Table of Contents

Namespace RoslynCSharp

Classes

AssemblyReferenceAsset

Represents a self contains assembly reference generated from a specific assembly file, but serialized as an asset so that the reference can be used on all platforms.

AssemblyReferenceSource

AssemblySource

Represents a specific assembly location with optional debug symbols.

CompileAsync

An awaitable object that is returned by async compile operations so you can wait for completion in a coroutine as well as access progress and status information. Used to wait until an async operation has been completed

CompileAsync<T>

An awaitable object that is returned by async compile operations so you can wait for completion in a coroutine as well as access progress and status information. Used to wait for an async operation to be completed with a result object.

CompileError

Represents a compiler message, warning or error relating to a specific compilation request.

CompileFlags

Represents a set of compiler flags to change the compilation and output behaviour of a compile request.

CompileOptions

Represents a set of options provided to the compiler for a specific compile request.

CompileRequest

Represents a single compilation request which can comprise of one or many source texts, files or syntax trees.

CompileResult

Represents a result of a specific compilation request by the compiler.

ExecutionSecuritySettings

The execution settings used to ensure that compiled code cannot cause the host app to crash by blocking the main thread for a long or infinite time. Note that execution security can only be applied to assemblies that were compiled by Roslyn C#, as safety checks need to be injected after the parsing phase.

RoslynCSharpSettings

Represents the main settings object for Roslyn C#. Used to store default and user settings in the project.

ScriptAssembly

Represents a Assembly that has been loaded into a Roslyn C# ScriptDomain.

ScriptDomain

A script domain stores a collection of assemblies that have been loaded or compiled, and provides API's for loading and compiling from a variety of input sources. Note that a domain is used mainly for organisation purposes and it is possible to have multiple script domains per project if required. This does not represent an app domain although it provides access to the current

domain via AppDomain property. Support for separate app domains is not currently supported/feasable with Unity/Mono.

ScriptFields

Represents a collection of fields defined on a type or instance. Used for quickly accessing field values with caching.

ScriptMethods

Represents a collection of methods defined on a type or instance. Used for calling methods or coroutines with caching.

ScriptProperties

Represents a collection of properties defined on a type or instance. Used for quickly accessing property values with caching.

ScriptProxy

Represents an instance of a type loaded and instantiated in a script domain.

ScriptType

Represents a specific type loaded into a script domain.

Interfaces

IAssemblyLoader

Used to load an assembly into memory from a specific location.

ICompilationProcessor

Called by the compiler when compilation has completed. Allows for additional IL processing to occur after the compilation request.

ICompilationReference

Represents an object that can be referenced as a dependency by the compiler.

IMainTypeSelector

Interface for selecting the main type defined inside an assembly. Use DefaultMainTypeSelector for default type selection/

IParserProcessor

Called by the compiler when parsing has completed. Allows for additional syntax tree processing to occur during the compilation request.

Enums

AssemblyReferenceSource.AssemblyReferencePathType

The type of assembly reference path, used to specify the same relative assembly location on different devices or versions.

ExecutionSecurityMode

The security mode used to prevent compiled code from crashing the game by hogging the main thread.

LogLevel

The log level used by the runtime.

ScriptSecurityMode

The security mode used when loading and compiling assemblies.

Class AssemblyReferenceAsset

Represents a self contains assembly reference generated from a specific assembly file, but serialized as an asset so that the reference can be used on all platforms.

Inheritance

object
Object
ScriptableObject
AssemblyReferenceAsset

Implements

ICompilationReference

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[CreateAssetMenu(menuName = "Roslyn C# 2.0/Assembly Reference Asset")]
public sealed class AssemblyReferenceAsset : ScriptableObject, ICompilationReference
```

Properties

AssemblyImage

The raw portable executable image of the compiled assembly, used for referencing purposes.

Declaration

```
public byte[] AssemblyImage { get; }
```

Property Value

TYPE	DESCRIPTION
byte[]	

CompilationReference

Get the metadata reference from the assembly image.

Declaration

```
public MetadataReference CompilationReference { get; }
```

Property Value

TYPE	DESCRIPTION
MetadataReference	

HasValidImage

Return a value indicating whether that reference asset contains a valid assembly image.

Declaration

```
public bool HasValidImage { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

ReferenceAssembly

The reference assembly that this reference asset was created from. Used to auto update the assembly reference when the assembly image is changed on disk.

Declaration

```
public AssemblyReferenceSource ReferenceAssembly { get; }
```

Property Value

TYPE	DESCRIPTION
AssemblyReferenceSource	

Methods

RefreshAssemblyReference()

Force a refresh of the assembly reference image from disk if available. This requires the assembly reference path to be valid and available on the current system.

Declaration

```
[ContextMenu("Refresh Assembly Reference")]
public void RefreshAssemblyReference()
```

Implements

ICompilationReference

Class AssemblyReferenceSource

Inheritance

object

AssemblyReferenceSource

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public sealed class AssemblyReferenceSource
```

Properties

AssemblyExists

Checks whether the specified assembly reference exists if it is set. False if it does not exist or if this asset does not yet reference an assembly.

Declaration

```
public bool AssemblyExists { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

AssemblyFullPath

Get the full path to the assembly. AssemblyPath may be relative to the Unity install or project folder, this property will resolve the full rooted path.

Declaration

```
public string AssemblyFullPath { get; }
```

Property Value

TYPE	DESCRIPTION
string	

AssemblyName

The name of the assembly that is referenced.

Declaration

```
public string AssemblyName { get; }
```

Property Value

TYPE	DESCRIPTION
string	

AssemblyPath

The original path of the assembly that is referenced.

Declaration

```
public string AssemblyPath { get; }
```

Property Value

TYPE	DESCRIPTION
string	

AssemblyPathType

The type of path that is stored by this assembly.

Declaration

```
public AssemblyReferenceSource.AssemblyReferencePathType AssemblyPathType { get; }
```

Property Value

TYPE	DESCRIPTION
AssemblyReferenceSource.AssemblyReferencePathType	

LastWriteTime

Get the last date and time that the assembly was compiled.

Declaration

```
public DateTime LastWriteTime { get; }
```

Property Value

TYPE	DESCRIPTION
DateTime	

LastWriteTimeUTC

Get the last time the assembly was compiled in UTC file time.

Declaration

```
public long LastWriteTimeUTC { get; }
```

Property Value

TYPE	DESCRIPTION
long	

Methods

AutoRefreshAssemblyReference()

Attempt to check and update the source assembly if it has been modified since last refresh.

Declaration

```
public bool AutoRefreshAssemblyReference()
```

Returns

TYPE	DESCRIPTION
bool	True if the assembly location is valid and the assembly was modified on disk, or false if not

RefreshAssemblyReference()

Declaration

```
public void RefreshAssemblyReference()
```

UpdateAssemblyReference(Assembly)

Update this reference asset with information for the provided assembly. Note that the specified assembly `Location` property must have a valid load path set, otherwise you should use UpdateAssemblyReference(string, string).

Declaration

```
public void UpdateAssemblyReference(Assembly assembly)
```

Parameters

TYPE	NAME	DESCRIPTION
Assembly	assembly	The assembly to update from

Exceptions

TYPE	CONDITION
ArgumentNullException	Assembly is null
ArgumentException	Assembly `Location` property is not valid

UpdateAssemblyReference(string, string)

Update this reference asset with the provided information.

Declaration

```
public void UpdateAssemblyReference(string assemblyPath, string assemblyName)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyPath	The path of the source assembly, used to read the assembly image

TYPE	NAME	DESCRIPTION
string	assemblyName	The name of the assembly. Should be a short name without public key information

Exceptions

TYPE	CONDITION
ArgumentException	Assembly path is null or empty
FileNotFoundException	Could not find the assembly path

Enum AssemblyReferenceSource.AssemblyReferencePathType

The type of assembly reference path, used to specify the same relative assembly location on different devices or versions.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public enum AssemblyReferenceSource.AssemblyReferencePathType
```

Fields

NAME	DESCRIPTION
Absolute	The path is an absolute path and is only usable on the device it was created for.
UnityInstall	The path is relative to the Unity editor install location.
UnityProject	The path is relative to the Unity project location.

Class AssemblySource

Represents a specific assembly location with optional debug symbols.

Inheritance

object

AssemblySource

Implements

ICompilationReference

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public abstract class AssemblySource : ICompilationReference
```

Properties

AssemblyHintPath

A hint path where the assembly is located on disk.

Declaration

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

Property Value

TYPE	DESCRIPTION
string	

CompilationReference

Get the metadata reference for this compiled assembly. It allows this object to be passed directly as a compiler reference.

Declaration

```
public abstract MetadataReference CompilationReference { get; }
```

Property Value

TYPE	DESCRIPTION
MetadataReference	

DebugSymbolsHintPath

A hint path where the debug symbols are located on disk.

Declaration

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

Property Value

TYPE	DESCRIPTION
string	

HasDebugSymbols

Returns true if Portable PDB debug symbols are present in the compilation output. Requires compiling with optimize disabled to produce a debug build.

Declaration

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

Property Value

TYPE	DESCRIPTION
bool	

Methods

FromFile(string, string)

Create an assembly source from the specified assembly path.

Declaration

```
public static AssemblySource FromFile(string assemblyPath, string debugSymbolsPath = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyPath	The path of the managed .dll PE assembly file
string	debugSymbolsPath	The optional path of the debug symbols if present (Usually replace the assembly path extension with .pdb extension)

Returns

TYPE	DESCRIPTION
AssemblySource	An assembly source representing the provided assembly path

FromMemory(byte[], byte[])

Create an assembly source from the specified assembly image bytes.

Declaration

```
public static AssemblySource FromMemory(byte[] assemblyImage, byte[] debugSymbolsImage = null)
```

Parameters

TYPE	NAME	DESCRIPTION
byte[]	assemblyImage	The assembly image bytes (.dll PE file bytes)

TYPE	NAME	DESCRIPTION
byte[]	debugSymbolsImage	The optional debug symbols bytes

Returns

TYPE	DESCRIPTION
AssemblySource	An assembly source representing the provided assembly image

GetSecuritySource()

Get the assembly source for code security verification purposes.

Declaration

```
public abstract AssemblySource GetSecuritySource()
```

Returns

TYPE	DESCRIPTION
AssemblySource	

LoadAssembly(IAssemblyLoader)

Load the compiled assembly into memory.

Declaration

```
public abstract Assembly LoadAssembly(IAssemblyLoader assemblyLoader)
```

Parameters

TYPE	NAME	DESCRIPTION
IAssemblyLoader	assemblyLoader	

Returns

TYPE	DESCRIPTION
Assembly	

LoadAssemblyImage()

Load the entire assembly image bytes.

Declaration

```
public abstract byte[] LoadAssemblyImage()
```

Returns

TYPE	DESCRIPTION
byte[]	

LoadDebugSymbolsImage()

Load the entire portable PDB debug symbols image bytes.

Declaration

```
public abstract byte[] LoadDebugSymbolsImage()
```

Returns

TYPE	DESCRIPTION
byte[]	

Implements

ICompilationReference

Class CompileAsync

An awaitable object that is returned by async compile operations so you can wait for completion in a coroutine as well as access progress and status information. Used to wait until an async operation has been completed

Inheritance

object
CompileAsync
CompileAsync<T>

Implements

IEnumerator

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public class CompileAsync : IEnumerator
```

Constructors

CompileAsync(bool)

Create new operation.

Declaration

```
public CompileAsync(bool awaitable = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	awaitable	Can this operation be awaited o the main thread, by blocking until the task has completed. This is only possible for requests that run on a background thread, and UnityWebRequest for example cannot support this

Fields

isSuccessful

Was the operation successful or did something go wrong.

Declaration

```
protected bool isSuccessful
```

Field Value

TYPE	DESCRIPTION
bool	

status

Get the current status of the async operation.

Declaration

```
protected string status
```

Field Value

TYPE	DESCRIPTION
string	

Properties

CompileResult

Get the compile result for this request.

Declaration

```
public CompileResult CompileResult { get; }
```

Property Value

TYPE	DESCRIPTION
CompileResult	

Current

IEnumerator.Current implementation.

Declaration

```
public object Current { get; }
```

Property Value

TYPE	DESCRIPTION
object	

IsDone

Returns true if the async operation has finished or false if it is still running.

Declaration

```
public bool IsDone { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsSuccessful

Returns true if the async operation completed successfully or false if an error occurred.

Declaration

```
public bool IsSuccessful { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Progress

Get the current progress of the async operation. This is a normalized value between 0-1.

Declaration

```
public float Progress { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
float	

ProgressPercentage

Get the current progress percentage of the async operation.

Declaration

```
public int ProgressPercentage { get; }
```

Property Value

TYPE	DESCRIPTION
int	

Status

Get the current status of the async operation.

Declaration

```
public string Status { get; protected set; }
```

Property Value

TYPE	DESCRIPTION
string	

Task

Get the Task for this async operation for use in C# async/await contexts.

Declaration

```
public Task Task { get; }
```

Property Value

TYPE	DESCRIPTION
Task	

Methods

Await(long)

Block the main thread until the async operation has completed. Use with caution. This can cause an infinite loop if the async operation never completes, if the operation is not true async, or if the async operation relies on data from the main thread.

Declaration

```
public void Await(long msTimeout = 10000)
```

Parameters

TYPE	NAME	DESCRIPTION
long	msTimeout	

Exceptions

TYPE	CONDITION
TimeoutException	The await operation took longer that the specified timeout milliseconds, so was aborted to avoid infinite waiting

Complete(bool, CompileResult)

Complete the operation with the specified success status. This will cause IsDone to become true and Progress to become 1.

Declaration

```
public void Complete(bool success, CompileResult compileResult)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	success	Was the operation completed successfully
CompileResult	compileResult	The compile result for the request

Error(string, CompileResult)

Complete the operation with an error status. This will cause IsDone to become true and Progress to become 1.

Declaration

```
public void Error(string status, CompileResult compileResult)
```

Parameters

TYPE	NAME	DESCRIPTION
string	status	The error message for the failure

TYPE	NAME	DESCRIPTION
CompileResult	compileResult	The compile result for the request

MoveNext()

IEnumerator.MoveNext() implementation.

Declaration

```
public bool MoveNext()
```

Returns

TYPE	DESCRIPTION
bool	True if the enumerator advanced successfully or false if not

Reset()

IEnumerator.Reset() implementation.

Declaration

```
public virtual void Reset()
```

UpdateProgress(int, int)

Update the load progress for this operation. Calculates the progress as a value from 0-1 based on the input values.

Declaration

```
protected CompileAsync UpdateProgress(int current, int total)
```

Parameters

TYPE	NAME	DESCRIPTION
int	current	The current number of tasks that have been completed
int	total	The total number of tasks that should be completed

Returns

TYPE	DESCRIPTION
CompileAsync	

UpdateProgress(float)

Update the load progress for this operation. The specified progress value should be in the range of 0-1, and will be clamped if not.

Declaration

```
protected CompileAsync UpdateProgress(float progress)
```

Parameters

TYPE	NAME	DESCRIPTION
float	progress	The current progress value between 0-1

Returns

TYPE	DESCRIPTION
CompileAsync	

UpdateStatus(string)

Update the status message for this operation. Useful to show the current status if a failure occurs.

Declaration

```
protected CompileAsync UpdateStatus(string status)
```

Parameters

TYPE	NAME	DESCRIPTION
string	status	The status message

Returns

TYPE	DESCRIPTION
CompileAsync	

UpdateTasks()

Called when the async operation can perform some logic.

Declaration

```
protected virtual void UpdateTasks()
```

Implements

IEnumerator

Class CompileAsync<T>

An awaitable object that is returned by async compile operations so you can wait for completion in a coroutine as well as access progress and status information. Used to wait for an async operation to be completed with a result object.

Inheritance

object
CompileAsync
CompileAsync<T>

Implements

IEnumerator

Inherited Members

CompileAsync.isSuccessful
CompileAsync.status
CompileAsync.Status
CompileAsync.Progress
CompileAsync.ProgressPercentage
CompileAsync.IsDone
CompileAsync.CompileResult
CompileAsync.IsSuccessful
CompileAsync.Current
CompileAsync.UpdateTasks()
CompileAsync.MoveNext()
CompileAsync.Await(long)
CompileAsync.Error(string, CompileResult)
CompileAsync.Complete(bool, CompileResult)

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public class CompileAsync<T> : CompileAsync, IEnumerator
```

Type Parameters

NAME	DESCRIPTION
T	The generic result type

Constructors

CompileAsync(bool)

Create a new instance.

Declaration

```
public CompileAsync(bool awaitable = true)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
bool	awaitable	Can this operation be awaited o the main thread, by blocking until the task has completed. This is only possible for requests that run on a background thread, and UnityWebRequest for example cannot support this

Properties

Result

Get the generic result of the async operation.

Declaration

```
public T Result { get; }
```

Property Value

TYPE	DESCRIPTION
T	

SecurityReport

Get the code security report for the loaded assembly if code validation was performed.

Declaration

```
public CodeSecurityReport SecurityReport { get; }
```

Property Value

TYPE	DESCRIPTION
CodeSecurityReport	

Task

Get the Task for this async operation for use in C# async/await contexts.

Declaration

```
public Task<T> Task { get; }
```

Property Value

TYPE	DESCRIPTION
Task<T>	

Methods

Complete(bool, CompileResult, CodeSecurityReport, T)

Complete the operation with the specified success status. This will cause IsDone to become true and Progress to become 1.

Declaration

```
public void Complete(bool success, CompileResult compileResult, CodeSecurityReport securityReport, T result)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	success	Was the operation completed successfully
CompileResult	compileResult	The compile result for the request
CodeSecurityReport	securityReport	The code security report if code security validation was performed
T	result	An optional result object

Error(string, CompileResult, CodeSecurityReport)

Complete the operation with an error status. This will cause IsDone to become true and Progress to become 1.

Declaration

```
public void Error(string status, CompileResult compileResult, CodeSecurityReport securityReport)
```

Parameters

TYPE	NAME	DESCRIPTION
string	status	The error message for the failure
CompileResult	compileResult	The compile result for the request
CodeSecurityReport	securityReport	The code security report if code security validation was performed

Reset()

Reset the async operation.

Declaration

```
public override void Reset()
```

Overrides

CompileAsync.Reset()

UpdateProgress(int, int)

Update the load progress for this operation. Calculates the progress as a value from 0-1 based on the input values.

Declaration

```
protected CompileAsync<T> UpdateProgress(int current, int total)
```

Parameters

TYPE	NAME	DESCRIPTION
int	current	The current number of tasks that have been completed
int	total	The total number of tasks that should be completed

Returns

TYPE	DESCRIPTION
CompileAsync<T>	

UpdateProgress(float)

Update the load progress for this operation. The specified progress value should be in the range of 0-1, and will be clamped if not.

Declaration

```
protected CompileAsync<T> UpdateProgress(float progress)
```

Parameters

TYPE	NAME	DESCRIPTION
float	progress	The current progress value between 0-1

Returns

TYPE	DESCRIPTION
CompileAsync<T>	

UpdateStatus(string)

Update the status message for this operation. Useful to show the current status if a failure occurs.

Declaration

```
protected CompileAsync<T> UpdateStatus(string status)
```

Parameters

TYPE	NAME	DESCRIPTION
string	status	The status message

Returns

TYPE	DESCRIPTION
CompileAsync<T>	

Implements

IEnumerator

Class CompileError

Represents a compiler message, warning or error relating to a specific compilation request.

Inheritance

object

CompileError

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class CompileError
```

Properties

Code

Get the error message code.

Declaration

```
public string Code { get; }
```

Property Value

TYPE	DESCRIPTION
string	

IsError

Returns true if this is a compiler error.

Declaration

```
public bool IsError { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsInfo

Returns true if this is a compiler message.

Declaration

```
public bool IsInfo { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsSuppressed

Returns true if this error is suppressed.

Declaration

```
public bool IsSuppressed { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsWarning

Returns true if this is a compiler warning.

Declaration

```
public bool IsWarning { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsWarningAsError

Returns true if this is a warning elevated to an error due to strict compile.

Declaration

```
public bool IsWarningAsError { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Message

Get the error message text.

Declaration

```
public string Message { get; }
```

Property Value

TYPE	DESCRIPTION
string	

SourceColumn

Get the associated column index if available for the error.

Declaration

```
public int SourceColumn { get; }
```

Property Value

TYPE	DESCRIPTION
int	

SourceFile

Get the associated source file if available for the error.

Declaration

```
public string SourceFile { get; }
```

Property Value

TYPE	DESCRIPTION
string	

SourceLine

Get the associated line number if available for the error.

Declaration

```
public int SourceLine { get; }
```

Property Value

TYPE	DESCRIPTION
int	

Methods

ToString()

Get this error as a string representation.

Declaration

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

Returns

TYPE	DESCRIPTION
string	

Overrides

object.ToString()

Class CompileFlags

Represents a set of compiler flags to change the compilation and output behaviour of a compile request.

Inheritance

object

CompileFlags

CompileOptions

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public class CompileFlags
```

Constructors

CompileFlags(CompileFlags)

Create a new instance.

Declaration

```
protected CompileFlags(CompileFlags other)
```

Parameters

TYPE	NAME	DESCRIPTION
CompileFlags	other	

CompileFlags(string, string, bool, bool, bool, bool, bool, bool, int, LanguageVersion, OutputKind, Platform, DebugInformationFormat)

Create a new instance.

Declaration

```
public CompileFlags(string outputDirectory = null, string outputName = null, bool allowUnsafe = false, bool allowOptimize = false, bool allowConcurrentCompile = true, bool deterministic = true, bool generateInMemory = true, bool generateSymbols = true, int warningLevel = 4, LanguageVersion languageVersion = LanguageVersion.Default, OutputKind outputKind = OutputKind.DynamicallyLinkedLibrary, Platform targetPlatform = Platform.AnyCpu, DebugInformationFormat debugFormat = DebugInformationFormat.PortablePdb)
```

Parameters

TYPE	NAME	DESCRIPTION
string	outputDirectory	
string	outputName	
bool	allowUnsafe	
bool	allowOptimize	
bool	allowConcurrentCompile	

TYPE	NAME	DESCRIPTION
bool	deterministic	
bool	generateInMemory	
bool	generateSymbols	
int	warningLevel	
LanguageVersion	languageVersion	
OutputKind	outputKind	
Platform	targetPlatform	
DebugInformationFormat	debugFormat	

Fields

Default

Get the default compiler flags.

Declaration

```
public static readonly CompileFlags Default
```

Field Value

TYPE	DESCRIPTION
CompileFlags	

Properties

AllowConcurrentCompile

Can the compilation request use multiple threads if available.

Declaration

```
public bool AllowConcurrentCompile { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

AllowOptimize

Should the compiled assembly be optimized for best performance.

Declaration

```
public bool AllowOptimize { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

AllowUnsafe

Should unsafe code be allowed to compile.

Declaration

```
public bool AllowUnsafe { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

DebugFormat

The debug symbol format produced by the compiler.

Declaration

```
public DebugInformationFormat DebugFormat { get; set; }
```

Property Value

TYPE	DESCRIPTION
DebugInformationFormat	

Deterministic

Should the compiler produce identical results for the same input.

Declaration

```
public bool Deterministic { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

GenerateInMemory

Should the compiler generate the compiled assembly in memory or on disk. Note that some platforms only support compile in memory, and for those platforms this option will be ignored.

Declaration

```
public bool GenerateInMemory { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

TYPE	DESCRIPTION

GenerateSymbols

Should the compiler generate portable PDB debug symbols.

Declaration

```
public bool GenerateSymbols { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

LanguageVersion

The C# language version to target.

Declaration

```
public LanguageVersion LanguageVersion { get; set; }
```

Property Value

TYPE	DESCRIPTION
LanguageVersion	

OutputDirectory

The optional directory where the compiled assembly will be generated if GenerateInMemory is disabled. If the directory is null or empty then the compiled assembly will be generated in the current executing directory (Project folder in case of Unity editor, and AppFolder in case of standalone).

Declaration

```
public string OutputDirectory { get; set; }
```

Property Value

TYPE	DESCRIPTION
string	

OutputKind

The output kind to produce from the compilation. Should always be .dll expect for very specific cases.

Declaration

```
public OutputKind OutputKind { get; set; }
```

Property Value

TYPE	DESCRIPTION
OutputKind	

OutputName

The optional name of the output assembly. If the name is null or empty then a unique name will be generated using a guid format.

Declaration

```
public string OutputName { get; set; }
```

Property Value

TYPE	DESCRIPTION
string	

TargetPlatform

The platform that the compiled assembly should target. AnyCPU is recommended for almost all cases.

Declaration

```
public Platform TargetPlatform { get; set; }
```

Property Value

TYPE	DESCRIPTION
Platform	

WarningLevel

The warning level used by the compiler.

Declaration

```
public int WarningLevel { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

Methods

FromSettings()

Create a new instance with settings copied from the specified Roslyn C# settings window. The resulting object is a clone of the main source settings, so can be modified without affecting the global settings.

Declaration

```
public static CompileFlags FromSettings()
```

Returns

TYPE	DESCRIPTION
CompileFlags	

FromSettings(RoslynCSharpSettings)

Create a new instance with settings copied from the specified Roslyn C# settings window. The resulting object is a clone of the main source settings, so can be modified without affecting the global settings.

Declaration

```
public static CompileFlags FromSettings(RoslynCSharpSettings settings)
```

Parameters

TYPE	NAME	DESCRIPTION
RoslynCSharpSettings	settings	

Returns

TYPE	DESCRIPTION
CompileFlags	

Exceptions

TYPE	CONDITION
ArgumentNullException	

GetCompileOptions()

Get the compilation options from this object.

Declaration

```
public CSharpCompilationOptions GetCompileOptions()
```

Returns

TYPE	DESCRIPTION
CSharpCompilationOptions	The compilation options created from this object

GetOutputExtension()

Get the output file extension based on the OutputKind.

Declaration

```
public string GetOutputExtension()
```

Returns

TYPE	DESCRIPTION
string	

GetParseOptions(IEnumerable<string>)

Get the parser options from this object.

Declaration

```
public CSharpParseOptions GetParseOptions(IEnumerable<string> defineSymbols)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<string>	defineSymbols	The optional parser define symbols

Returns

TYPE	DESCRIPTION
CSharpParseOptions	Parser options created from this object

Class CompileOptions

Represents a set of options provided to the compiler for a specific compile request.

Inheritance

object
CompileFlags
CompileOptions

Inherited Members

CompileFlags.OutputDirectory
CompileFlags.OutputName
CompileFlags.AllowUnsafe
CompileFlags.AllowOptimize
CompileFlags.AllowConcurrentCompile
CompileFlags.Deterministic
CompileFlags.GenerateInMemory
CompileFlags.GenerateSymbols
CompileFlags.WarningLevel
CompileFlags.LanguageVersion
CompileFlags.OutputKind
CompileFlags.TargetPlatform
CompileFlags.DebugFormat
CompileFlags.GetOutputExtension()
CompileFlags.GetParseOptions(IEnumerable<string>)
CompileFlags.GetCompileOptions()
CompileFlags.FromSettings()
CompileFlags.FromSettings(RoslynCSharpSettings)

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public class CompileOptions : CompileFlags
```

Constructors

CompileOptions(CompileFlags, LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>, IEnumerable<IParserProcessor>, IEnumerable<ICompilationProcessor>, ExecutionSecuritySettings, IMainTypeSelector)

Create a new instance.

Declaration

```
public CompileOptions(CompileFlags compilerFlags = null, LogLevel compilerLogLevel = LogLevel.Errors,
IEnumerable<string> defineSymbols = null, IEnumerable<ICompilationReference> references = null,
IEnumerable<IParserProcessor> parserProcessors = null, IEnumerable<ICompilationProcessor>
compilationProcessors = null, ExecutionSecuritySettings executionSecuritySettings = null, IMainTypeSelector
mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
CompileFlags	compilerFlags	

TYPE	NAME	DESCRIPTION
LogLevel	compilerLogLevel	
IEnumerable<string>	defineSymbols	
IEnumerable<ICompilationReference>	references	
IEnumerable<IParserProcessor>	parserProcessors	
IEnumerable<ICompilationProcessor>	compilationProcessors	
ExecutionSecuritySettings	executionSecuritySettings	
IMainTypeSelector	mainTypeSelector	

CompileOptions(CompileOptions, IEnumerable<string>, IEnumerable<ICompilationReference>)

Create a new instance.

Declaration

```
protected CompileOptions(CompileOptions other, IEnumerable<string> extraDefineSymbols = null,
IEnumerable<ICompilationReference> extraReferences = null)
```

Parameters

TYPE	NAME	DESCRIPTION
CompileOptions	other	
IEnumerable<string>	extraDefineSymbols	
IEnumerable<ICompilationReference>	extraReferences	

Fields

Default

Get the default compiler options.

Declaration

```
public static readonly CompileOptions Default
```

Field Value

TYPE	DESCRIPTION
CompileOptions	

Properties

CompilationProcessors

The compilation processors that will run after the compilation has completed.

Declaration

```
public IList<ICompilationProcessor> CompilationProcessors { get; }
```

Property Value

TYPE	DESCRIPTION
IList<ICompilationProcessor>	

CompilerLogLevel

Should compiler warnings and errors be logged to the Unity console.

Declaration

```
public LogLevel CompilerLogLevel { get; set; }
```

Property Value

TYPE	DESCRIPTION
LogLevel	

DefineSymbols

The define symbols used for this request.

Declaration

```
public IList<string> DefineSymbols { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

ExecutionSecuritySettings

The execution security settings used when compiling.

Declaration

```
public ExecutionSecuritySettings ExecutionSecuritySettings { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExecutionSecuritySettings	

MainTypeSelector

The selector used to identify the main type defined within an assembly.

Declaration

```
public IMainTypeSelector MainTypeSelector { get; set; }
```

Property Value

TYPE	DESCRIPTION
IMainTypeSelector	

ParserProcessors

The parser processors that will be executed after the parsing stage has completed.

Declaration

```
public IList<IParserProcessor> ParserProcessors { get; }
```

Property Value

TYPE	DESCRIPTION
IList<IParserProcessor>	

References

The referenced assemblies used for this request.

Declaration

```
public IList<ICompilationReference> References { get; }
```

Property Value

TYPE	DESCRIPTION
IList<ICompilationReference>	

Methods

FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>)

Create a new instance with settings copied from the Roslyn C# settings window. The resulting object is a clone of the main source settings, so can be modified without affecting the global settings.

Declaration

```
public static CompileOptions FromSettings(LogLevel compilerLogLevel = LogLevel.Errors, IEnumerable<string>
extraDefineSymbols = null, IEnumerable<ICompilationReference> extraReferences = null)
```

Parameters

TYPE	NAME	DESCRIPTION
LogLevel	compilerLogLevel	
IEnumerable<string>	extraDefineSymbols	
IEnumerable<ICompilationReference>	extraReferences	

Returns

TYPE	DESCRIPTION
CompileOptions	

FromSettings(RoslynCSharpSettings, LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>)

Create a new instance with settings copied from the specified Roslyn C# settings window. The resulting object is a clone of the main source settings, so can be modified without affecting the global settings.

Declaration

```
public static CompileOptions FromSettings(RoslynCSharpSettings settings, LogLevel compilerLogLevel =
LogLevel.Errors, IEnumerable<string> extraDefineSymbols = null, IEnumerable<ICompilationReference>
extraReferences = null)
```

Parameters

TYPE	NAME	DESCRIPTION
RoslynCSharpSettings	settings	The settings asset to copy the options from
LogLevel	compilerLogLevel	
IEnumerable<string>	extraDefineSymbols	
IEnumerable<ICompilationReference>	extraReferences	

Returns

TYPE	DESCRIPTION
CompileOptions	

Class CompileRequest

Represents a single compilation request which can comprise of one or many source texts, files or syntax trees.

Inheritance

object

CompileRequest

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class CompileRequest
```

Constructors

CompileRequest(CompileFlags, IEnumerable<IParserProcessor>, IEnumerable<ICompilationProcessor>)

Create a new compilation request.

Declaration

```
public CompileRequest(CompileFlags flags = null, IEnumerable<IParserProcessor> parserProcessors = null,
IEnumerable<ICompilationProcessor> compilationProcessors = null)
```

Parameters

TYPE	NAME	DESCRIPTION
CompileFlags	flags	The compiler flags to use or null for default
IEnumerable<IParserProcessor>	parserProcessors	The optional parser processor to use for additional parser post processing
IEnumerable<ICompilationProcessor>	compilationProcessors	The optional compilation processor to use for additional IL post processing

Methods

CompileSyntaxTree(SyntaxTree, IEnumerable<ICompilationReference>)

Compile the specified syntax tree with the provided references.

Declaration

```
public CompileResult CompileSyntaxTree(SyntaxTree syntaxTree, IEnumerable<ICompilationReference> references)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	The syntax tree to compile

TYPE	NAME	DESCRIPTION
IEnumerable<ICompilationReference>	references	The optional references to compile with

Returns

TYPE	DESCRIPTION
CompileResult	The result of the compile request

CompileSyntaxTrees(SyntaxTree[], IEnumerable<ICompilationReference>)

Compile the specified syntax trees as a batch with the provided references.

Declaration

```
public CompileResult CompileSyntaxTrees(SyntaxTree[] syntaxTrees, IEnumerable<ICompilationReference>
references)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	The syntax trees to compile
IEnumerable<ICompilationReference>	references	The optional references to compile with

Returns

TYPE	DESCRIPTION
CompileResult	The result of the compile request

ParseFile(string, IEnumerable<string>)

Parse the specified source file into syntax trees. Note that parser processors will be run at this stage.

Declaration

```
public SyntaxTree[] ParseFile(string cSharpFile, IEnumerable<string> defineSymbols = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	The C# source file to parse
IEnumerable<string>	defineSymbols	The optional define symbols used to determine which define conditions are met

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The syntax trees representing the parsed and processed code

Exceptions

TYPE	CONDITION
ArgumentException	The file path is null or empty

ParseFiles(IEnumerable<string>, IEnumerable<string>)

Parse the specified source files into syntax trees. Note that parser processors will be run at this stage.

Declaration

```
public SyntaxTree[] ParseFiles(IEnumerable<string> cSharpFiles, IEnumerable<string> defineSymbols = null)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<string>	cSharpFiles	The C# source files to parse
IEnumerable<string>	defineSymbols	The optional define symbols used to determine which define conditions are met

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The syntax trees representing the parsed and processed code

Exceptions

TYPE	CONDITION
ArgumentNullException	The file paths are null

ParseSource(string, IEnumerable<string>)

Parse the specified source file into syntax trees. Note that parser processors will be run at this stage.

Declaration

```
public SyntaxTree[] ParseSource(string cSharpSource, IEnumerable<string> defineSymbols = null)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
string	cSharpSource	The C# source string to parse
IEnumerable<string>	defineSymbols	The optional define symbols used to determine which define conditions are met

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The syntax trees representing the parsed and processed code

Exceptions

TYPE	CONDITION
ArgumentException	Input source is null or empty

ParseSources(IEnumerable<string>, IEnumerable<string>)

Parse the specified source strings into syntax trees. Note that parser processors will be run at this stage.

Declaration

```
public SyntaxTree[] ParseSources(IEnumerable<string> cSharpSources, IEnumerable<string> defineSymbols = null)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<string>	cSharpSources	The C# source strings to parse
IEnumerable<string>	defineSymbols	The optional define symbols used to determine which define conditions are met

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The syntax trees representing the parsed and processed code

Exceptions

TYPE	CONDITION
ArgumentNullException	Input sources is null

ProcessSyntaxTree(SyntaxTree)

Run parser processors on the specified C# syntax tree.

Declaration

```
public SyntaxTree[] ProcessSyntaxTree(SyntaxTree syntaxTree)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	The syntax tree to process

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The processed syntax trees

Exceptions

TYPE	CONDITION
ArgumentNullException	Input syntax tree is null

ProcessSyntaxTrees(SyntaxTree[])

Run parser processors on the specified C# syntax trees.

Declaration

```
public SyntaxTree[] ProcessSyntaxTrees(SyntaxTree[] syntaxTrees)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	The syntax trees to process

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The processed syntax trees

Exceptions

TYPE	CONDITION
ArgumentNullException	Input syntax trees is null

Class CompileResult

Represents a result of a specific compilation request by the compiler.

Inheritance

object

CompileResult

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class CompileResult
```

Properties

Assembly

Get the compiled assembly if compilation was successful, or null if not.

Declaration

```
public AssemblySource Assembly { get; }
```

Property Value

TYPE	DESCRIPTION
AssemblySource	

Errors

Get all compilation messages, warnings and errors reported by the compiler.

Declaration

```
public CompileError[] Errors { get; }
```

Property Value

TYPE	DESCRIPTION
CompileError[]	

Success

Was the compilation successful.

Declaration

```
public bool Success { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Methods

ReportCompilationErrors(LogLevel, bool)

Log all compiled diagnostic messages to the Unity console which meet the specified log level.

Declaration

```
public void ReportCompilationErrors(LogLevel logLevel, bool logHeader = true)
```

Parameters

TYPE	NAME	DESCRIPTION
LogLevel	logLevel	The log level used to restrict the amount of detail reported
bool	logHeader	Should a header message be logged to the console indicating the start of the report

Enum ExecutionSecurityMode

The security mode used to prevent compiled code from crashing the game by hogging the main thread.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Flags]
public enum ExecutionSecurityMode : uint
```

Fields

NAME	DESCRIPTION
ExecutionIterations	Ensure that user code will return after a set number of iterations to avoid infinite or very long execution cycles.
ExecutionTimeout	Ensure that user code will time out if a long or infinite execution cycle is entered. The timeout time can be set via execution settings.
None	No execution security is injected. Recommended only if absolute performance is required for tight loops.

Class ExecutionSecuritySettings

The execution settings used to ensure that compiled code cannot cause the host app to crash by blocking the main thread for a long or infinite time. Note that execution security can only be applied to assemblies that were compiled by Roslyn C#, as safety checks need to be injected after the parsing phase.

Inheritance

object

ExecutionSecuritySettings

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public sealed class ExecutionSecuritySettings
```

Constructors

ExecutionSecuritySettings(ExecutionSecurityMode, float, int)

Create a new instance.

Declaration

```
public ExecutionSecuritySettings(ExecutionSecurityMode securityMode = ExecutionSecurityMode.ExecutionTimeout |
ExecutionSecurityMode.ExecutionIterations, float timeoutSeconds = 2, int maxIterations = 5000)
```

Parameters

TYPE	NAME	DESCRIPTION
ExecutionSecurityMode	securityMode	The execution security flags to use
float	timeoutSeconds	The optional time in seconds for loop timeout"/>
int	maxIterations	The optional max iterations for loops

Properties

SecurityMode

The security mode used to process compiled code.

Declaration

```
public ExecutionSecurityMode SecurityMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExecutionSecurityMode	

TimeoutIterations

The maximum number of iterations of a looping execution cycle before execution is forcefully terminated in user code.

Declaration

```
public int TimeoutIterations { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

TimeoutSeconds

The amount of time in seconds before execution is forcefully terminated in user code.

Declaration

```
public float TimeoutSeconds { get; set; }
```

Property Value

TYPE	DESCRIPTION
float	

Interface IAssemblyLoader

Used to load an assembly into memory from a specific location.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public interface IAssemblyLoader
```

Methods

LoadAssembly(byte[], byte[])

Load an assembly from the specified PE image bytes.

Declaration

```
Assembly LoadAssembly(byte[] assemblyImage, byte[] debugSymbolsImage = null)
```

Parameters

TYPE	NAME	DESCRIPTION
byte[]	assemblyImage	The portable executable image
byte[]	debugSymbolsImage	The optional debug symbols image

Returns

TYPE	DESCRIPTION
Assembly	

LoadAssembly(string, string)

Load an assembly from the specified path.

Declaration

```
Assembly LoadAssembly(string assemblyPath, string debugSymbolsPath = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyPath	The assembly path to use
string	debugSymbolsPath	The optional path to the pdb symbols path

Returns

TYPE	DESCRIPTION
Assembly	

Interface ICompilationProcessor

Called by the compiler when compilation has completed. Allows for additional IL processing to occur after the compilation request.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public interface ICompilationProcessor
```

Properties

Priority

Get the processor priority in the case that multiple processors are used. Default priority is 100 and implementation with a higher priority will run after those with a lower priority.

Declaration

```
int Priority { get; }
```

Property Value

TYPE	DESCRIPTION
int	

Methods

OnPostProcess(AssemblySource)

Called by the compiler with the compiled assembly.

Declaration

```
AssemblySource OnPostProcess(AssemblySource assembly)
```

Parameters

TYPE	NAME	DESCRIPTION
AssemblySource	assembly	The assembly that was compiled from the request

Returns

TYPE	DESCRIPTION
AssemblySource	A modified/IL post processed compilation or the same assembly if no modification needs to take place

Interface ICompilationReference

Represents an object that can be referenced as a dependency by the compiler.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public interface ICompilationReference
```

Properties

CompilationReference

Get the compiler reference metadata.

Declaration

```
MetadataReference CompilationReference { get; }
```

Property Value

TYPE	DESCRIPTION
MetadataReference	

Interface IMainTypeSelector

Interface for selecting the main type defined inside an assembly. Use DefaultMainTypeSelector for default type selection/

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public interface IMainTypeSelector
```

Methods

SelectMainType(ScriptAssembly, IReadOnlyList<ScriptType>)

Select the main type from the given assembly context.

Declaration

```
ScriptType SelectMainType(ScriptAssembly definingAssembly, IReadOnlyList<ScriptType> allTypes)
```

Parameters

TYPE	NAME	DESCRIPTION
ScriptAssembly	definingAssembly	The assembly where the types are defined
IReadOnlyList<ScriptType>	allTypes	All non-nested types defined within the assembly, both public and non-public

Returns

TYPE	DESCRIPTION
ScriptType	The type identified as the main type in this assembly

Interface IParserProcessor

Called by the compiler when parsing has completed. Allows for additional syntax tree processing to occur during the compilation request.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public interface IParserProcessor
```

Properties

Priority

Get the processor priority in the case that multiple processors are used. Default priority is 100 and implementation with a higher priority will run after those with a lower priority.

Declaration

```
int Priority { get; }
```

Property Value

TYPE	DESCRIPTION
int	

Methods

OnPostProcess(SyntaxTree[])

Called by the compiler with the parsed syntax tree.

Declaration

```
SyntaxTree[] OnPostProcess(SyntaxTree[] syntaxTrees)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	The syntax trees that were parsed from the compile request

Returns

TYPE	DESCRIPTION
SyntaxTree[]	A modified array of syntax trees with the same or differing number of trees returned (Add or remove syntax trees is supported), or the same syntax tree array if no modification needs to take place

Enum LogLevel

The log level used by the runtime.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public enum LogLevel
```

Fields

NAME	DESCRIPTION
All	Same as Messages. Messages, Warnings and Errors should be logged to the Unity console.
Errors	Only errors should be logged to the Unity console.
Messages	Messages, Warnings and Errors should be logged to the Unity console.
None	Nothing should be logged to the Unity console. Not recommended for most users as it makes it easy to miss critical messages if a problem occurs.
Warnings	Only Warnings and Errors should be logged to the Unity console. This is the default option.

Class RoslynCSharpSettings

Represents the main settings object for Roslyn C#. Used to store default and user settings in the project.

Inheritance

object
Object
ScriptableObject
RoslynCSharpSettings

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public sealed class RoslynCSharpSettings : ScriptableObject
```

Properties

AssemblyReferences

The default assembly references that will be used for compilation unless custom options are provided.

Declaration

```
public IList<AssemblyReferenceAsset> AssemblyReferences { get; }
```

Property Value

TYPE	DESCRIPTION
IList<AssemblyReferenceAsset>	

CodeSecurityRestrictions

The code security restrictions used for static security verification of compiled assemblies before loading.

Declaration

```
public CodeSecurityRestrictionValidator CodeSecurityRestrictions { get; set; }
```

Property Value

TYPE	DESCRIPTION
CodeSecurityRestrictionValidator	

CompilerFlags

The compiler options that are used for compilation requests unless custom options are provided.

Declaration

```
public CompileFlags CompilerFlags { get; set; }
```

Property Value

TYPE	DESCRIPTION
CompileFlags	

DefaultSettings

Get the default settings for the project. Should not be modified in most cases, because changes will be overwritten when updating the asset.

Declaration

```
public static RoslynCSharpSettings DefaultSettings { get; }
```

Property Value

TYPE	DESCRIPTION
RoslynCSharpSettings	

DefineSymbols

The preprocessor define symbols that will be used for compilation unless custom options are provided.

Declaration

```
public IList<string> DefineSymbols { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

ExecutionSecuritySettings

The execution security settings used for compiled code.

Declaration

```
public ExecutionSecuritySettings ExecutionSecuritySettings { get; set; }
```

Property Value

TYPE	DESCRIPTION
ExecutionSecuritySettings	

LogLevel

The log level used to determine which messages should be logged to the Unity console.

Declaration

```
public LogLevel LogLevel { get; set; }
```

Property Value

TYPE	DESCRIPTION
LogLevel	

SecurityCheckCode

Should loaded code be statically evaluated for potentially harmful code.

Declaration

```
public bool SecurityCheckCode { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

UserSettings

Get the user settings for the project.

Declaration

```
public static RoslynCSharpSettings UserSettings { get; }
```

Property Value

TYPE	DESCRIPTION
RoslynCSharpSettings	

Class ScriptAssembly

Represents a Assembly that has been loaded into a Roslyn C# ScriptDomain.

Inheritance

object

ScriptAssembly

Implements

ICompilationReference

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public abstract class ScriptAssembly : ICompilationReference
```

Constructors

ScriptAssembly(ScriptDomain, AssemblySource, CompileResult, CodeSecurityReport, IMainTypeSelector)

Create a new instance.

Declaration

```
protected ScriptAssembly(ScriptDomain domain, AssemblySource source, CompileResult compileResult,
CodeSecurityReport securityReport, IMainTypeSelector mainTypeSelector)
```

Parameters

TYPE	NAME	DESCRIPTION
ScriptDomain	domain	
AssemblySource	source	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
IMainTypeSelector	mainTypeSelector	

Exceptions

TYPE	CONDITION
ArgumentNullException	

Fields

DefaultMainTypeSelector

The default type selector used for identifying the main type defined with an assembly. The default behaviour is to search for public MonoBehaviour types, then public classes, then public any type, and finally fall back to first defined type. Note that the special `Module` type is not considered as a possible MainType.

Declaration

```
public static readonly IMainTypeSelector DefaultMainTypeSelector
```

Field Value

TYPE	DESCRIPTION
IMainTypeSelector	

Properties

AssemblyPath

Get the file path where the assembly was loaded from if it was loaded from disk. This value will be null if the assembly was loaded in memory.

Declaration

```
public virtual string AssemblyPath { get; }
```

Property Value

TYPE	DESCRIPTION
string	

CompilationReference

Get the metadata reference information so that this assembly object can be referenced as a dependency during compilation.

Declaration

```
public virtual MetadataReference CompilationReference { get; }
```

Property Value

TYPE	DESCRIPTION
MetadataReference	

CompilationResult

Get the CompilationResult for this assembly if it was runtime compiled.

Declaration

```
public virtual CompileResult CompilationResult { get; }
```

Property Value

TYPE	DESCRIPTION
CompileResult	

DebugSymbolsPath

Get the file path where the debug symbols were loaded if the associated assembly was loaded from disk. This value will be null if the associated assembly was loaded in memory, of if debug symbols (.pdb) are not present along side the assembly.

Declaration

```
public virtual string DebugSymbolsPath { get; }
```

Property Value

TYPE	DESCRIPTION
string	

Domain

Get the ScriptDomain that this assembly was loaded into.

Declaration

```
public virtual ScriptDomain Domain { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptDomain	

FullName

Get the full name of the assembly.

Declaration

```
public virtual string FullName { get; }
```

Property Value

TYPE	DESCRIPTION
string	

HasDebugSymbols

Return a value indicating whether the assembly was loaded with .pdb debug symbols or not.

Declaration

```
public virtual bool HasDebugSymbols { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsRuntimeCompiled

Return a value indicating whether this assembly was runtime compiled or simply loaded from a source.

Declaration

```
public virtual bool IsRuntimeCompiled { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsSecurityVerified

Return a value indicating whether this assembly has passed code security verification. This value will be true if code security verification was not run during loading.

Declaration

```
public virtual bool IsSecurityVerified { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

MainType

Try to get the main type defined in the assembly. The default behaviour is to search for public MonoBehaviour types, then public classes, then public any type, and finally fall back to first defined type. Note that the special `Module` type is not considered as a possible MainType.

Declaration

```
public virtual ScriptType MainType { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptType	

Name

Get the name of the assembly.

Declaration

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

Property Value

TYPE	DESCRIPTION
string	

SecurityReport

Get the Trivial.CodeSecurity.CodeSecurityReport for this assembly if code security verification occurred during loading.

Declaration

```
public virtual CodeSecurityReport SecurityReport { get; }
```

Property Value

TYPE	DESCRIPTION
CodeSecurityReport	

Source

Get the AssemblySource where this assembly was loaded from.

Declaration

```
public virtual AssemblySource Source { get; }
```

Property Value

TYPE	DESCRIPTION
AssemblySource	

SystemAssembly

Get the Assembly that is represented by this assembly.

Declaration

```
public abstract Assembly SystemAssembly { get; }
```

Property Value

TYPE	DESCRIPTION
Assembly	

Types

Get all types defined within the assembly. Includes all public and non-public types.

Declaration

```
public abstract ScriptType[] Types { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptType[]	

Version

Get the version of the assembly.

Declaration

```
public virtual Version Version { get; }
```

Property Value

TYPE	DESCRIPTION
Version	

Methods

EnumerateAllMonoBehaviourTypes(bool, bool)

Enumerate all types defined in this ScriptAssembly that inherit from UnityEngine.MonoBehaviour.

Declaration

```
public IEnumerable<ScriptType> EnumerateAllMonoBehaviourTypes(bool includeNonPublic = true, bool
enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	enumerateNestedTypes	

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of matching results

EnumerateAllScriptableObjectTypes(bool, bool)

Enumerate all types defined in this ScriptAssembly that inherit from UnityEngine.ScriptableObject.

Declaration

```
public IEnumerable<ScriptType> EnumerateAllScriptableObjectTypes(bool includeNonPublic = true, bool
enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	enumerateNestedTypes	

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of matching results

EnumerateAllSubTypesOf(Type, bool, bool)

Enumerate all types defined in this ScriptAssembly that inherits from the specified type.

Declaration

```
public virtual IEnumerable<ScriptType> EnumerateAllSubTypesOf(Type subType, bool includeNonPublic = true, bool
enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritance chain

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	Should the search include non public types
bool	enumerateNestedTypes	Should nested types be included in the search

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of matching results

EnumerateAllSubTypesOf<T>(bool, bool)

Enumerate all types defined in this ScriptAssembly that inherit from the specified generic type.

Declaration

```
public IEnumerable<ScriptType> EnumerateAllSubTypesOf<T>(bool includeNonPublic = true, bool
enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	enumerateNestedTypes	

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of matching results

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

EnumerateAllTypes(bool, bool)

Enumerate all defined types in this ScriptAssembly.

Declaration

```
public virtual IEnumerable<ScriptType> EnumerateAllTypes(bool includeNonPublic = true, bool
enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	enumerateNestedTypes	

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of all results

EnumerateAllUnityTypes(bool, bool)

Enumerate all types defined in this ScriptAssembly that inherit from UnityEngine.Object.

Declaration

```
public IEnumerable<ScriptType> EnumerateAllUnityTypes(bool includeNonPublic = true, bool enumerateNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	enumerateNestedTypes	

Returns

TYPE	DESCRIPTION
IEnumerable<ScriptType>	Enumerable of matching results

FindAllMonoBehaviourTypes(bool, bool)

Attempts to find all types defined in this ScriptAssembly that inherit from UnityEngine.MonoBehaviour.
If there are no types that inherit from UnityEngine.MonoBehaviour then the return value will be an empty array.

Declaration

```
public ScriptType[] FindAllMonoBehaviourTypes(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	findNestedTypes	

Returns

TYPE	DESCRIPTION
ScriptType[]	(Not Null) An array of ScriptType or an empty array if no matching type was found

FindAllScriptableObjectTypes(bool, bool)

Attempts to find all types defined in this ScriptAssembly that inherit from UnityEngine.ScriptableObject.
If there are no types that inherit from UnityEngine.ScriptableObject then the return value will be an empty array.

Declaration

```
public ScriptType[] FindAllScriptableObjectTypes(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	findNestedTypes	

Returns

TYPE	DESCRIPTION
ScriptType[]	(Not Null) An array of ScriptType or an empty array if no matching type was found

FindAllSubTypesOf(Type, bool, bool)

Attempts to find all types defined in this ScriptAssembly that inherits from the specified type. If there are no types that inherit from the specified type then the return value will be an empty array.

Declaration

```
public virtual ScriptType[] FindAllSubTypesOf(Type subType, bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritance chain
bool	includeNonPublic	Should the search include non public types
bool	findNestedTypes	Should nested types be included in the search

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
ScriptType[]	(Not Null) An array of ScriptType or an empty array if no matching type was found

FindAllSubTypesOf<T>(bool, bool)

Attempts to find all types defined in this ScriptAssembly that inherit from the specified generic type. If there are no types that inherit from the specified type then the return value will be an empty array.

Declaration

```
public ScriptType[] FindAllSubTypesOf<T>(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	findNestedTypes	

Returns

TYPE	DESCRIPTION
ScriptType[]	(Not Null) An array of ScriptType or an empty array if no matching type was found

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

FindAllTypes(bool, bool)

Returns an array of all defined types in this ScriptAssembly.

Declaration

```
public virtual ScriptType[] FindAllTypes(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	findNestedTypes	

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
ScriptType[]	An array of ScriptType representing all types defined in this ScriptAssembly

FindAllUnityTypes(bool, bool)

Attempts to find all types defined in this ScriptAssembly that inherit from UnityEngine.Object.
If there are no types that inherit from UnityEngine.Object then the return value will be an empty array.

Declaration

```
public ScriptType[] FindAllUnityTypes(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	
bool	findNestedTypes	

Returns

TYPE	DESCRIPTION
ScriptType[]	(Not Null) An array of ScriptType or an empty array if no matching type was found

FindEmbeddedResourceName(string)

Attempt to search for a full embedded resource name from the given search string. Useful for quickly finding resources without needing to include the resource namespace.

Declaration

```
public string FindEmbeddedResourceName(string searchName)
```

Parameters

TYPE	NAME	DESCRIPTION
string	searchName	The search term used to find the closest matching embedded resource

Returns

TYPE	DESCRIPTION
string	The full path of the embedded resource or null

FindSubTypeOf(Type, bool, bool)

Attempts to find a type defined in this ScriptAssembly that inherits from the specified base type. If there is more than one type that inherits from the specified base type, then the first matching type will be returned. If you want to find all types then use FindAllSubTypesOf(Type, bool, bool).

Declaration

```
public virtual ScriptType FindSubTypeOf(Type subType, bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritance chain
bool	includeNonPublic	Should the search include non public types
bool	findNestedTypes	Should nested types be included in the search

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

FindSubTypeOf(Type, string)

Attempts to find a type defined in this ScriptAssembly that inherits from the specified base type and matches the specified name. Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public virtual ScriptType FindSubTypeOf(Type subType, string name)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritance chain
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

FindSubTypeOf<T>(bool, bool)

Attempts to find a type defined in this ScriptAssembly that inherits from the specified generic type. If there is more than one type that inherits from the specified generic type, then the first matching type will be returned. If you want to find all types then use FindAllSubTypesOf<T>(bool, bool).

Declaration

```
public ScriptType FindSubTypeOf<T>(bool includeNonPublic = true, bool findNestedTypes = true)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	includeNonPublic	Should the search include non public types
bool	findNestedTypes	Should nested types be included in the search

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

FindSubTypeOf<T>(string)

Attempts to find a type defined in this ScriptAssembly that inherits from the specified generic type and matches the specified name. Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public ScriptType FindSubTypeOf<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

FindType(string)

Attempts to find a type defined in this ScriptAssembly with the specified name. Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public virtual ScriptType FindType(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

FindType(Type)

Attempts to find a type defined in this ScriptAssembly from the specified system type.

Declaration

```
public virtual ScriptType FindType(Type type)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	type	The system type to look for

Returns

TYPE	DESCRIPTION
ScriptType	An instance of ScriptType representing the found type or null if the type could not be found

GetEmbeddedResourceBytes(string)

Try to find and read the embedded resource content as a byte array from the given search name.

Declaration

```
public byte[] GetEmbeddedResourceBytes(string resourceName)
```

Parameters

TYPE	NAME	DESCRIPTION
string	resourceName	The name of the embedded resource or shortened search term to find the resource by partial name

Returns

TYPE	DESCRIPTION
byte[]	A byte array containing the raw contents of the embedded resource or null

GetEmbeddedResourceNames()

Get the full names of all embedded resources including namespace. Array will be empty if there are no embedded resources in the assembly.

Declaration

```
public string[] GetEmbeddedResourceNames()
```

Returns

TYPE	DESCRIPTION
string[]	An array of all embedded resources full names

GetEmbeddedResourceNamesImpl()

Get the full names of all embedded resources including namespace. Array should be empty if there are no embedded resources in the assembly.

Declaration

```
protected virtual string[] GetEmbeddedResourceNamesImpl()
```

Returns

TYPE	DESCRIPTION
string[]	An array of all embedded resources full names

GetEmbeddedResourceStream(string)

Try to find the embedded resource stream for reading with the given search name.

Declaration

```
public virtual Stream GetEmbeddedResourceStream(string resourceName)
```

Parameters

TYPE	NAME	DESCRIPTION
string	resourceName	The name of the embedded resource or shortened search term to find the resource by partial name

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
Stream	A readable stream or null

GetEmbeddedResourceText(string)

Try to find and read the embedded resource content as a string from the given search name.

Declaration

```
public string GetEmbeddedResourceText(string resourceName)
```

Parameters

TYPE	NAME	DESCRIPTION
string	resourceName	The name of the embedded resource or shortened search term to find the resource by partial name

Returns

TYPE	DESCRIPTION
string	A string containing the contents of the embedded resource or null

GetName()

Get the AssemblyName of this assembly.

Declaration

```
public virtual AssemblyName GetName()
```

Returns

TYPE	DESCRIPTION
AssemblyName	

HasSubTypeOf(Type)

Returns true if this ScriptAssembly defines one or more types that inherit from the specified type. The specified type may be a base class or interface type.

Declaration

```
public virtual bool HasSubTypeOf(Type subType)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritace chain

Returns

TYPE	DESCRIPTION
bool	True if there are one or more defined types that inherit from the specified type

HasSubTypeOf(Type, string)

Returns true if this ScriptAssembly defines a type that inherits from the specified type and matches the specified name. Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public virtual bool HasSubTypeOf(Type subType, string name)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check for in the inheritance chain
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
bool	True if a type that inherits from the specified type and has the specified name is defined

HasSubTypeOf<T>()

Returns true if this ScriptAssembly defined one or more types that inherit from the specified generic type. The specified generic type may be a base class or interface type.

Declaration

```
public bool HasSubTypeOf<T>()
```

Returns

TYPE	DESCRIPTION
bool	True if there are one or more defined types that inherit from the specified generic type

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

HasSubTypeOf<T>(string)

Returns true if this ScriptAssembly defines a type that inherits from the specified genric type and matches the specified name.

Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public bool HasSubTypeOf<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
bool	True if a type that inherits from the specified type and has the specified name is defined

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for in the inheritance chain

HasType(string)

Returns true if this ScriptAssembly defines a type with the specified name. Depending upon settings, name comparison may or may not be case sensitive.

Declaration

```
public virtual bool HasType(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the type to look for

Returns

TYPE	DESCRIPTION
bool	True if a type with the specified name is defined

LoadAssemblyImage()

Load the assembly image bytes into memory.

Declaration

```
public virtual byte[] LoadAssemblyImage()
```

Returns

TYPE	DESCRIPTION
byte[]	

LoadDebugSymbolsImage()

Load the assembly debug symbols bytes into memory if present. This will return null if debug symbols are not available for the source assembly.

Declaration

```
public virtual byte[] LoadDebugSymbolsImage()
```

Returns

TYPE	DESCRIPTION
byte[]	

Implements

ICompilationReference

Class ScriptDomain

A script domain stores a collection of assemblies that have been loaded or compiled, and provides API's for loading and compiling from a variety of input sources. Note that a domain is used mainly for organisation purposes and it is possible to have multiple script domains per project if required. This does not represent an app domain although it provides access to the current domain via AppDomain property. Support for separate app domains is not currently supported/feasable with Unity/Mono.

Inheritance

object

ScriptDomain

Implements

IDisposable

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class ScriptDomain : IDisposable
```

Constructors

ScriptDomain(AppDomain, bool)

Create a new script domain instance, used to make load or compile requests.

Declaration

```
public ScriptDomain(AppDomain sandbox = null, bool keepDomainAlive = false)
```

Parameters

TYPE	NAME	DESCRIPTION
AppDomain	sandbox	The optional app domain used to load compiled code. Requires all dependencies to be loaded ahead of time
bool	keepDomainAlive	Should the provided app domain be unloaded when this object is disposed, or should it be kept alive

Fields

IsIL2CPPBackend

Used to check whether the current backend is IL2CPP or not. If false then we can assume Mono-Jit backend.

Declaration

```
public static readonly bool IsIL2CPPBackend
```

Field Value

TYPE	DESCRIPTION
bool	

Properties

AppDomain

Get the app domain associated with this script domain. Currently just returns the Unity created domain.

Declaration

```
public AppDomain AppDomain { get; }
```

Property Value

TYPE	DESCRIPTION
AppDomain	

Assemblies

Get all ScriptAssembly that are currently loaded into this domain. This will include any assembly that was loaded or compiled from this domain.

Declaration

```
public ScriptAssembly[] Assemblies { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptAssembly[]	

AssemblyLoader

Get the assembly loader for this script domain.

Declaration

```
public IAssemblyLoader AssemblyLoader { get; }
```

Property Value

TYPE	DESCRIPTION
IAssemblyLoader	

CompiledAssemblies

Get only ScriptAssembly that have been compiled into this domain.

Declaration

```
public ScriptAssembly[] CompiledAssemblies { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptAssembly[]	

EnumerateAssemblies

Enumerate all ScriptAssembly that are currently loaded into this domain. This will include any assembly that was loaded or compiled from this domain.

Declaration

```
public IEnumerable<ScriptAssembly> EnumerateAssemblies { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ScriptAssembly>	

EnumerateCompiledAssemblies

Enumerate only ScriptAssembly that have been compiled into this domain.

Declaration

```
public IEnumerable<ScriptAssembly> EnumerateCompiledAssemblies { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ScriptAssembly>	

IsDisposed

Check if this domain has been disposed.

Declaration

```
public bool IsDisposed { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Methods

CompileAndLoadDirectory(string, SearchOption, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadDirectory(string directoryPath, SearchOption searchOption, CompileOptions
options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadDirectory(string, SearchOption, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadDirectory(string directoryPath, SearchOption searchOption, out
CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadDirectoryAsync(string, SearchOption, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadDirectoryAsync(string directoryPath, SearchOption
searchOption, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadFile(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadFile(string cSharpFile, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadFile(string, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadFile(string cSharpFile, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadFileAsync(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadFileAsync(string cSharpFile, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadFiles(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadFiles(string[] cSharpFiles, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadFiles(string[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadFiles(string[] cSharpFiles, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileResult	compileResult	

TYPE	NAME	DESCRIPTION
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadFilesAsync(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadFilesAsync(string[] cSharpFiles, CompileOptions options =
null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadMainDirectory(string, SearchOption, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainDirectory(string directoryPath, SearchOption searchOption, CompileOptions
options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainDirectory(string, SearchOption, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainDirectory(string directoryPath, SearchOption searchOption, out
CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainDirectoryAsync(string, SearchOption, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainDirectoryAsync(string directoryPath, SearchOption
searchOption, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainFile(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainFile(string cSharpFile, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainFile(string, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainFile(string cSharpFile, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainFileAsync(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainFileAsync(string cSharpFile, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainFiles(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainFiles(string[] cSharpFiles, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainFiles(string[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainFiles(string[] cSharpFiles, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileResult	compileResult	

TYPE	NAME	DESCRIPTION
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainFilesAsync(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainFilesAsync(string[] cSharpFiles, CompileOptions options =
null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainProject(CSharpProject, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainProject(CSharpProject project, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainProject(CSharpProject, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainProject(CSharpProject project, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSource(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSource(string cSharpSource, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSource(string, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSource(string cSharpSource, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSourceAsync(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainSourceAsync(string cSharpSource, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainSources(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSources(string[] cSharpSources, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSources(string[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSources(string[] cSharpSources, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSourcesAsync(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainSourcesAsync(string[] cSharpSources, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileOptions	options	

TYPE	NAME	DESCRIPTION
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainSyntaxTree(SyntaxTree, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSyntaxTree(SyntaxTree syntaxTree, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSyntaxTree(SyntaxTree, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSyntaxTree(SyntaxTree syntaxTree, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSyntaxTreeAsync(SyntaxTree, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainSyntaxTreeAsync(SyntaxTree syntaxTree, CompileOptions
options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadMainSyntaxTrees(SyntaxTree[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSyntaxTrees(SyntaxTree[] syntaxTrees, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSyntaxTrees(SyntaxTree[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptType CompileAndLoadMainSyntaxTrees(SyntaxTree[] syntaxTrees, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptType	

CompileAndLoadMainSyntaxTreesAsync(SyntaxTree[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptType> CompileAndLoadMainSyntaxTreesAsync(SyntaxTree[] syntaxTrees, CompileOptions
options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptType>	

CompileAndLoadProject(CSharpProject, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadProject(CSharpProject project, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileOptions	options	

TYPE	NAME	DESCRIPTION
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadProject(CSharpProject, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadProject(CSharpProject project, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadProjectAsync(CSharpProject, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadProjectAsync(CSharpProject project, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadSource(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSource(string cSharpSource, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSource(string, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSource(string cSharpSource, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSourceAsync(string, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadSourceAsync(string cSharpSource, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadSources(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSources(string[] cSharpSources, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSources(string[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSources(string[] cSharpSources, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileResult	compileResult	

TYPE	NAME	DESCRIPTION
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSourcesAsync(string[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadSourcesAsync(string[] cSharpSources, CompileOptions options
= null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadSyntaxTree(SyntaxTree, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSyntaxTree(SyntaxTree syntaxTree, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSyntaxTree(SyntaxTree, out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSyntaxTree(SyntaxTree syntaxTree, out CompileResult compileResult, out CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSyntaxTreeAsync(SyntaxTree, CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadSyntaxTreeAsync(SyntaxTree syntaxTree, CompileOptions options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileAndLoadSyntaxTrees(SyntaxTree[], CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSyntaxTrees(SyntaxTree[] syntaxTrees, CompileOptions options = null,
ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSyntaxTrees(SyntaxTree[], out CompileResult, out CodeSecurityReport, CompileOptions, ScriptSecurityMode)

Declaration

```
public ScriptAssembly CompileAndLoadSyntaxTrees(SyntaxTree[] syntaxTrees, out CompileResult compileResult, out
CodeSecurityReport securityReport, CompileOptions options = null, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileResult	compileResult	
CodeSecurityReport	securityReport	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

CompileAndLoadSyntaxTreesAsync(SyntaxTree[], CompileOptions, ScriptSecurityMode)

Declaration

```
public CompileAsync<ScriptAssembly> CompileAndLoadSyntaxTreesAsync(SyntaxTree[] syntaxTrees, CompileOptions
options = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	
ScriptSecurityMode	securityMode	

Returns

TYPE	DESCRIPTION
CompileAsync<ScriptAssembly>	

CompileDirectory(string, SearchOption, CompileOptions)

Declaration

```
public CompileResult CompileDirectory(string directoryPath, SearchOption searchOption, CompileOptions options
= null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileResult	

CompileDirectoryAsync(string, SearchOption, CompileOptions)

Declaration

```
public CompileAsync CompileDirectoryAsync(string directoryPath, SearchOption searchOption, CompileOptions
options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	directoryPath	
SearchOption	searchOption	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

CompileFile(string, CompileOptions)

Compile the specified C# source file. Use CompileFiles(string[], CompileOptions) if you need to compile multiple file paths as a batch.

Declaration

```
public CompileResult CompileFile(string cSharpFile, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	The C# file path to compile (.cs usually)
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileResult	The CompileResult for the compile request

Exceptions

TYPE	CONDITION
ArgumentException	File path is null or empty

CompileFileAsync(string, CompileOptions)

Declaration

```
public CompileAsync CompileFileAsync(string cSharpFile, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpFile	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

CompileFiles(string[], CompileOptions)

Compile the specified C# source files as a batch.

Declaration

```
public CompileResult CompileFiles(string[] cSharpFiles, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	The C# file paths to compile (.cs usually)
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileResult	The CompileResult for the compile request

Exceptions

TYPE	CONDITION
ArgumentNullException	File path array is null
ArgumentException	File path array is empty

CompileFilesAsync(string[], CompileOptions)

Declaration

```
public CompileAsync CompileFilesAsync(string[] cSharpFiles, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpFiles	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

CompileProject(CSharpProject, CompileOptions)

Declaration

```
public CompileResult CompileProject(CSharpProject project, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileResult	

CompileProjectAsync(CSharpProject, CompileOptions)

Declaration

```
public CompileAsync CompileProjectAsync(CSharpProject project, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
CSharpProject	project	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

CompileSource(string, CompileOptions)

Compile the specified C# source code string. Use CompileSources(string[], CompileOptions) if you need to compile multiple strings as a batch.

Declaration

```
public CompileResult CompileSource(string cSharpSource, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	The C# source code string to compile
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileResult	The CompileResult for the compile request

Exceptions

TYPE	CONDITION
ArgumentException	Source string is null or empty
ArgumentNullException	Compile options are null

CompileSourceAsync(string, CompileOptions)

Compile the specified C# source code string asynchronously.

Declaration

```
public CompileAsync CompileSourceAsync(string cSharpSource, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpSource	The C# source code string to compile
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileAsync	An operation that can be awaited in a coroutine or async await context and provides access to the CompileResult

CompileSources(string[], CompileOptions)

Compile the specified C# source code strings as a batch.

Declaration

```
public CompileResult CompileSources(string[] cSharpSources, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	An array of C# source code strings to compile

TYPE	NAME	DESCRIPTION
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileResult	The CompileResult for the compile request

Exceptions

TYPE	CONDITION
ArgumentNullException	Source code array is null
ArgumentException	Source code array has a length of zero

CompileSourcesAsync(string[], CompileOptions)

Compile the specified C# source code strings as a batch asynchronously.

Declaration

```
public CompileAsync CompileSourcesAsync(string[] cSharpSources, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string[]	cSharpSources	An array of C# source code strings to compile
CompileOptions	options	The options for the compile request. Use FromSettings(LogLevel, IEnumerable<string>, IEnumerable<ICompilationReference>) to create options from settings asset

Returns

TYPE	DESCRIPTION
CompileAsync	An operation that can be awaited in a coroutine or async await context and provides access to the CompileResult

CompileSyntaxTree(SyntaxTree, CompileOptions)

Declaration

```
public CompileResult CompileSyntaxTree(SyntaxTree syntaxTree, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileResult	

CompileSyntaxTreeAsync(SyntaxTree, CompileOptions)

Declaration

```
public CompileAsync CompileSyntaxTreeAsync(SyntaxTree syntaxTree, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree	syntaxTree	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

CompileSyntaxTrees(SyntaxTree[], CompileOptions)

Declaration

```
public CompileResult CompileSyntaxTrees(SyntaxTree[] syntaxTrees, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileResult	

CompileSyntaxTreesAsync(SyntaxTree[], CompileOptions)

Declaration

```
public CompileAsync CompileSyntaxTreesAsync(SyntaxTree[] syntaxTrees, CompileOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	
CompileOptions	options	

Returns

TYPE	DESCRIPTION
CompileAsync	

Dispose()

Dispose this domain. Note that this will just clear all stored assemblies and will not actually unload the Unity app domain.

Declaration

```
public void Dispose()
```

LoadAssembly(byte[], byte[], ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly image from the provided bytes that should represent a value PE assembly image.

Declaration

```
public ScriptAssembly LoadAssembly(byte[] assemblyImage, byte[] debugSymbolsImage = null, ScriptSecurityMode
securityMode = ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
byte[]	assemblyImage	The assembly image bytes
byte[]	debugSymbolsImage	The optional debug symbols image (usually contents of .pdb file)
ScriptSecurityMode	securityMode	The security mode used to load the assembly
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptAssembly	The loaded assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentNullException	Assembly image is null

LoadAssembly(string, ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly from the specified path.

Declaration

```
public ScriptAssembly LoadAssembly(string assemblyPath, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyPath	The path of the assembly to load
ScriptSecurityMode	securityMode	The security mode used to load the assembly
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptAssembly	The loaded assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentException	The assembly path is null or empty

LoadAssemblyFromResources(string, string, ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly from a text asset using the Unity resources API.

Declaration

```
public ScriptAssembly LoadAssemblyFromResources(string assemblyResourcesPath, string debugSymbolsResourcesPath
= null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector =
null)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
string	assemblyResourcesPath	The resources path where the assembly text asset is located
string	debugSymbolsResourcesPath	The optional resources path of the debug symbols file if applicable (.pdb usually)
ScriptSecurityMode	securityMode	The security mode to use when loading the assembly
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptAssembly	The loaded assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentException	Assembly resources path is null or empty
DllNotFoundException	Could not find the text asset or load the dll at the specified path

LoadMainAssembly(byte[], byte[], ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly image from the provided bytes that should represent a value PE assembly image.

Declaration

```
public ScriptType LoadMainAssembly(byte[] assemblyImage, byte[] debugSymbolsImage = null, ScriptSecurityMode
securityMode = ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
byte[]	assemblyImage	The assembly image bytes
byte[]	debugSymbolsImage	The optional debug symbols image (usually contents of .pdb file)
ScriptSecurityMode	securityMode	The security mode used to load the assembly

TYPE	NAME	DESCRIPTION
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptType	The loaded main type for the compiled assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentNullException	Assembly image is null

LoadMainAssembly(string, ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly from the specified path.

Declaration

```
public ScriptType LoadMainAssembly(string assemblyPath, ScriptSecurityMode securityMode =
ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyPath	The path of the assembly to load
ScriptSecurityMode	securityMode	The security mode used to load the assembly
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptType	The loaded main type for the compiled assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentException	The assembly path is null or empty

LoadMainAssemblyFromResources(string, string, ScriptSecurityMode, IMainTypeSelector)

Load a managed assembly from a text asset using the Unity resources API.

Declaration

```
public ScriptType LoadMainAssemblyFromResources(string assemblyResourcesPath, string debugSymbolsResourcesPath = null, ScriptSecurityMode securityMode = ScriptSecurityMode.UseSettings, IMainTypeSelector mainTypeSelector = null)
```

Parameters

TYPE	NAME	DESCRIPTION
string	assemblyResourcesPath	The resources path where the assembly text asset is located
string	debugSymbolsResourcesPath	The optional resources path of the debug symbols file if applicable (.pdb usually)
ScriptSecurityMode	securityMode	The security mode to use when loading the assembly
IMainTypeSelector	mainTypeSelector	The optional main type selector used to determine which type in the assembly is considered to be the main type

Returns

TYPE	DESCRIPTION
ScriptType	The loaded main type for the compiled assembly or null if an error occurred

Exceptions

TYPE	CONDITION
ArgumentException	Assembly resources path is null or empty
DllNotFoundException	Could not find the text asset or load the dll at the specified path

RegisterAssembly(ScriptAssembly)

Declaration

```
public ScriptAssembly RegisterAssembly(ScriptAssembly asm)
```

Parameters

TYPE	NAME	DESCRIPTION
ScriptAssembly	asm	

Returns

TYPE	DESCRIPTION
ScriptAssembly	

SecurityCheckAssembly(AssemblySource, ScriptSecurityMode, out CodeSecurityReport, ICodeSecurityValidator)

Run code security checks on the specified AssemblySource to determine whether it passes code security validation. Note that security checks will be skipped if EnsureLoad is passed.

Declaration

```
public bool SecurityCheckAssembly(AssemblySource source, ScriptSecurityMode securityMode, out
CodeSecurityReport securityReport, ICodeSecurityValidator securityValidator = null)
```

Parameters

TYPE	NAME	DESCRIPTION
AssemblySource	source	The assembly source to check
ScriptSecurityMode	securityMode	The security mode used for the check
CodeSecurityReport	securityReport	The generated code security report if security validation did run
ICodeSecurityValidator	securityValidator	An optional Trivial.CodeSecurity.ICodeSecurityValidator to use in place of the default code validation rules

Returns

TYPE	DESCRIPTION
bool	True if the assembly passed code security verification or false if not

Implements

IDisposable

Class ScriptFields

Represents a collection of fields defined on a type or instance. Used for quickly accessing field values with caching.

Inheritance

object

ScriptFields

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class ScriptFields
```

Properties

this[string]

Get oor set the value of a field by name.

Declaration

```
public object this[string name] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the field to access

Property Value

TYPE	DESCRIPTION
object	The value of the field with the specified name

Methods

Get<T>(string)

Get the value of a field with the specified name as the provided generic type.

Declaration

```
public T Get<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the field to get

Returns

TYPE	DESCRIPTION
T	The field value as the specified generic type

Type Parameters

NAME	DESCRIPTION
T	The generic type to get the field value as

Exceptions

TYPE	CONDITION
TargetException	Could not convert field value to specified generic type

Set<T>(string, T)

Set the value of a field with the specified name.

Declaration

```
public void Set<T>(string name, T value)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the field to set
T	value	The value to assign to the field

Type Parameters

NAME	DESCRIPTION
T	The generic type to set the field value as

Class ScriptMethods

Represents a collection of methods defined on a type or instance. Used for calling methods or coroutines with caching.

Inheritance

object

ScriptMethods

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class ScriptMethods
```

Methods

Call(string, params object[])

Call a method with the specified name and arguments.

Declaration

```
public object Call(string name, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the method to call
object[]	args	The optional arguments for the method

Returns

TYPE	DESCRIPTION
object	The method return value or null if the method does not return anything

CallCoroutine(string, params object[])

Call a method with the specified name as a coroutine. Note that the method must be an instance method defined on a MonoBehaviour derived type. Not supported for static methods since coroutines implicitly need an associated behaviour component to run and manage the coroutine.

Declaration

```
public Coroutine CallCoroutine(string name, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the method to call

TYPE	NAME	DESCRIPTION
object[]	args	The optional arguments for the method

Returns

TYPE	DESCRIPTION
Coroutine	A coroutine that has been started

Exceptions

TYPE	CONDITION
InvalidOperationException	Method is not a coroutine (Does not return IEnumerator) or is defined as static or on a type not deriving from MonoBehaviour

Call<T>(string, params object[])

Call a method with the specified name and arguments, and returns the value as the specified generic type.

Declaration

```
public T Call<T>(string name, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the method to call
object[]	args	The optional arguments for the method

Returns

TYPE	DESCRIPTION
T	The method return value or null if the method does not return anything

Type Parameters

NAME	DESCRIPTION
T	The generic type to get the return value as

Exceptions

TYPE	CONDITION
TargetException	Could not convert the return value of the called method to the specified generic type

Class ScriptProperties

Represents a collection of properties defined on a type or instance. Used for quickly accessing property values with caching.

Inheritance

object

ScriptProperties

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class ScriptProperties
```

Properties

this[string]

Get or set the value of a property by name.

Declaration

```
public object this[string name] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the property to access

Property Value

TYPE	DESCRIPTION
object	The value of the property with the specified name

Methods

Get<T>(string)

Get the value of a property with the specified name as the provided generic type.

Declaration

```
public T Get<T>(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the property to get

Returns

TYPE	DESCRIPTION
T	The property value as the specified generic type

Type Parameters

NAME	DESCRIPTION
T	The generic type to get the property value as

Exceptions

TYPE	CONDITION
TargetException	Could not convert property value to specified generic type

Set<T>(string, T)

Set the value of a property with the specified name.

Declaration

```
public void Set<T>(string name, T value)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the property to set
T	value	The value to assign to the property

Type Parameters

NAME	DESCRIPTION
T	The generic type to set the property value as

Class ScriptProxy

Represents an instance of a type loaded and instantiated in a script domain.

Inheritance

object
ScriptProxy

Implements

IDisposable

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public abstract class ScriptProxy : IDisposable
```

Constructors

ScriptProxy()

Create a new instance.

Declaration

```
protected ScriptProxy()
```

Properties

ComponentInstance

Returns the wrapped instance as UnityEngine.Component if it is derived from, or null if not.

Declaration

```
public virtual Component ComponentInstance { get; }
```

Property Value

TYPE	DESCRIPTION
Component	

Fields

Fields

Quick access to the fields defined on this instance. Note that static fields should be accessed via the StaticFields.

Declaration

```
public ScriptFields Fields { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptFields	

Instance

Get the raw wrapped instance.

Declaration

```
public abstract object Instance { get; }
```

Property Value

TYPE	DESCRIPTION
object	

IsComponent

Returns true if the wrapped instance is derived from UnityEngine.Component.

Declaration

```
public virtual bool IsComponent { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsDisposed

Check if this instance has been disposed.

Declaration

```
public bool IsDisposed { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsMonoBehaviour

Returns true if the wrapped instance is derived from UnityEngine.MonoBehaviour.

Declaration

```
public virtual bool IsMonoBehaviour { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsScriptableObject

Returns true if the wrapped instance is derived from UnityEngine.ScriptableObject.

Declaration

```
public virtual bool IsScriptableObject { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsUnityObject

Returns true if the wrapped instance is derived from UnityEngine.Object.

Declaration

```
public virtual bool IsUnityObject { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Methods

Quick access to the methods defined on this instance. Note that static methods should be accessed via the StaticMethods.

Declaration

```
public ScriptMethods Methods { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptMethods	

MonoBehaviourInstance

Returns the wrapped instance as UnityEngine.MonoBehaviour if it derived from, or null if not.

Declaration

```
public virtual MonoBehaviour MonoBehaviourInstance { get; }
```

Property Value

TYPE	DESCRIPTION
MonoBehaviour	

Properties

Quick access to the properties defined on this instance. Note that static properties should be accessed via the StaticProperties.

Declaration

```
public ScriptProperties Properties { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptProperties	

ScriptType

Get the ScriptType of the wrapped instance.

Declaration

```
public abstract ScriptType ScriptType { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptType	

ScriptableObjectInstance

Returns the wrapped instance as UnityEngine.ScriptableObject if it derived from, or null if not.

Declaration

```
public virtual ScriptableObject ScriptableObjectInstance { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptableObject	

UnityInstance

Returns the wrapped instance as UnityEngine.Object if it derived from, or null if not.

Declaration

```
public virtual Object UnityInstance { get; }
```

Property Value

TYPE	DESCRIPTION
Object	

Methods

CheckDisposed()

Check if this object is disposed and throw an exception if that is the case.

Declaration

```
protected void CheckDisposed()
```

Exceptions

TYPE	CONDITION
ObjectDisposedException	This object has already been disposed

Dispose()

Dispose of this object and the managed instance. This will cause UnityEngine.Object instances to be destroyed, and for `Dispose`

method to be called if the type implements IDisposable

Declaration

```
public void Dispose()
```

GetInstanceAs<T>(bool, T)

Attempts to get the managed instance as the specified generic type.

Declaration

```
public virtual T GetInstanceAs<T>(bool throwOnError, T errorValue = default)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	throwOnError	When false, any exceptions caused by the conversion will be caught and will result in a default value being returned. When true, any exceptions will not be handled.
T	errorValue	The value to return when 'throwOnError' is false and an error occurs"/>

Returns

TYPE	DESCRIPTION
T	The managed instance as the specified generic type or the default value for the generic type if an error occurred

Type Parameters

NAME	DESCRIPTION
T	The generic type to return the instance as

GetInstanceType()

Get the type of the instance that is managed by this object.

Declaration

```
public virtual Type GetInstanceType()
```

Returns

TYPE	DESCRIPTION
Type	The type of this wrapped instance

Exceptions

TYPE	CONDITION
InvalidOperationException	Instance is null or has been destroyed

IsInstanceOf<T>()

Checks whether this is an instance of the specified generic type.

Declaration

```
public virtual bool IsInstanceOf<T>()
```

Returns

TYPE	DESCRIPTION
bool	True if the managed instance is assignable from the specified generic type, or false if not

Type Parameters

NAME	DESCRIPTION
T	The generic type to check for an instance of

MakePersistent()

If the wrapped instance is a Unity type then this method will call 'Don'tDestroyOnLoad' to ensure that the object is able to survive scene loads.

Declaration

```
public virtual void MakePersistent()
```

OnDispose()

Dispose of this object and the managed instance. This will cause UnityEngine.Object instances to be destroyed, and for `Dispose` method to be called if the type implements IDisposable

Declaration

```
protected virtual void OnDispose()
```

Implements

IDisposable

Enum ScriptSecurityMode

The security mode used when loading and compiling assemblies.

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public enum ScriptSecurityMode
```

Fields

NAME	DESCRIPTION
EnsureLoad	Security verification will be skipped.
EnsureSecurity	Security verification will be used.
UseSettings	Use the security mode from the Roslyn C# settings window.

Class ScriptType

Represents a specific type loaded into a script domain.

Inheritance

object

ScriptType

Namespace: RoslynCSharp

Assembly: RoslynCSharp.dll

Syntax

```
public abstract class ScriptType
```

Constructors

ScriptType(ScriptAssembly, ScriptType)

Create a new instance.

Declaration

```
protected ScriptType(ScriptAssembly assembly, ScriptType parent)
```

Parameters

TYPE	NAME	DESCRIPTION
ScriptAssembly	assembly	The declaring assembly
ScriptType	parent	The optional parent type if this type is nested

Properties

Assembly

Get the script assembly that this type is defined in.

Declaration

```
public ScriptAssembly Assembly { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptAssembly	

FullName

Get the full name of the type including the namespace name if applicable.

Declaration

```
public virtual string FullName { get; }
```

Property Value

TYPE	DESCRIPTION
string	

IsComponent

Return a value indicating whether this type is derived from UnityEngine.Component.

Declaration

```
public bool IsComponent { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsMonoBehaviour

Return a value indicating whether this type is derived from UnityEngine.MonoBehaviour.

Declaration

```
public bool IsMonoBehaviour { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsNested

Return a value indicating whether the type is a nested type defined within another type.

Declaration

```
public virtual bool IsNested { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsPublic

Return a value indicating whether the type is public.

Declaration

```
public virtual bool IsPublic { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsScriptableObject

Return a value indicating whether this type is derived from UnityEngine.ScriptableObject.

Declaration

```
public bool IsScriptableObject { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

IsUnityObject

Return a value indicating whether this type is derived from UnityEngine.Object.

Declaration

```
public bool IsUnityObject { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Name

Get the name of the type without namespace.

Declaration

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

Property Value

TYPE	DESCRIPTION
string	

Namespace

Get the namespace name of the type.

Declaration

```
public virtual string Namespace { get; }
```

Property Value

TYPE	DESCRIPTION
string	

Parent

Get the parent type if this is a nested type.

Declaration

```
public ScriptType Parent { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptType	

StaticFields

Quick access to the static fields defined on this type.

Declaration

```
public ScriptFields StaticFields { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptFields	

StaticMethods

Quick access to the static methods defined on this type.

Declaration

```
public ScriptMethods StaticMethods { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptMethods	

StaticProperties

Quick access to the static properties defined on this type.

Declaration

```
public ScriptProperties StaticProperties { get; }
```

Property Value

TYPE	DESCRIPTION
ScriptProperties	

SystemType

Get the Type represented by this object.

Declaration

```
public abstract Type SystemType { get; }
```

Property Value

TYPE	DESCRIPTION
Type	

Methods

CreateInstance(GameObject, params object[])

Create a proxy instance of this type. A proxy instance is an object which wraps the instance and provides a quick and easy way to access members and perform reflection. Note that this will construct the type using the correct method. For example: MonoBehaviour will use `AddComponent`, ScriptableObject will use `CreateInstance`, and other types will use the best matching or default constructor.

Declaration

```
public ScriptProxy CreateInstance(GameObject parent = null, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	parent	The optional game object parent which is required if the type is a MonoBehaviour
object[]	args	The optional arguments for the target constructor if the type is a non-Unity object

Returns

TYPE	DESCRIPTION
ScriptProxy	A ScriptProxy object for the newly created instance

Exceptions

TYPE	CONDITION
InvalidOperationException	Parent is null but the type is derived from MonoBehaviour

CreateInstanceAs(GameObject, params object[])

Create a new instance of this type and return the raw instance. Note that this will construct the type using the correct method. For example: MonoBehaviour will use `AddComponent`, ScriptableObject will use `CreateInstance`, and other types will use the best matching or default constructor.

Declaration

```
public object CreateInstanceAs(GameObject parent = null, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	parent	The optional game object parent which is required if the type is a MonoBehaviour

TYPE	NAME	DESCRIPTION
object[]	args	The optional arguments for the target constructor if the type is a non-Unity object

Returns

TYPE	DESCRIPTION
object	The raw instance

Exceptions

TYPE	CONDITION
InvalidOperationException	Parent is null but the type is derived from MonoBehaviour

CreateInstanceAs<T>(GameObject, params object[])

Create a new instance of this type and return the object as the specified generic type, base type or interface. Return value will be null if the type is not convertible to the specified generic type. Note that this will construct the type using the correct method. For example: MonoBehaviour will use `AddComponent`, ScriptableObject will use `CreateInstance`, and other types will use the best matching or default constructor.

Declaration

```
public T CreateInstanceAs<T>(GameObject parent = null, params object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	parent	The optional game object parent which is required if the type is a MonoBehaviour
object[]	args	The optional arguments for the target constructor if the type is a non-Unity object

Returns

TYPE	DESCRIPTION
T	The instance as T or null if the instance could bnot be converted to T

Type Parameters

NAME	DESCRIPTION
T	The generic type to create the instance as

Exceptions

TYPE	CONDITION
InvalidOperationException	Parent is null but the type is derived from MonoBehaviour

CreateInstanceComponentImpl(GameObject)

Create a new instance of a component on the specified game object. This will create this instance using AddComponent(Type).

Declaration

```
protected abstract ScriptProxy CreateInstanceComponentImpl(GameObject parent)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	parent	The game object to attach the component to

Returns

TYPE	DESCRIPTION
ScriptProxy	

CreateInstanceImpl(object[])

Create a new instance of a type using the constructor appropriate to the provided arguments.

Declaration

```
protected abstract ScriptProxy CreateInstanceImpl(object[] args)
```

Parameters

TYPE	NAME	DESCRIPTION
object[]	args	The optional arguments of the constructor

Returns

TYPE	DESCRIPTION
ScriptProxy	

CreateInstanceScriptableObjectImpl()

Create a new instance of a scriptable object. This will create the instance using CreateInstance(Type).

Declaration

```
protected abstract ScriptProxy CreateInstanceScriptableObjectImpl()
```

Returns

TYPE	DESCRIPTION
ScriptProxy	

IsSubTypeOf(Type)

Return a value indicating whether this type is derived from the specified type.

Declaration

```
public virtual bool IsSubTypeOf(Type subType)
```

Parameters

TYPE	NAME	DESCRIPTION
Type	subType	The type to check

Returns

TYPE	DESCRIPTION
bool	

IsSubTypeOf<T>()

Return a value indicating whether this type is derived from the specified generic type.

Declaration

```
public virtual bool IsSubTypeOf<T>()
```

Returns

TYPE	DESCRIPTION
bool	

Type Parameters

NAME	DESCRIPTION
T	The generic type to check

Namespace RoslynCSharp.CodeSecurity

Classes

CodeSecurityAssemblyRestriction

CodeSecurityRestrictionValidator

DefaultCodeValidator

Provides the default code security validation behaviour with the option to override any checks and fallback to defaults for others. Useful if you are only interested in restricting certain types for example, but want to retain the default behaviour for other aspects.

Class CodeSecurityAssemblyRestriction

Inheritance

object
AssemblyRestriction
CodeSecurityAssemblyRestriction

Inherited Members

AssemblyRestriction.Name
AssemblyRestriction.NamedTypeRestrictions
AssemblyRestriction.RootTypeRestrictions
AssemblyRestriction.Allowed
AssemblyRestriction.ClearCache()
AssemblyRestriction.ToString()
AssemblyRestriction.IsNamespaceAllowed(TypeReference, RestrictionAllow)
AssemblyRestriction.IsTypeAllowed(TypeReference, RestrictionAllow)
AssemblyRestriction.Sort()
AssemblyRestriction.GetAllowedNamespacesAndTypesCombined(out int, out int)
AssemblyRestriction.SetAllowedNamespacesAndTypesCombined(RestrictionAllow)
AssemblyRestriction.SetNamespaceAllowed(string, RestrictionAllow, bool)
AssemblyRestriction.SetTypeAllowed(string, RestrictionAllow)
AssemblyRestriction.FromRuntimeAssembly<T>()
AssemblyRestriction.HasTypes

Namespace: RoslynCSharp.CodeSecurity

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public sealed class CodeSecurityAssemblyRestriction : AssemblyRestriction
```

Constructors

CodeSecurityAssemblyRestriction()

Create a new instance.

Declaration

```
public CodeSecurityAssemblyRestriction()
```

CodeSecurityAssemblyRestriction(Assembly, RestrictionAllow)

Create a new instance from the assembly.

Declaration

```
public CodeSecurityAssemblyRestriction(Assembly assembly, RestrictionAllow allowed = RestrictionAllow.Allow)
```

Parameters

TYPE	NAME	DESCRIPTION
Assembly	assembly	The assembly to create from

TYPE	NAME	DESCRIPTION
RestrictionAllow	allowed	Are the members allowed

Properties

LastWriteTimeUTC

Get the last time the assembly was compiled in UTC file time.

Declaration

```
public long LastWriteTimeUTC { get; }
```

Property Value

TYPE	DESCRIPTION
long	

ReferenceAssemblyAsset

The optional assembly reference asset used to auto update

Declaration

```
public AssemblyReferenceAsset ReferenceAssemblyAsset { get; }
```

Property Value

TYPE	DESCRIPTION
AssemblyReferenceAsset	

Class CodeSecurityRestrictionValidator

Inheritance

object

RestrictionValidator<CodeSecurityAssemblyRestriction>

CodeSecurityRestrictionValidator

Implements

ICodeSecurityValidator

Inherited Members

RestrictionValidator<CodeSecurityAssemblyRestriction>.RuntimeAssemblyRestrictions
RestrictionValidator<CodeSecurityAssemblyRestriction>.AssemblyRestrictions
RestrictionValidator<CodeSecurityAssemblyRestriction>.AllowPInvoke
RestrictionValidator<CodeSecurityAssemblyRestriction>.AllowUnsafe
RestrictionValidator<CodeSecurityAssemblyRestriction>.ClearCache()
RestrictionValidator<CodeSecurityAssemblyRestriction>.Sort()
RestrictionValidator<CodeSecurityAssemblyRestriction>.IsAssemblyReferenceAllowed(AssemblyReference)
RestrictionValidator<CodeSecurityAssemblyRestriction>.IsNamespaceReferenceAllowed(TypeReference)
RestrictionValidator<CodeSecurityAssemblyRestriction>.IsTypeReferenceAllowed(TypeReference)
RestrictionValidator<CodeSecurityAssemblyRestriction>.IsMemberReferenceAllowed(MemberReference)
RestrictionValidator<CodeSecurityAssemblyRestriction>.SetNamespaceAllowed(string, RestrictionAllow, bool)
RestrictionValidator<CodeSecurityAssemblyRestriction>.SetTypeAllowed(string, RestrictionAllow)
RestrictionValidator<CodeSecurityAssemblyRestriction>.RuntimeAndAssemblyRestrictions

Namespace: RoslynCSharp.CodeSecurity

Assembly: RoslynCSharp.dll

Syntax

```
[Serializable]
public sealed class CodeSecurityRestrictionValidator : RestrictionValidator<CodeSecurityAssemblyRestriction>,
ICodeSecurityValidator
```

Methods

CreatedDefaultRestrictions()

Declaration

```
public static CodeSecurityRestrictionValidator CreatedDefaultRestrictions()
```

Returns

TYPE	DESCRIPTION
CodeSecurityRestrictionValidator	

Implements

Trivial.CodeSecurity.ICodeSecurityValidator

Class DefaultCodeValidator

Provides the default code security validation behaviour with the option to override any checks and fallback to defaults for others. Useful if you are only interested in restricting certain types for example, but want to retain the default behaviour for other aspects.

Inheritance

object

DefaultCodeValidator

Implements

ICodeSecurityValidator

Namespace: RoslynCSharp.CodeSecurity

Assembly: RoslynCSharp.dll

Syntax

```
public class DefaultCodeValidator : ICodeSecurityValidator
```

Properties

AllowPInvoke

Should platform invoke feature be allowed in security checked code to call native code.

Declaration

```
public virtual bool AllowPInvoke { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

AllowUnsafeCode

Should unsafe code be allowed in security checked code to use things like unsafe contexts, fixed buffers, and pointers.

Declaration

```
public virtual bool AllowUnsafeCode { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Methods

IsAssemblyReferenceAllowed(AssemblyReference)

Check whether the specified assembly reference is allowed to be referenced.

Declaration

```
public virtual bool IsAssemblyReferenceAllowed(AssemblyReference assemblyReference)
```

Parameters

TYPE	NAME	DESCRIPTION
AssemblyReference	assemblyReference	The assembly reference to check

Returns

TYPE	DESCRIPTION
bool	True if the reference is allowed of false if it is considered illegal

IsMemberReferenceAllowed(MemberReference)

Check whether the specified member reference is allowed to be referenced.

Declaration

```
public virtual bool IsMemberReferenceAllowed(MemberReference memberReference)
```

Parameters

TYPE	NAME	DESCRIPTION
MemberReference	memberReference	The member reference to check which could be a field, property, method or event reference

Returns

TYPE	DESCRIPTION
bool	True if the reference is allowed of false if it is considered illegal

IsNamespaceReferenceAllowed(TypeReference)

Check whether the specified named type reference is allowed to be referenced. Note that a type reference is passed here but only the Namespace property should be considered and the type reference is just for context.

Declaration

```
public virtual bool IsNamespaceReferenceAllowed(TypeReference namedTypeReference)
```

Parameters

TYPE	NAME	DESCRIPTION
TypeReference	namedTypeReference	The named type reference to check

Returns

TYPE	DESCRIPTION
bool	True if the reference is allowed of false if it is considered illegal

IsTypeReferenceAllowed(TypeReference)

Check whether the specified type reference is allowed to be referenced.

Declaration

```
public virtual bool IsTypeReferenceAllowed(TypeReference typeReference)
```

Parameters

TYPE	NAME	DESCRIPTION
TypeReference	typeReference	The type reference to check

Returns

TYPE	DESCRIPTION
bool	True if the reference is allowed of false if it is considered illegal

Implements

Trivial.CodeSecurity.ICodeSecurityValidator

Namespace RoslynCSharp.ExecutionSecurity

Classes

ExecutionSecurityProcessor

Used to post process syntax trees in order to inject execution security checks for loop statements.

Class ExecutionSecurityProcessor

Used to post process syntax trees in order to inject execution security checks for loop statements.

Inheritance

object

ExecutionSecurityProcessor

Implements

IParserProcessor

Namespace: RoslynCSharp.ExecutionSecurity

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class ExecutionSecurityProcessor : IParserProcessor
```

Constructors

ExecutionSecurityProcessor(ExecutionSecuritySettings)

Create a new instance.

Declaration

```
public ExecutionSecurityProcessor(ExecutionSecuritySettings securitySettings)
```

Parameters

TYPE	NAME	DESCRIPTION
ExecutionSecuritySettings	securitySettings	The execution security settings to use

Properties

Priority

The priority of this processor. Should run quite late if not last

Declaration

```
public int Priority { get; }
```

Property Value

TYPE	DESCRIPTION
int	

Methods

OnPostProcess(SyntaxTree[])

Called by the compilation pipeline when post processing should run on the provided syntax trees.

Declaration

```
public SyntaxTree[] OnPostProcess(SyntaxTree[] syntaxTrees)
```

Parameters

TYPE	NAME	DESCRIPTION
SyntaxTree[]	syntaxTrees	The syntax trees to postprocess

Returns

TYPE	DESCRIPTION
SyntaxTree[]	The new syntax trees with security checks injected

Implements

IParserProcessor

Namespace RoslynCSharp.Project

Classes

CSharpProject

Represents a C# project file (.csproj) that can be used for compilation.

Class CSharpProject

Represents a C# project file (.csproj) that can be used for compilation.

Inheritance

object

CSharpProject

Namespace: RoslynCSharp.Project

Assembly: RoslynCSharp.dll

Syntax

```
public sealed class CSharpProject
```

Properties

AssemblyName

The name of the output assembly.

Declaration

```
public string AssemblyName { get; set; }
```

Property Value

TYPE	DESCRIPTION
string	

AssemblyReferences

Get the names or paths of all assemblies that are referenced by this project.

Declaration

```
public IList<string> AssemblyReferences { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

DefineSymbols

Get all define symbols that are defined in this project.

Declaration

```
public IList<string> DefineSymbols { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

ParseException

Get the exception that was thrown during parsing if any.

Declaration

```
public Exception ParseException { get; }
```

Property Value

TYPE	DESCRIPTION
Exception	

ProjectReferences

Get the name of all other CSharp projects that are referenced by this project.

Declaration

```
public IList<string> ProjectReferences { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

Sources

Get all C# source files that are defined in this project.

Declaration

```
public IList<string> Sources { get; }
```

Property Value

TYPE	DESCRIPTION
IList<string>	

Methods

GetAssemblyReferences()

Get all referenced in this project as an enumerable of ICompilationReference that can be used in a compile request.

Declaration

```
public IEnumerable<ICompilationReference> GetAssemblyReferences()
```

Returns

TYPE	DESCRIPTION
IEnumerable<ICompilationReference>	An enumerable of referenceable objects

Parse(string)

Create a new project parsed from the specified .csproj file path. If there is an error during loading then ParseException will be set with the load exception.

Declaration

```
public static CSharpProject Parse(string cSharpProjectFile)
```

Parameters

TYPE	NAME	DESCRIPTION
string	cSharpProjectFile	The path to the .csproj file

Returns

TYPE	DESCRIPTION
CSharpProject	The loaded C# project