



# A Guide to XSLT Functions

2017-02-15

# Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
1.1	Who Should Read This Guide	3
1.2	Getting Started	3
<b>2</b>	<b>CREATING XSLT FUNCTIONS.....</b>	<b>4</b>
<b>3</b>	<b>EDITING FUNCTIONS .....</b>	<b>7</b>
3.1	Editing General Settings	8
3.2	Editing Debug Settings	9
<b>4</b>	<b>CREATING INPUT PARAMETERS.....</b>	<b>11</b>
<b>5</b>	<b>MAKING FUNCTION CALLS.....</b>	<b>13</b>
5.1	Setting Parameters of Called Functions	13
5.2	Common CMS Functions	14
5.3	Data-Centric Functions	15
<b>6</b>	<b>EDITING THE TEMPLATE.....</b>	<b>16</b>
6.1	Using Input Parameters	16
6.2	Using Function Call Results	17
6.3	Using XSLT Extension Functions	17
6.3.1	<i>DateFormatting</i>	17
6.3.2	<i>Globalization</i>	18
6.3.3	<i>MarkupParser</i>	18
6.4	Using C# Inline Functions	19
<b>7</b>	<b>PREVIEWING FUNCTIONS.....</b>	<b>21</b>
7.1	Input Preview	21
7.2	Output Preview	22
<b>8</b>	<b>TEST YOUR KNOWLEDGE .....</b>	<b>23</b>
8.1	Task: Create an XSLT Function	23
8.2	Task: Add a Function Call	23
8.3	Task: Edit the Template	23
8.4	Task: Add an Input Parameter	23
8.5	Task: Use an Extension Function	23
8.6	Task: Use XSLT to Switch between Two Date Formats	23

# 1 Introduction

XSLT functions play a key role in designing and developing websites in C1 CMS. In this guide you will learn how to create XSLT functions, edit XSLT functions by adding input parameters and making calls to other CMS functions, edit their markup as well as preview and debug the functions.

## 1.1 Who Should Read This Guide

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

As a web developer, you must be an expert in XML and XSLT and know how to work with C1 CMS and in its CMS Console. Knowing C# is a plus as some topics include examples in C# as a scripting language used in XSLT.

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

## 1.2 Getting Started

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

Getting Started		
Step	Activity	Chapter or section
1	Create a function	<a href="#"><i>Creating XSLT Functions</i></a>
2	Edit its general settings	<a href="#"><i>Editing General Settings</i></a>
3	Edit its debug settings	<a href="#"><i>Editing Debug Settings</i></a>
4	Add input parameters	<a href="#"><i>Creating Input Parameters</i></a>
5	Make function calls	<a href="#"><i>Making Function Calls</i></a>
6	Edit its XSLT markup	<a href="#"><i>Editing the Template</i></a>
7	Preview the function	<a href="#"><i>Previewing Functions</i></a>

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

## 2 Creating XSLT Functions

An XSLT function is a key function in C1 CMS. And as other types of [CMS functions](#), an XSLT function is a logical unit that you can insert on a page, in markup of a page template or another function.

It serves three major purposes or the combination thereof in C1 CMS:

- **Content Reuse:** XSLT functions make parts of content or functionality reusable within a single website and across multiple websites.
- **Dynamic Content Rendering:** XSLT functions present dynamic content and allow its customization.
- **Integration:** XSLT functions integrate external content or functionality by wrapping it up in format usable in C1 CMS.

As its name suggests, an XSLT function can transform anything that can be passed to it in XML format by using XSLT. It then outputs the transformed markup either as XHTML or XML. Internally, a XSLT function can use outputs of other CMS functions available in the system.

To create an XSLT function:

1. In the Functions perspective, select **XSLT Functions**.
2. Click **Add XSLT Function** on the toolbar.
3. In the Add New XSLT Function window, specify its parameters:
  - **Name:** The name of the function
  - **Namespace:** The namespace it should belong to
  - **Output type:** The type of the function's output: XHTML or XML
  - **Copy from:** Keep the option "(New XSLT function)" to create a new function, or select an existing XSLT function to copy from.
4. Click **OK**.

**ADD NEW XSLT FUNCTION**

Name  
MyFirstXsltFunction

Namespace  
Demo

Output type  
XHTML

Copy from  
(New XSLT function)

OK CANCEL

Figure 1: Adding an XSLT function

The newly added function opens in the function editor.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3   xmlns:in="https://www.composite.net/ns/transformation/input/1.0"
4   xmlns:lang="http://www.composite.net/ns/localization/1.0"
5   xmlns:if="http://www.composite.net/ns/function/1.0"
6   xmlns="http://www.w3.org/1999/xhtml"
7   exclude-result-prefixes="xsl in lang f">
8
9   <xsl:template match="/"> |
10  <html>
11    <head>
12      <!-- markup placed here will be shown in the head section of the rendered page -->
13    </head>
14
15    <body>
16      <!-- markup placed here will be the output of this rendering -->
17      <div>
18        Value of input parameter 'TestParam':
19        <xsl:value-of select="/in:inputs/in:param[@name='Input parameter name']" />
20      </div>
21      <div>
22        Value of function call 'TestCall':
23        <xsl:value-of select="/in:inputs/in:result[@name='Function call local name']" />
24      </div>
25    </body>
26  </html>
27 </xsl:template>
28
29 </xsl:stylesheet>
30

```

Figure 2: XSLT function editor

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.

The functions that output XML are only available in the Select Function dialog invoked from the source code editor. XHTML-based functions are available both for the source code and visual content editors.

### 3 Editing Functions

Once you have created an XSLT function, it features only default markup which practically does nothing.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:in="http://www.composite.net/ns/transformation/input/1.0"
xmlns:lang="http://www.composite.net/ns/localization/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0"
xmlns="http://www.w3.org/1999/xhtml"
exclude-result-prefixes="xsl in lang f">

  <xsl:template match="/">
    <html>
      <head>
        <!-- markup placed here will be shown in the
head section of the rendered page -->
      </head>

      <body>
        <!-- markup placed here will be the output of
this rendering -->
        <div>
          Value of input parameter 'TestParam':
          <xsl:value-of
select="/in:inputs/in:param[@name='Input parameter name']" />
        </div>
        <div>
          Value of function call 'TestCall':
          <xsl:value-of
select="/in:inputs/in:result[@name='Function call local name']" />
        </div>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Listing 1: Default function markup

It is your task now to implement the desired logic. In most cases, you will add or edit its template's markup.

Besides, you can add one or more input parameters to the function and call other CMS functions to use their results as kind of input or their logic as sub-functions.

You can always keep track of the function's work by previewing its current input and 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 XSLT function:

1. In the **Functions** perspective, expand **XSLT 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 **Input Parameters** tab, add, edit or delete its [input parameters](#).
5. On the **Function Calls** tab, add, edit or delete [calls to other CMS functions](#).
6. On the **Template** tab, edit the function's [XSLT markup](#).
7. On the **Preview** tab, [preview the function's input and output](#) if necessary.
8. Click **Save**.

### 3.1 Editing General Settings

You 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.

SETTINGS INPUT PARAMETERS FUNCTION CALLS TEMPLATE PREVIEW

**SETTINGS**

Name  
MyFirstXsltFunction

Namespace  
Demo

Description  
My first XSLT function that shows all its capabilities

Output type  
XHTML ✓

Figure 3: General settings

This is where you can provide a description for the function. It is a good practice for all CMS functions, as the description appears in:

- the **Function Properties** window (in **Advanced** mode) when the user inserts the function or edits its properties in the content



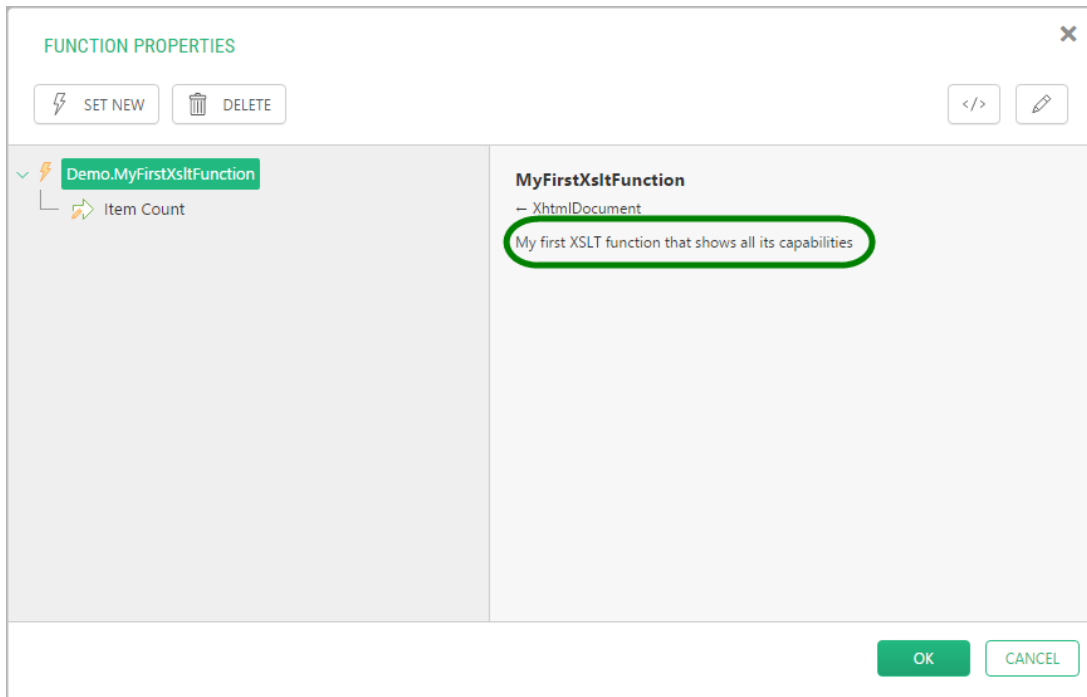


Figure 4: Description in the Function Properties window (Advanced mode)

- the **Documentation** view when the user generates documentation for CMS functions

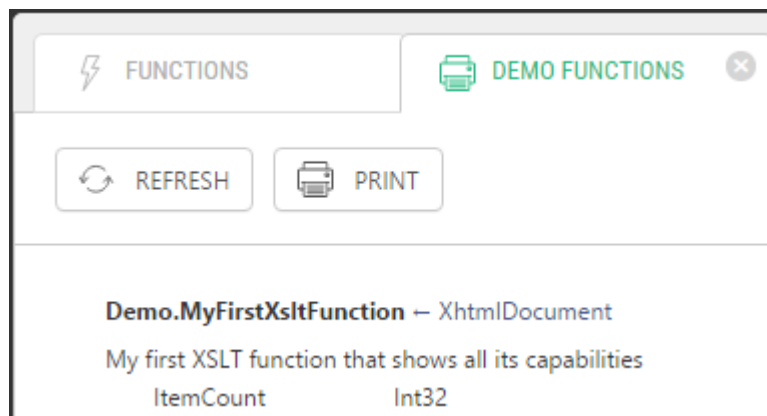


Figure 5: Description in the Function documentation


The general settings of an XSLT function include:

- **Name:** The name of the function
- **Namespace:** The namespace it belongs to
- **Description:** The description of the function
- **Output type:** The type of the function's output: XHTML or XML

### 3.2 Editing Debug Settings

Along with the general settings of an XSLT function, you can manage its debug settings on the **Settings** tab.

## DEBUG



The image shows a 'DEBUG' settings panel with three sections:

- Page:** A dropdown menu with the text 'VENUS STARTER SITE...' and a question mark icon to its right.
- Data scope:** A dropdown menu with the text 'Administrative' and a question mark icon to its right.
- Language:** A dropdown menu with the text 'English, US' and a question mark icon to its right.

Figure 6: Debug settings

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

However, if your function uses, for example, an active page's Id filter and thus depends on a specific page, previewing it might fail or output no result.

This is where the debug settings come in handy. They 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 Creating Input Parameters

Input parameters enable users of an XSLT function to customize its appearance and behavior, which makes it flexible and highly adjustable.

The values specified in the input parameters can be further used in [function calls](#) and the [function's template markup](#).

For example, if your function outputs a list of items, it may output all of them by default. One way is to limit the number of items to display by editing the function's template markup. This would be a hard-coded value, which makes your function inflexible in this respect.

The other and better way is to use a variable in the markup that will get its value from an input parameter set by the user of the function.

To add an input parameter to a function:

1. Edit an XSLT function.
2. On the **Input Parameters** tab, select **List of input parameters**.
3. Click **Add New**. The input parameter's property editor opens on the right.
4. Set the parameters properties where required or necessary:
  - **Parameter name:** The name of the parameter. The name is used by the system to identify this parameter. Names must be unique and may not contain spaces and other special characters. Use names like 'Title', 'StartDate', 'LargeImage' etc.
  - **Label:** The text that users should see when specifying a value for this parameter. This is the 'human name' for the parameter.
  - **Help:** Write a short text that tells the user what to do with the parameter.
  - **Parameter type:** The type of this parameter.
  - **Default value:** You can specify a default value for this parameter. If a parameter has a default value, users are not required to specify it when calling the function.
  - **Test value:** When previewing you can test with different input parameter values using this field. If this is left blank, the default value will be used for previews.
  - **Widget type:** You can select which type of input widget (like a textbox) to use when specifying a value for this parameter. Widgets are only available for simple types.
  - **Position:** The position of the parameter. This controls the order of the parameters.
5. Click **Save**.

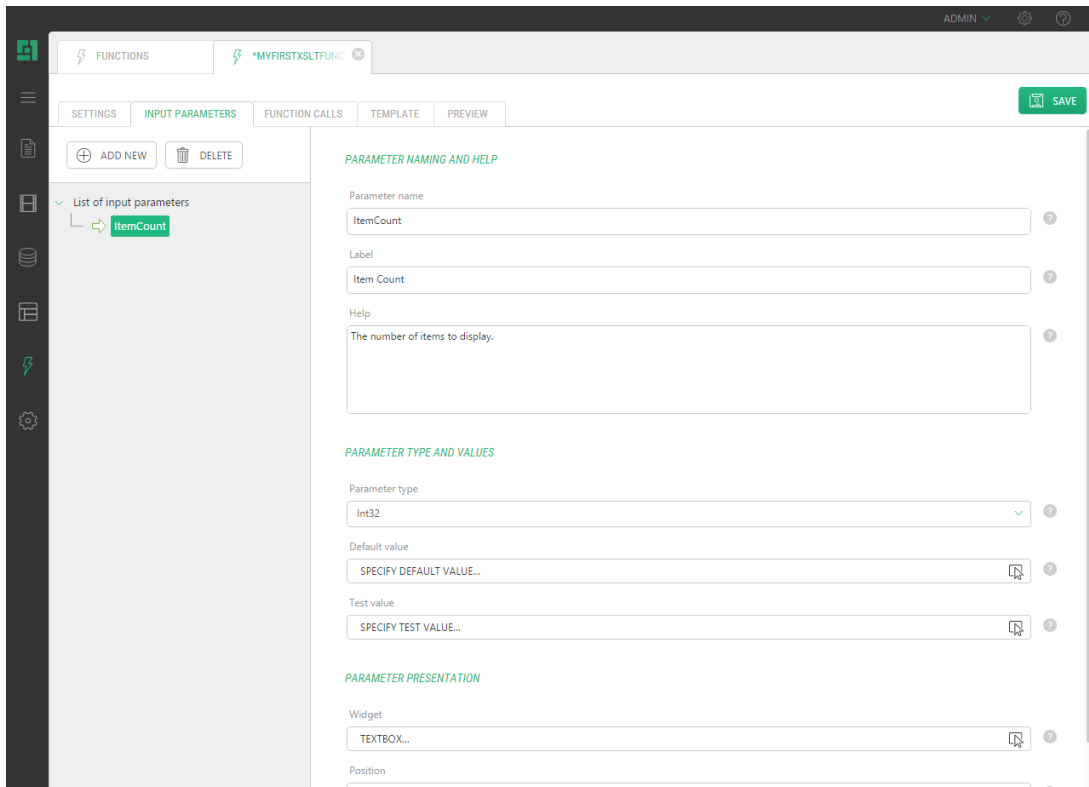


Figure 7: Input parameters

Now you can [refer to the parameter from the function's template markup](#).

Please note that the input parameter's value may be also used as a value for the parameter in a function call. Its type must match that of the parameter of the [function called](#) to become available for selection.

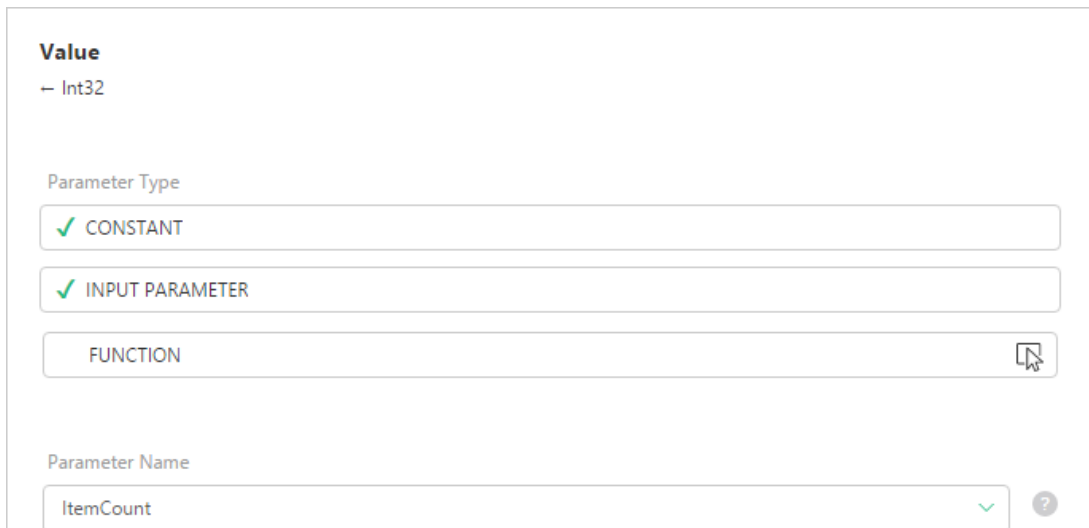


Figure 8: Input parameter as a function's parameter type

## 5 Making Function Calls

Each XSLT function can call other CMS functions to extend its functionality and use external content or CMS- and browser-related information.

Very common is calling a function that retrieves data from a specific data type in XML format to further [process and transform it](#).

A number of functions extend the functionality available for the function by default. For example, if one of the data item fields is in XHTML format, it makes sense to parse and present it properly in a browser. A number of functions help format strings and dates in multiple ways.

The XSLT function can call both built-in and custom functions.

To make a function call:

1. Edit an XSLT function.
2. On the **Function Calls** tab, click **Add New**. The **Select Function** window opens.
3. Expand **All functions**, namespaces and select the function.
4. Click **OK**. The function (with its parameters if any) appears in the function list.
5. If necessary, select the function in the list and override its **Local name** set by default.
6. Select and set its parameters where required or necessary.
7. Click **Save**.

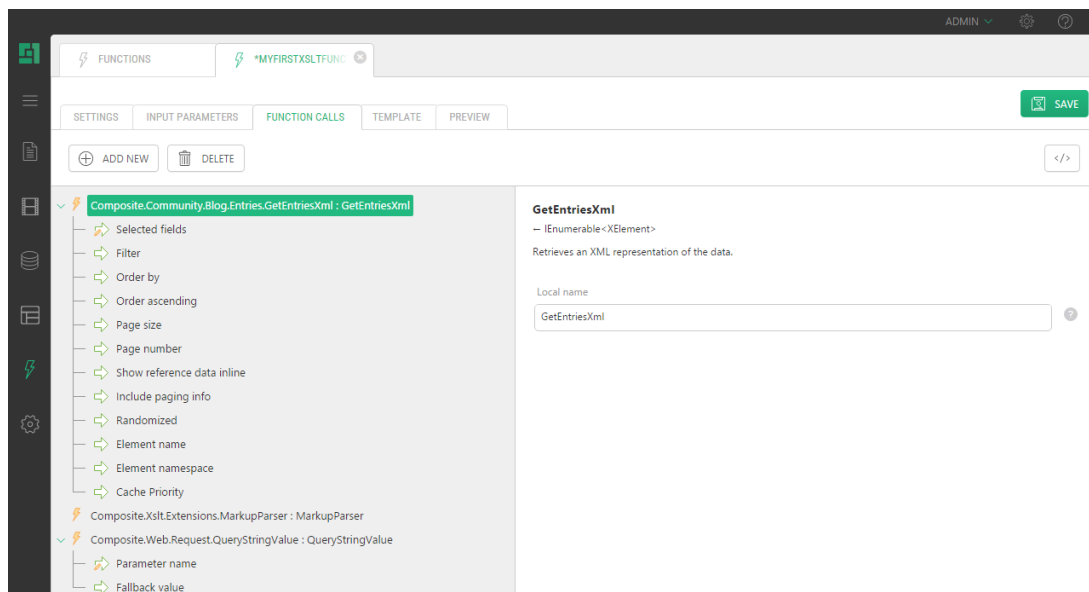


Figure 9: Function calls

Now you can [refer to the function call's result from the function's template markup](#).

### 5.1 Setting Parameters of Called Functions

Each function that you call from your XSLT function may come with no, one or many parameters.

Some parameters are required, and you have to set them; others are optional and set to some reasonable defaults, which you can override when necessary or keep as they are.

Setting the parameters of the called functions is no different from setting the parameters of the CMS functions inserted in the Content or Source editors in C1 CMS.

For information about setting parameters in CMS functions, please refer to [How to Set Function Parameters](#) in the *Guide to CMS functions*.

## 5.2 Common CMS Functions

The built-in CMS functions are grouped by namespaces. The following are most common namespace the functions may belong to:

- Constant
- Core.Xml
- Datatypes
- Mail
- Pages
- Utils
- Utils.Caching
- Utils.Compare
- Utils.Globalization
- Utils.Integer
- Utils.String
- Utils.Date
- Web.Client
- Web.Html.Template
- Web.Request
- Web.Server
- Web.Response
- Xslt.Extensions

There are a number of useful functions you may want to use when creating your own XSLT function.

If you want to set a parameter to a specific integer value or a text string, you may use one of the Composite.Constant functions such as Composite.Constant.Integer or Composite.Constant.String.

To get information related to pages, you should use Composite.Pages functions (for example, to get the current page's GUID, use Composite.Pages.GetPageld).

A number of useful functions are located within the Composite.Utils namespace. For example, you can get the current date and time by calling Composite.Utils.Date.Now.

The Composite.Web functions deal with a web server and web client information and their communication (request-response). For example, you can get a string value of the URL parameter by employing Composite.Web.Request.QueryStringValue.

The Composite.Web.Html.Template functions handle most common template markup tasks such as generating some specific tags or including content from page template features.

(The Xslt.Extensions functions are discussed in [Using XSLT Extension Functions](#).)

In addition to built-in CMS functions, when installed, CMS add-ons also register their own functions within their own namespaces. These functions can be also called from XSLT functions.

As you create your own functions, they also become available for calls. Thus, you can conveniently split the logic between several functions and call them from one parent function.

## 5.3 Data-Centric Functions

Along with the built-in, add-on-based or custom functions, there are functions that C1 CMS generates automatically for each data type created in the system. These are the so-called “data-centric” functions.

The topic of using data-centric functions in function calls is beyond the scope of this guide. The following is just a brief introduction.

The function is placed within the namespace where the data type is, and its name follows this pattern:

Get<Type>XML

where <Type> stands for the name of the data type in question.

You can add the call to such a function and then iterate the data type's fields to get values using XPath. The function itself has a number of parameters by which you can fine tune its output you want in your markup.

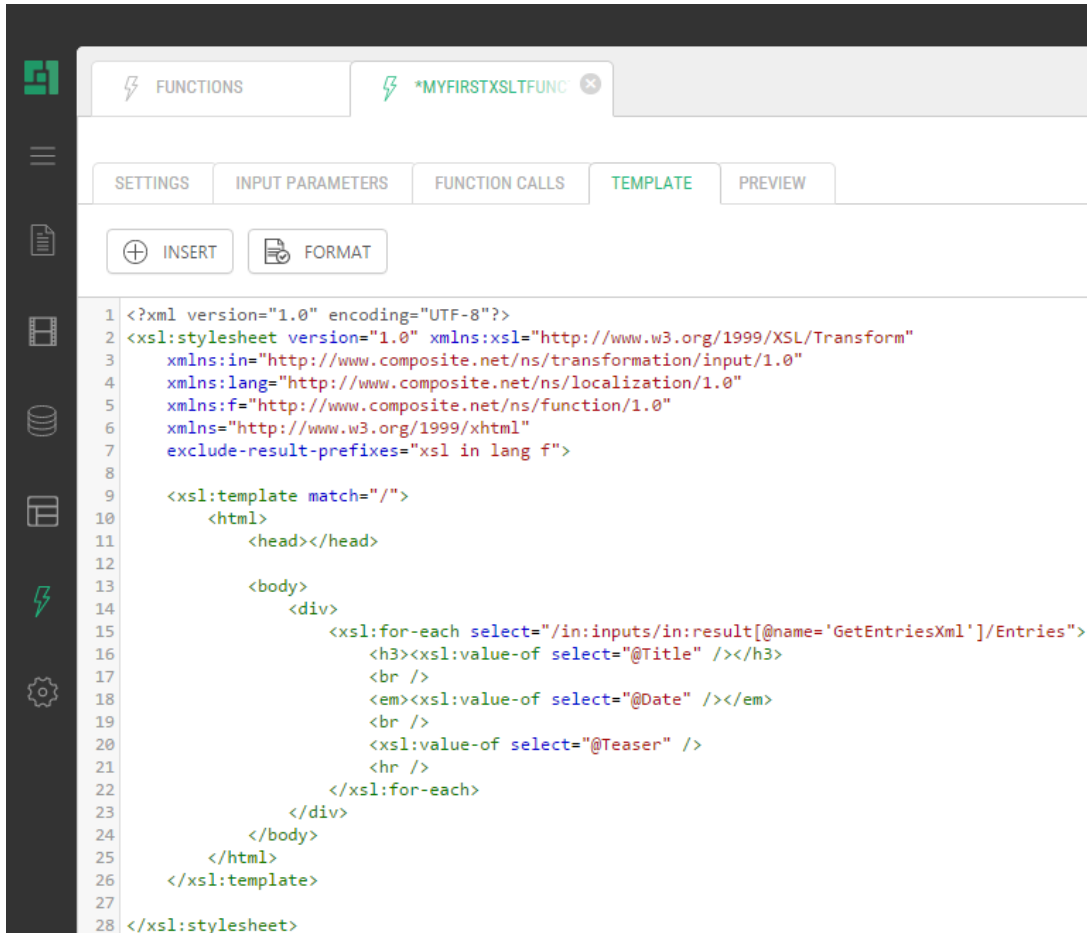
For example, if you have created a global data type named My.Demo.Users, its data-centric function becomes available as My.Demo.GetUsersXml. Having added the call to this function with the local name of GetUsersXml, you can now iterate its data items as follows:

```
<xsl:for-each select="/in:inputs/in:result[@name='GetUsersXml']/Users"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  User Name: <xsl:value-of select="@Name" /> <br/>
  Email Address: <xsl:value-of select="@Email" /> <br/>
</xsl:for-each>
```

Listing 2: Iterating a data type via its data-centric function

## 6 Editing the Template

The key activities of creating and editing an XSLT function take place on the Template tab. This is where the logic of the function is implemented, putting all things together. As this is an “XSLT” function, you write out this logic using standard XSLT.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3   xmlns:in="http://www.composite.net/ns/transformation/input/1.0"
4   xmlns:lang="http://www.composite.net/ns/localization/1.0"
5   xmlns:f="http://www.composite.net/ns/function/1.0"
6   xmlns="http://www.w3.org/1999/xhtml"
7   exclude-result-prefixes="xsl in lang f">
8
9   <xsl:template match="/">
10     <html>
11       <head></head>
12
13       <body>
14         <div>
15           <xsl:for-each select="/in:inputs/in:result[@name='GetEntriesXml']/Entries">
16             <h3><xsl:value-of select="@Title" /></h3>
17             <br />
18             <em><xsl:value-of select="@Date" /></em>
19             <br />
20             <xsl:value-of select="@Teaser" />
21             <hr />
22           </xsl:for-each>
23         </div>
24       </body>
25     </html>
26   </xsl:template>
27
28 </xsl:stylesheet>
```

Figure 10: XSLT markup

Teaching XSLT is beyond the scope of this guide. However, there is some C1 CMS-related XSLT logic that is worth a closer examination.

As you have learned previously, you can add parameters to your XSLT functions to get the user’s input when needed and call other CMS functions to use implement some extra functionality. The next two sections discuss how to use [the input values](#) and [function call results](#) in your template markup.

Besides, you may want to consider [using some extension functions](#) that handle the tasks of formatting content properly or [using C# as a scripting language in your XSLT](#).

### 6.1 Using Input Parameters

You can refer to the parameter in the function’s template markup as the value of the **name** attribute of the **param** element using regular XSLT:

```
/in:inputs/in:param[@name='Input parameter name']
```

For example, if you have a parameter named Count, you can get its value as:



```
<xsl:value-of select="/in:inputs/in:param[@name='Count']"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

## 6.2 Using Function Call Results

You can refer to the function call's result in the function's template markup as the value of the **name** attribute of the **result** element using XSLT:

```
/in:inputs/in:result[@name=' Function call local name']
```

For example, if you call a function named `GetNames`, you can get its output value as:

```
<xsl:value-of select="/in:inputs/in:result[@name='GetNamesXml']"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

Please note that you should specify the local name of the function.

## 6.3 Using XSLT Extension Functions

As you work with some values in your XSLT, you may want to present them to the end user in a specific format.

For example, if you use dates in your markup, it makes sense to make them more human readable or if one of your functions return an XHTML content, you may choose to parse it

You can use three `Composite.Xslt.Extensions` functions that extend your XSLT with formatting capabilities:

- `DateFormatting`
- `Globalization`
- `MarkupParser`

To use these extension functions:

1. Add a [function call](#) to one of the extension functions.
2. Add a corresponding namespace to the `<xsl:stylesheet>` element and specify the prefix.
3. Use one of the formatting functions with the corresponding prefix appended.

```
<xsl:copy-of select="mp:ParseXhtmlBodyFragment (@Content) "
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

Listing 3: Using the extension function `ParseXhtmlBodyFragment`

### 6.3.1 DateFormatting

`Composite.Xslt.Extensions.DateFormatting` provides a number of localized date formatting functions to use in XSLT. The functions format date and time presenting their short and long versions as well as format days, months and years as numbers or strings, format dates following specific date-formatting syntax, and retrieve the current date and time.

To start using date formatting functions, register its namespace - `"#dateExtensions"`:

```
<xsl:stylesheet xsl:df = "#dateExtensions"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Listing 4: Registering date extensions namespace

Now you can use one of the date formatting functions on XML-formatted dates in your XSLT. The following functions are available:

- String ns:Now()
- String ns:LongDateFormat(String xmlFormattedDate)
- String ns:LongTimeFormat(String xmlFormattedDate)
- String ns:ShortDateFormat(String xmlFormattedDate)
- String ns:ShortTimeFormat(String xmlFormattedDate)
- Int32 ns:Day(String xmlFormattedDate)
- Int32 ns:Month(String xmlFormattedDate)
- Int32 ns:Year(String xmlFormattedDate)
- String ns:LongMonthName(Int32 monthNumber)
- String ns:ShortMonthName(Int32 monthNumber)
- String ns:Format(String xmlFormattedDate, String DateFormat)

The names of the above functions are self-explanatory.

```
<xsl:value-of select="df:LongDateFormat (@Now) "
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" />
```

Listing 5: Using date extensions

### 6.3.2 Globalization

Composite.Xslt.Extensions.Globalization provides globalization functions to use in XSLT. There are two kinds of globalization functions. If you keep text strings in the global resource files, you can get these strings with the GetGlobalResourceString function. The other two functions present the long and short names of the month passed by its number.

To start using globalization functions, register its namespace - "#globalizationExtensions":

```
<xsl:stylesheet xsl:ge = "#globalizationExtensions"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Listing 6: Registering globalization extensions namespace

Now you can use one of the globalization functions in your XSLT. The following functions are available:

- String ns:GetGlobalResourceString(String resourceClassKey, String resourceKey)
- String ns:LongMonthName(Int32 monthNumber)
- String ns:ShortMonthName(Int32 monthNumber)

```
<xsl:value-of select="ge:ShortMonthName (@Month) "
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" />
```

Listing 7: Using globalization extensions

### 6.3.3 MarkupParser

Composite.Xslt.Extensions.MarkupParser provides functions that parse encoded XML documents or XHTML fragments into nodes. You should use this extension when you have XML or XHTML as a string and need to copy it to the output or do transformations on it.

To start using markup parsing functions, register its namespace - "#MarkupParserExtensions":

```
<xsl:stylesheet xsl:mp = "#MarkupParserExtensions"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Listing 8: Registering markup parser extensions namespace

Now you can use one of the globalization functions in your XSLT. The following functions are available:

- XPathNavigator ns:ParseWellformedDocumentMarkup(String wellformedMarkupString)
- XPathNavigator ns:ParseXhtmlBodyFragment(String xhtmlBodyFragmentString)

```
<xsl:copy-of select="mp:ParseXhtmlBodyFragment (@Content) "
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

Listing 9: Using markup parser extensions namespace

## 6.4 Using C# Inline Functions

You can create and use C# functions right in your XSLT function using C# as a scripting language that accesses all the capabilities of .NET Framework.

To start using this functionality:

1. Add two namespaces to the <xsl:stylesheet /> element:
  - xmlns:csharp="http://c1.composite.net/sample/csharp"
  - xmlns:msxsl="urn:schemas-microsoft-com:xslt"
2. Add the <msxsl:script> element to the template's markup within which you will add your C# functions.

```
<msxsl:script implements-prefix="csharp" language="C#"
xmlns:msxsl="urn:schemas-microsoft-com:xslt">
<!-- C# functions are defined here -->
</msxsl:script>
```

3. Add references to the assemblies your C# code depends on within the <msxsl:script> element:

```
<msxsl:assembly name="Composite" xmlns:msxsl="urn:schemas-microsoft-com:xslt"/>
```

4. Add **using** directives required by your C# code within the <msxsl:script> element:

```
<msxsl:using namespace="Composite.Data" xmlns:msxsl="urn:schemas-microsoft-com:xslt"/>
```

5. Add the function within the <msxsl:script> element. Keep all your functions within a CDATA element.

```
<div>
<![CDATA[
public string GetCurrentPageId() {
    return Composite.Data.Navigation.CurrentPage.Id.ToString();
}
]]>
</div>
```

6. Now you can directly call the C# function in your XSLT prefixing its name with "csharp" namespace.

```
<xsl:value-of select="csharp:GetCurrentPageId()"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:csharp="http://c1.composite.net/sample/csharp"/>
```

The following is the sample markup that puts all the above things together:

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml"
xmlns:csharp="http://cl.composite.net/sample/csharp"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
exclude-result-prefixes="xsl csharp">
  <xsl:template match="/">
    <html>
      <head />
      <body>
        <div>
          CurrentPageId: <xsl:value-of select="csharp:GetCurrentPageId()" />
        </div>
      </body>
    </html>
  </xsl:template>
  <msxsl:script implements-prefix="csharp" language="C#">
    <msxsl:assembly name="Composite" />
    <msxsl:using namespace="Composite.Data" />
  <![CDATA[
public string GetCurrentPageId(){
    return Composite.Data.Navigation.CurrentPage.Id.ToString();
}
]]>
  </msxsl:script>
</xsl:stylesheet>

```

Listing 10: Using C# as a scripting language in a XSLT function

(For more information on XSLT style sheet scripting with C#, please refer <http://msdn.microsoft.com/en-us/library/533texsx%28VS.71%29.aspx> )

## 7 Previewing Functions

The Preview tab consists of two panes: Input and Output.

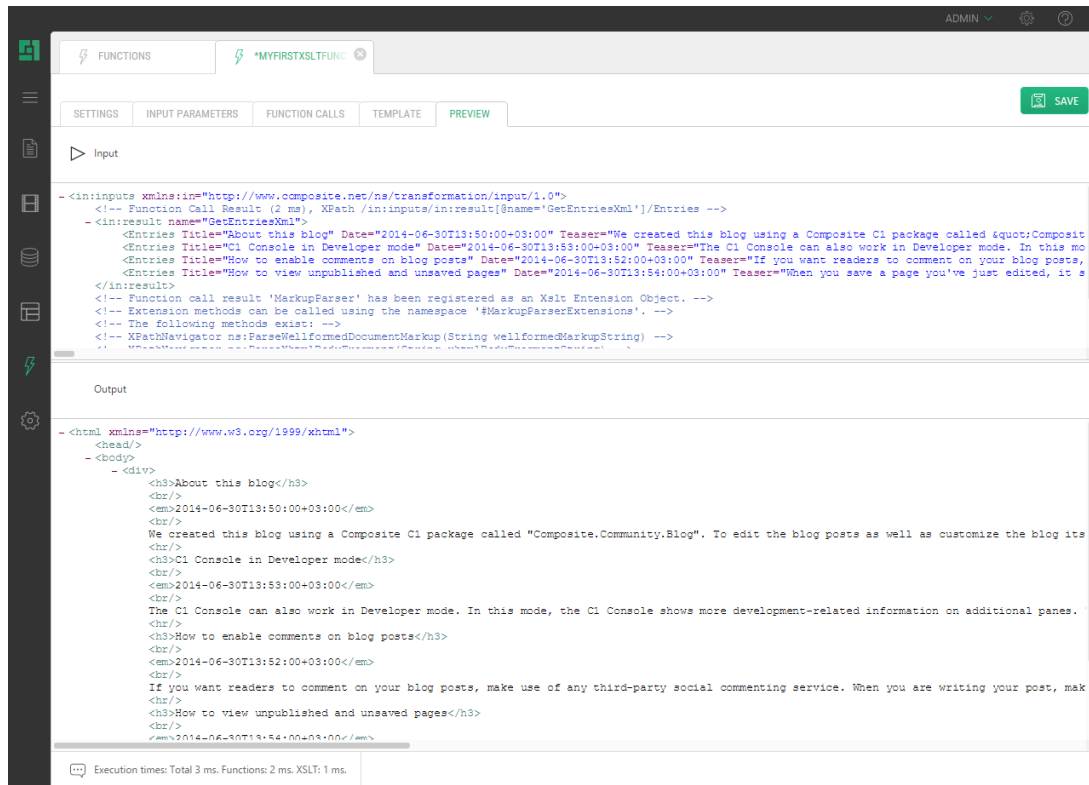


Figure 11: Function preview

### 7.1 Input Preview

In the Input pane, you can see all the inputs used in your functions, which include input parameters with values if any and function call results if any.

```
<in:inputs xmlns:in="http://www.composite.net/ns/transformation/input/1.0">
  <!-- Input Parameter, XPath /in:inputs/in:param[@name='DisplayMode'] -->
  <in:param name="DisplayMode">true</in:param>
</in:inputs>
```

Listing 11: Input parameter

```
<in:inputs xmlns:in="http://www.composite.net/ns/transformation/input/1.0">
  <!-- Function Call Result (65 ms), XPath
  /in:inputs/in:result[@name='GetUsersXml']/Users -->

  <in:result name="GetUsersXml">

  <Users Id="1e4b9093-9047-4e88-8496-e46bcb48d3c1" Name="John Doe"
  Email="john.doe@omnicorp.org" PublicationStatus="published" xmlns=""/>

  <Users Id="b8d9208a-3050-4991-b595-c86ebcd35119" Name="Jane Doe" Email="
  jane.doe@omnicorp.org" PublicationStatus="published" xmlns=""/>

  </in:result>
</in:inputs>
```

Listing 12: Function call results

When you add an input parameter or a function call to your function and switch to the Preview tab, a comment is generated for the input parameter and the function call, showing how to refer them in your XSLT.

The function call's comment also features the function call execution time.

```
<!-- Function Call Result (65 ms), XPath
/in:inputs/in:result[@name='GetUsersXml']/Users -->
```

Listing 13: Function call result with execution time

The comment to extension functions such as `Composite.Xslt.Extensions` also includes a list of the functions you can use in your XSLT.

```
<!-- Function call result 'DateFormatting' has been registered as an
Xslt Extension Object. -->
<!-- Extension methods can be called using the namespace
'#dateExtensions'. -->
<!-- The following methods exist: -->
<!-- String ns:Now() -->
<!-- String ns:LongDateFormat(String xmlFormattedDate) -->
<!-- String ns:LongTimeFormat(String xmlFormattedDate) -->
<!-- String ns:ShortDateFormat(String xmlFormattedDate) -->
<!-- String ns:ShortTimeFormat(String xmlFormattedDate) -->
<!-- Int32 ns:Day(String xmlFormattedDate) -->
<!-- Int32 ns:Month(String xmlFormattedDate) -->
<!-- Int32 ns:Year(String xmlFormattedDate) -->
<!-- String ns:LongMonthName(Int32 monthNumber) -->
<!-- String ns:ShortMonthName(Int32 monthNumber) -->
<!-- String ns:Format(String xmlFormattedDate, String DateFormat) -->
```

Listing 14: List of extension functions

If the input parameter has a value, it will be presented within the `<in:param>` element. If the input function contains data, it will be presented within the `<in:result>` element.

## 7.2 Output Preview

In the Output pane, you can see the markup that your XSLT function will emit when used.

The actual output depends on the [Debug settings](#). For example, if your XSLT requires a specific page to work properly, you should choose a page in the Debug settings.

Wrong debug settings might sometimes account for empty output.

If an error is displayed instead of markup in the output, there might be an error in your XSLT and you need to validate it before previewing the function.

The lower part of the output pane shows the execution time for the entire XSLT function as well as for its constituent parts: functions and XSLT itself.

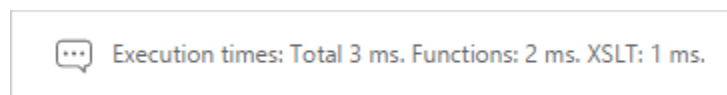


Figure 12: Execution time

## 8 Test Your Knowledge

### 8.1 Task: Create an XSLT Function

1. Add an XSLT function.
2. Specify its name as "ShowCurrentDate".
3. Specify its namespace as "Demo".
4. In the function editor, on the Settings tab, write its description.
5. Save the function.
6. In the Functions perspective, expand All Functions, select Demo and click Generate Documentation on the toolbar.
7. Make sure the function has the description.

### 8.2 Task: Add a Function Call

1. Add a function call to `Composite.Utills.Date.Now`.
2. Specify its local name as "CurrentDate".
3. Save the function.

### 8.3 Task: Edit the Template

1. On the Template tab, remove the default contents between the `<body>` and `</body>` tags.
2. Use the called `Composite.Utills.Date.Now` function's output in the markup by referring to it by its local name "CurrentDate".
3. Save the function.
4. Preview the function and make sure that the current date appears in the input and output panes.

### 8.4 Task: Add an Input Parameter

1. Add the Boolean parameter "LongDate".
2. Edit its default value.
3. Add `Composite.Constant.Boolean` and set its value to "True".
4. Save the function.

### 8.5 Task: Use an Extension Function

1. Add a call to the extension function `Xslt.Extensions.DateFormating`.
2. On the Template tab, register the `'xsl:df = "#dateExtensions"'` namespace.
3. Use the `'df: LongDateFormat'` extension function on the function call's output in the template.
4. Save the function.
5. Preview the output of the function and make sure the long date appears now.

### 8.6 Task: Use XSLT to Switch between Two Date Formats

1. Use the XSLT syntax to have two branches for the current date's format (`<xsl:choose>` etc).
2. Use the input parameter "LongDate" and its value ('true', 'false') to specify each branch of the current date's format (`<xsl:when test="...">` etc).
3. For the "LongDate" set to 'true' use the `'df:LongDateFormat'` extension function on the "CurrentDate" function's output
4. For the "LongDate" set to 'false' use the `'df:ShortDateFormat'` extension function on the "CurrentDate" function's output.

5. Save the function.
6. Insert the function on a page in the Content perspective.
7. Set its LongDate parameter to 'false'.
8. Preview the page and make sure that the date is displayed in short format.