CSharpEditor

1.1.2

Generated by Doxygen 1.8.18

Chapter 1

CSharpEditor: A C# source code editor with syntax highlighting, intelligent code completion and real-time compilation error checking

1.1 Introduction

CSharpEditor is a C# source code editor control for `Avalonia` applications.

This library provides a control that can be added to Avalonia windows and integrates:

- The main code editor with search/replace functions and intellisense-like code completion.
- A panel showing errors in the code.
- A panel to add/remove assembly references.
- A panel showing the save history of the file (that can persist across different sessions, if the application implements it properly).
- A panel with general settings.

The code entered in this control can also be "debugged": by entering a comment `/* Breakpoint */` before a statement in the code, execution of the code will pause at that point, and the editor will enter a debugger-like state in which the code is read-only and the value of local variables is shown in a panel on the right. A breakpoint can also be entered by clicking next to the line number where the breakpoint is to be inserted; if it is allowed in that position, the breakpoint comment will be inserted automatically.

CSharpEditor is a .NET Standard 2.1 library, and should be usable in .NET Core 3.0+ and .NET 5.0+ projects. It is released under a GPLv3 licence. You can find the full documentation for this library at the `documentation website`. A `PDF reference manual` is also available.

1.2 Getting started

First of all, you need to install the `NuGet package` in your project.

The editor control cannot be added directly to the Window in XAML code, because it requires some non-trivial initialisation; you can create a new `Editor` control using the static method `Editor.Create` and then add it to the window:

```
using CSharpEditor;
// ...
    Editor editor = await Editor.Create();
    Grid grid = this.FindControl<Grid>("EditorContainer");
    grid.Children.Add(editor);
```

The first time an `Editor` control is added to your window may take some time to initialise; subsequent `Editor` controls will be created much faster.

The `Editor.Create` static method has multiple parameters, all of which are optional:

- `string initialText`: this is simply the initial source code that is shown in the control when it is created.
- `string preSource`: the code provided in this parameter is not shown to the user in the editor, but it is prepended to the code in the editor when compiling and checking for errors. This makes it possible e.g. to allow the user to edit just a single method, without having to show boilerplate class and namespace definitions in the editor.
- `string postSource`: same as `preSource`, but this code is included *after* the code the user has entered in the control.
- `IEnumerable<CachedMetadataReference> references`: this parameter determines the assembly references that are used to compile the code (the user can add or remove references using the panel in the interface). If this is not provided, the control will automatically build a list of references based on the assemblies that are loaded in the current `AppDomain`.
- `CSharpCompilation compilationOptions`: these are the `CompilationOptions` used to compile the code. If this is not provided, the control will use default compilation options with a `DynamicallyLinkedLibrary` output target.
- `string guid`: this parameter provides an identifier for the control. This will be used, in particular, to store the save history of the file. If the control is initialised with the same `guid` across different sessions, the save history of the file will be restored.
- `Shortcut[] additionalShortcuts`: this makes it possible to display additional application-specific shortcuts in the shortcut section of the settings panel. Note that this does not actually implement the shortcut behaviour (which needs to be implemented separately by the developer) - it is simply provided so that users can open the settings panel and see all the shortcuts that can be used with the editor in the same place.

Take a look at the `MainWindow.xaml.cs file` in the CSharpEditorDemo project to see how this works in practice.

1.3 Compiling and breakpoints

If you wish, you can use the classes and methods in the `Microsoft.CodeAnalysis.CSharp` namespace to compile the source code entered by the user. However, the `Editor` control also provides a `Compile` method that does it for you, returning an awaitable `Task` that returns a tuple of an `Assembly` and a `CSharpCompilation`. If the compilation attempt was successful, the `Assembly` will contain the compiled assembly and the `CSharp↩ Compilation` should hold no error messages; if the compilation was not successful, the `Assembly` will be `null` and the `CSharpCompilation` can be used to retrieve the compilation error messages.

The `Compile` method has two optional arguments:

- `Func<BreakpointInfo, bool> synchronousBreak`
- `Func<BreakpointInfo, Task<bool>> asynchronousBreak`

If these are provided, at any point in the code where the comment `/* Breakpoint */` is found the code will be altered to call the function that has been provided. The function will be called with a `BreakpointInfo` argument that contains information about the name and value of local variables that have been captured at the breakpoint.

The `synchronousBreak` function is called for breakpoints that happen within synchronous code; the `asynchronousBreak` function will be called for breakpoints that happen within asynchronous code. The functions should return a boolean value, indicating whether the breakpoint should be hit again if the code is executed again (think e.g. a breakpoint within a `for` or `while` loop).

If you wish to enable the default UI for breakpoints, you can just pass the `SynchronousBreak` and `AsynchronousBreak` instance methods of the `Editor` as arguments to the `Compile` method.

There is a catch, however: if synchronous code is running in the UI thread, it is not possible to handle breakpoints using the default UI, as this would cause a deadlock because the UI thread is paused waiting for the user to resume it *through the UI*, which is not possible. To prevent this, the default `SynchronousBreak` handler checks whether it has been invoked from the UI thread and, if so, does not actually enter the breakpoint.

1.4 Debugging using a separate process

A way to address the issue of synchronous breakpoints in the UI thread is to use a separate process to display the breakpoint UI. This can be achieved using the `InterprocessDebuggerServer` and `Interprocess←DebuggerClient` classes.

To use this approach, you need to create two separate processes, one for the client and one for the server.

The server process contains the main UI of the application, e.g. the `Editor` control and any associated paraphernalia. Within this project, you should create an `InterprocessDebuggerServer` object, providing it with the path to the executable of the client process. Then, when you invoke the `Compile` method to compile the code in the `Editor`, instead of invoking it with the `SynchronousBreak` and `AsynchronousBreak` methods of the `Editor`, you provide the same methods from the `InterprocessDebuggerServer` object. For example

```
using CSharpEditor;
// ...
    InterprocessDebuggerServer server = new InterprocessDebuggerServer(@"path/to/client.exe");
    Editor editor = await Editor.Create();
    // ...

    // Instead of...
    Assembly assembly = (await editor.Compile(editor.SynchronousBreak, editor.AsynchronousBreak)).Assembly;
    // Use this:
    Assembly assembly = (await editor.Compile(server.SynchronousBreak(editor),
       server.AsynchronousBreak(editor))).Assembly;
```

The client process consists of just a single window, containing an `InterprocessDebuggerClient` control that has been created by supplying it with the command-line argument with which the process has been invoked.

When the server process executes the compiled code and hits a breakpoint, the client process is notified; the server process then waits for a signal from the client process to resume the execution of the code. If the default UI is being used, this is all handled rather transparently by the `InterprocessDebuggerServer` and `Interprocess←DebuggerClient` classes.

Take a look at the `CSharpEditorIPCDemoServer` and `CSharpEditorIPCDemoClient` projects for an example of this approach.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 CSharpEditor Namespace Reference

Classes

- class BreakpointInfo

  A class to hold information about breakpoints.
- class CachedMetadataReference

  This class represents a cached MetadataReference. When an instance of this class is created using the CreateFromFile(string, string) method, we check whether a CachedMetadataReference to the same file has already been created and, in that case, return a reference to that object, rathern than creating a new one.
- class CompilationEventArgs

  A class to hold data for an event where a background compilation has completed.
- class Editor

  A C# source code editor for Avalonia.
- class InterprocessDebuggerClient

  A control that shows breakpoint information for breakpoints reached on a server process. This control contains a read-only CSharpEditor.Editor to display the code, which is reused as much as possible to reduce the initialization time.
- class InterprocessDebuggerServer

  A class used to analyse breakpoints on a separate process (to avoid deadlocks with breakpoints in synchronous code).
- class SaveEventArgs

  A class to hold data for an event where the user has requested to save the document.
- class Shortcut

  Represents a keyboard shortcut.

Enumerations

- enum SyntaxHighlightingModes { SyntaxHighlightingModes.None, SyntaxHighlightingModes.Syntactic, SyntaxHighlightingModes.Semantic }

  Represents syntax highlighting modes.

Variables

- string **propertyName**
- string bool **isProperty**

5.1.1 Enumeration Type Documentation

5.1.1.1 SyntaxHighlightingModes

```
enum CSharpEditor.SyntaxHighlightingModes  [strong]
```

Represents syntax highlighting modes.

Enumerator

None	No syntax highliting is perfomed.
Syntactic	Syntax highlighting is performed only based on syntactic information.
Semantic	Syntax highlighting is performed based on syntactic and semantic information.

Definition at line 812 of file Editor.public.cs.

Chapter 6

Class Documentation

6.1 CSharpEditor.BreakpointInfo Class Reference

A class to hold information about breakpoints.

Properties

- TextSpan BreakpointSpan `[get]`

    The location in the source code of the breakpoint, including any prepended or appended source code.
- Dictionary< string, object > LocalVariables `[get]`

    A dictionary containing the names and values of the local variables in scope at the breakpoint.

6.1.1 Detailed Description

A class to hold information about breakpoints.

Definition at line 35 of file BreakpointInfo.cs.

6.1.2 Property Documentation

6.1.2.1 BreakpointSpan

```
TextSpan CSharpEditor.BreakpointInfo.BreakpointSpan [get]
```

The location in the source code of the breakpoint, including any prepended or appended source code.

Definition at line 40 of file BreakpointInfo.cs.

6.1.2.2 LocalVariables

```
Dictionary<string, object> CSharpEditor.BreakpointInfo.LocalVariables  [get]
```

A dictionary containing the names and values of the local variables in scope at the breakpoint.

Definition at line 45 of file BreakpointInfo.cs.

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

- CSharpEditor/BreakpointInfo.cs

6.2 CSharpEditor.CachedMetadataReference Class Reference

This class represents a cached MetadataReference. When an instance of this class is created using the CreateFromFile(string, string) method, we check whether a CachedMetadataReference to the same file has already been created and, in that case, return a reference to that object, rathern than creating a new one.

Public Member Functions

- CachedMetadataReference (MetadataReference reference)

  Creates a new CachedMetadataReference wrapping the specified reference .

Static Public Member Functions

- static CachedMetadataReference CreateFromFile (string path, string xmlDocumentationPath=null)

  Creates a new CachedMetadataReference from an assembly file (optionally including the XML documentation).
- static implicit operator MetadataReference (CachedMetadataReference reference)

  Converts a CachedMetadataReference into a MetadataReference.

6.2.1 Detailed Description

This class represents a cached MetadataReference. When an instance of this class is created using the CreateFromFile(string, string) method, we check whether a CachedMetadataReference to the same file has already been created and, in that case, return a reference to that object, rathern than creating a new one.

Definition at line 28 of file CachedMetadataReference.cs.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 CachedMetadataReference()

```
CSharpEditor.CachedMetadataReference.CachedMetadataReference (
            MetadataReference reference )
```

Creates a new CachedMetadataReference wrapping the specified *reference* .

Parameters

reference	The MetadataReference wrap in a new CachedMetadataReference.

Definition at line 38 of file CachedMetadataReference.cs.

6.2.3 Member Function Documentation

6.2.3.1 CreateFromFile()

```
static CachedMetadataReference CSharpEditor.CachedMetadataReference.CreateFromFile (
            string path,
            string xmlDocumentationPath = null )  [static]
```

Creates a new CachedMetadataReference from an assembly file (optionally including the XML documentation).

Parameters

path	The path to the assembly file.
xmlDocumentationPath	The path to the XML documentation file for the assembly.

Returns

If a CachedMetadataReference has already been created from the same assembly file and the same XML documentation, a reference to the previously created object. Otherwise, a new CachedMetadataReference wrapping a MetadataReference created from the specified assembly file.

Definition at line 52 of file CachedMetadataReference.cs.

6.2.3.2 operator MetadataReference()

```
static implicit CSharpEditor.CachedMetadataReference.operator MetadataReference (
            CachedMetadataReference reference )  [static]
```

Converts a CachedMetadataReference into a MetadataReference.

Parameters

reference	The CachedMetadataReference to convert.

Definition at line 73 of file CachedMetadataReference.cs.

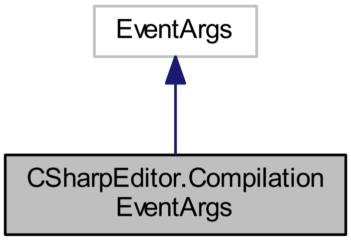
The documentation for this class was generated from the following file:

- CSharpEditor/CachedMetadataReference.cs

6.3 CSharpEditor.CompilationEventArgs Class Reference

A class to hold data for an event where a background compilation has completed.

Inheritance diagram for CSharpEditor.CompilationEventArgs:



Properties

- CSharpCompilation Compilation `[get]`

  A CSharpCompilation object containing information about the compilation that has completed, which can be used to `Emit` *an assembly, if successful.*

6.3.1 Detailed Description

A class to hold data for an event where a background compilation has completed.

Definition at line 769 of file Editor.public.cs.

6.3.2 Property Documentation

6.3.2.1 Compilation

```
CSharpCompilation CSharpEditor.CompilationEventArgs.Compilation  [get]
```

A CSharpCompilation object containing information about the compilation that has completed, which can be used to `Emit` an assembly, if successful.

Definition at line 774 of file Editor.public.cs.

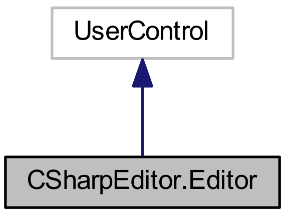
The documentation for this class was generated from the following file:

- CSharpEditor/Editor.public.cs

6.4 CSharpEditor.Editor Class Reference

A C# source code editor for Avalonia.

Inheritance diagram for CSharpEditor.Editor:



Public Types

- enum AccessTypes { AccessTypes.ReadWrite, AccessTypes.ReadOnlyWithHistoryAndErrors, AccessTypes.ReadOnly }

  Describes the actions that the user can perform on the code.

Public Member Functions

- Editor ()

  Public constructor. This is only provided for compatibility with Avalonia (`see issue #2593`*). Please use Editor.Create instead.*
- async Task SetText (string text)

  Sets the text of the document.
- async Task SetText (SourceText text)

  Sets the text of the document.
- bool SynchronousBreak (BreakpointInfo info)

  A function to handle breakpoints in synchronous methods. Pass this as an argument to Compile(Func< BreakpointInfo, bool>, Func< Brea
  To prevent deadlocks, this function will have no effect if called from the UI thread.
- async Task< bool > AsynchronousBreak (BreakpointInfo info)

  A function to handle breakpoints in asynchronous methods. Pass this as an argument to Compile(Func< BreakpointInfo, bool>, Func< Bre
- async Task<(Assembly Assembly, CSharpCompilation Compilation)> Compile (Func< BreakpointInfo, bool > synchronousBreak=null, Func< BreakpointInfo, Task< bool >> asynchronousBreak=null)

  Compile the source code to an Assembly.
- async Task< CSharpCompilation > CreateCompilation ()

  Compile the source code to a CSharpCompilation. Note that breakpoints will be disabled.
- void Save ()

  Add the current text of the document to the save history (if enabled) and invoke the SaveRequested event.

Static Public Member Functions

- static async Task< Editor > Create (string initialText="", string preSource="", string postSource="", I↩ Enumerable< CachedMetadataReference > references=null, CSharpCompilationOptions compilation↩ Options=null, string guid=null, Shortcut[ ] additionalShortcuts=null)

  Create a new Editor instance.

Public Attributes

- SyntaxHighlightingModes SyntaxHighlightingMode => this.EditorControl.SyntaxHighlightingMode

  The current syntax highlighting mode.
- bool ShowLineChanges => this.EditorControl.ShowLineChanges

  A boolean value indicating whether changed lines are highlighted on the left side of the control.
- bool ShowScrollbarOverview => this.EditorControl.ShowScrollbarOverview

  A boolean value indicating whether a summary of the changed lines, errors/warning, search results, breakpoints and the position of the caret should be shown over the vertical scrollbar.
- int AutosaveInterval => this.AutoSaver.MillisecondsInterval

  The timeout between consecutive autosaves, in milliseconds.
- int CompilationTimeout => this.CompilationErrorChecker.MillisecondsInterval

  The timeout for automatic compilation after the user stops typing, in milliseconds.

Properties

- EventHandler< EventArgs > TextChanged

  Event raised when the document text is changed.
- string PreSource `[get]`

  Source code to be prepended to the text of the document when compiling it.
- string PostSource = "" `[get]`

  Source code to be appended after the text of the document when compiling it.
- string Text `[get]`

  The source code of the document as a string.
- SourceText SourceText `[get]`

  The source code of the document as a SourceText.
- string FullSource `[get]`

  Full source code, including the PreSource, the Text, and the PostSource.
- AccessTypes AccessType `[get, set]`

  Determines whether the text of the document can be edited by the user.
- CSharpCompilationOptions CompilationOptions `[get]`

  Compilation options used to compile the source code.
- string Guid `[get]`

  A unique identifier for the document being edited.
- string SaveDirectory `[get]`

  The full path to the directory where the autosave file and the save history for the current document are kept.
- string AutoSaveFile `[get]`

  The full path to the autosave file.
- bool KeepSaveHistory = true `[get]`

  A boolean value indicating whether a history of the saved versions of the document is kept.
- bool AutoOpenSuggestions = true `[get]`

  A boolean value indicating whether the suggestion panel should open automatically while the user is typing.
- bool AutoOpenParameters = true `[get]`

A boolean value indicating whether the parameter list tooltip should open automatically while the user is typing.

- bool AutoFormat = true `[get]`

  A boolean value indicating whether the source text should be formatted automatically while the user is typing.
- ImmutableList< MetadataReference > References `[get]`

  The list of MetadataReferences for which the compiled assembly will have bindings.
- bool IsReferencesButtonEnabled `[get, set]`

  A boolean value indicating whether the button allowing the user to add or remove assembly references is enabled or not.
- TextSpan Selection `[get, set]`

  Gets or sets the selected text span.

Events

- EventHandler< SaveEventArgs > SaveRequested

  Event raised when the user uses the keyboard shortcut or pressed the button to save the document.
- EventHandler< SaveEventArgs > Autosave

  Event raised when the document is automatically saved.
- EventHandler< CompilationEventArgs > CompilationCompleted

  Event raised when a background compilation of the document completes.

6.4.1 Detailed Description

A C# source code editor for Avalonia.

Definition at line 43 of file Editor.axaml.cs.

6.4.2 Member Enumeration Documentation

6.4.2.1 AccessTypes

`enum CSharpEditor.Editor.AccessTypes [strong]`

Describes the actions that the user can perform on the code.

Enumerator

ReadWrite	The code can be edited freely.
ReadOnlyWithHistoryAndErrors	The code cannot be edited, but the list of errors and warnings is displayed, and the user can load previous versions of the file.
ReadOnly	The code can only be read. No advanced features are provided beyond syntax highlighting.

Definition at line 138 of file Editor.public.cs.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 Editor()

```
CSharpEditor.Editor.Editor ( )
```

Public constructor. This is only provided for compatibility with Avalonia (`see issue #2593`). Please use Editor.Create instead.

Definition at line 133 of file Editor.axaml.cs.

6.4.4 Member Function Documentation

6.4.4.1 AsynchronousBreak()

```
async Task<bool> CSharpEditor.Editor.AsynchronousBreak (
            BreakpointInfo info )
```

A function to handle breakpoints in asynchronous methods. Pass this as an argument to Compile(Func<BreakpointInfo, bool>, Func<

Parameters

info	A BreakpointInfo object containing information about the location of the breakpoint and the current value of local variables.

Returns

A Task that completes when code execution resumes after the breakpoint.

Definition at line 452 of file Editor.public.cs.

6.4.4.2 Compile()

```
async Task<(Assembly Assembly, CSharpCompilation Compilation)> CSharpEditor.Editor.Compile (
            Func< BreakpointInfo, bool > synchronousBreak = null,
            Func< BreakpointInfo, Task< bool >> asynchronousBreak = null )
```

Compile the source code to an Assembly.

Parameters

synchronousBreak	The function to handle synchronous breakpoints. If this is `null`, these breakpoints will be skipped. If you want to enable the default UI for breakpoints, use SynchronousBreak(BreakpointInfo) (or a function that calls it after performing additional operations).
asynchronousBreak	The function to handle asynchronous breakpoints. If this is `null`, these breakpoints will be skipped. If you want to enable the default UI for breakpoints, use AsynchronousBreak(BreakpointInfo) (or a function that calls it after performing additional operations).

Returns

An Assembly containing the compiled code, or `null` if the compilation fails, as well as a CSharpCompilation that also contains information about any compilation errors.

Definition at line 563 of file Editor.public.cs.

6.4.4.3 Create()

```
static async Task<Editor> CSharpEditor.Editor.Create (
              string initialText = "",
              string preSource = "",
              string postSource = "",
              IEnumerable< CachedMetadataReference > references = null,
              CSharpCompilationOptions compilationOptions = null,
              string guid = null,
              Shortcut[] additionalShortcuts = null )  [static]
```

Create a new Editor instance.

Parameters

initialText	The initial text of the editor.
preSource	The source code that should be prepended to the text of the document when compiling it.
postSource	The source code that should be appended to the text of the document when compiling it.
references	A list of MetadataReferences for which the compiled assembly will have bindings. Make sure to include an appropriate DocumentationProvider, if you would like documentation comments to appear in code completion windows. If this is `null`, references to all of the assemblies loaded in the current AppDomain will be added.
compilationOptions	The compilation options used to compile the code. If this is `null`, a `new CSharp↩ CompilationOptions(OutputKind.DynamicallyLinkedLibrary)` will be used.
guid	A unique identifier for the document being edited. If this is `null`, a new System.Guid is generated. If the same identifier is used multiple times, the save history of the document will be available, even if the application has been closed between different sessions.
additionalShortcuts	Additional application-specific shortcuts (for display purposes only - you need to implement your own logic).

Returns

A fully initialised Editor instance.

Definition at line 328 of file Editor.public.cs.

6.4.4.4 CreateCompilation()

```
async Task<CSharpCompilation> CSharpEditor.Editor.CreateCompilation ( )
```

Compile the source code to a CSharpCompilation. Note that breakpoints will be disabled.

Returns

A CSharpCompilation containing the compiled code, which can be used to `Emit` an assembly.

Definition at line 701 of file Editor.public.cs.

6.4.4.5 Save()

```
void CSharpEditor.Editor.Save ( )
```

Add the current text of the document to the save history (if enabled) and invoke the SaveRequested event.

Definition at line 729 of file Editor.public.cs.

6.4.4.6 SetText() [1/2]

```
async Task CSharpEditor.Editor.SetText (
            SourceText text )
```

Sets the text of the document.

Parameters

text	The new text of the document.

Returns

A Task that completes when the text has been updated.

Definition at line 394 of file Editor.public.cs.

6.4.4.7 SetText() [2/2]

```
async Task CSharpEditor.Editor.SetText (
            string text )
```

Sets the text of the document.

Parameters

text	The new text of the document.

Returns

A Task that completes when the text has been updated.

Definition at line 384 of file Editor.public.cs.

6.4.4.8 SynchronousBreak()

```
bool CSharpEditor.Editor.SynchronousBreak (
            BreakpointInfo info )
```

A function to handle breakpoints in synchronous methods. Pass this as an argument to Compile(Func<BreakpointInfo, bool>, Func<E
To prevent deadlocks, this function will have no effect if called from the UI thread.

Parameters

info	A BreakpointInfo object containing information about the location of the breakpoint and the current value of local variables.

Returns

`true` if further occurrences of the same breakpoint should be ignored; `false` otherwise.

Definition at line 405 of file Editor.public.cs.

6.4.5 Member Data Documentation

6.4.5.1 AutosaveInterval

```
int CSharpEditor.Editor.AutosaveInterval => this.AutoSaver.MillisecondsInterval
```

The timeout between consecutive autosaves, in milliseconds.

Definition at line 260 of file Editor.public.cs.

6.4.5.2 CompilationTimeout

```
int CSharpEditor.Editor.CompilationTimeout => this.CompilationErrorChecker.Milliseconds↩
Interval
```

The timeout for automatic compilation after the user stops typing, in milliseconds.

Definition at line 265 of file Editor.public.cs.

6.4.5.3 ShowLineChanges

```
bool CSharpEditor.Editor.ShowLineChanges => this.EditorControl.ShowLineChanges
```

A boolean value indicating whether changed lines are highlighted on the left side of the control.

Definition at line 250 of file Editor.public.cs.

6.4.5.4 ShowScrollbarOverview

```
bool CSharpEditor.Editor.ShowScrollbarOverview => this.EditorControl.ShowScrollbarOverview
```

A boolean value indicating whether a summary of the changed lines, errors/warning, search results, breakpoints and the position of the caret should be shown over the vertical scrollbar.

Definition at line 255 of file Editor.public.cs.

6.4.5.5 SyntaxHighlightingMode

```
SyntaxHighlightingModes CSharpEditor.Editor.SyntaxHighlightingMode => this.EditorControl.↩
SyntaxHighlightingMode
```

The current syntax highlighting mode.

Definition at line 245 of file Editor.public.cs.

6.4.6 Property Documentation

6.4.6.1 AccessType

```
AccessTypes CSharpEditor.Editor.AccessType  [get], [set]
```

Determines whether the text of the document can be edited by the user.

Definition at line 161 of file Editor.public.cs.

6.4.6.2 AutoFormat

```
bool CSharpEditor.Editor.AutoFormat = true  [get]
```

A boolean value indicating whether the source text should be formatted automatically while the user is typing.

Definition at line 240 of file Editor.public.cs.

6.4.6.3 AutoOpenParameters

```
bool CSharpEditor.Editor.AutoOpenParameters = true  [get]
```

A boolean value indicating whether the parameter list tooltip should open automatically while the user is typing.

Definition at line 235 of file Editor.public.cs.

6.4.6.4 AutoOpenSuggestions

```
bool CSharpEditor.Editor.AutoOpenSuggestions = true  [get]
```

A boolean value indicating whether the suggestion panel should open automatically while the user is typing.

Definition at line 230 of file Editor.public.cs.

6.4.6.5 AutoSaveFile

```
string CSharpEditor.Editor.AutoSaveFile  [get]
```

The full path to the autosave file.

Definition at line 220 of file Editor.public.cs.

6.4.6.6 CompilationOptions

```
CSharpCompilationOptions CSharpEditor.Editor.CompilationOptions  [get]
```

Compilation options used to compile the source code.

Definition at line 205 of file Editor.public.cs.

6.4.6.7 FullSource

```
string CSharpEditor.Editor.FullSource  [get]
```

Full source code, including the PreSource, the Text, and the PostSource.

Definition at line 126 of file Editor.public.cs.

6.4.6.8 Guid

```
string CSharpEditor.Editor.Guid  [get]
```

A unique identifier for the document being edited.

Definition at line 210 of file Editor.public.cs.

6.4.6.9 IsReferencesButtonEnabled

```
bool CSharpEditor.Editor.IsReferencesButtonEnabled  [get], [set]
```

A boolean value indicating whether the button allowing the user to add or remove assembly references is enabled or not.

Definition at line 277 of file Editor.public.cs.

6.4.6.10 KeepSaveHistory

```
bool CSharpEditor.Editor.KeepSaveHistory = true  [get]
```

A boolean value indicating whether a history of the saved versions of the document is kept.

Definition at line 225 of file Editor.public.cs.

6.4.6.11 PostSource

```
string CSharpEditor.Editor.PostSource = ""  [get]
```

Source code to be appended after the text of the document when compiling it.

Definition at line 97 of file Editor.public.cs.

6.4.6.12 PreSource

```
string CSharpEditor.Editor.PreSource  [get]
```

Source code to be prepended to the text of the document when compiling it.

Definition at line 80 of file Editor.public.cs.

6.4.6.13 References

```
ImmutableList<MetadataReference> CSharpEditor.Editor.References  [get]
```

The list of MetadataReferences for which the compiled assembly will have bindings.

Definition at line 270 of file Editor.public.cs.

6.4.6.14 SaveDirectory

```
string CSharpEditor.Editor.SaveDirectory  [get]
```

The full path to the directory where the autosave file and the save history for the current document are kept.

Definition at line 215 of file Editor.public.cs.

6.4.6.15 Selection

```
TextSpan CSharpEditor.Editor.Selection  [get], [set]
```

Gets or sets the selected text span.

Definition at line 304 of file Editor.public.cs.

6.4.6.16 SourceText

```
SourceText CSharpEditor.Editor.SourceText  [get]
```

The source code of the document as a SourceText.

Definition at line 114 of file Editor.public.cs.

6.4.6.17 Text

```
string CSharpEditor.Editor.Text  [get]
```

The source code of the document as a string.

Definition at line 102 of file Editor.public.cs.

6.4.6.18 TextChanged

```
EventHandler<EventArgs> CSharpEditor.Editor.TextChanged  [add], [remove]
```

Event raised when the document text is changed.

Definition at line 62 of file Editor.public.cs.

6.4.7 Event Documentation

6.4.7.1 Autosave

```
EventHandler<SaveEventArgs> CSharpEditor.Editor.Autosave
```

Event raised when the document is automatically saved.

Definition at line 52 of file Editor.public.cs.

6.4.7.2 CompilationCompleted

```
EventHandler<CompilationEventArgs> CSharpEditor.Editor.CompilationCompleted
```

Event raised when a background compilation of the document completes.

Definition at line 57 of file Editor.public.cs.

6.4.7.3 SaveRequested

```
EventHandler<SaveEventArgs> CSharpEditor.Editor.SaveRequested
```

Event raised when the user uses the keyboard shortcut or pressed the button to save the document.

Definition at line 47 of file Editor.public.cs.

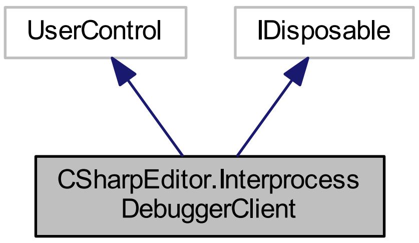
The documentation for this class was generated from the following files:

- CSharpEditor/Editor.axaml.cs
- CSharpEditor/Editor.public.cs

6.5 CSharpEditor.InterprocessDebuggerClient Class Reference

A control that shows breakpoint information for breakpoints reached on a server process. This control contains a read-only CSharpEditor.Editor to display the code, which is reused as much as possible to reduce the initialization time.

Inheritance diagram for CSharpEditor.InterprocessDebuggerClient:



Public Member Functions

- InterprocessDebuggerClient (string[ ] args)

  Creates a new InterprocessDebuggerClient, using the information provided by the InterprocessDebuggerServer to open the pipes to communicate with it.
- void Dispose ()

  Closes the pipes uses by this instance.

Events

- EventHandler< EventArgs > ParentProcessExited

  Invoked when the server process that started this client has been closed or has signaled that all client activity should cease.
- EventHandler< EventArgs > BreakpointHit

  Invoked when the server process signals that a breakpoint has been reached.
- EventHandler< EventArgs > BreakpointResumed

  Invoked when the user signals that code execution can resume.

6.5.1 Detailed Description

A control that shows breakpoint information for breakpoints reached on a server process. This control contains a read-only CSharpEditor.Editor to display the code, which is reused as much as possible to reduce the initialization time.

Definition at line 453 of file InterprocessDebugger.cs.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 InterprocessDebuggerClient()

```
CSharpEditor.InterprocessDebuggerClient.InterprocessDebuggerClient (
             string[] args )
```

Creates a new InterprocessDebuggerClient, using the information provided by the InterprocessDebuggerServer to open the pipes to communicate with it.

Parameters

args	The arguments with which the InterprocessDebuggerServer started the client process.

Definition at line 485 of file InterprocessDebugger.cs.

6.5.3 Member Function Documentation

6.5.3.1 Dispose()

```
void CSharpEditor.InterprocessDebuggerClient.Dispose ( )
```

Closes the pipes uses by this instance.

Definition at line 729 of file InterprocessDebugger.cs.

6.5.4 Event Documentation

6.5.4.1 BreakpointHit

```
EventHandler<EventArgs> CSharpEditor.InterprocessDebuggerClient.BreakpointHit
```

Invoked when the server process signals that a breakpoint has been reached.

Definition at line 471 of file InterprocessDebugger.cs.

6.5.4.2 BreakpointResumed

```
EventHandler<EventArgs> CSharpEditor.InterprocessDebuggerClient.BreakpointResumed
```

Invoked when the user signals that code execution can resume.

Definition at line 476 of file InterprocessDebugger.cs.

6.5.4.3 ParentProcessExited

```
EventHandler<EventArgs> CSharpEditor.InterprocessDebuggerClient.ParentProcessExited
```

Invoked when the server process that started this client has been closed or has signaled that all client activity should cease.

Definition at line 466 of file InterprocessDebugger.cs.

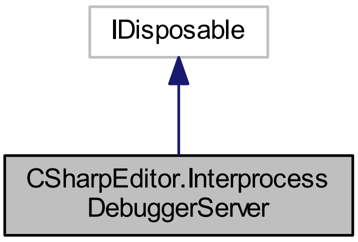
The documentation for this class was generated from the following file:

- CSharpEditor/InterprocessDebugger.cs

6.6 CSharpEditor.InterprocessDebuggerServer Class Reference

A class used to analyse breakpoints on a separate process (to avoid deadlocks with breakpoints in synchronous code).

Inheritance diagram for CSharpEditor.InterprocessDebuggerServer:



Public Member Functions

- InterprocessDebuggerServer (string clientExePath)

  Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.
- InterprocessDebuggerServer (string clientExePath, IEnumerable< string > initialArguments)

  Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.
- InterprocessDebuggerServer (string clientExePath, IEnumerable< string > initialArguments, Func< int, int > getClientPid)

  Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.
- Func< BreakpointInfo, bool > SynchronousBreak (Editor editor)

  Returns a function to handle breakpoints in synchronous methods by transferring the breakpoint information to the client process. Pass the output of this method as an argument to Editor.Compile(Func<BreakpointInfo, bool>, Func<BreakpointInfo, Task The function will lock until the client process signals that execution can resume.
- Func< BreakpointInfo, Task< bool > > AsynchronousBreak (Editor editor)

  Returns a function to handle breakpoints in asynchronous methods by transferring the breakpoint information to the client process. Pass the output of this method as an argument to Editor.Compile(Func<BreakpointInfo, bool>, Func<BreakpointInfo, Task This function will actually execute synchronously and lock until the client process signals that execution can resume.
- void Dispose ()

  Kills the debugger client process and frees the pipe resources.

6.6.1 Detailed Description

A class used to analyse breakpoints on a separate process (to avoid deadlocks with breakpoints in synchronous code).

Definition at line 39 of file InterprocessDebugger.cs.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 InterprocessDebuggerServer() [1/3]

```
CSharpEditor.InterprocessDebuggerServer.InterprocessDebuggerServer (
            string clientExePath )
```

Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.

Parameters

clientExePath	The path to the executable of the client process.

Definition at line 55 of file InterprocessDebugger.cs.

6.6.2.2 InterprocessDebuggerServer() [2/3]

```
CSharpEditor.InterprocessDebuggerServer.InterprocessDebuggerServer (
            string clientExePath,
            IEnumerable< string > initialArguments )
```

Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.

Parameters

clientExePath	The path to the executable of the client process.
initialArguments	The arguments that will be used to start the client process. The additional arguments specific to the InterprocessDebuggerServer will be appended after these.

Definition at line 65 of file InterprocessDebugger.cs.

6.6.2.3 InterprocessDebuggerServer() [3/3]

```
CSharpEditor.InterprocessDebuggerServer.InterprocessDebuggerServer (
            string clientExePath,
            IEnumerable< string > initialArguments,
            Func< int, int > getClientPid )
```

Initializes a new InterprocessDebuggerServer, starting the client process and establishing pipes to communicate with it.

Parameters

clientExePath	The path to the executable of the client process.
initialArguments	The arguments that will be used to start the client process. The additional arguments specific to the InterprocessDebuggerServer will be appended after these.
getClientPid	A method that returns the process identifier (PID) of the client debugger process. The argument of this method is the PID of the process that has been started by the server. If this is `null`, it is assumed that the process started by the server is the client debugger process.

Definition at line 76 of file InterprocessDebugger.cs.

6.6.3 Member Function Documentation

6.6.3.1 AsynchronousBreak()

```
Func<BreakpointInfo, Task<bool> > CSharpEditor.InterprocessDebuggerServer.AsynchronousBreak (
            Editor editor )
```

Returns a function to handle breakpoints in asynchronous methods by transferring the breakpoint information to the client process. Pass the output of this method as an argument to Editor.Compile(Func<BreakpointInfo, bool>, Func<BreakpointInfo,
This function will actually execute synchronously and lock until the client process signals that execution can resume.

Parameters

editor	The Editor whose code will be debugged. Note that no reference to this object is kept after this method returns.

Returns

A function to handle breakpoints in asynchronous methods by transferring the breakpoint information to the client process. If the client process is not executing when a breakpoint occurs, it is started again.

Definition at line 267 of file InterprocessDebugger.cs.

6.6.3.2 Dispose()

```
void CSharpEditor.InterprocessDebuggerServer.Dispose ( )
```

Kills the debugger client process and frees the pipe resources.

Definition at line 442 of file InterprocessDebugger.cs.

6.6.3.3 SynchronousBreak()

```
Func<BreakpointInfo, bool> CSharpEditor.InterprocessDebuggerServer.SynchronousBreak (
             Editor editor )
```

Returns a function to handle breakpoints in synchronous methods by transferring the breakpoint information to the client process. Pass the output of this method as an argument to Editor.Compile(Func<BreakpointInfo, bool>, Func<BreakpointInfo,
The function will lock until the client process signals that execution can resume.

Parameters

editor	The Editor whose code will be debugged. Note that no reference to this object is kept after this method returns.

Returns

A function to handle breakpoints in synchronous methods by transferring the breakpoint information to the client process. If the client process is not executing when a breakpoint occurs, it is started again.

Definition at line 143 of file InterprocessDebugger.cs.

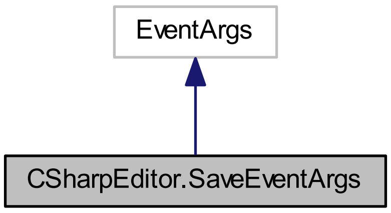
The documentation for this class was generated from the following file:

- CSharpEditor/InterprocessDebugger.cs

6.7 CSharpEditor.SaveEventArgs Class Reference

A class to hold data for an event where the user has requested to save the document.

Inheritance diagram for CSharpEditor.SaveEventArgs:



Properties

- string Text `[get]`

  The text of the document to save (not including any prepended or appended source code).

6.7.1 Detailed Description

A class to hold data for an event where the user has requested to save the document.

Definition at line 753 of file Editor.public.cs.

6.7.2 Property Documentation

6.7.2.1 Text

```
string CSharpEditor.SaveEventArgs.Text  [get]
```

The text of the document to save (not including any prepended or appended source code).

Definition at line 758 of file Editor.public.cs.

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

- CSharpEditor/Editor.public.cs

6.8 CSharpEditor.Shortcut Class Reference

Represents a keyboard shortcut.

Public Member Functions

- Shortcut (string name, string[ ][ ] shortcuts)

  Creates a new Shortcut instance.

Properties

- string Name `[get]`

  The name of the action performed by the shortcut.
- string[ ][ ] Shortcuts `[get]`

  The keys that have to be pressed together to perform the action.

6.8.1 Detailed Description

Represents a keyboard shortcut.

Definition at line 785 of file Editor.public.cs.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Shortcut()

```
CSharpEditor.Shortcut.Shortcut (
              string name,
              string shortcuts[ ][ ] )
```

Creates a new Shortcut instance.

Parameters

name	The name of the action performed by the shortcut (e.g. "Copy").
shortcuts	The keys that have to be pressed together to perform the action (e.g. [["Ctrl", "C"], ["Ctrl", "Ins"]] to specify that either `Ctrl+C` or `Ctrl+Ins` can be used. "Ctrl" will automatically be converted to "Cmd" on macOS.

Definition at line 802 of file Editor.public.cs.

6.8.3 Property Documentation

6.8.3.1 Name

```
string CSharpEditor.Shortcut.Name  [get]
```

The name of the action performed by the shortcut.

Definition at line 790 of file Editor.public.cs.

6.8.3.2 Shortcuts

```
string [][] CSharpEditor.Shortcut.Shortcuts  [get]
```

The keys that have to be pressed together to perform the action.

Definition at line 795 of file Editor.public.cs.

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

- CSharpEditor/Editor.public.cs

Index