



A Guide to C# Functions

2020-10-09

Contents

1	INTRODUCTION	3
1.1	Who Should Read This Guide	3
1.2	Getting Started	3
2	C# FUNCTIONS	4
3	CREATING INLINE C# FUNCTIONS.....	5
3.1	Templates for Inline C# Functions	6
4	EDITING INLINE C# FUNCTIONS.....	10
4.1	Editing General and Debug Settings	10
4.2	Editing Source Code	11
4.3	Selecting Assembly References	11
4.4	Using Input Parameters	13
4.5	Previewing Output Result	13
5	CREATING EXTERNAL C# FUNCTIONS.....	15
5.1	Creating C# Method	15
5.1.1	<i>Adding Description</i>	16
5.1.2	<i>Adding Parameters</i>	16
6	ADDING EXTERNAL C# FUNCTIONS PROGRAMMATICALLY.....	18
6.1	Registering a Static Method	18
6.2	Registering an Instance Method	19
7	ADDING EXTERNAL C# FUNCTIONS IN CMS CONSOLE.....	22
8	INTEGRATING WITH VISUAL EDITOR.....	25
9	USING C# FUNCTIONS.....	27
10	XHTML AND C# FUNCTIONS.....	29
10.1	Returning XHTML from C# Functions	29
10.2	Parsing XHTML in C# Functions	29
11	TEST YOUR KNOWLEDGE	31
11.1	Task 1: Create an Inline C# Function	31
11.2	Task 2: Use an Input Parameter in an Inline C# Function	31
11.3	Task 3: Create an External C# Function	31
11.4	Task 4: Use a C# Function on a Page	31
11.5	Task 5: Use a C# Function in an XSLT Function	31

1 Introduction

As C1 CMS is a .NET Framework based CMS, it is only natural that a web developer can use C# in the development. C# functions allow even the end users to use code written in C# as regular CMS functions.

With all the power of .NET Framework and C# on the one hand and programmatic access to a C1 CMS website and all its constituents via C1 CMS APIs on the other hand, C# functions are effective tools in web development and web design in C1 CMS.

1.1 Who Should Read This Guide

This guide is intended for developers who want to learn how to create C# functions in C1 CMS.

As a developer, you must be an expert in .NET Framework and C# and know how to work with C1 CMS and in the CMS Console as well as a C# code editor of your choice (for example, Visual Studio). Knowing XML and XSLT is desirable as some topics and tasks include references to XSLT functions and XML.

You need to have access to the Functions perspective with sufficient permissions to create, edit and delete C# functions. To use the C# functions on pages and page templates, you might also need to have access to the Content and Layout perspectives.

1.2 Getting Started

To get started with C# Functions, you are supposed to take a number of steps.

Getting Started		
Step	Activity	Chapter or section
1	Create inline C# functions	Creating Inline C# Functions
2	Edit inline C# functions	Editing Inline C# Functions
3	Create external C# functions	Creating External C# Functions
4	Add external C# functions in the CMS Console	Adding External C# Functions in CMS Console
5	Make C# available in Visual Editor	Integrating with Visual Editor
6	Use C# functions	Using C# Functions

In the following few chapters, you will learn more about these and other activities.

2 C# Functions

A C# function is one of several types of [CMS functions](#), written in C#. There are two types of C# functions in C1 CMS:

- Inline
- External

Inline C# functions are created and edited in the CMS Console in its source code editor.

External C# functions are created and edited in an external source code editor such as Visual Studio, normally placed in the App_Code folder and added as CMS functions in the CMS Console.

(Please note that also in the manner described in the following chapters⁶ as the source of external C# function, you can use public static C# methods contained in assemblies in the Bin folder as well as in shared .NET Framework assemblies, e.g. System.Guid.NewGuid(). However, this topic is beyond the scope of this guide, which only deals with the C# methods from App_Code.)

When writing C# functions, in addition to the standard .NET Framework, you can also use C1 CMS API. Please see <http://api.composite.net/> for more information.

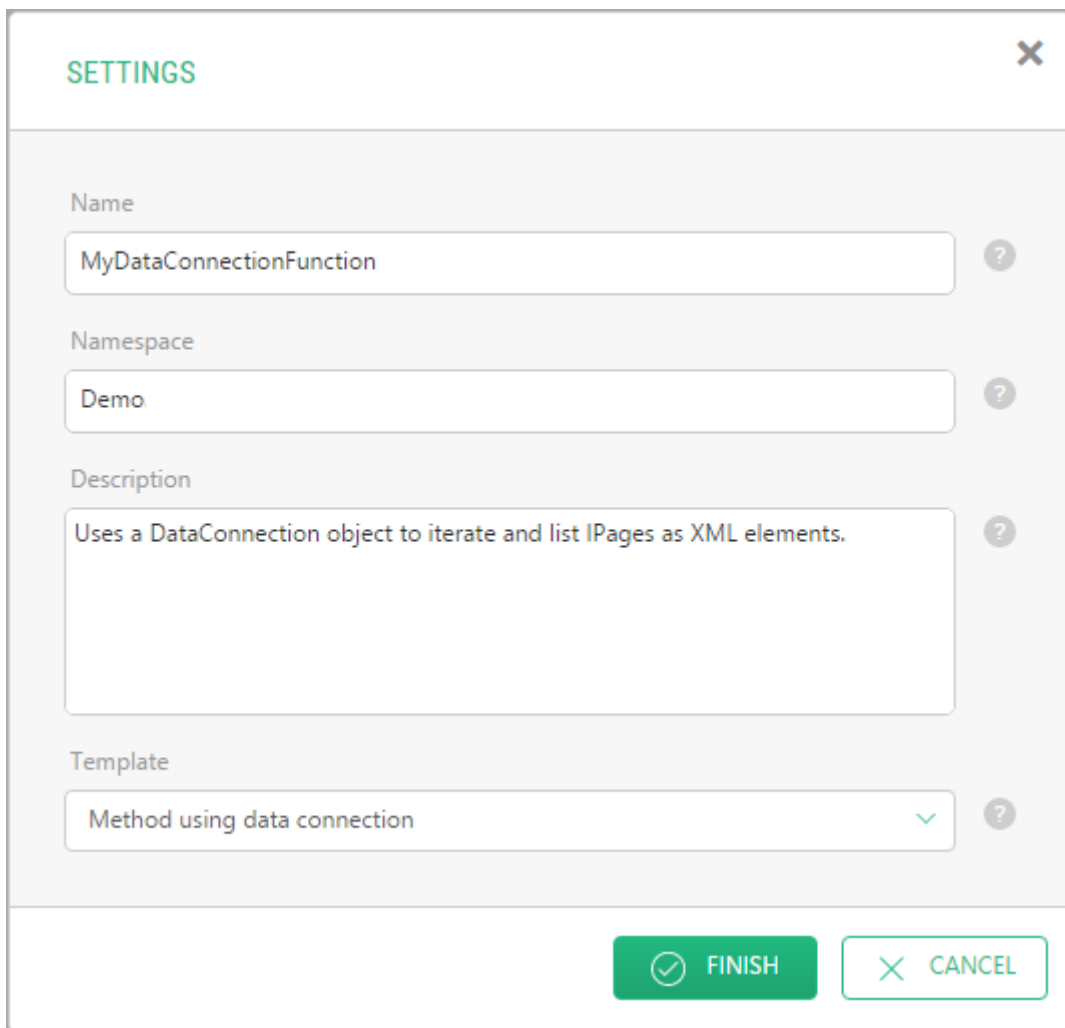
In the following few chapters you will learn how to create and use both inline and external C# functions in C1 CMS.

3 Creating Inline C# Functions

An inline C# function is a CMS function written in C# and normally created in the CMS Console (unlike external C# functions, created in external editors and added to C1 CMS).

To create an inline C# function:

1. In **Functions**, select **C# Functions** and click **Add Inline C# Function**.
2. In the Settings window, enter information in these fields.
 - **Name:** The name of the function
 - **Namespace:** The namespace it should belong to
 - **Description:** A short description of the function
 - **Template:** A [template](#) to use when creating a C# methods
3. Click **Finish**.



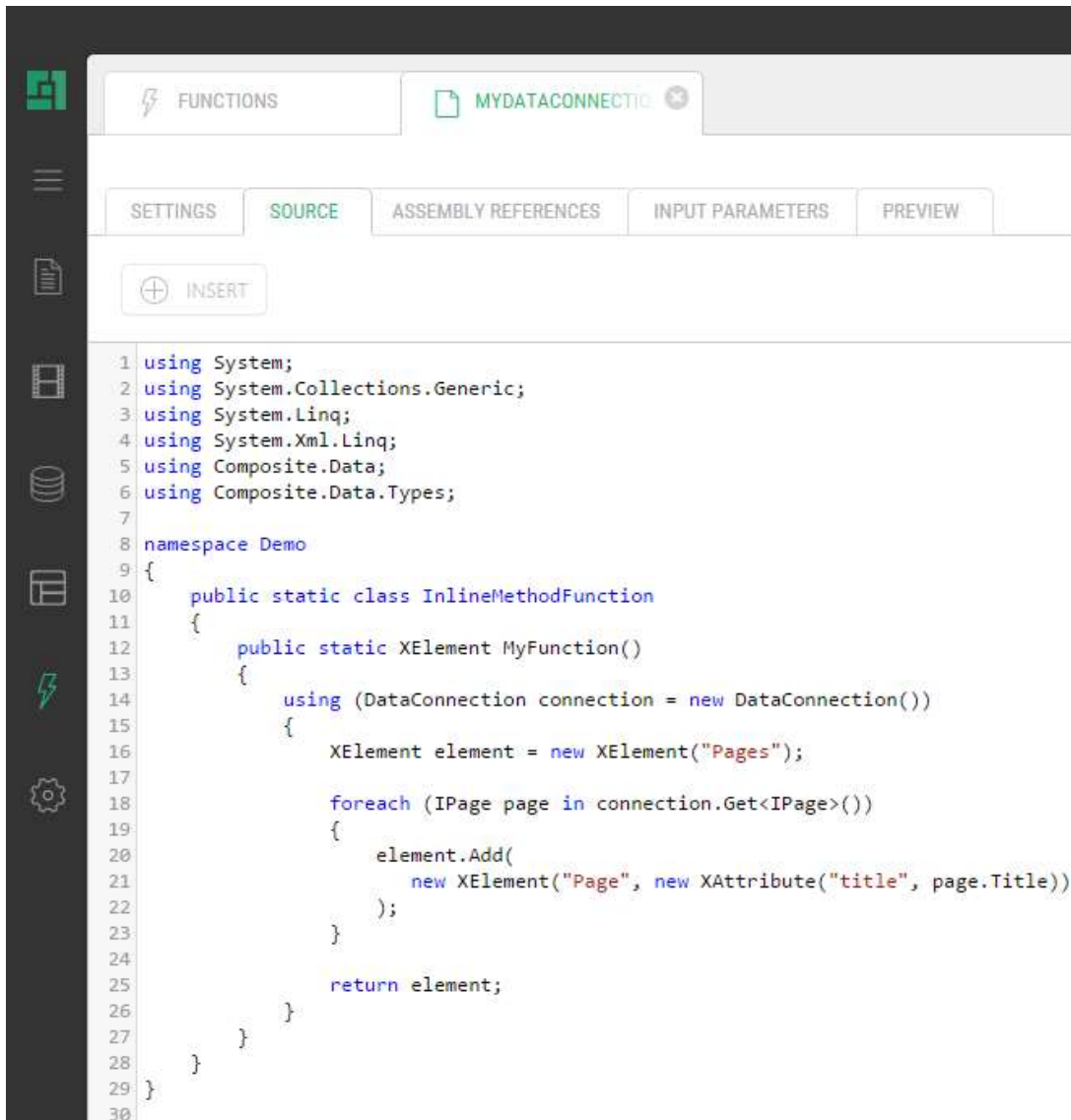
The screenshot shows a 'SETTINGS' dialog box with a close button (X) in the top right corner. It contains four input fields, each with a help icon (?) to its right:

- Name:** MyDataConnectionFunction
- Namespace:** Demo
- Description:** Uses a DataConnection object to iterate and list IPages as XML elements.
- Template:** Method using data connection (with a dropdown arrow and a help icon)

At the bottom of the dialog are two buttons: a green 'FINISH' button with a checkmark icon and a white 'CANCEL' button with an 'X' icon.

Figure 1: Adding an inline C# function

The newly added function opens in the function editor.



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Xml.Linq;
5 using Composite.Data;
6 using Composite.Data.Types;
7
8 namespace Demo
9 {
10     public static class InlineMethodFunction
11     {
12         public static XElement MyFunction()
13         {
14             using (DataConnection connection = new DataConnection())
15             {
16                 XElement element = new XElement("Pages");
17
18                 foreach (IPage page in connection.Get<IPage>())
19                 {
20                     element.Add(
21                         new XElement("Page", new XAttribute("title", page.Title))
22                     );
23                 }
24
25                 return element;
26             }
27         }
28     }
29 }
30
```

Figure 2: Source code of an inline C# function

You can also select a namespace in Step 1 and add a function to it. In this case, the namespace will be automatically entered in the Namespace field.

Please note that the name of the function serves as its identifier in the code, so it must only contain English letters (a-z, A-Z) and digits (0-9) and must start with a letter.

Please also note that on pages you can use C# functions when switched to the Source view. To learn to make them available in Visual Editor, too, please see "[Integrating with Visual Editor](#)".

3.1 Templates for Inline C# Functions

When you create an inline C# function, you can choose one of three templates for its initial code:

- Empty method
- Method with parameters
- Method using data connection

The source code of the **Empty method** template is very simplistic:

- it uses no input parameters
- its code is just a return statement
- it returns 'true'

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using Composite.Data;
using Composite.Data.Types;

namespace Demo
{
    public static class InlineMethodFunction
    {
        public static bool MyEmptyMethod()
        {
            return true;
        }
    }
}
```

Listing 1: Empty method

The source code of the **Method with parameters** template is similar to that of the Empty method but it also uses two input parameters (int and string).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using Composite.Data;
using Composite.Data.Types;

namespace Demo
{
    public static class InlineMethodFunction
    {
        public static bool MyMethodWithParams(int myIntValue, string
myStringValue)
        {
            return true;
        }
    }
}
```

Listing 2: Method with parameters (source code)

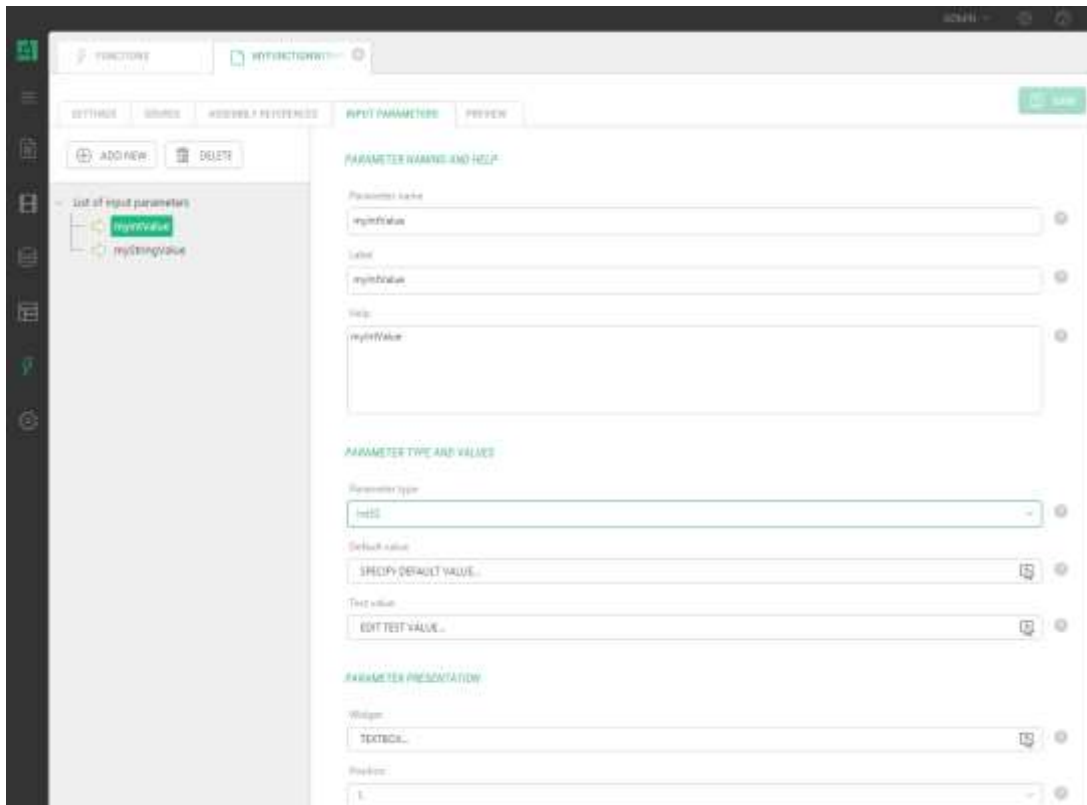


Figure 3: Method with input parameters (Input Parameters tab)

The source code of the **Method using data connection** template is more sophisticated:

- it uses no parameters
- it uses C1 CMS API to establish a data connection, iterates all IPage objects and creates a list of <Page> elements that specify their page title.
- it returns XML (XElement) as a result


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using Composite.Data;
using Composite.Data.Types;

namespace Demo
{
    public static class InlineMethodFunction
    {
        public static XElement MyDataConnectionMethod()
        {
            using (DataConnection connection = new
DataConnection(PublicationScope.Unpublished))
            {
                XElement element = new XElement("Pages");

                foreach (IPage page in connection.Get<IPage>())
                {
                    element.Add(
                        new XElement("Page", new XAttribute("title", page.Title))
                    );
                }

                return element;
            }
        }
    }
}

```

Listing 3: Method using data connection

4 Editing Inline C# Functions

Based on the template selected when creating the function, you get the initial source code in the source code editor.

It is now your task to implement the desired logic by [editing the source code](#) within the created method.

Besides, you can [add one or more input parameters](#) to the function.

You can keep track of the function's work by [previewing its current output](#) (debugging) as you continue to edit it. [Debugging can be configured](#) to work in the environment your function is intended to be.

To edit an inline C# function:

1. In the **Functions** perspective, expand **C# Functions** and expand namespaces the function belongs to.
2. Select the function and click **Edit** on the toolbar. The function opens in the function editor.
3. On the **Settings** tab, edit its [general and debug settings](#).
4. On the **Source** tab, [edit the function's source code](#).
5. On the **Assembly References** tab, [select assemblies](#), references to which are required by your source code.
6. On the **Input Parameters** tab, [add, edit or delete its input parameters](#).
7. On the **Preview** tab, debug the function by [previewing its return result](#).
8. Click **Save**.

4.1 Editing General and Debug Settings

You normally set a function's general settings when [creating it](#). However, you can later change its general settings on the **Settings** tab of the function editor.

The general settings of an inline C# function include:

- **Name:** The name of the function
- **Namespace:** The namespace it belongs to
- **Description:** The description of the function

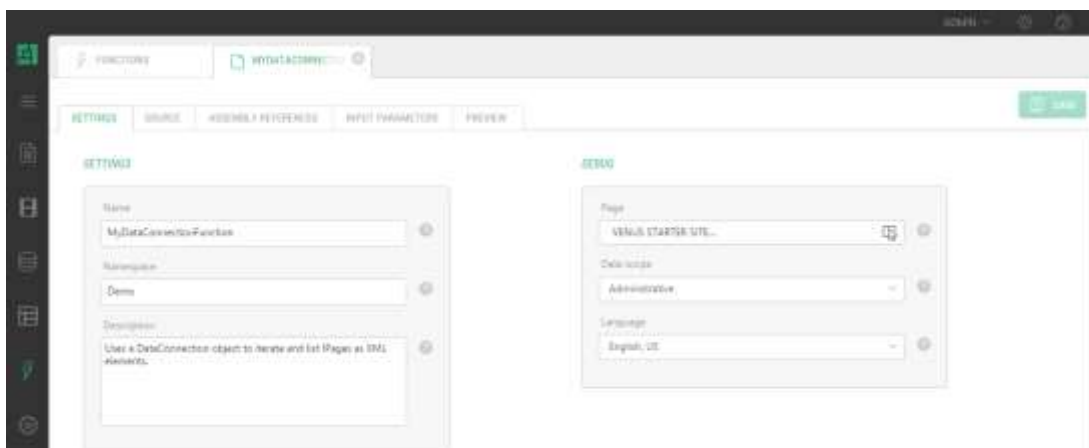


Figure 4: Settings tab

Along with the general settings, you can manage the function's debug settings here.

If you want to quickly see what this function will output without using it in the content yet, you can debug it by [previewing](#) it on the Preview tab.

The debug settings of an inline C# function include:

- **Page:** The page used as the context for rendering when debugging
- **Data scope:** The public or development version of data as the context for rendering when debugging
- **Language:** The language to use for debugging

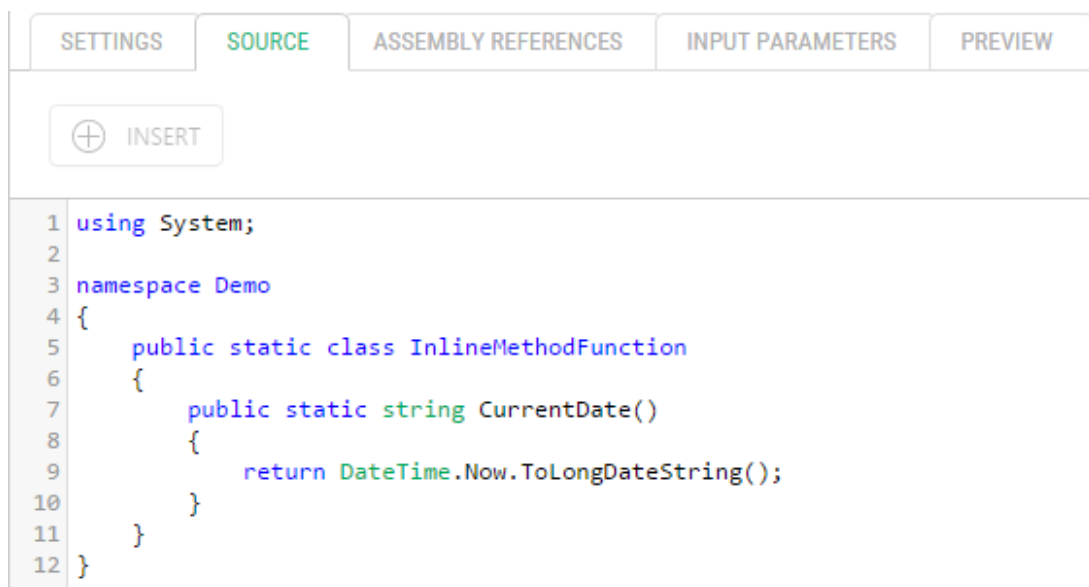
4.2 Editing Source Code

When an inline C# function has been created, it already has initial source code based on the selected template.

The code usually includes:

- The list of used namespaces (using...)
- The namespace this method belongs to (namespace Demo {...})
- The system-generated public static class for inline C# methods (public static class InlineMethodFunction {...})
- The public static method that serves as an inline C# function in C1 CMS (public static XElement Show3(...) {...})

Normally, you make your changes in the method.



The screenshot shows a code editor interface with five tabs: SETTINGS, SOURCE (selected), ASSEMBLY REFERENCES, INPUT PARAMETERS, and PREVIEW. Below the tabs is an 'INSERT' button with a plus sign icon. The main area displays C# source code with line numbers 1 through 12. The code defines a namespace 'Demo' containing a public static class 'InlineMethodFunction' with a public static method 'CurrentDate()' that returns the current date as a long date string.

```
1 using System;
2
3 namespace Demo
4 {
5     public static class InlineMethodFunction
6     {
7         public static string CurrentDate()
8         {
9             return DateTime.Now.ToLongDateString();
10        }
11    }
12 }
```

Figure 5: Source tab

Please note that if you change the namespace and the name of the method in the source code, you should also change them on the Settings tab.

Please also note that you cannot change the class name. The class and the method should both be public and static.

When inserting the “using” statements, make sure you have included required [assembly references](#).

If you need to return XHTML from a function, consider [using XmlDocument](#).

4.3 Selecting Assembly References

The code you write in your inline C# function normally requires one or more assembly references.

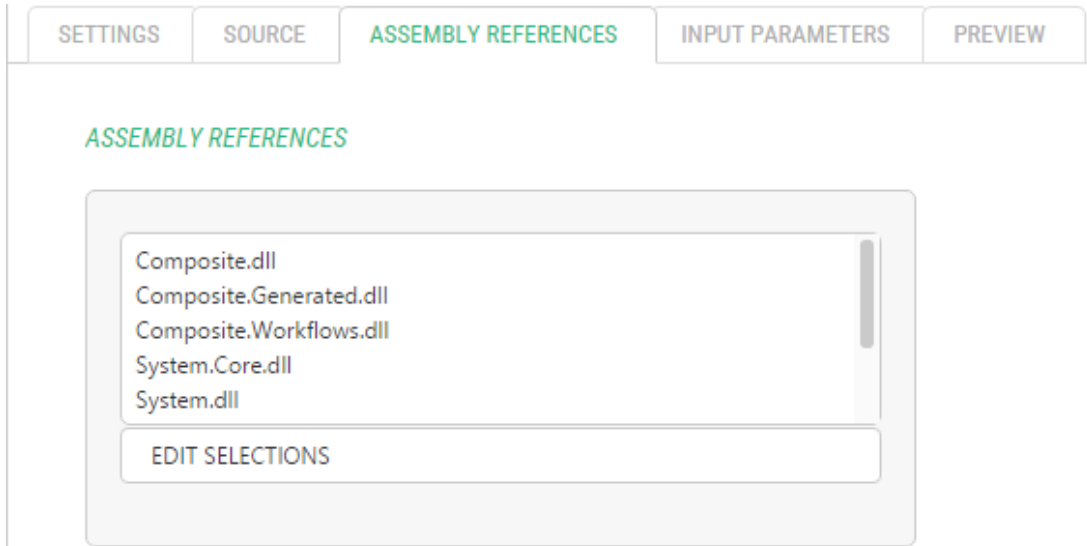


Figure 6: Assembly References tab

When you create the function, the default set of assembly references is added. You can however choose to include or exclude references at will.

To manage assembly references:

1. Edit an inline C# function.
2. On the Assembly References tab, click Edit Selections.
3. In the window that opens:
 - a. move to the right list the assemblies the references to which you want to include
 - b. move to the left list the assemblies the references to which you want to exclude

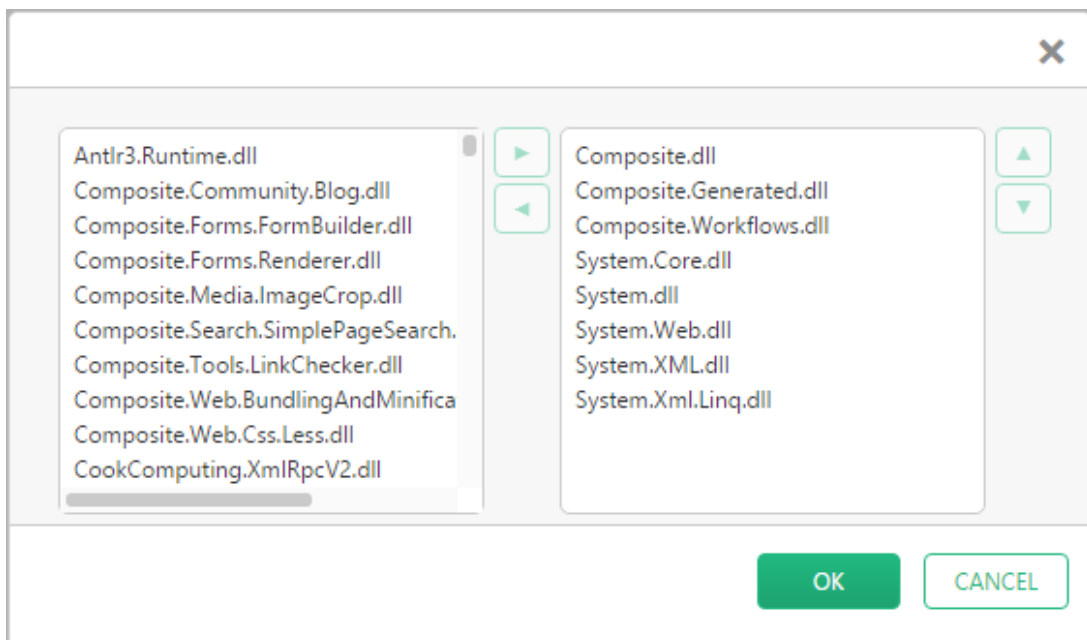


Figure 7: Including and excluding assembly references

4. Click OK and save the function.

4.4 Using Input Parameters

Input parameters enable users of an inline C# function to customize its behavior. The values specified in the input parameters can be further used in the source code.

Please note that if you add an input parameter in the source code as the parameter of the corresponding method, you must add the same parameter (name, type) on the Input Parameter tab and vice versa.

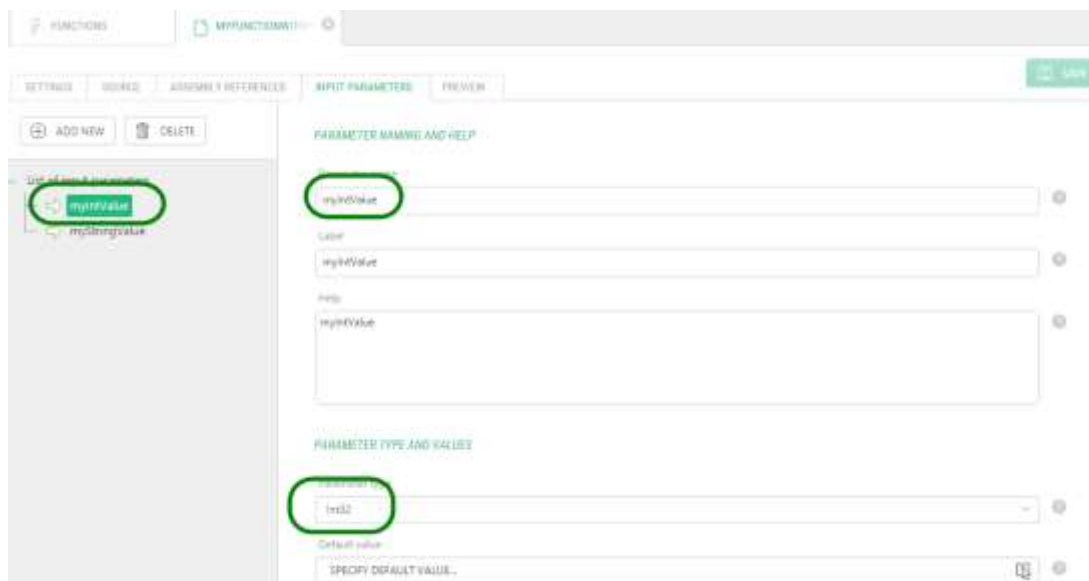
To add an input parameter to an inline C# function:

1. Edit an inline C# function.
2. On the **Source** tab, add a parameter to the method (for example, `int myIntParam`)

```
8 namespace Demo
9 {
10     public static class InlineMethodFunction
11     {
12         public static bool MyFunctionWithParameter(int myIntValue, string myStringValue)
13         {
14             return true;
15         }
16     }
17 }
```

Figure 8: Parameters added in source code

3. On the **Input Parameters** tab, add the parameter with the same name and type. (For information on creating input parameters, please refer to the [Guide to Function Parameters](#).)



The screenshot shows the 'INPUT PARAMETERS' tab in a development tool. On the left, there is a list of parameters with 'myIntValue' selected and circled in green. On the right, the configuration form for 'myIntValue' is shown, with the name, label, help text, and parameter type (int) all circled in green. The 'Default Value' field is empty and labeled 'SPECIFY DEFAULT VALUE...'.

Figure 9: Parameters added on the Input Parameters tab

4. Click **Save**.

4.5 Previewing Output Result

You can preview the result you inline C# function should return on the Preview tab.



Figure 10: Previewing the function's return result

This is also a way of debugging your function. If something is wrong, the error will be displayed in the preview instead of the result.

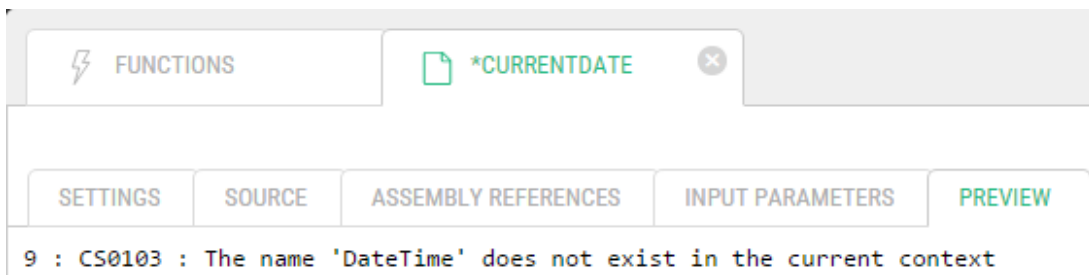


Figure 11: Error displayed

5 Creating External C# Functions

An external C# function is a CMS function written in C# and created in an external code editor such as Visual Studio (unlike internal C# functions created directly in the CMS Console).

To create an external C# function:

1. [Create a C# method](#) in a code editor of your choice (for example, Visual Studio) and move it to the website.
2. [Add this method to C1 CMS](#) as an external C# function in the CMS Console.

Please note that you can:

- Keep your C# method in a .cs file in the /App_Code folder. (You may want to create /App_Code as there might be no /App_Code in C1 CMS initially.)
- Build a class library with this method and place it in the /Bin folder.

Please also note that on pages you can use C# functions when switched to the Source view. To learn to make them available in Visual Editor, too, please see [“Integrating with Visual Editor”](#).

5.1 Creating C# Method

To create the C# method in a .cs file in /App_Code:

1. In your code editor, create a C# class in the website's App_Code folder (or elsewhere and then move it to App_Code).
2. Add using statements and references if required in your code.
3. Add namespaces if necessary.
4. Make sure the class is public.
5. Create a public static method in this class.
6. Save the .cs file.

```
using System;

namespace Demo
{
    public class Functions
    {
        public static string CurrentDate()
        {
            return DateTime.Now.ToLongDateString();
        }
    }
}
```

Listing 4: Demo.Functions.CurrentDate method

C1 CMS must become aware of this newly added class and all its public static methods.

Please note that the file should be placed in the App_Code folder or one of its subfolders. Both the class and its method(s) must be public and the method must also be static.

You can create as many classes in one file and methods in one class as you need.

If you need to return XHTML from a function, consider [using XmlDocument](#).

Alternatively, you can create a class library with the method:

1. In Visual Studio, create a class library project.
2. Add using statements and references if required in your code.
3. Add namespaces if necessary.

4. Make sure the class is public.
5. Create a public static method in this class.
6. Build the code.
7. Copy the output DLL to the /Bin folder of your website.

5.1.1 Adding Description

A regular CMS function can have a description. To provide a description for an external C# function, use the `Function` attribute on the C# method setting its `Description` parameter to user-friendly text:

```
using System;

namespace Demo
{
    public class Functions
    {
        [Function(Description="Returns the current date in long format")]
        public static string CurrentDate()
        {
            return DateTime.Now.ToLongDateString();
        }
    }
}
```

Listing 5: C# method with some description

5.1.2 Adding Parameters

You can add parameters to such C# methods.

```
using System;

namespace Demo
{
    public class Functions
    {
        public static string ShowSomeInfo(string text, int number)
        {
            return text + ": " + number.ToString();
        }
    }
}
```

Listing 6: C# method with parameters

A regular CMS function can have a user-friendly label, some help text, a default value and a widget other than the default one. They appear when you use the function.

You can provide these settings for external C# functions as well - by using the `FunctionParameter` attribute on the C# method:


```

using System;
using Composite.Functions;

namespace Demo
{
    public class Functions
    {
        [FunctionParameter(Name = "text",
            Label = "Some Text",
            Help = "The text to show",
            DefaultValue = "Amount",
            WidgetMarkup =
@"<f:widgetfunction name='Composite.Widgets.String.TextArea'
xmlns:f='http://www.composite.net/ns/function/1.0' />")]
        [FunctionParameter(Name = "number",
            Label = "Some Number",
            Help = "The number to show",
            DefaultValue = 5)]
        public static string ShowSomeInfo(string text, int number)
        {
            return text + ": " + number.ToString();
        }
    }
}

```

Listing 7: Using the FunctionParameter attribute

As you can see in the example above, in the FunctionParameter attribute, you need to specify the name of the parameter in the Name parameter, and the needed settings in the following parameters:

- Label
- Help
- DefaultValue
- WidgetMarkup

You can provide the widget markup via the <f:widgetfunction> element specifying its name and its namespace. If the widget have parameters, you can specify the parameters, too, via the <f:param> element:

```

[FunctionParameter(Name = "text", WidgetMarkup =
@"<f:widgetfunction name="Composite.Widgets.String.TextBox"
xmlns:f="http://www.composite.net/ns/function/1.0"><f:param
name="SpellCheck" value="False" /></f:widgetfunction>")]

```

Listing 8: Using the widget with parameters

Note: The FunctionParameter attribute is available in Composite C1 (now C1 CMS) version 3.2 or later.

6 Adding External C# Functions Programmatically

It is possible to register a C# method as a C1 function from the code of your custom assembly so that it can be used in C1 CMS in markups or directly as a native C1 function by end users. You can use the default `CodeBasedFunctionProvider` to create one without creating your own function provider.

With this provider, you can easily register both static and instance methods of your custom class. Imagine your package has the following `DataProvider` class with the static method `GetDataFromStaticMethod` and the instance method `GetDataFromInstanceMethod`.

```
using System.Collections.Generic;

namespace Orkestra.Example.Assembly
{
    /// <summary>
    /// Example class
    /// </summary>
    public class DataProvider
    {
        /// <summary>
        /// Get data from a static method
        /// </summary>
        public static IEnumerable<string> GetDataFromStaticMethod()
        {
            return new [] { "data 1", "data 2" };
        }

        /// <summary>
        /// Get data from an instance method
        /// </summary>
        public IEnumerable<string> GetDataFromInstanceMethod()
        {
            return new [] { "data 3", "data 4" };
        }
    }
}
```

Listing 9: Data Provider class

6.1 Registering a Static Method

To register a static method as a C1 function you have to take the following steps for your assembly:

- Add the `StartupHandler` class, augmented with `[ApplicationStartup]` attribute (if it does not exist yet)
- In the `OnInitialized()` method, register the C# method as a C1 function, calling the `RegisterMethod` method of the `CodeBasedFunctionRegistry` class as in the example below.

```
public static void OnInitialized()
{
    CodeBasedFunctionRegistry.RegisterMethod<DataProvider>(
        nameof(DataProvider.GetDataFromStaticMethod),
        "Parent.Category.ExampleStatic",
        "Description 1");
}
```

Listing 10: OnInitialized method

And that's all. After you have installed your assembly as a part of a C1 package or even copied a compiled DLL to the bin folder of the website and restarted it, this method will become available as a C1 function.

6.2 Registering an Instance Method

To register an instance method, you should take the same steps as described above for a static method.

However, to avoid creating a new object on each method call (if it is not necessary) in the StartupHandler class in the ConfigureServices method, register a singleton for a desired type.

```
public static void ConfigureServices(IServiceCollection collection)
{
    collection.AddSingleton(typeof(DataProvider));
}
```

Listing 11: Registering a singleton

In this case, the same object will be used with each GetDataFromInstanceMethod call. As a result, the StartupHandler class should look as below.

```
using Composite.Core.Application;
using
Composite.Plugins.Functions.FunctionProviders.CodeBasedFunctionProvider;
using Microsoft.Extensions.DependencyInjection;
using Orckestra.Example.Assembly;

namespace Orckestra.Test.Assembly
{
    [ApplicationStartup()]
    public static class StartupHandler
    {
        public static void OnBeforeInitialize() { }
        public static void OnInitialized()
        {
            CodeBasedFunctionRegistry.RegisterMethod<DataProvider>(
                nameof(DataProvider.GetDataFromStaticMethod),
                "Parent.Category.ExampleStatic",
                "Description 1");

            CodeBasedFunctionRegistry.RegisterMethod<DataProvider>(
                nameof(DataProvider.GetDataFromInstanceMethod),
                "Parent.Category.ExampleInstance",
                "Description 2");
        }

        public static void ConfigureServices(IServiceCollection collection)
        {
            collection.AddSingleton(typeof(DataProvider));
        }
    }
}
```

Listing 12: StartupHandler class

After you have copied a compiled assembly to the bin folder of the website and restarted it (for example, as part of the package installation), the GetDataFromStaticMethod and GetDataFromInstanceMethod methods will become available as C1 functions ExampleStatic and ExampleInstance.

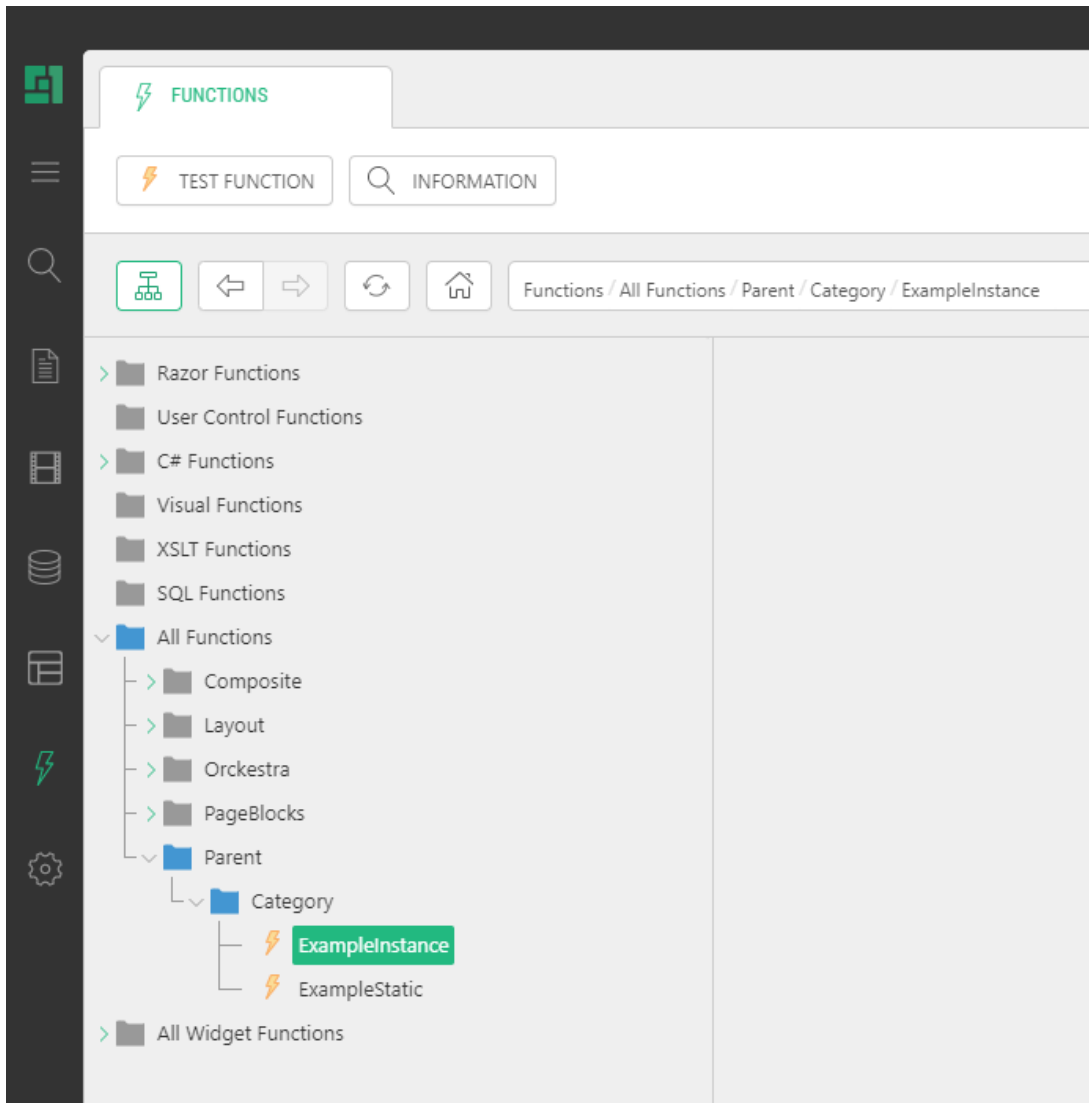


Figure 12: Step 1: The methods available as C1 functions

The end user can call these registered C1 functions directly in the C1 CMS Console or use the functions in markups and other C1 CMS functionality.

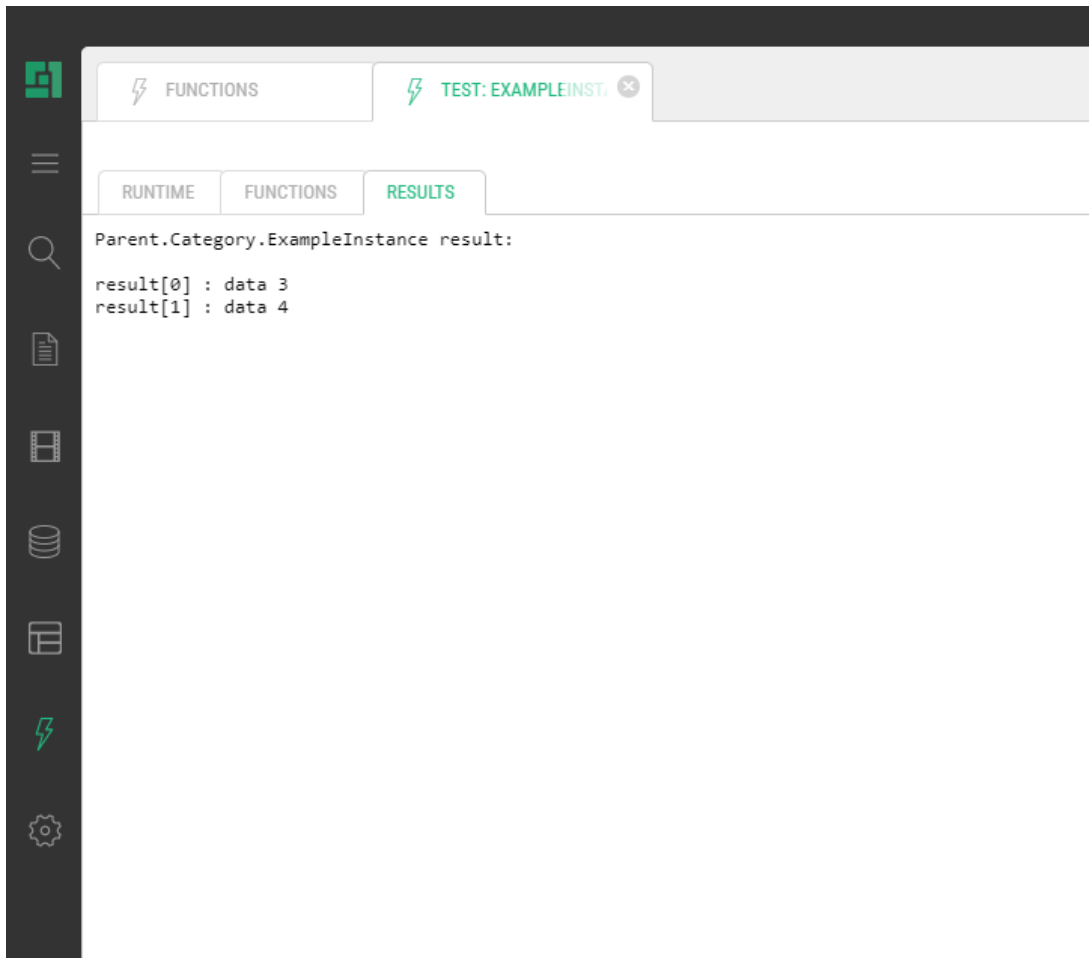
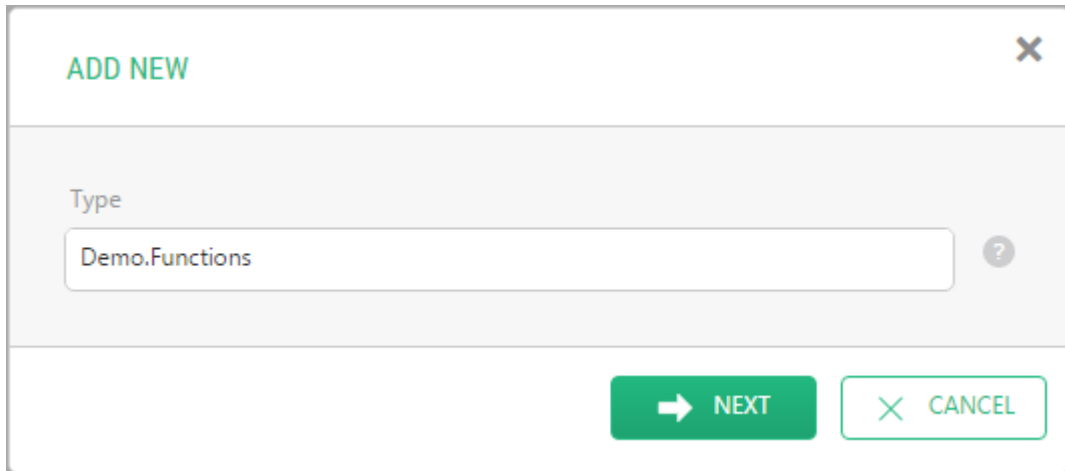


Figure 13: Viewing the results of a function

7 Adding External C# Functions in CMS Console

To add one of the C# methods (created externally) as an external C# function in C1 CMS:

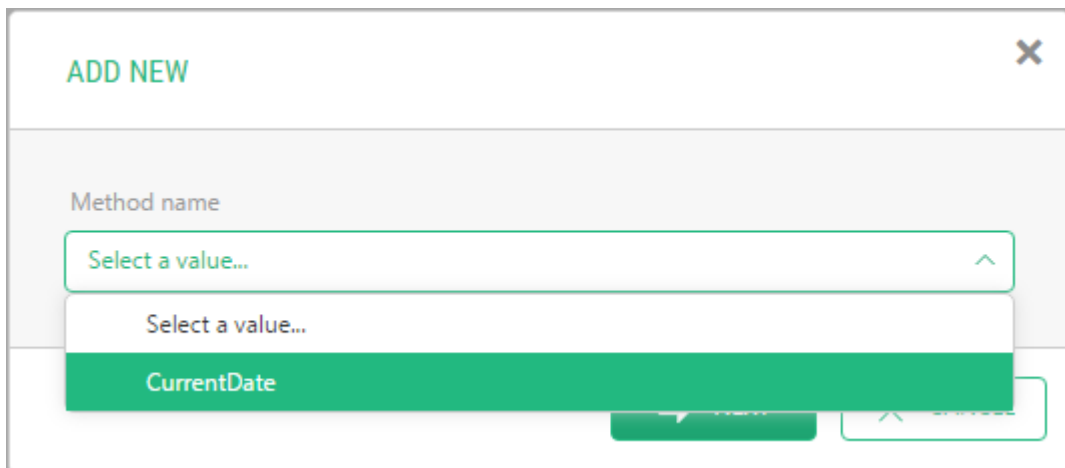
1. In **Functions**, select **C# Functions** and click **Add External C# Function**.
2. In Step 1 of the wizard, in the **Type** field specify the class the method belongs to (including namespaces) (e.g. "Demo.Functions").



The screenshot shows a dialog box titled "ADD NEW" with a close button (X) in the top right corner. Below the title bar, there is a "Type" label and a text input field containing "Demo.Functions". To the right of the input field is a question mark icon. At the bottom of the dialog, there are two buttons: a green "NEXT" button with a right-pointing arrow and a white "CANCEL" button with a green border and an X icon.

Figure 14: Step 1: Specifying the C# method's type

3. In Step 2, select the **Method name** from the list (e.g. "CurrentDate").



The screenshot shows the same "ADD NEW" dialog box, but now the "Method name" label is above a dropdown menu. The dropdown menu is open, showing a list of options. The top option is "Select a value..." with a small upward arrow. Below it is another "Select a value..." option. The bottom option is "CurrentDate" and is highlighted with a green background. To the right of the dropdown menu is a small upward arrow icon. At the bottom of the dialog, there are two buttons: a green "NEXT" button and a white "CANCEL" button with a green border and an X icon.

Figure 15: Step 2: Selecting the C# method by name

4. In Step 3, if necessary, change the **Method Name** and **Namespace Name** suggested by C1 CMS for the external C# function.

The image shows a dialog box titled "ADD NEW" with a close button in the top right corner. Below the title bar, there are two text input fields. The first is labeled "Method Name" and contains the text "CurrentDate". The second is labeled "Namespace Name" and contains the text "Demo.Functions". At the bottom right of the dialog, there are two buttons: a green button with a white checkmark icon and the text "FINISH", and a white button with a green border, a green 'X' icon, and the text "CANCEL".

Figure 16: Step 3: Specifying the name and the namespace for the external C# function

5. Click **Finish**.

The external C# function will appear under C# Functions in the Functions perspective.

Please note that values in Step 1 and 2 of the wizard (Type, Method name) are taken from the C# method you have created in your code editor and must be exact. The values in Step 3 of the wizard (Method Name, Namespace Name) make up the name of the external C# function in C1 CMS and can be arbitrary.

If you edit such a function in C1 CMS in the Functions perspective, you will be able to change the following settings:

- **Method Name:** The name of the external C# function (in C1 CMS)
- **Namespace Name:** The name of the namespace the external C# function belongs to (in C1 CMS)
- **Type:** The actual name of the namespace(s) and the class the C# method belongs to (in external code editor)
- **Method:** The actual name of the C# method (in external code editor)

The screenshot shows a web interface for editing a function. At the top, there are two tabs: 'FUNCTIONS' and 'CURRENTDATE'. Below the tabs, the main heading is 'EDIT METHOD BASED QUERY'. The form contains four input fields, each with a question mark icon to its right:

- Method Name: CurrentDate
- Namespace Name: Demo.Functions
- Type: Demo.Functions
- Method: CurrentDate

Figure 17: Editing an external C# function's properties

To edit the code behind this function, you should open the corresponding C# file (.cs) in your code editor and edit the corresponding C# method therein.

8 Integrating with Visual Editor

The return type of an inline or external C# function influences whether the function will be available in Visual Editor (Visual mode of the page editor) in the Select Function dialog.

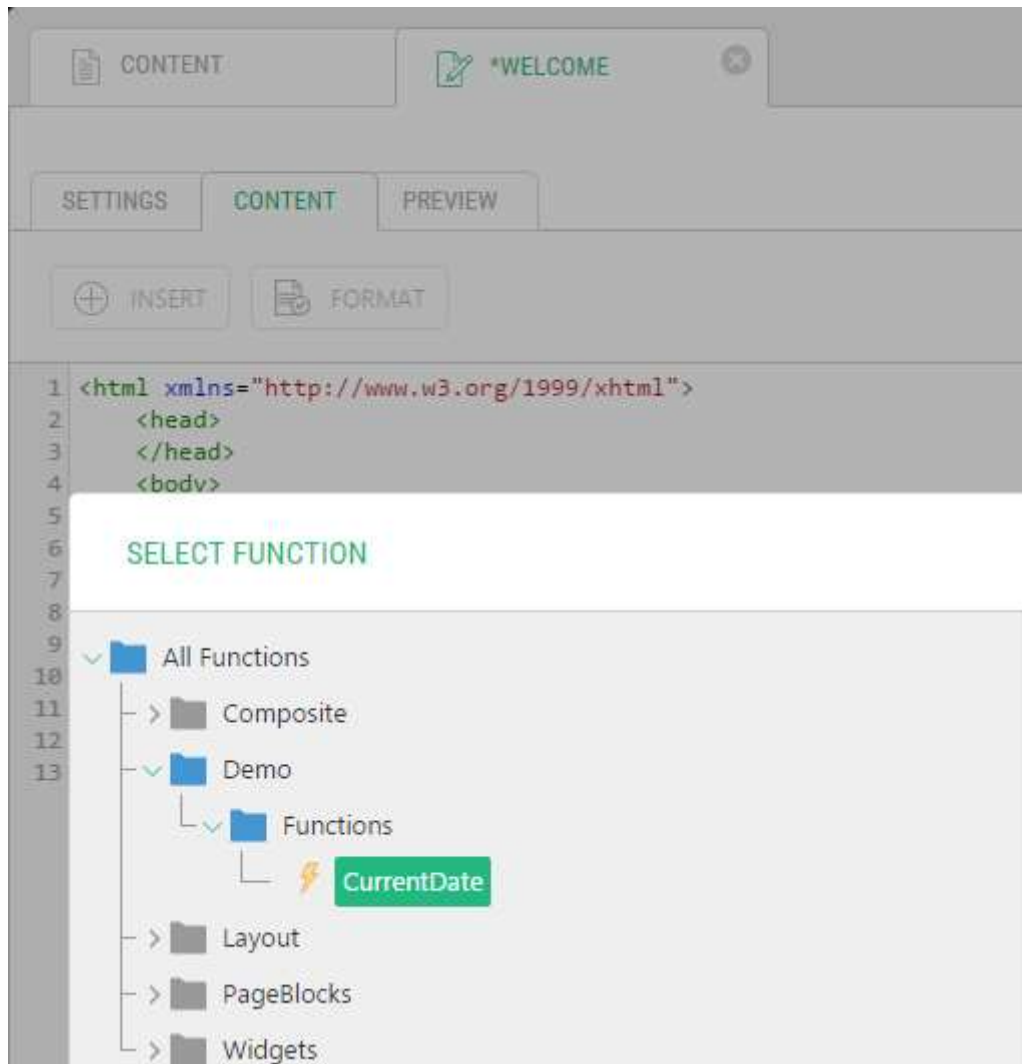


Figure 18: C# Function available in Source Editor

C# functions are always available in Source Editor (source code mode of the page editor).

To make a C# function also available for insertions from Visual Editor, too, make sure that its return value is of one of these types:

- Composite.Core.Xml.XhtmlDocument
- System.Web.UI.Control

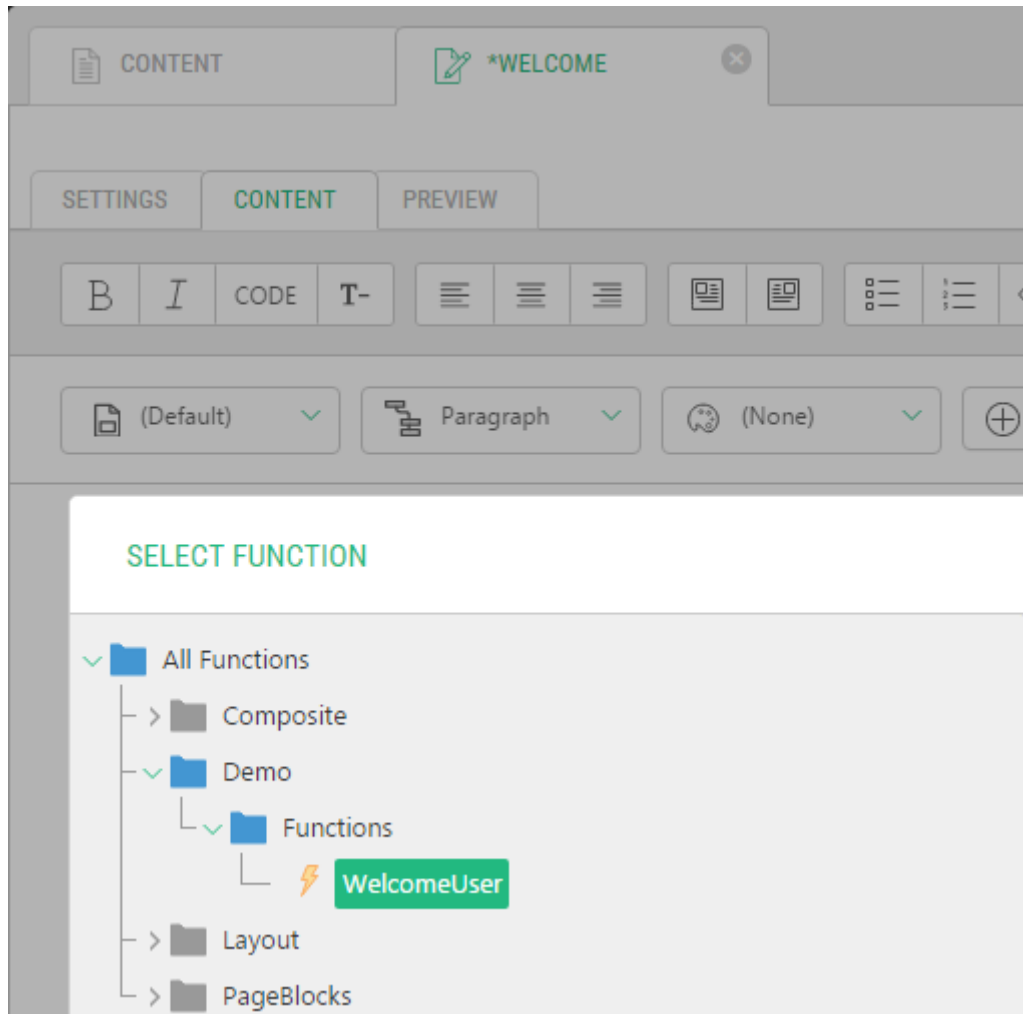


Figure 19: C# Function available in Visual Editor

For information about using `Composite.Core.Xml.XhtmlDocument` in C# functions, please see [“XHTML and C# Functions”](#).

For information about `System.Web.UI.Control`, please see [“Control Class”](#).

9 Using C# Functions

You can use both inline and external C# functions as any other CMS functions. You can insert them on pages or page templates. You can make function calls to them from, or use them in templates of, XSLT functions. You can use C# functions to set values of input parameters and data fields.

Please note that on pages you can use C# functions when switched to the Source view. To learn to make them available in Visual Editor, too, please see "[Integrating with Visual Editor](#)".

To use a C# function on a page:

1. In **Content**, select a page and click **Edit Page**.
2. Switch to **Source**.
3. Place the cursor where you want the function inserted.
4. Click **Insert > Function Markup**.
5. In the **Select Function** window, expand All Functions, namespaces, select the function and click OK.

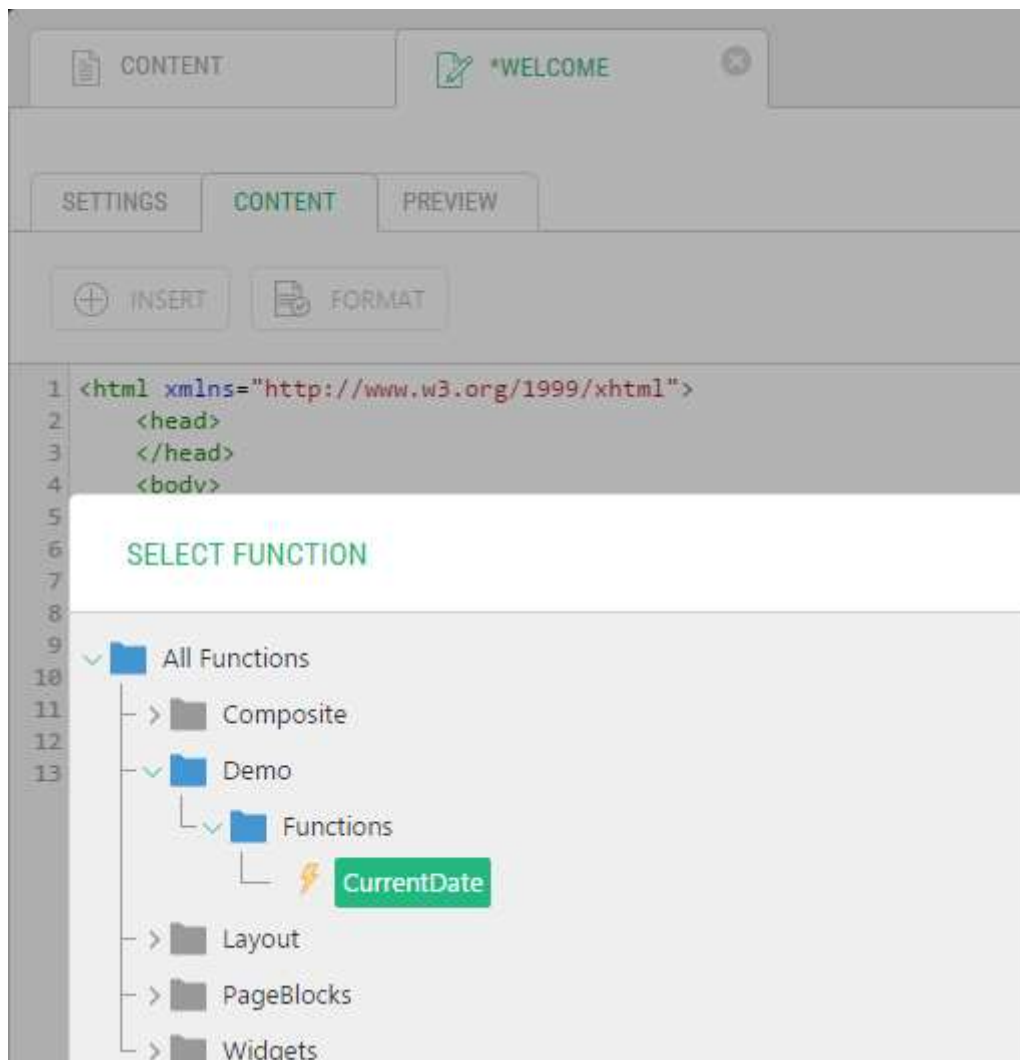


Figure 20: Inserting a C# function on a page (Source view)

6. In the **Function properties** window, set the function's parameters if required and click OK.

The function will appear on a page.

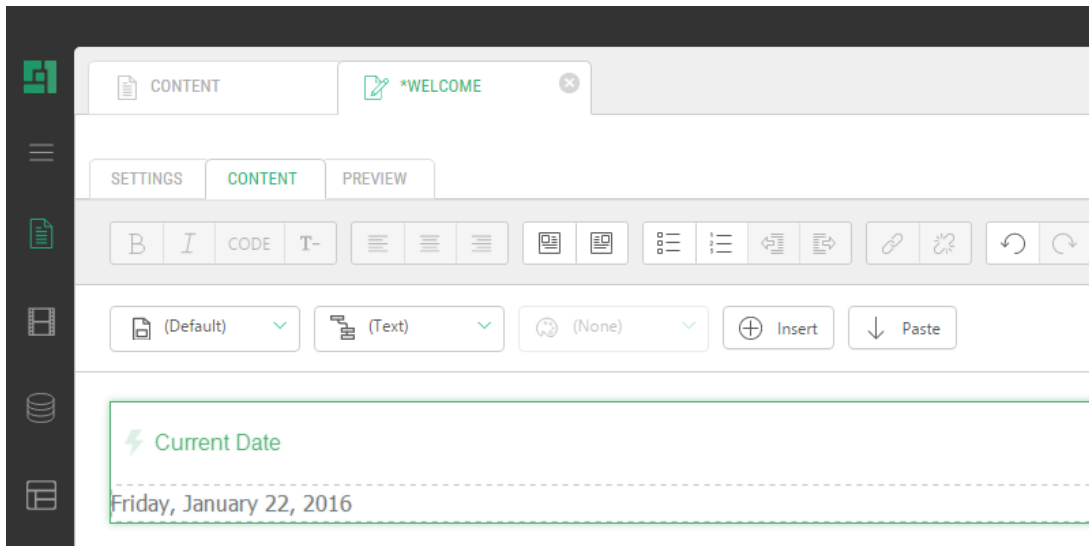


Figure 21: A C# function inserted on a page (Visual view)

In the same manner, you can insert C# function in page templates and XSLT functions.

Please also see "[How to Use Functions](#)" in the "[Guide to CMS Functions](#)".

10 XHTML and C# Functions

You may use a C# function to create some markup that will be further used in some other functions, page templates or on pages.

Also, in general for CMS functions to be available in Visual Editor for insertion on pages, they must return XHTML. Otherwise they will be only visible in Source Editor.

For these purposes, you may want to consider having your C# function to return XHTML.

As many CMS functions return XHTML, in some cases you may need to use the XHTML parsed rather than raw.

Let's have a quick look at how you can:

- [Return XHTML from C# functions](#)
- [Parse XHTML in C# functions](#)

10.1 Returning XHTML from C# Functions

If you want to return XHTML from a C# function, you should use `XhtmlDocument` (**Composite.Core.Xml.XhtmlDocument**) to create required markup and return it as an `XhtmlDocument` object.

The `XhtmlDocument` class is part of C1 CMS API and is inherited from .NET Framework class `System.Xml.Linq.XDocument`.

You can add XHTML content to **XhtmlDocument.Head** (<head/> section of the document) and **XhtmlDocument.Body** (<body/> section of the document) by using the `Add` method.

```
public static XhtmlDocument WelcomeUser(string userName)
{
    string welcomeMessage = String.Format("Hi {0}, welcome to my website!",
        userName);

    XhtmlDocument document = new XhtmlDocument();

    document.Body.Add(
        new XElement(Namespaces.Xhtml + "p", welcomeMessage));

    return document;
}
```

Listing 13: A sample C# function returning XHTML

Please note XML namespaces are important here. In the above sample an HTML <p /> element is created as `Namespaces.Xhtml+"p"`. `Namespaces.Xhtml` ensures that the <p/> element is in the XHTML namespace.

When you use `XhtmlDocument` as the C# function's return value, it will also be available in Visual Editor so you will not need to switch to the Source mode.

As the function on a page or in a page template is nothing but markup (<f:function />), you can also use `XhtmlDocument` to build the markup of a certain CMS function and return it as a value of your C# function.

10.2 Parsing XHTML in C# Functions

If you use a value, for example, from a data field, that contains markup (`XhtmlDocument`, `XElement` etc), you should consider using [XElement.Parse](#) method to parse it first.

So rather than using raw markup:

```
myXelement.Add(teamMember.HtmlDescriptionField);
```

use it parsed:

```
myXelement.Add(XElement.Parse(teamMember.HtmlDescriptionField));
```

As to ASP.NET Web Forms and MVC scenarios, you should print the value raw as you normally would.

11 Test Your Knowledge

11.1 Task 1: Create an Inline C# Function

1. Create an inline C# function "Demo.ListPages".
2. Select the Method using data connection template for the initial code.
3. Change the code so that the returned XML should also include Page.Id values in the id attributes of <Page /> elements
4. Preview the return value.

11.2 Task 2: Use an Input Parameter in an Inline C# Function

1. Edit the Demo.ListPages function.
2. Manually add a Boolean parameter "Includelds" to the function (on both the Source and Input Parameters tabs) setting its default value to "false".
3. Use the parameter value in the source code to choose whether to include the id attribute in the <Page/> elements.
4. Preview the return value.
5. Change the default value to "true" and preview the return value.

11.3 Task 3: Create an External C# Function

1. In Visual Studio, create a C# method "Demo.CurrentDate".
2. Change its source code so that it should return the current date and time as a string.
3. Add the method as an external C# function to C1 CMS.

11.4 Task 4: Use a C# Function on a Page

1. Edit a page.
2. Insert the Demo.CurrentDate function on the page.
3. Preview the page and check for the correct date and time.

11.5 Task 5: Use a C# Function in an XSLT Function

1. Create an XSLT function "Demo.ListPageTitles".
2. Make a function call to the Demo.ListPages function.
3. Preview the input of the XSLT function.
4. Change the template of the function so that it uses the returned XML of the Demo.ListPages function to list all page titles.
5. Preview the output of the XSLT function.