GetNodePosition(this Node N)
```

Parameters

N Node

The Godot.Node that will be used.

Returns

[Vector3D](#)

Returns a [Vector3D](#) containing the current position of the Godot.Node object.

## GetNodeRotation(Node)

Gets the current rotation of the Godot.Node object.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static Vector3D GetNodeRotation(this Node N)
```

## Parameters

**N** Node

The Godot.Node that will be used.

## Returns

[Vector3D](#)

Returns a [Vector3D](#) containing the current rotation of the Godot.Node object.

## GetNodeScale(Node)

Gets the current scale of the Godot.Node object.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static Vector3D GetNodeScale(this Node N)
```

## Parameters

**N** Node

The Godot.Node that will be used.

## Returns

[Vector3D](#)

Returns a Vector3D containing the current scale of the Node object.

## Print(Node, params object[])

Converts one or more arguments of any type to string in the best way possible and prints them to the console.

Note: Consider using [PushError\(string\)](#) and [PushWarning\(string\)](#) to print error and warning messages instead of [Print\(params object\[\]\)](#). This distinguishes them from print messages used for debugging purposes, while also displaying a stack trace when an error or warning is printed.

```
public static void Print(this Node N, params object[] args)
```

## Parameters

**N** Node

The Godot.Node that will be used.

**args** [object\[\]](#)

The arguments passed will be printed to the console.

## Examples

```
var a = new int[] { 1, 2, 3 };
GD.Print("a", "b", a); // Prints ab[1, 2, 3]
```

## SetNodePosition(Node, Vector3D)

Allows you to define the position of the Godot.Node object.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static void SetNodePosition(this Node N, Vector3D position)
```

## Parameters

**N** Node

The Godot.Node that will be used.

**position** [Vector3D](#)

The new position of the Godot.Node object.

## SetNodeRotation(Node, Vector3D)

Allows you to define the rotation of the Godot.Node object.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static void SetNodeRotation(this Node N, Vector3D rotation)
```

### Parameters

**N** Node

The Godot.Node that will be used.

**rotation** [Vector3D](#)

The new rotation of the Godot.Node object.

## SetNodeScale(Node, Vector3D)

Allows you to define the scale of the Godot.Node object.

This method will only take effect when the Godot.Node object inherits the Godot.Node2D class or the Godot.Spatial class.

```
public static void SetNodeScale(this Node N, Vector3D scale)
```

### Parameters

**N** Node

The Godot.Node that will be used.

**scale** [Vector3D](#)

The new size or scale of the Godot.Node object.

# Class Rect2\_GD\_CB\_Extension

Namespace: [Godot](#)

Assembly: com.cobilas.godot.utility.dll

Extension to the Godot.Rect2 struct.

```
public static class Rect2_GD_CB_Extension
```

## Inheritance

[object](#) ← Rect2\_GD\_CB\_Extension

## Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#)

## Methods

### Bottom(Rect2)

Gets the bottom position of Godot.Rect2.

```
public static float Bottom(this Rect2 R)
```

#### Parameters

R Rect2

Godot.Rect2 that will be used.

#### Returns

[float](#)

Returns a floating-point value with the bottom position of Godot.Rect2.

## Left(Rect2)

Gets the left position of Godot.Rect2.

```
public static float Left(this Rect2 R)
```

### Parameters

R Rect2

Godot.Rect2 that will be used.

### Returns

[float](#)

Returns a floating-point left position of Godot.Rect2.

## Right(Rect2)

Gets the right position of Godot.Rect2.

```
public static float Right(this Rect2 R)
```

### Parameters

R Rect2

Godot.Rect2 that will be used.

### Returns

[float](#)

Returns a floating-point right position of Godot.Rect2.

## Top(Rect2)

Gets the top position of Godot.Rect2.

```
public static float Top(this Rect2 R)
```

## Parameters

R Rect2

Godot.Rect2 that will be used.

## Returns

[float](#)

Returns a floating-point with the top position of Godot.Rect2.