A Guide to XML Templates in C1 CMS

2017-02-14

Orckestra, Europe
Nygårdsvej 16
DK-2100 Copenhagen
Phone +45 3915 7600

http://c1.orckestra.com

Contents

1 Introduction

This document introduces key concepts and tools in C1 CMS that front-end developers are required to be familiar with in order to build customized sites with C1 CMS.

It can be used both as reading material to hand out while conducting trainings on C1 CMS and a self-study guide to learning C1 CMS.

1.1 Who Should Read This Guide

This document is an introduction to C1 CMS for front-end developers who want to learn to create and work with XML layout templates in C1 CMS.

However, part of this document might be useful to a broader audience. Not everyone is required to know the technical details, but it helps when you know the concepts on a more detailed level.

This document assumes that you have basic knowledge and/or experience in working with C1 CMS and know how to create a new empty site.

C1 CMS uses a number of open web standards, so you are supposed to have at least basic understanding of XML, XHTML, XSLT and CSS.

Since C1 CMS is based on a number of Microsoft technologies, you will want to have an experience with Internet Information Services, Visual Studio and .NET Framework, C#, ASP.NET, Windows Workflow Foundation, .NET User Controls.

1.2 System Requirements

While working with this document, you should have access to a working C1 CMS environment. It should be one of the following options:

- (minimum) A laptop or PC that meets requirements to install C1 CMS (see below).
- (preferred) A laptop or PC that has C1 CMS already installed and running.
- (in some cases) Web access to an external C1 CMS installation (one website per attendee who wants to complete hands-on exercises)

(See “C1 CMS System Requirements” for more information on the requirements.)

1.3 Document Overview

In **Introduction**, you will get an overview of what this document is about, and learn what prerequisites must be in place.

The Anatomy of a C1 CMS Web Page gives you a quick introduction to what a standard C1 CMS page is and what the role a C1 CMS template plays in creating pages.

In **Creating Layout Templates in C1 CMS**, you will learn how to create XML templates in C1 CMS.

A Closer Look at the Template Markup will thoroughly examine the core C1 CMS template elements you can use when creating or editing templates.

Editing Templates focuses on how to use a built-in C1 CMS template editor to edit templates as well as what you need to know to edit templates in external editors. It also discusses how to use external resources within your templates such as style sheets (CSS), JavaScript and media files.

Localizing Templates covers the topic of template localization on multiple-language websites in C1 CMS.

In **Test Your Knowledge**, you can check how good you have mastered the topics discussed in this document.

1.4 What You Will Learn

When you finish the document, you should know:

- How to create XML layout templates in C1 CMS
- What core C1 CMS template elements to use in XML templates
- How to edit templates in C1 CMS built-in XML template editor
- What it takes to edit XML templates in external editors
- How to use external CSS, JavaScript and media in XML templates
- How to localize XML templates

1.5 References and Links

For further information on C1 CMS's concepts, tools, packages, visit http://docs.c1.orckestra.com where you can read many articles and tutorials on various C1 CMS's subjects.

C1 CMS's FAQ is another source of information you can refer to, available at Layout FAQ.

You can always ask questions or look for existing information on our forums at: http://compositec1.codeplex.com/discussions.

If you are a Composite's partner, you can also sign up and report your issues at http://support.composite.net.

For information about standards and technologies used in C1 CMS such as XML, you should use external sources (e.g. http://www.w3schools.com).

2 The Anatomy of a C1 Page

A web page in C1 CMS is constructed from the following elements:

- A page object containing titles, page metadata and content elements
- A layout template defined using XHTML / Razor or Master Pages and C1 CMS-related markup or code
- Functions that allow reusing some of the content and create dynamic elements such as navigation tools or news lists

Please note that you can create layout templates based on XMLXHTML, Razor or Master Pages. This guide entirely focuses on creating XML templates.

2.1 The Page Object, Contents and Metadata

C1 CMS is partly a page-based CMS, which means that the system comes with a core set of data objects and features that predefine what a page is in its minimal form. You can extend on this model by adding content areas and metadata to pages. And use the extended data in layout templates and functions to create required designs and features for the website.

2.2 XML Layout Templates

Each page has one layout template. Layout templates are shared across pages and define the overall layout of a page. XML-based templates typically contain the backbone XHTML markup of the page, placeholders for page content as well as function calls to dynamic elements such as navigation tools or news lists.

2.3 Functions

Functions are a key part of C1 CMS. While page objects and layout templates only deliver data, functions are what deliver dynamically generated output to pages and as a front-end developer you will both use, and create, functions when building websites.

Typical examples of functions that deliver page output are:

- Navigation elements, bread crumbs, sitemaps, menus
- Contact Forms
- Image and document libraries
- News lists, RSS feeds
- Teasers
- Search results

Both content editors and developers can embed functions on pages using either a visual Function Designer or XML markup.

(You can read more on C1 CMS functions in “A Guide to CMS Functions”.)

3 Creating XML Templates

To create pages, you should first create templates. To make templates, you take your working design, the XHTML pages, and transform them into templates.

Let's start with a simple design.

To create a template:

1. In the **Layout** perspective, select **Page Templates** and click **Add Template** on the toolbar.

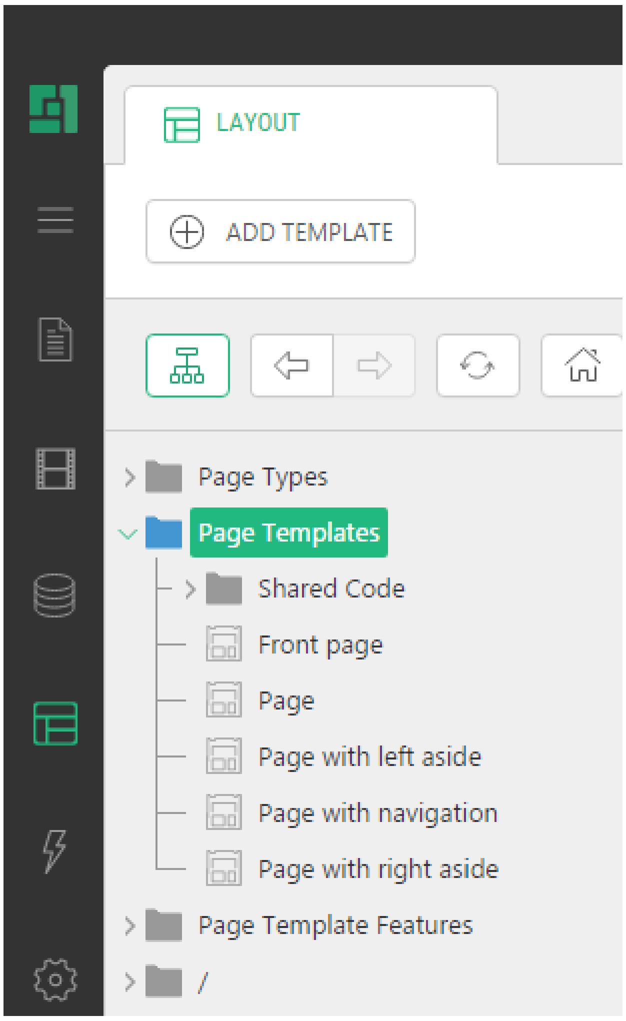
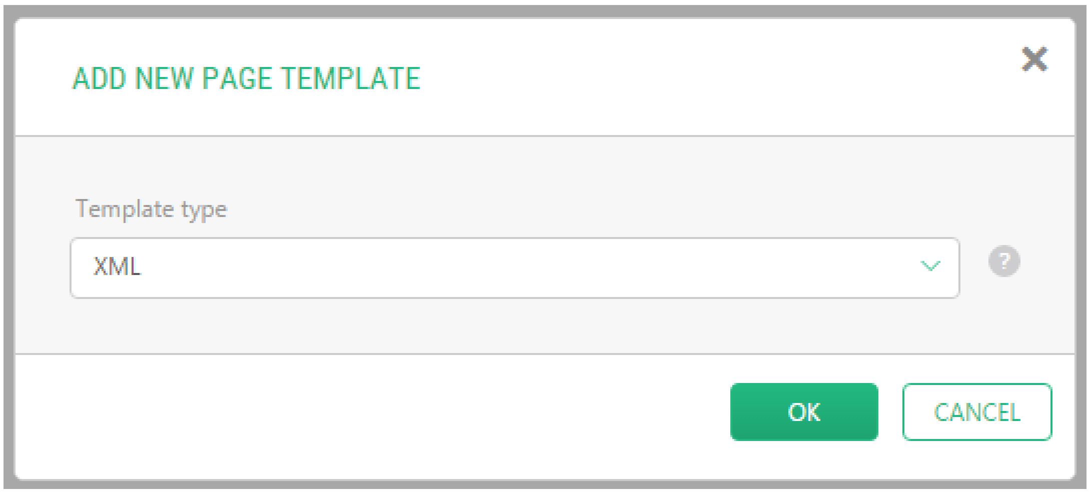


Figure 1: Adding a new website template

2. For the **Template type**, select *XML*.

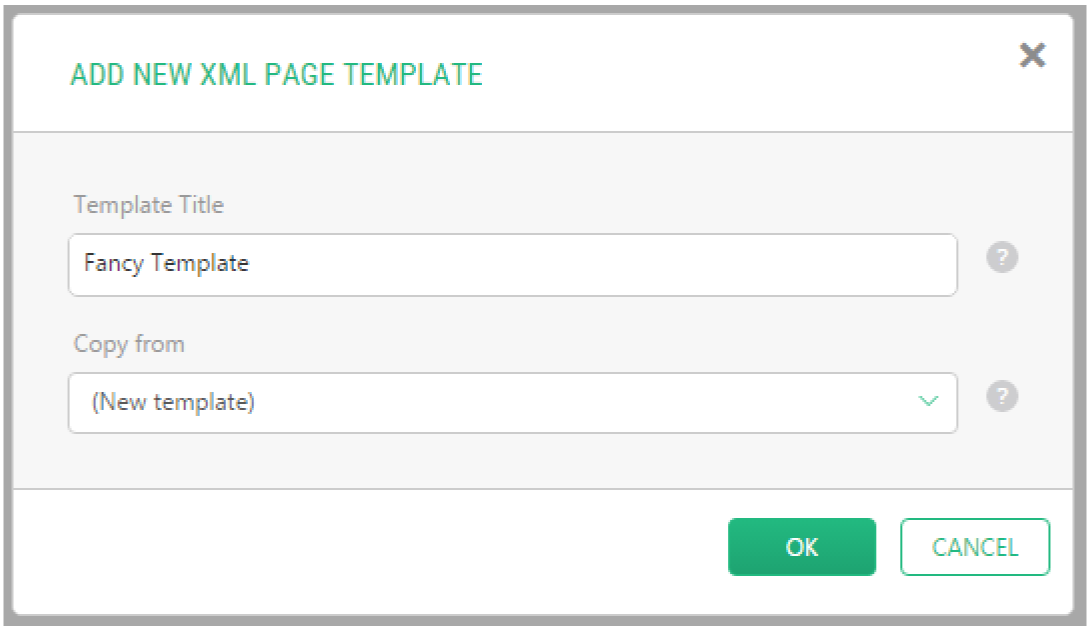

ADD NEW PAGE TEMPLATE

Template type

XML

Figure 2: Selecting XML as the template type

3. Enter the title of your template in the **Template Title** field.


ADD NEW XML PAGE TEMPLATE

Template Title

Fancy Template

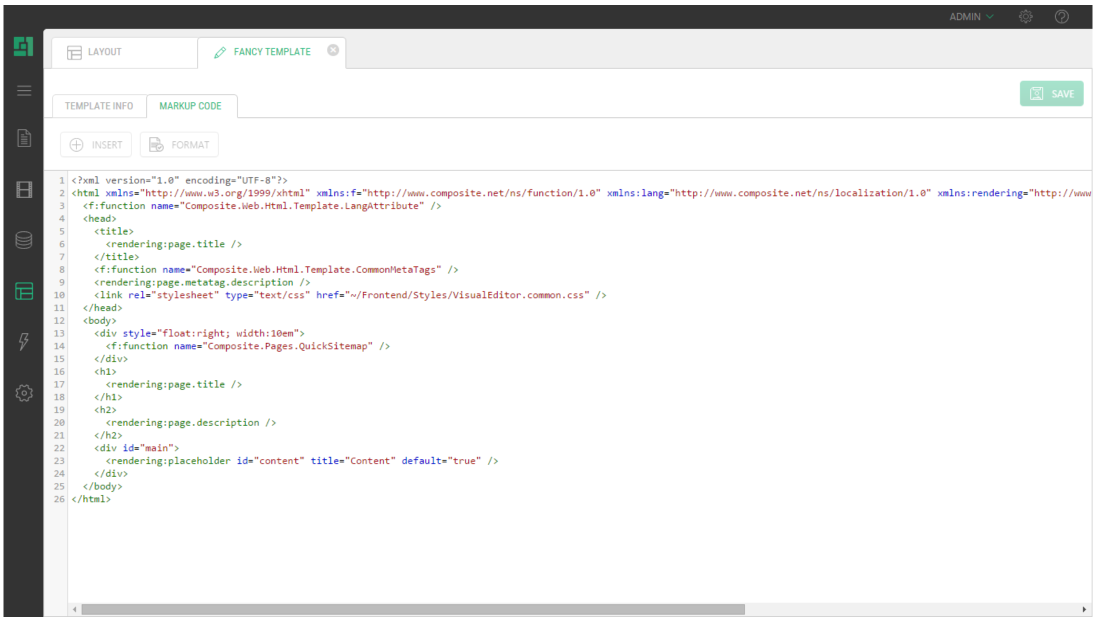
Copy from

(New template)

Figure 3: Specifying the template's title

4. Keep "(New template)" in the Copy from field if you want to create a new template. Otherwise, select an existing XML template to copy the markup from.
5. Click **OK**.

The new template opens in the working area and switches to the **Markup Code** tab.



```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="http://www.composite.net/ns/function/1.0" xmlns:lang="http://www.composite.net/ns/localization/1.0" xmlns:rendering="http://www
  <f:function name="Composite.Web.Html.Template.LangAttribute" />
  <head>
    <title>
      <rendering:page.title />
    </title>
    <f:function name="Composite.Web.Html.Template.CommonMetaTags" />
    <rendering:page.metatag.description />
    <link rel="stylesheet" type="text/css" href="~/Frontend/Styles/VisualEditor.common.css" />
  </head>
  <body>
    <div style="float:right; width:10em">
      <f:function name="Composite.Pages.QuickSitemap" />
    </div>
    <h1>
      <rendering:page.title />
    </h1>
    <h2>
      <rendering:page.description />
    </h2>
    <div id="main">
      <rendering:placeholder id="content" title="Content" default="true" />
    </div>
  </body>
</html>
```

Figure 4: Markup code of the newly created template

When creating a new template, C1 CMS inserts some very basic template markup for you.

The generated default template code might look as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://www.composite.net/ns/function/1.0"
xmlns:lang="http://www.composite.net/ns/localization/1.0"
xmlns:rendering="http://www.composite.net/ns/rendering/1.0"
xmlns:asp="http://www.composite.net/ns/asp.net/controls">
  <f:function name="Composite.Web.Html.Template.LangAttribute" />
  <head>
       <title>
         <rendering:page.title />
       </title>
       <f:function name="Composite.Web.Html.Template.CommonMetaTags" />
       <rendering:page.metatag.description />
       <link rel="stylesheet" type="text/css"
href="~/Frontend/Styles/VisualEditor.common.css" />
  </head>
  <body>
       <div style="float:right; width:10em">
         <f:function name="Composite.Pages.QuickSitemap" />
       </div>
       <h1>
         <rendering:page.title />
       </h1>
       <h2>
         <rendering:page.description />
       </h2>
       <div id="main">
         <rendering:placeholder id="content" title="Content" default="true"
/>
       </div>
  </body>
</html>
```

Listing 1: Default markup code of the newly created template

This is a valid and working template. You can save it and create pages that use it.

Let’s have a quick look at the code.

In the <html> element, you can see 4 namespaces added:

- xmlns:f=http://www.composite.net/ns/function/1.0
- xmlns:lang="http://www.composite.net/ns/localization/1.0"
- xmlns:rendering=http://www.composite.net/ns/rendering/1.0
- xmlns:asp=http://www.composite.net/ns/asp.net/controls

This ensures that our template should be valid with the use of the specific C1 CMS XML elements for rendering, functions and so on.

You might have also noticed that the code is written in XHTML, not HTML. (Read more on the namespaces and XHTML in the chapter "A Closer Look at the Template Markup".)

When you look further, you will notice some specific C1 CMS elements using the "rendering" namespace:

- <rendering:page.title />
- <rendering:page.description />
- <rendering:placeholder id="content" title="Content" default="true" />

These and other core C1 CMS template elements are explained in the following chapter.

4 A Closer Look at the Template Markup

As you already know, C1 CMS inserts some very basic template markup for you in a newly created XML template. Let’s have a closer look at each major element of this inserted markup.

4.1 XML Namespaces and XHTML

C1 CMS utilizes XML namespaces to identify different types of markup and it is highly recommended that developers have a good conceptual understanding of XML namespaces when working with front-end development in C1 CMS.

4.1.1 XML Namespaces

In short, XML namespaces are a mechanism that allows a computer program (and humans) to uniquely distinguish XML elements from each other, even in XML documents that contain a mix of data from different subsystems where element names might clash. By declaring that an element belongs to a certain namespace you typically declare that it should be handled by a specific subsystem, making other subsystems take a hands-off approach to that particular element.

If you examine the markup of a page template, you will notice that the <html /> element contain a lot of namespace declarations in the form of xmlns=“(URL1)” and xmlns:xxx=“(URL2)”. These items define that ‘from this element and down, elements with no prefix belong to the namespace (URL1), and elements with the prefix “xxx” belong to the namespace (URL2)’. “

An element with no prefix looks like this <form /> while a prefixed element looks like this <xxx:form />. Both of them can belong to a namespace, though.

```
<sample xmlns="http//collection/" xmlns:a="http://animal/"
xmlns:p="http://plant/">
       <title>This belongs to the collection namespace</title>
       <a:title>This belongs to the animal namespace</a:title>
       <p:title>This belongs to the plant namespace</p:title>
</sample>
```

Listing 2: Sample of using namespaces

In the XML snippet above you can see how the same element “title” can be used by multiple subsystems without interfering with each other. The “animal” subsystem can simply ignore elements not bound to the “http://animal/” namespace.

4.1.2 XHTML Namespace

One of the most fundamental namespaces in C1 CMS is the XHTML namespace, http://www.w3.org/1999/xhtml . Simply stated, XML templates should always use XHTML as their overall structure in order to allow the full feature set of C1 CMS to be utilized.

The minimal structure of a page template’s markup code should look like this:

```
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
              <title>...</title>
       </head>
       <body>...</body>
</html>
```

Listing 3: Minimal structure of a template’s markup

One of the features of C1 CMS is the ability to inject markup (for example, script and CSS declarations and includes) into a page’s <head /> section from sub-functions written in

Razor, ASP.NET, XSLT and more. This is highly convenient for front-end developers since you do not need to change the central page templates and can keep your code and markup in one place.

In order to provide this service, C1 CMS must be able to identify the resulting pages <head /> element with 100% certainty and to do so a well-formed XHTML document is expected.

(You can read more on this in "A Guide to XSLT Functions".)

4.1.3 XHTML and Validation

Every result part rendered by C1 CMS must be a well-formed XML document. C1 CMS validates the results and throws exceptions when they are not valid. If the output of the XML template is not structured as an XHTML document, all XHTML-related services in C1 CMS are automatically disabled.

XHTML has the same depth of expression as HTML, but also conforms to XML syntax. This means XHTML can be validated. To most, XHTML will feel stricter than HTML.

(Please refer to the standard documentation on XML and XHTML for more information.)

4.2 The Role of the id Attribute

The id attribute on XHTML elements is actively used by C1 CMS in the following ways:

- The value of the id attribute must be unique – two elements cannot have the same ID.
- An XHTML container such as <div /> can be assigned an id, which will automatically make it available to ASP.NET developers (see the section on <asp:placeholder />).
- XHTML elements (for example, style or script includes) added to a page's <head /> section will be "replaced" by <head /> element output, if the same ID is used. (For more information about replacing <head /> elements, please refer to "A Guide to XSLT Functions".)

4.3 Core XML Template Elements

Apart from having the overall layout of an XHTML document, XML templates also contain markup that controls the placement of content and functionality. In the following few sections is a description of the core elements you can use in XML templates.

Please note that the namespace prefixes in the tag names used in this section expect the following namespace declarations in your template:

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0"
       xmlns:f="http://www.composite.net/ns/function/1.0"
       xmlns:lang="http://www.composite.net/ns/localization/1.0"
       xmlns:asp="http://www.composite.net/ns/asp.net/controls">
       ...
</html>
```

Listing 4: Namespaces required by core C1 CMS template elements

4.3.1 rendering:page.title

The <rendering:page.title /> element inserts the title of the page being rendered. The value is user-defined and can be set on the Settings tab when editing a page.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0" >
       <head>
              <title><rendering:page.title /></title>
       </head>
       <body>
              <h1><rendering:page.title /></h1>
       </body>
</html>
```

Listing 5: <rendering:page.title /> used

4.3.2 rendering:page.description

The <rendering:page.description /> element inserts the description of the page being rendered. The value is user-defined and can be set on the Settings tab when editing a page.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0" >
       <head>
              <title><rendering:page.title /></title>
       </head>
       <body>
              <h1><rendering:page.title /></h1>
              <rendering:page.metatag.description />
       </body>
</html>
```

Listing 6: <rendering:page.description /> used

4.3.3 rendering:placeholder

This element defines a unique content placeholder. When rendered, the content for the current page will be inserted at its place. You can have multiple (or zero) <rendering:placeholder /> elements in your template.

When editing pages based on a particular XML template, the system will automatically show WYSIWYG editors for each <rendering:placeholder /> element defined in the template's markup. You can thus add a new content area to a template by simply adding a new <rendering:placeholder /> element to it, and pages based on this template will automatically support the new placeholder.

```
<rendering:placeholder
       id="{a locally unique ID}"
       title="{a human readable title}"
       default="{true | false}"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0"/>
```

Listing 7: <rendering:placeholder /> with attributes

When adding a new <rendering:placeholder /> element, you should set its mandatory attributes in a proper way:

- The **id** attribute uniquely identifies the placeholder and ensures that you can move the element around and still keep a relation to the existing content defined on pages.
  Note: If you have multiple templates, it is highly recommended that you should use the same IDs across those templates. This ensures that the user editing a page can switch between different templates and move the content seamlessly to the new template's placeholders.
- The **title** attribute serves as the label for a template in the page's WYSIWYG editor.
- The **default** attribute defines which placeholder should be shown by default when users edit a page. Only one placeholder can have the default attribute set to 'true'.

Note that when the <rendering:placeholder/> element is rendered, the page content is wrapped in a ASP.NET Placeholder Control with the specified ID, making it possible to read and manipulate the content from ASP.NET code.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0" >
       <head>
          <title>...</title>
       </head>
       <body>
          <rendering:placeholder id="content" title="Content" />
       </body>
</html>
```

Listing 8: <rendering:placeholder /> used

For two content areas you should use different IDs and titles:

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0" >
       <head />
       <body>
       <rendering:placeholder id="content" title="Content" default="true"/>
       <rendering:placeholder id="contentright" title="Content (Right)"
default="false" />
       </body>
</html>
```

Listing 9: Creating two content areas

The IDs and titles should be unique within the template.

The specific content areas, where an editor can type XHTML in, fully depend on the template related to the page.

4.3.4 f:function

This element defines that a C1 function should be invoked and its result - inserted in its place. The <f:function /> elements can invoke Razor functions, XSLT functions, User Control functions, Visual functions, C# functions and SQL functions. The function invoked depends on the name specified.

You can provide parameter values to functions by nesting <f:param /> elements within the <f:function /> element, and even nesting other function calls or placeholders in parameters:

```
<f:function name="…" xmlns:f="http://www.composite.net/ns/function/1.0">
       <f:param name="…" value="…" />
       <f:param name="…">
              <f:function name="…" />
       </f:param>
       <f:param name="…">
              <rendering:placeholder id="…"
       xmlns:rendering="http://www.composite.net/ns/rendering/1.0"/>
       </f:param>
</f:function>
```

Listing 10: A sample of a function call with parameters nesting another function call

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:f="http://www.composite.net/ns/function/1.0" >
       <head>
              <title>...</title>
       </head>
       <body>
              <f:function name="Composite.Pages.QuickSitemap" />
       </body>
</html>
```

Listing 11: A simple function call

(You can read more on using functions in “A Guide to CMS Functions”.)

4.3.5 lang:string

This element defines that a localized string – a string matching the language of the current page request - should be retrieved from a string resource repository and inserted in its place. The value of the key attribute defines which repository and string to use.

Using the <lang:string /> element makes sense when you would like to keep all strings specific to languages in a central repository for easy language versioning.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:lang="http://www.composite.net/ns/localization/1.0" >
       <head>
              <title>...</title>
       </head>
       <body>
              <lang:string key="Resource, Resources.Resource.Name" />
       </body>
</html>
```

Listing 12: <lang:string /> used

Please refer to the chapter “Localizing Templates“ in this guide for more information.

4.3.6 lang:switch

This element defines that a specific string or markup fragment should be selected based on the language of the page being rendered. This enables you to write localized content inline in your markup.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:lang="http://www.composite.net/ns/localization/1.0" >
       <head>
              <title>...</title>
       </head>
       <body>
              <lang:switch>
                     <lang:when culture="da-DK">
                            <img src="~/dk-logo.png" title="Dansk logo"/>
                     </lang:when>
                     <lang:when culture="en-US">
                            <img src="~/us-logo.png" title="American logo"/>
                     </lang:when>
                     <lang:default>
                            No logo available
                     </lang:default>
              </lang:switch>
       </body>
</html>
```

Listing 13: <lang:switch /> used

Each localized version of the content should be placed within the <lang:when/> element. The <lang:switch/> element can contain as many <lang:when/> elements as you need – normally for each language installed on the website.

In case a language is used, which is not represented with one of the <lang:when> element, the system will use the default content, which you place within the <lang:default/> element. There can be only one <lang:default/> element in <lang:switch/>.

You can use as many <lang:switch/> elements as you need within a template.

4.3.7 asp:form

This element defines that an ASP.NET Form should be created and inserted on a page. ASP.NET Controls that must run inside an ASP.NET form should be placed within this element.

Typically it makes sense to have the <asp:form /> tag as a normal part of a template, ensuring that ASP.NET features are always available.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:asp="http://www.composite.net/ns/asp.net/controls" >
       <head>
              <title>...</title>
       </head>
       <body>
              <asp:form>
                     <h1>ASP.NET post-back supported here</h1>
              </asp:form>
       </body>
</html>
```

Listing 14: <asp:form /> used

4.3.8 asp:placeholder

This element defines that an ASP.NET Placeholder with the specified ID should be inserted in its place. ASP.NET code running on the page will be able to get a reference to the placeholder by its ID using the ASP.NET’s this.Page.FindControl(id) method and modify the content of this section of the page.

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:asp="http://www.composite.net/ns/asp.net/controls" >
       <head>
              <title>...</title>
       </head>
       <body>
              <asp:form>
              <asp:placeholder id="info">
                     <h1>this.Page.FindControl("info") finds me!</h1>
              </asp:placeholder>
              </asp:form>
       </body>
</html>
```

Listing 15: <asp:placeholder /> used

5 Editing XML Templates

When editing XML layout templates, you can use the built-in XML template editor in C1 CMS as well as any external XML-based editors.

Since a template in C1 CMS mostly includes the content reused on multiple pages, it makes sense to place all the reusable style, script or media code or link to them within the template.

While adding the markup requires that you directly edit the template markup code, which is nothing else but XHTML, keeping the CSS and JavaScript in external files as well as inserting images requires that you know where to put these files and how to link to them within a template.

5.1 Location of Templates

Apart from accessing the templates from within C1 CMS's backend, you can also access the templates on the file level of the web server. This allows you to use other tools, such as Visual Studio, your favorite text editor or other relevant editing tools.

It also enables you to source-control templates in your CVS system such as Team Foundation in the Visual Studio environment.

The templates are accessible in the following directory:

<website_directory>\ App_Data\PageTemplates

For example:

C:\inetpub\website\App_Data\PageTemplates

Note: This is also why a good name is important. The title of the template can be changed afterwards, but the initial title will be used to create the template file and that filename will not change.

5.2 Where to Put Style Sheet, Script and Media Files

It is recommended that you put web-development related files (template images, style sheets, scripts) directly on the file system, and not in the Media Archive. Putting your web-developer related files on the file system allows you to use most editors to work with the files.

You can put your template-related files (scripts, style sheets and images) on the file system where you desire as long as you stay clear of the ASP.NET and application specific folders such as ~/App_Data, ~/App_Code, ~/bin and ~/Composite.

By default C1 CMS is promoting that you place your frontend-related files in the ~/Frontend folder, but you are free to put your items in other locations. However, a good approach is to create an ad-hoc folder under ~/Frontend and keep your files there. If necessary, you can also create 3 subfolders therein such as Styles, Scripts, and Image.

Predefined functionality such as navigation or news lists installed through the C1 CMS Package system will typically put all frontend-related files within the ~/Frontend folder.

You can also create a hierarchy of folders that reflects some hierarchy of namespaces that you might follow. Packages in C1 CMS normally follow the hierarchy here. For example, the Composite.Feeds.RssReader related files are placed in the following folder:

~/Frontend/Composite/Feeds/RssReader

5.2.1 How to Show File Folders on Layout

In the Layout perspective in the C1 CMS administrative console access to the ~/Frontend folder is active by default.

You can control what other file folders are available here, too:

1. In the **System** perspective, expand the “/” node.
2. Browse to the folder you want to make available in Layout.
3. Right-click the folder to open its context menu.
4. In the menu, click to set the check mark next to the **Show in Layout** menu item. This will make the folder available in Layout.

To remove the folder from Layout:

- Click the **Show in Layout** again to clear the check mark.

5.3 Editing Templates in C1 CMS Built-In Editor

C1 CMS comes with a built-in XML template editor which gives you access to the template markup code and a number of operations available via its menu.

When editing an XML template, you can use XHTML to create the content directly, insert content placeholders, insert different kinds of links including links to external CSS and script files and insert C1 CMS functions.

5.3.1 Inserting Page, Image and Media URLs

You can create links to pages or insert images and other media files available on your website by inserting their URLs in the corresponding XHTML tags (e.g. <a href=“…”>, <img src=“…”>) used in the template.

1. In the XML template, place the cursor where the attribute value is in the relevant tag (e.g. *href* in <a>)
2. On the template’s menu bar, click **Insert**.
3. In the dropdown menu, click one of the following:
   - Page URL
   - Image URL
   - Media URL
4. In the window that appears, browse to, and select, the file you need and click **OK**.

The URL will be inserted where the cursor was in the form required by C1 CMS, for example:

~page(4b647577-f326-4ab2-a8ee-d7c2b75649ce)

~media(aa1082d0-807f-46b0-9cd2-81e9125ef35a)

When published, the page URLs will be rendered to more user-friendly form on the front-end (in the user’s web browser), for example:

/News/Archive

When you click Page URL on the Insert menu, you will access your website page structure and can select an individual page. The Image URL accesses folders with images files of various formats in the Media Archive, which you can preview immediately. The Media URL gives you access to media files that include zip files, PDF files, Flash files and other formats.

5.3.2 Inserting Frontend URLs

When editing your XML template, you are not limited by the template markup itself. You can link to external CSS and JavaScript files as well as media files available on your web server, which will customize the appearance and behavior of the pages based on this template.

Before you insert an external link, make sure that you have uploaded or created the required file under the ~/Frontend folder where necessary.

To insert external link:

1. In the XML template, place the cursor where the attribute value is in the relevant tag (e.g. *href* in <link>)
2. On the template's menu bar, click **Insert**.
3. In the dropdown menu, click **Insert Frontend URL**.
4. In the window that appears, browse to and select, the file you need and click **OK**.

The URL will be inserted where the cursor was in the form required by C1 CMS, for example:

/Frontend/Styles/Layouts/Enhanced.css

When you click Frontend URL on the Insert menu, you will access files and folders available under ~/Frontend.

5.3.3 Inserting Function Markup

As it was mentioned above, functions play a key role in C1 CMS. Not only can you use a function to add reusable content but also render dynamic content. You can insert functions in XML templates as markup.

To insert a function:

1. In the XML template, place the cursor where you want the function-rendered content to appear (if required).
2. On the template's menu bar, click **Insert**.
3. In the dropdown menu, click **Insert Function Markup**.
4. In the window that appears, browse to, and select, the function you want to insert and click **OK**.

This will insert the function markup in the XML template in the form required by C1 CMS, for example:

```
<f:function xmlns:f="http://www.composite.net/ns/function/1.0"
name="Composite.Media.FlashViewer">
       <f:param name="FlashFile" value="MediaArchive:b261615c-6d3d-488b-
b402-75691cd4b8d5" />
</f:function>
```

Listing 16: The function markup inserted

You can use functions to encapsulate logical parts of your design. For example, your template may consist of:

- The header
- The navigation pane or bar
- The content part or parts
- The footer

If some of these parts can be shared between multiple templates (for example, the header, the navigation, the footer), you may want to consider moving their content or logic to functions and use these function instead in the template.

(You can read more on functions in "A Guide to CMS Functions".)

5.3.4 Formatting and Validating the Markup

As you continue editing the XML template, you add or edit more and more XHTML elements in its markup. At some point, it might be hard to read especially for another front-end developer. You can make the markup more readable by formatting it. It will use indentation and properly nest elements in the markup.

To format the template's markup:

- On the template's menu bar, click **Format**.

Not only will the Format command properly align the XHTML elements but also validate the XHTML code. When encountering a validation error, the built-in XHTML validator will inform you of its type and place in the code so that you can review the code and correct it.

5.4 Styling with CSS

All layout-related markup emitted by C1 CMS is under the control of the front-end developer, including markup related to styling. C1 CMS does not come with a proprietary styling system and front-end developers can leverage their existing XHTML, CSS and JavaScript knowledge when building sites.

When editing an XML template, you have full access to the structural markup. You can thus specify inline styling or include style sheets in the <head /> section or put styling on individual elements as desired.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
       <head>
              <title>...</title>
              <link rel="stylesheet" href="~/screen.css" type="text/css"
media="screen" />
              <link rel="stylesheet" href="~/print.css" type="text/css"
media="print" />
              <style type="text/css">
                     div#infotext { padding: 5px; }
              </style>
       </head>
       <body>
              <div style="border: 1px solid black" id="infotext">
                     <h1>Includes, head defined and inline styles are all
possible</h1>
              </div>
       </body>
</html>
```

Listing 17: Using CSS inline, included and on individual elements

5.4.1 Where to Place CSS Markup

One of the unique features of C1 CMS is the ability to let functions add elements to the page's <head /> section such as <link /> and <style />. This feature is available to XSLT and ASP.NET developers and it gives you the choice of putting style-related markup directly into the XML template or putting it in the function code, and having it appended to the page markup when needed.

Placing style-related markup in a function code makes a lot of sense. Since the markup will automatically be added to the <head /> section when pages with functions are rendered, the net result is that you can keep the template's markup simple and focused on the overall layout, and let the function code handle its own styling. You can put function-specific styling and scripts directly into the XML template, but this can make templates and functions harder to maintain for other developers.

In short, you should put functionality-specific styling in the function that emits its markup and only put overall and cross-functional styling in templates. Determining if a piece of CSS relates to a specific function or not can be easily answered by the question “does this CSS selector work on elements emitted by a Function or does it work on elements in my Template?”.

(You can read more on functions in “A Guide to CMS Functions”.)

5.5 Resolving Paths to Files by Using Tilde-Based Paths

The page sitemap in C1 CMS is built as a hierarchical structure, mapping 1:1 with the page structure in the Content navigator. Pages that are nested at deeper levels will have a URL that reflects the path to the page, like “/Products/HybridCars”. This makes it impossible to use relative paths in your XHTML.

For instance an image reference like <img src=“images/logo.png” /> on the before mentioned page would translate to the path “/Home/Products/images/logo.png”, which not a valid file reference.

All files paths you specify in markup should be absolute paths, meaning that they should start with “~/”. The “tilde slash” sign (~/) is caught by C1 CMS and translated into the correct path depending on whether C1 CMS is running in the web root or in a sub folder.

If C1 CMS is running as the root application, “~/images/logo.png” will be automatically translated to “/images/logo.png”.

If C1 CMS is running in a sub-folder “SubSite1”, “~/images/logo.png” will be automatically translated to “/SubSite1/images/logo.png”.

Typically C1 CMS will be running as the root application and you will experience that using “/images/logo.png” (without “~/”) will work just fine, but you should consider using the tilde-based path since it will ensure that the site can be moved back and forth to sub-folders without breaking links.

Moving a site to and from a sub-folder could make sense in development scenarios and users using some operating systems as the web server are forced to run C1 CMS in a sub-folder.

6 Localizing XML Templates

In the section discussing Core XML Template Elements, you have been introduced to two localization-related elements that you can use in XML templates on a website that has multiple-language versions (localizations).

In addition, you can use an ad-hoc package that provides functionality to automatically localize strings in XML templates.

Let’s take a look at these options. It makes sense, however, to start from the information on the resource files, their content and on how their content can be used for localization.

6.1 Using Resource Files for Localization

C1 CMS makes use of a standard .NET resource file to keep the localization data centrally, in one place. The use of resource files is not C1 CMS template-specific; however, we will focus on using resource files for C1 CMS templates here.

If you run two language versions of your website, you might want to have strings used within XML templates in those two languages. And when the user opens a page in one of the languages, he or she can see these strings in the correct language.

To start using localized version of a string you need:

- A kind of *variable* that you use instead of the string in one of the languages
- A list of matches between these variables and their *values* – strings - in the corresponding language

It makes sense to have a separate list for each language.

When the user opens a page in one of the languages used on the website, the system looks for these “localization” variables in the currently used template, reads their values from the list that matches the current language and replaces the variable with the string value - and the user can see the text in the language of the localized website.

The .NET resource file contains this list in XML format. Here is the sample of such a list in the resource file:

```
<root>
  <!-- skipped -->
  <data name="Hello" xml:space="preserve">
    <value>Hello</value>
  </data>
  <data name="HelloScandinavia" xml:space="preserve">
    <value>Hello Scandinavia</value>
  </data>
  <data name="HelloWorld" xml:space="preserve">
    <value>Hello world</value>
  </data>
</root>
```

Listing 18: String resource matches

The **<data>** element specifies the name of the variable in its **name** attribute while the **<value>** element contains the string translated into a corresponding language.

Note: You can use the Resource File Editor in Visual Studio, which provides you with a more user-friendly way of editing resource file lists.

As it has been mentioned above, each language used on the website should have its own resource file with the same variables as well as strings in proper languages.

6.1.1 Location and Naming of Resource Files

For C1 CMS to find and use the correct resource file for localization, the resource files must be placed in a proper place within the website's file system on the web server. They should also be properly named.

Each resource file that handles string localization must be placed in the ~/App_GlobalResources folder. The file must follow the schema for .NET resource file. (Please refer to the standard documentation on .NET resource file schema.)

When you create a resource file, you should give it a meaningful name. When you create a resource file for the default language on your website, you should name it as:

[filename].resx

However, when you create a file for any other non-default language, the file name must be the same but you should also insert the culture code between its name and its extension:

[filename].[culturecode].resx

For example, you have English as your default language and Danish as the other, non-default language. You should create resource files for each language as:

- Resource.resx (English)
- Resource.da-dk.resx (Danish)

It is quite reasonable to create the first file for the default language, provide all the entries for localization and then create localized versions of this file by copying and inserting the proper culture code in its filename.

Now that you are done creating localization resources, you should use these resources in your templates in a proper way.

6.2 Using Localized Strings from Resource Files in XML Templates

As you learned in the section on the <lang:string> element, you can use the latter to provide language-specific strings or texts in XML templates.

```
<lang:string key="Resource, Resources.Resource.Name"
xmlns:lang="http://www.composite.net/ns/localization/1.0"/>
```

In its key attribute, you should correctly refer to the variable that represents the string in the XML template.

The reference consists of 4 parts. The first 2 parts are the same in all <lang:string> elements:

- The **Resource** word followed by a comma and a space
- The **Resources** word followed by a period

The last 2 parts specify the *name of the resource file* and the *name of the variable*:

- The name of the resource file without the extension and the culture code (followed by a period)
- The name of the variable that represents the string to be used in its place.

For example, if you have a variable named Customers:

```
<root>
  <!-- skipped -->
  <data name="Customers" xml:space="preserve">
    <value>Kunder</value>
  </data>
</root>
```

in the resource file for non-default Danish:

~/App_GlobalResources/Templates.da-dk.resx

the reference string in the template must be:

```
<lang:string key="Resource, Resources.Templates.Customers"
xmlns:lang="http://www.composite.net/ns/localization/1.0"/>
```

where “Templates” is the name of the resource file (without the extension “.resx” and culture code “da-dk”) and “Customers” is the name of the variable to refer to the string by.

This will be correctly rendered as “Kunder” on a web page on the Danish version of the website.

6.3 Using Localized Content within the Same Template

In some cases, you might need to keep the content within the same template but target different language versions of the website. It is more than reasonable to take this approach, for example, for images that might differ from language to language.

In this case, you should use the <lang:switch> elements with its sub-elements as described in the section on this element.

6.4 Automating Localization of Strings with Front-End Localizer

The use of Composite.Tools.FrontendLocalizer has been depricated.

7 Test Your Knowledge

7.1 Exercise: Create an XML template with multiple-content areas

1. Create an XML template and save it.
2. Create a page with this XML template.
3. Preview the page and publish it.
4. Add several content placeholders in the template.
5. Create another XML template with several content placeholders, too.
6. Try switching between these two templates on pages.
7. View the content in the different content areas of the page.

   Note: If the content disappears, make sure you are not using different IDs.

7.2 Exercise: Use external CSS, JavaScript and media files in XML templates

1. Upload CSS, JavaScript and image files you are going to use in XML templates to the server.
2. Edit an XML template and insert links to these files.
3. Make sure that these links will be correctly processed on the front-end.
4. Create a page based on this template.
5. Preview the page and make sure that the resource files you linked to in the template are rendered and used properly.

7.3 Exercise: Localize an XML template

1. Make sure you have two languages installed on your website.
2. Create an XML template with some content, which includes some text and images.
3. Localize the text by using .NET resource files and a proper XML template element.
4. Localize the use of images in the template by specifying different images for different languages.