A Guide to Console Applications

2017-02-14

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

www.orckestra.com

C1 CMS

Contents

1 Introduction

Being a highly customizable content management system, C1 CMS allows you to change the way you work with it in the Administrative console.

By using the XML-based Tree Definition feature, you can customize the structure, content and commands in the CMS Console's tree structure by defining your own console applications via XML documents.

Each application can vary from simply attaching a custom command to an existing element in the tree structure (for example, a page in the Content perspective) to creating complex tree structures in a custom perspective by combining and grouping multiple data types and attaching various commands to the tree elements at will.

The tree structure can consist of "static" folders, data folders and grouping folders, which empowers XML-oriented developers to build trees in the CMS Console exactly to their liking.

Generic data commands (add, edit, delete) can be attached to tree elements as well as one can invoke one's own ASP.NET, XSLT Functions or "advanced" Workflow Foundation-based UIs to get customized editors up and running.

These features are available to you:

- Using IntelliSense (XSD) and validation with verbose logging
- Using flat XML files in ~/App_Data/Composite/TreeDefinitions/
- Having the CMS Console automatically pick up on new files and changes
- Declaring what structure you want using nested XML elements
- Mixing elements of different types in one tree
- Defining simple elements, data elements or data element-driven grouping folders
- Attaching various commands to elements: add, edit, delete, custom workflow, ASP.NET pages (with query string parameters) or XSLT Functions
- Using the CMS Function system to use or create advanced data filters
- Using customized editing forms where needed
- Sorting items as you desire
- Grouping by variable depth with date driven folders, reference fields, ranges etc.

You can watch these videos for an introduction to Tree Definitions in C1 CMS.

Tree Driven Applications - Part 1

Tree Driven Applications - Part 2

1.1 Who Should Read This Guide?

This guide is intended for developers experienced in XML. However, for the full experience of creating console applications, you should be good at XSLT, ASP.NET, Workflow Foundation.

1.2 Getting Started

To get started with console applications, you are supposed to take a number of steps.

Getting Started		
Step	**Activity**	**Chapter or section**
1	Auto-attach an application	*How to Auto-Attach Applications*
2	Allow manually attaching application	*How to Allow Attaching Applications Manually*

3	Add simple elements	*Simple Elements*
4	Add data elements	*Data Elements*
5	Group elements with data folders	*How to Group Data Elements*
6	Sort data elements	*How to Sort Data Elements*
7	Filter data elements	*How to Filter Data Elements*
8	Execute C1 CMS-specific data workflows	*How to Execute Standard Data Workflows*
9	Execute custom workflows	*How to Execute Custom Workflows*
10	Open ASPX pages	*How to Open ASPX Pages*
11	Execute CMS functions	*How to Execute CMS Functions*
12	Show message boxes	*How to Display Message Boxes*
13	Show confirmation boxes	*How to Display Confirmation Boxes*

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

1.3 Terms and Abbreviations

The following is the list of terms and their definitions used throughout this guide.

Terms and Definitions	
Term	**Definition**
Action	An operation executed on an element or in general when users click corresponding button on the toolbar or in the context menu
Attaching	Relating an element (as a child element) or an action (as a context-sensitive command) to an element in existing or custom tree structures
CMS function	An application logic encapsulated as code or markup for processing and outputting data. Based on XSLT, C# or SQL.
Console application	An application based on an XML Tree Definition intended to customize the structure, content or commands available in the CMS Console's tree structures
Data elements	An element that retrieves items from a specific CMS data type as child elements within its parent element
Data item	A single instance of structured data retrieved from a data type displayed as a data element
Data folder	An element that groups data elements or other data folders by a specific field
Filtering	Selecting and displaying data items that only match certain logical condition

Grouping	Distributing data items among several groups by a specific field or condition
Simple element	A single user-defined element in console applications, normally not based on data items (data elements) of a specific data type
Sorting	Listing data elements in a defined order by a specific field
Tree structure	A hierarchy of elements in the CMS Console
Tree definition	A definition of the tree structure of elements and actions attached to these elements, which constitutes a console application and extends or customize the CMS Console.
Workflow	An encapsulated application logic for processing and outputting data. Normally, based on Workflow Foundation

2 An Overview of Console Applications

Let's start with a few scenarios that help illustrate the use of console applications in C1 CMS.

- Scenario 1: A custom action on tree elements in one of the C1 CMS perspectives.

By using a console application, you can attach a custom action to an element in the tree structure in one of the perspective.

For example, you want to display the GUID of a selected page in the Content perspective.

The simplest way would be to have a message box pop up and show the GUID.

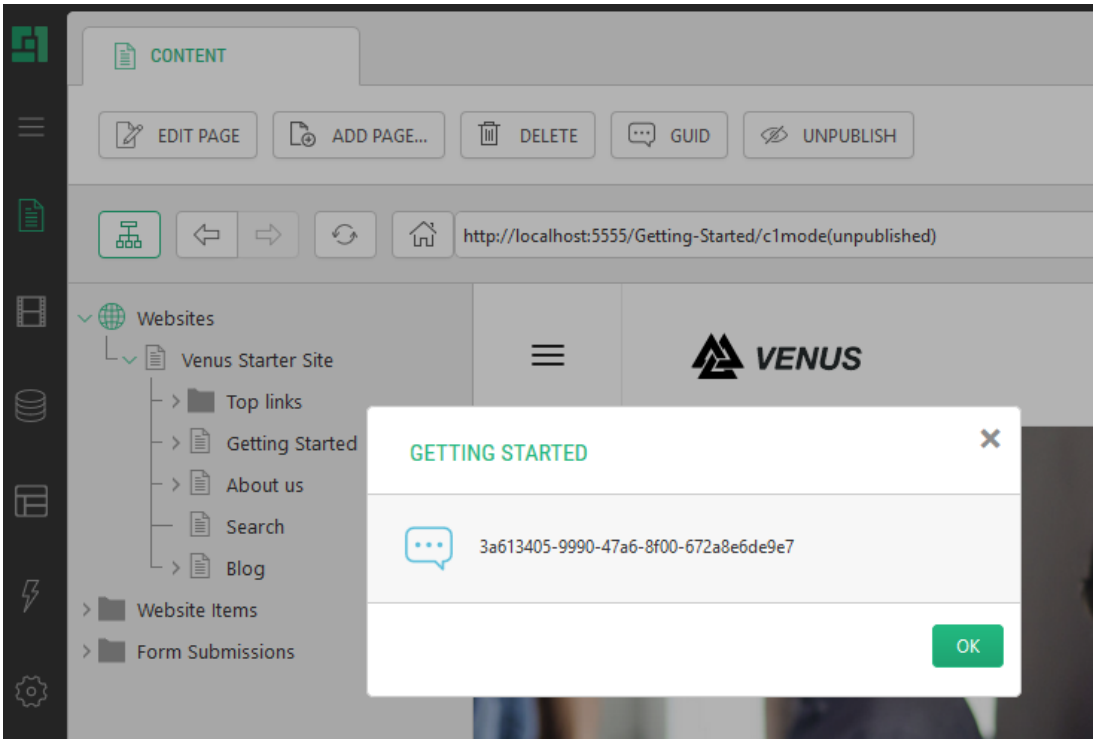


Figure 1: A custom action attached to a tree element

You can implement it by attaching the message box action to the IPage data type that represents pages in C1 CMS. An additional button appears on the toolbar and in the context menu when you select a page. When you click the button, the logic that gets the current page's GUID is implemented and the message box displays it to you.

```
<ElementStructure
xmlns="http://www.composite.net/ns/management/trees/treemarkup/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0">
  <ElementStructure.AutoAttachments>
    <DataType Type="Composite.Data.Types.IPage" />
  </ElementStructure.AutoAttachments>
  <ElementRoot>
    <Actions>
      <MessageBoxAction Label="GUID"
MessageBoxTitle="${C1:Data:Composite.Data.Types.IPage:Title}"
MessageBoxMessage="${C1:Data:Composite.Data.Types.IPage:Id}"/>
    </Actions>
  </ElementRoot>
</ElementStructure>
```

Listing 1: Sample code: Attaching a custom action to an existing element

- Scenario 2: A custom tree in one of the perspectives

By using a console application, you can make a tree structure automatically appear in one of the perspectives along with other existing tree structures.

For example, Composite C1 (now C1 CMS) version 1.3 or later exposes the functionality called “Page Types”.

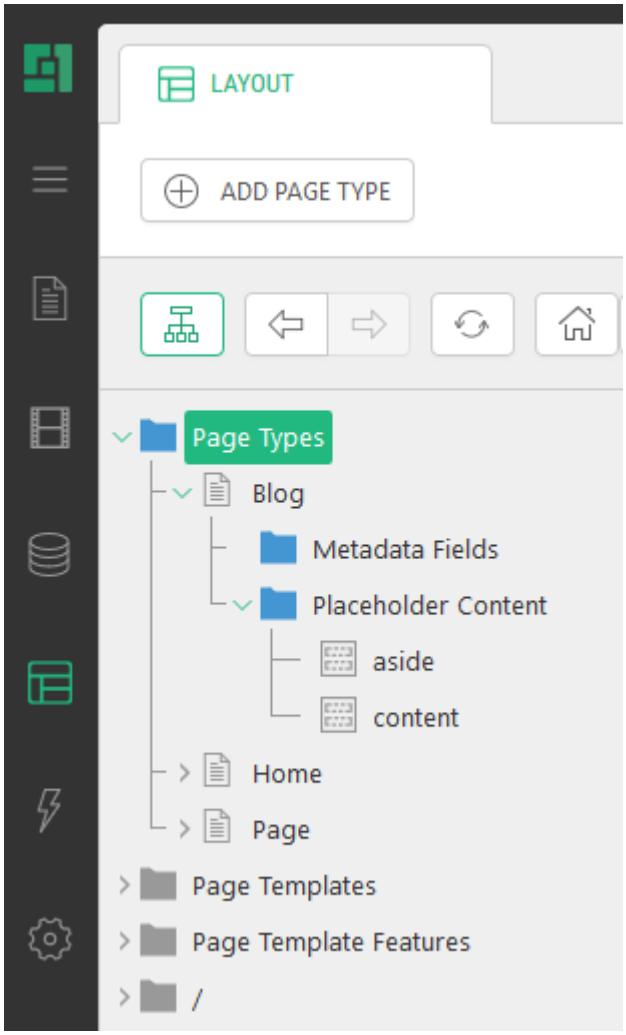


Figure 2: A custom tree in an existing perspective

The Page Types tree structure appears in the Layout perspective along with the Website Templates tree structure. The user can use the Page Types application to effectively manage the page types in C1 CMS.

For a code sample, open ~/App_Data/Composite/TreeDefinitions/PageType.xml

- Scenario 3: A custom tree in a custom perspective

By using a console application, you can make a tree structure automatically appear in a custom perspective.

For example, by changing only one attribute in the above mentioned PageTypes.xml, you can move the Page Types application from the Layout perspectives to its own Page Types perspective.

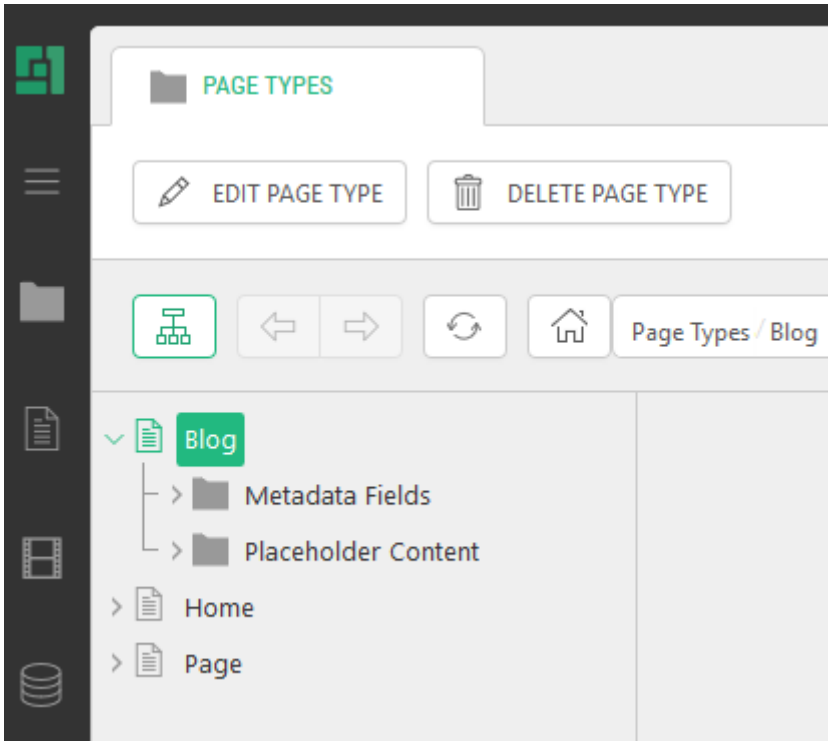


Figure 3: A custom tree in its own perspective

Having an application in a separate perspective can be regarded as a specific case of Scenario 2 where you application appears as a tree structure along with existing ones in one of the perspectives.

- Scenario 4: A custom tree manually added to elements

By using a console application, you can allow users to attach a tree structure to existing tree elements.

For example, you can allow users to manually attach the Blog application to a page in the Content perspective.

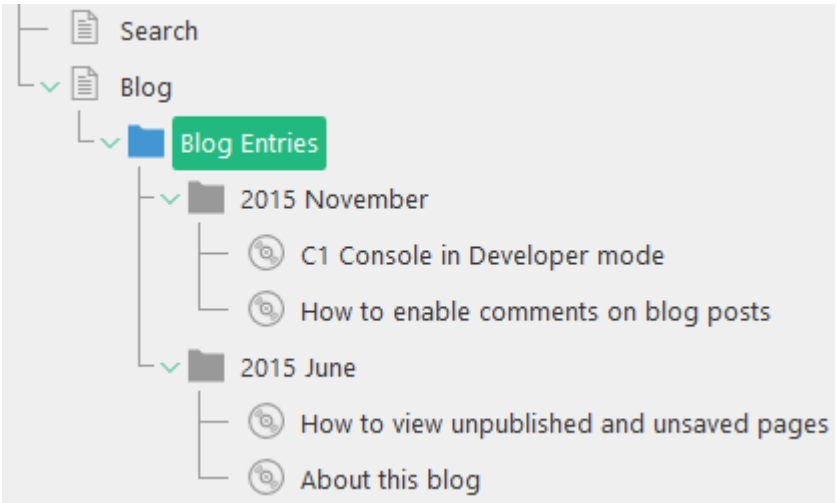


Figure 4: A custom tree manually added to an element

As a result, users can start using the page as a blog page immediately. The Blog's tree structure contains all the elements and action needed to maintain the blog. It keeps track of

blog entries and comments to those entries and the user can add, edit or delete the blog entries.

For a code sample, install the Blog add-on and open ~/App_Data/Composite/TreeDefinitions/Composite.Community.Blog.Entries.xml.

Please note that you can explore more add-ons that use tree definitions for sample markup.

2.1 A General Procedure of Creating Console Applications

For you to have a bird's eye view of how to create a console application, let's have a look at the major steps you should to take.

1. Create an XML file in \App_Data\Composite\TreeDefinitions.
2. Add the tree definition's root element and specify the required namespaces.

```
<ElementStructure
xmlns="http://www.composite.net/ns/management/trees/treemarkup/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0">
<!-- tree definition  -->
</ElementStructure>
```

3. Define how and where the application must appear in the console.

```
<ElementStructure.AutoAttachments>
  <!-- required elements -->
</ElementStructure.AutoAttachments>
```

or

```
<ElementStructure.AllowedAttachments>
 <!-- required elements -->
</ElementStructure.AllowedAttachments>
```

4. Add the starting element of the tree structure:

```
<ElementRoot>
  <!-- elements of the tree structure -->
</ElementRoot>
```

5. Create a tree structure combining simple elements and data elements if necessary.

```
<Children>
  <Element>
    <Children>
      <DataElements>
        <!-- other nested elements if needed -->
      </DataElements>
    </Children>
  </Element>
</Children>
```

6. Group the elements by using data folder elements if necessary.

```
<DataFolderElements>
  <Children>
    <DataElements>
        <!-- other nested elements if needed -->
    </DataElements>
  </Children>
</DataFolderElements>
```

7. Sort the data elements if necessary.

```
<DataElements>
  <OrderBy>
    <Field />
  </OrderBy>
</DataElements>
```

8. Filter the data elements if necessary.

```
<DataElements>
  <Filters>
    <FieldFilter />
  </Filters>
</DataElements>
```

9. Attach actions to the tree elements if necessary.

```
<DataElements>
  <Actions>
    <EditDataAction />
    <WorkflowAction />
    <CustomUrlAction />
    <ReportFunctionAction />
  </Actions>
</DataElements>
```

2.2 Creating a Tree Definition File

As you can see in the steps above, you are supposed to create a tree definition file for your console application in a specific folder on your website: ~\App_Data\Composite\TreeDefinitions.

You have two options here:

- Since the tree definition file is a file in XML format, you can create the file in your favorite XML editor and upload it to the folder via the CMS Console.
- You can create the tree definition file in the required folder in the CMS Console directly.

The first option makes more sense since many XML editors (for example, the built-in XML Editor in Visual Studio 2010) can provide the developers with additional conveniences such as IntelliSense and validation.

To upload the file to the TreeDefinitions folder:

1. Log into the CMS Console.
2. In the System perspective, expand \App_Data\Composite\TreeDefinitions.
3. Select TreeDefinitions.
4. Click Upload File on the toolbar.
5. In the Upload File window, click File Upload button, browse to, and select, the file, and click Open.
6. Click OK.

Please note that if you have access to the web server's file system (for example, via the FTP), you can simply copy (upload) the file to the above mentioned folder.

To create the tree definition file in C1 CMS directly:

1. Log into the CMS Console.
2. In the System perspective, expand \App_Data\Composite\TreeDefinitions.
3. Select TreeDefinitions.
4. Click New File on the toolbar.
5. In the Add New File window, type the name of your application and add the ".xml" extension: e.g. "MyApplication.xml"

6. Click OK. The file will open in the right view.
7. Add the content to the file and save it.

Please note that once the tree definition file has been placed in the TreeDefinitions folder, C1 CMS picks it up automatically and displays in the console whatever it defines.

2.2.1 Schema Definition File

The console application's XML schema is defined in a standard schema definition file:

~\Composite\schemas\Trees\Tree.xsd

You can link to the XSD file to your XML tree definition file in Visual Studio 2010 and start using IntelliSense and validation when creating your console applications.

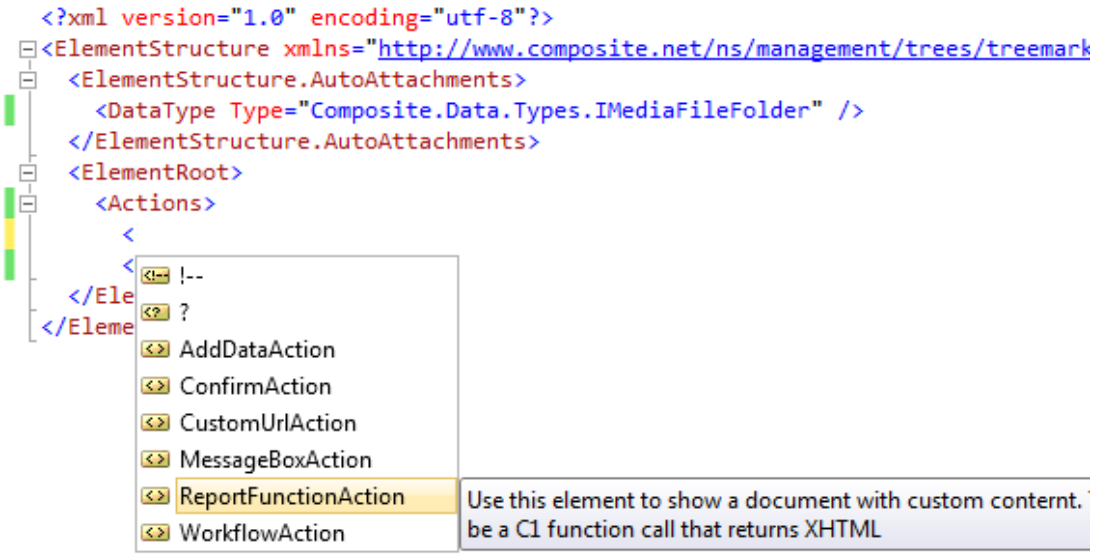


Figure 5: Using IntelliSense

2.3 A Quick Overview of a Tree Definition

The following figure illustrates the sample structure of the tree definition file.

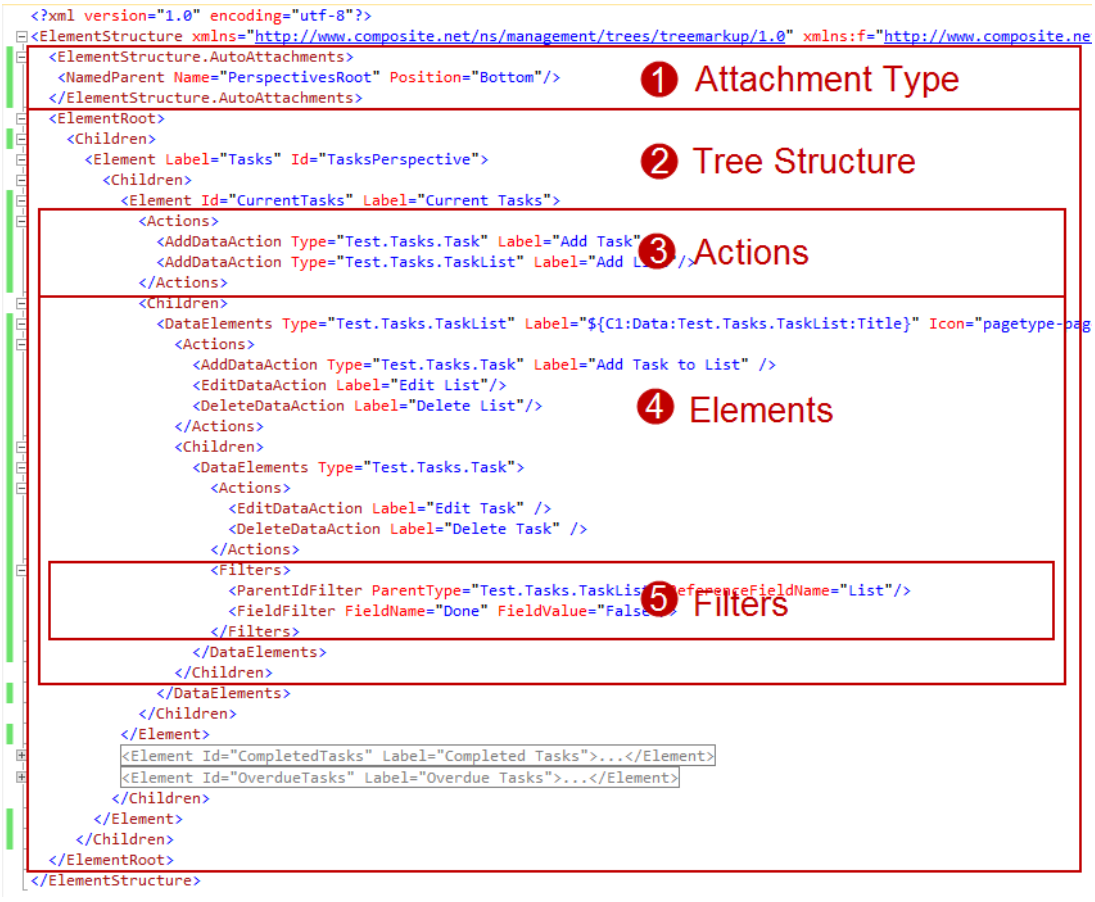


Figure 6: Sample Structure of a tree definition

As you can see from the figure above, the file structure has two main parts: the *type of attachment* (1) and the *tree structure* (2).

The tree structure consists of *elements*. The elements may have *child elements* (4) and *actions* (3). The *data elements* (one of the three element types) can also include *sorting* and *filtering elements* (5).

2.3.1 Root Element and Namespaces

You always start your tree definition file with the root ElementStructure element. This element must have two required namespaces specified:

- xmlns=http://www.composite.net/ns/management/trees/treemarkup/1.0
- xmlns:f=http://www.composite.net/ns/function/1.0

The first namespace defines all the elements of a console application. The second namespace is a standard namespace used to define CMS functions. Some elements in console applications make use of CMS functions. That is why the function's namespace must be mentioned, too.

```
<ElementStructure
xmlns="http://www.composite.net/ns/management/trees/treemarkup/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0">
</ElementStructure>
```

Listing 2: ElementStructure

The ElementStructure element normally includes:

- one of the two application attachment elements: ElementStructure.AutoAttachments or ElementStructure.AllowedAttachments
- and the required starting element of the tree structure: ElementRoot

3 How to Attach Console Applications

Console applications can vary from a single action attached to an element in an existing tree structure such as a page to a stand-alone highly-hierarchical tree structure of elements with attached actions.

To have the application appear in the console GUI, you should always specify:

- Where the application should be attached
- How the application should be attached

A console application can be attached to:

- An existing element of a specific type (for example, a page)
- A root (normally implicit) of an existing perspective (for example, “Content”)
- A root of its own perspective

The application can appear in all these points of attachment:

- Automatically
- Only when attached manually by the user

Normally attaching an application manually is relevant only for existing elements of specific types. In this case, the element gets two new actions on its context menu:

- “Add application”
- “Remove application”

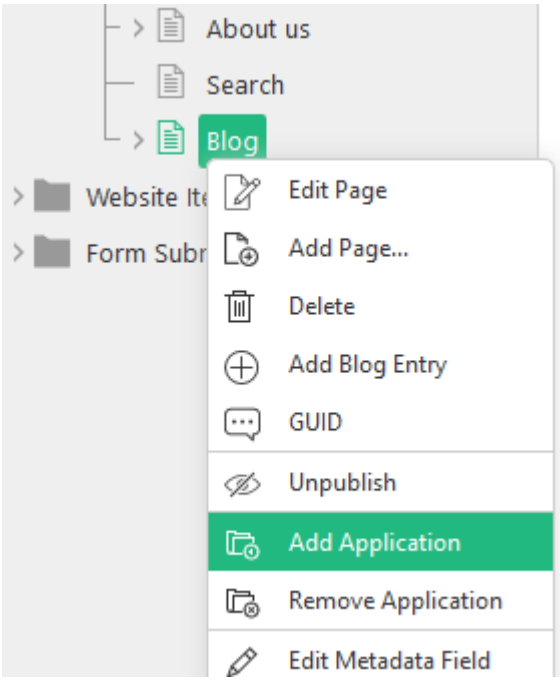


Figure 7: Attaching applications via the context menu

The user can thus add (attach) and remove (detach) an application. The manual attachment is regarded as “allowed”.

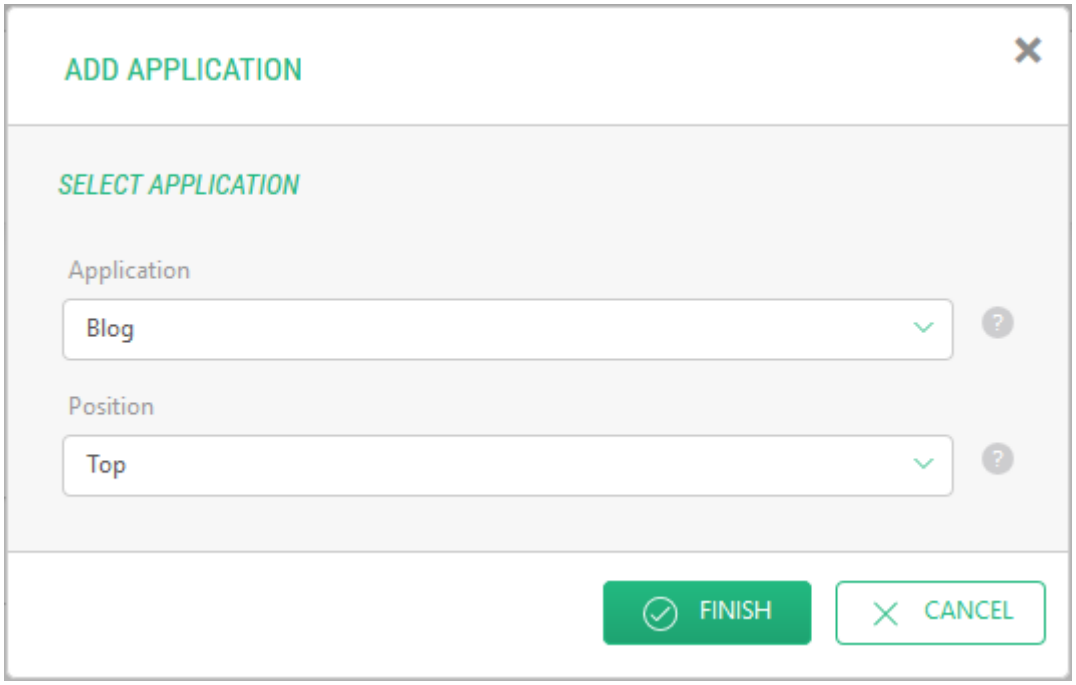


Figure 8: Selecting an application to attach

Based on the way you intend your application to appear in the console, you should specify whether you want to:

- auto-attach an application
- allow users to attach application manually

When that decided, you should proceed to select where you want your application to appear.

3.1 How to Auto-Attach Applications

The application can automatically appear as a tree structure in one of the perspectives or in its own new perspective. This type of application is regarded as an “Auto Attachment”. For example, the Page Types application appears in the Layout perspective automatically.

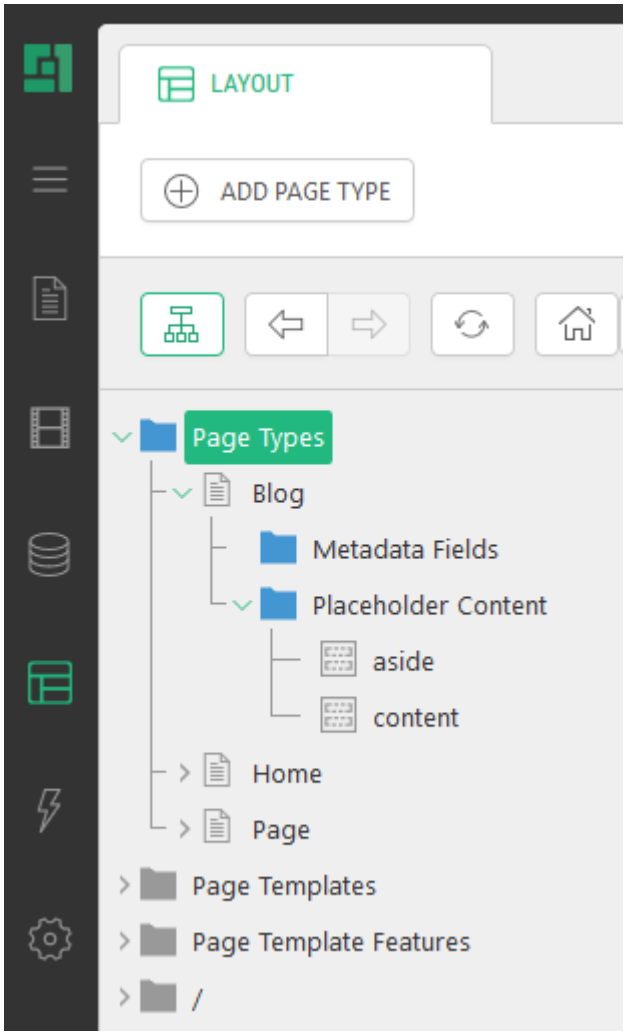


Figure 9: Auto-attached application

To auto-attach an application in a perspective:

1. Add the ElementStructure.AutoAttachments element within the ElementStructure element.
2. Add the NamedParent element within the ElementStructure.AutoAttachments element.
3. Set its required attribute:
   - Name: The name of the perspective to attach the application in.

If necessary, set its optional attribute:

- Position: The position in the tree structure the application should appear at

```
<ElementStructure.AutoAttachments>
  <NamedParent Name="Layout" Position="Top" />
</ElementStructure.AutoAttachments>
```

Listing 3: Auto-attaching the application in Layout

Please note that the application can only appear in one perspective.

Normally, you can select one of the existing perspectives here. However, you have two more options:

- You can have the application appear below the Website Items in the Content perspective (“Content.WebsiteItems”)

- You can have the application appear in its own perspective (“PerspectivesRoot”).

Please note that to make the new perspective available in the Administrative console, you should grant access to this perspective to proper users or user groups.

You may as well auto-attach the application to an existing element (for example, a page):

1. Add the ElementStructure.AutoAttachments element within the ElementStructure element.
2. Add the DataType element within the ElementStructure.AutoAttachments element.
3. Set its required attribute:
   - Type: The data type, the items of which the application will be attached to

If necessary, set its optional attribute:

   - Position: The position in the tree structure the application will appear at

```
<ElementStructure.AutoAttachments>
  <DataType Type="Composite.Data.Types.IPage" Position="Top" />
</ElementStructure.AutoAttachments>
```

Listing 4: Auto-attaching the application to an element of a specific data type

Please note that you can attach the application to multiple data types.

```
<ElementStructure.AutoAttachments>
  <DataType Type="Composite.Data.Types.IPage" Position="Top" />
  <DataType Type="Composite.Data.Types.IPageTypes" Position="Bottom" />
</ElementStructure.AutoAttachments>
```

Listing 5: Auto-attaching to elements of multiple types

3.2 How to Allow Attaching Applications Manually

You can have users manually add the application to an existing element (for example, a page or a data item). The application will thus appear as a child element. This type of application is regarded as an “Allowed Attachment”.

Figure 10: Manually attached application

Two new actions (“Add application” and “Remove application”) appear in the element’s context menu, which enable users to add or remove this application at will.

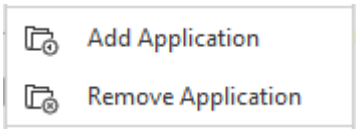


Figure 11: Menu commands to attach applications

To allow users to attach an application manually:

1. Add the ElementStructure.AllowedAttachments element within the ElementStructure element.
2. Set its required attribute:
   - ApplicationName: The name of the application to refer to when adding or removing it
3. Add the DataType element within the ElementStructure.AllowedAttachments element.
4. Set its required attribute:
   - Type: The data type, the items of which the application is allowed to be attached to

If necessary, set its optional attribute:

   - Position: The position in the tree structure the application will appear at

```
<ElementStructure.AllowedAttachments ApplicationName="Blog">
  <DataType Type="Composite.Data.Types.IPage" Position="Bottom" />
</ElementStructure.AllowedAttachments>
```

Listing 6: Allowing manually attaching the application

You can add as many data types as you need.

4 How to Attach Elements to Tree Structures

Creating a console application implies creating a tree-like structure of elements.

The elements often are retrieved as data items from a specific data type. And you can group, sort and filter them.

You can also create your own elements not based on data types. You can use these simple custom elements as, for example, parents for data type based items or data folders.

Besides you can nest elements within elements thus creating quite a hierarchy of them.

Figure 12: A hierarchy of elements in an application

The tree structure always starts with the tree's root element – ElementRoot. This element is required and can contain the Children and Actions elements. (You will learn more about Actions in How to Attach Actions to Tree Elements.)

```
<ElementRoot>
  <Children>
    <!-- child elements if needed -->
  </Children>
  <Actions>
    <!-- action elements if needed -->
  </Actions>
</ElementRoot>
```

Listing 7: Children and Actions are child elements of ElementRoot

Please note that ElementRoot is only the root of the application tree structure and is different from the application root - ElementStructure. The ElementStructure element contains the ElementRoot element.

Please also note that you should always use the Children element when you nest elements in other elements.

```
<!-- parent element: start tag -->
  <Children>
    <!-- child element -->
  </Children>
<!-- parent element: end tag -->
```

Listing 8: Using Children when nesting elements

The child elements appear in the GUI as children of elements which the application is attached to.

You can use 3 kinds of elements when creating a tree structure:

- Simple elements
- Data elements
- Data folder elements

In this chapter, you will learn to create and use *simple elements* and *data elements*. *Data folder elements* used to group data items are covered in How to Group Data Elements.

4.1 Simple Elements

A simple element stands for a single item in a tree structure. Normally it is not data type based. This element may serve as a container (parent) for other elements such as data elements or data folder elements.

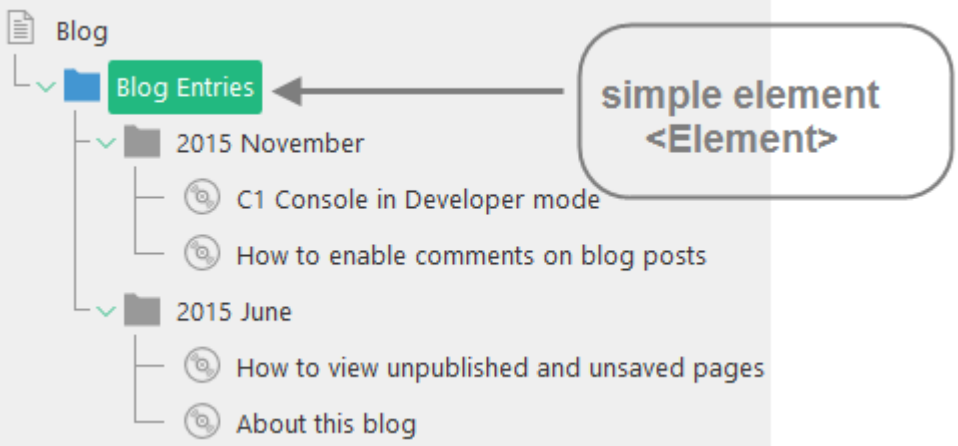


Figure 13: Simple elements

(A good practice is to have at least one simple element contained in ElementRoot.)

To add a simple element to a tree structure, you should use an Element element:

1. Locate an element to serve as a parent.
2. Add a Children element within this parent element.
3. Add an Element element within the Children element.
4. Specify its required attributes:
   - Id: A unique string in the tree to identify the element
   - Label: The label of the element

If necessary, specify its optional attributes:

- ToolTip: The tooltip of the element
- Icon: The icon of the element when collapsed (closed)
- OpenedIcon: The icon of the element when expanded (opened)

```
<ElementRoot>
  <Children>
    <Element
      Id="TasksRoot"
      Label="Tasks"
      ToolTip="Tasks"
      Icon="pagetype-pagetype-rootfolder"
      OpenedIcon="pagetype-pagetype-rootfolder-open">
    </Element>
  </Children>
</ElementRoot>
```

Listing 9: Using Element

You can add as many Element elements as you need.

Please note that you cannot add an Element element within a DataFolderElements element.

4.2 Data Elements

Data elements stand for data items from a specific data type. Data items retrieved as data elements can be grouped, sorted and filtered.

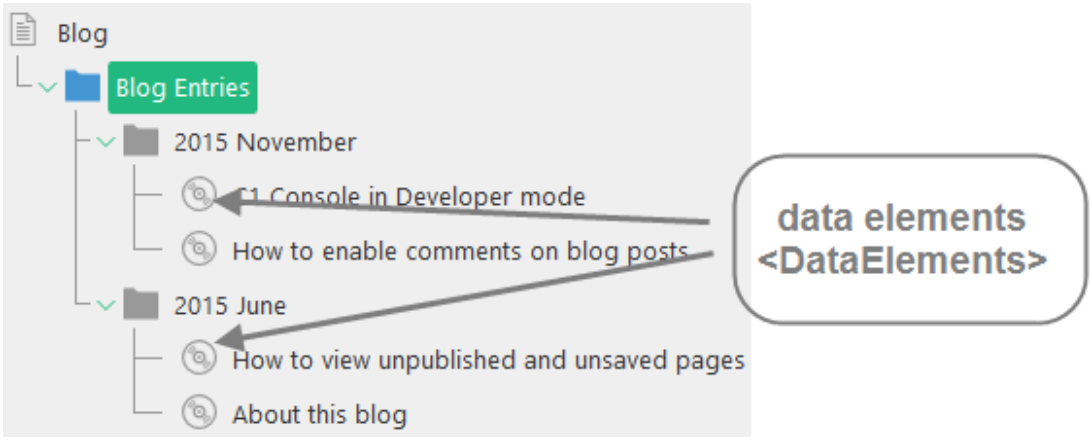


Figure 14: Data elements

It is a good practice to nest data elements within a container-like element such as a simple element (Element) or data folder elements (DataFolderElements).

To add data elements to a tree structure, you should use a DataElements element:

1. Locate an element to serve as a parent.
2. Add a Children element within this parent element.
3. Add a DataElements element within the Children element.
4. Specify its required attribute:
   - Type: The data interface type name

If necessary, specify its optional attributes:

- Label: A custom label for each data item
- ToolTip: A custom tooltip for each data item
- ShowForeignItems: When set to “true”, data items not yet localized are displayed with a “Localize” action
- Display: A mode in which an element that might have child elements is displayed (Auto, Lazy, Compact)
- Icon: The icon of the element when collapsed (closed)
- OpenedIcon: The icon of the element when expanded (open)

```
<DataElements
  Type="Composite.Community.Blog.Entries"
  Label="${C1:Data:Composite.Community.Blog.Entries:Title}"
  Display="Auto">
</DataElements>
```

Listing 10: Using DataElements

4.3 How to Use Values from Data Type Fields

When you retrieve items from a data type, you are likely to need the values of the data type’s specific fields, different for each item (for example, for the Label attribute of DataElements).

Use the following model to get a value from a data type field:

${C1:Data:DataType:Field}

replacing “DataType” with the full name of the data type (including namespaces) and “Field” - with the name of the field.

For example,

${C1:Data:Composite.Community.Blog.Entries:Title}

In the same manner, you can pass a specific value to custom commands you can attach to tree elements (e.g. a CMS function).

4.4 How to Use Localized Strings

If you want your application to match the language of the console GUI, consider use localized strings in its definition file.

While a hard-coded string read “Add Task”, the localized string will read “${TasksApp, AddTaskButtonLabel}”.

Here “TaskApp” stands for the name of a specific localization file without the culture/language code suffix and extension (e.g. “TaskApp.en-us.xml”) and “AddTaskButtonLabel” is the name of the string with the right text:

```
<string key="AddTaskButtonLabel" name="Add Task" />
```

In most cases, you will localize the GUI elements such as button labels and tree element labels, which is why you need to use this approach.

Please note that you have to register localization files used by your application in ~\App_Data\Composite\Composite.config to start using this approach.

Please refer Localizing Strings in Newsletter as a model for your localization.

4.5 How to Display Elements in Custom Perspectives

If you want your application to appear in a custom perspective, make sure you have followed these 3 major steps:

1. Attach the application to the perspectives’ root.
2. Use a simple element as the root of the tree of elements to provide the name for the perspective.
3. Grant access to the new perspective to required users and/or user groups.

First of all:

1. Add the ElementStructure.AutoAttachments element within the ElementStructure element.
2. Add the NamedParent element within the ElementStructure.AutoAttachments element.
3. Set its Name attribute to “PerspectivesRoot” and, if necessary, set its Position attribute.

```
<ElementStructure.AutoAttachments>
  <NamedParent Name="PerspectivesRoot" Position="Top" />
</ElementStructure.AutoAttachments>
```

Listing 11: Auto-attaching the application in its own perspective

(See “How to Auto-Attach Applications” for more information.)

Next:

1. Add a Children element within the ElementRoot element, and then an Element element within this Children element.
2. Specify its Id and Label attributes. Please note that the Label's value will serve as the name of the new perspective.

```
<ElementRoot>
  <Children>
    <Element Id="TasksRoot" Label="Tasks">
    </Element>
  </Children>
</ElementRoot>
```

Listing 12: Using Element to provide a name for the new perspective

(See "Simple Elements" for more information.)

Finally:

1. In the Users perspective, edit the user and/or user group you want to grant access to the new perspective.
2. Check the new perspective in the list of perspectives. (For a user, see the Perspectives tab; for a user group, see the Perspectives group box.)
3. Save the changes.
4. Reload the Administrative console (F5).

(See "Security" for more information.)

The perspective should appear in the console.

4.6 How to Group Multiple Tree Definitions in One Perspective

In Composite C1 (now C1 CMS) version 3.2 or later, you can display multiple console applications, each defined in its own tree definition file, in the same perspective, new or existing (as a subfolder).

Let's assume that you have five related tree definition files that you want to show in your own perspectives. Using one custom perspective for each tree definition would clutter the navigation pane in C1 CMS.

It makes sense to create a single custom perspective and display these tree definitions in it. One way is to merge all five tree definitions into one big tree definition file. However, it would be hard to maintain the markup in this way.

A better approach is to group all the five tree definitions in one perspective.

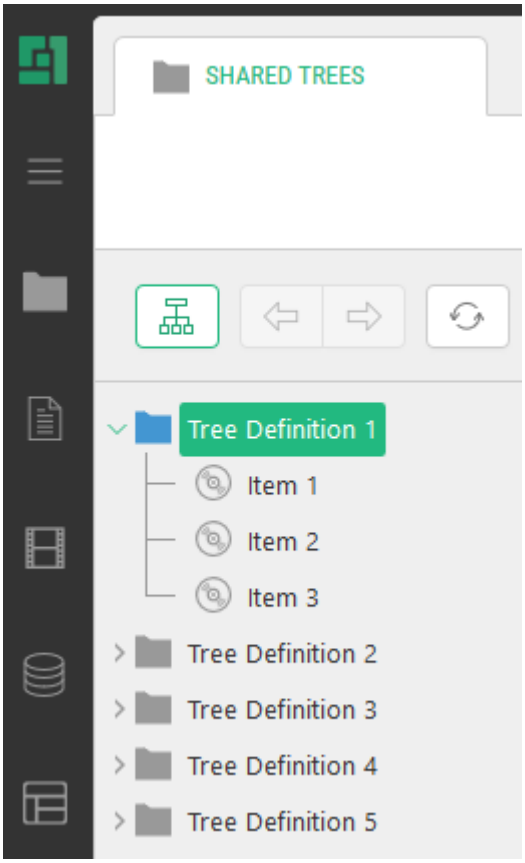


Figure 15: Several tree definitions grouped in one custom perspective

It can be a custom perspective or an existing one. In the latter case, the tree definitions will be grouped under one subfolder.

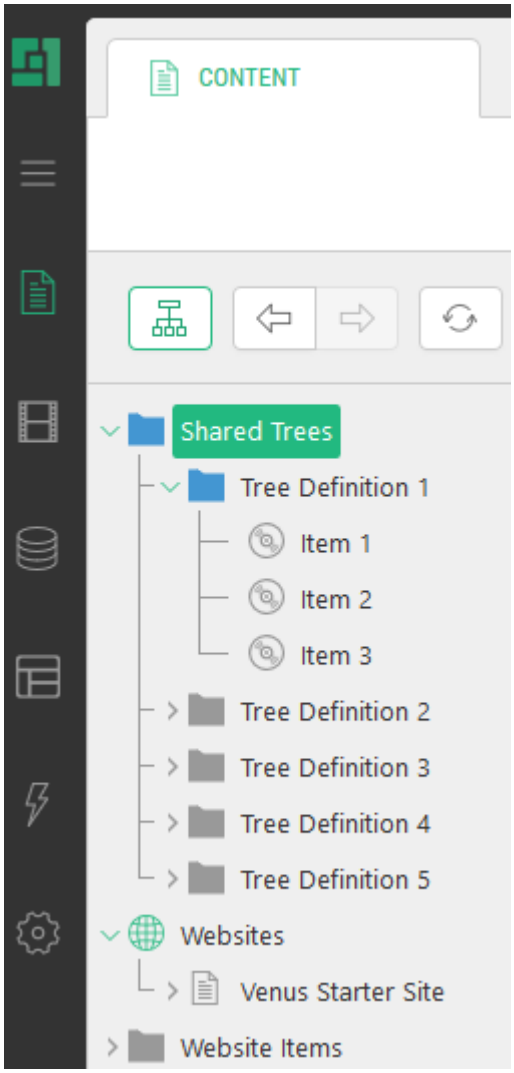


Figure 16: Several tree definitions grouped in one existing perspective

To group tree definitions in this manner, in each tree definition file:

1. Add and set the ShareRootElementById attribute to “true” on the ElementRoot element.
2. Set the Id attribute on the “root” Element element to the same value.

(Here the “root” Element element stands for the child of the ElementRoot element.)

Now C1 CMS will put in the same perspective all the trees that use the ShareRootElementById=”true” and the same Id in their “root” element.

```
<ElementStructure
xmlns="http://www.composite.net/ns/management/trees/treemarkup/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0">
  <ElementStructure.AutoAttachments>
    <NamedParent Name="PerspectivesRoot" Position="Top" />
  </ElementStructure.AutoAttachments>
  <ElementRoot ShareRootElementById="true">
    <Children>
      <Element Label="Shared Trees" Id="SharedTreesId">
        <Children>
          <Element Label="Tree Definition 1" Id="Tree1Id">
            <Children>
               <!-- data elements etc -->
            </Children>
          </Element>
        </Children>
      </Element>
    </Children>
  </ElementRoot>
</ElementStructure>
```

Listing 13: A tree to appear in a custom perspective (“Shared Trees”)

Not only can you display trees in a new perspective but also in one of the existing perspectives, for example, “Content”.

```
<ElementStructure
xmlns="http://www.composite.net/ns/management/trees/treemarkup/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0">
  <ElementStructure.AutoAttachments>
    <NamedParent Name="Content" Position="Top" />
  </ElementStructure.AutoAttachments>
  <ElementRoot ShareRootElementById="true">
    <Children>
      <Element Label="Shared Trees" Id="SharedTreesId">
        <Children>
          <Element Label="Tree Definition 1" Id="Tree1Id">
            <Children>
               <!-- data elements etc -->
            </Children>
          </Element>
        </Children>
      </Element>
    </Children>
  </ElementRoot>
</ElementStructure>
```

Listing 14: A tree to appear in an existing perspective (“Content”)

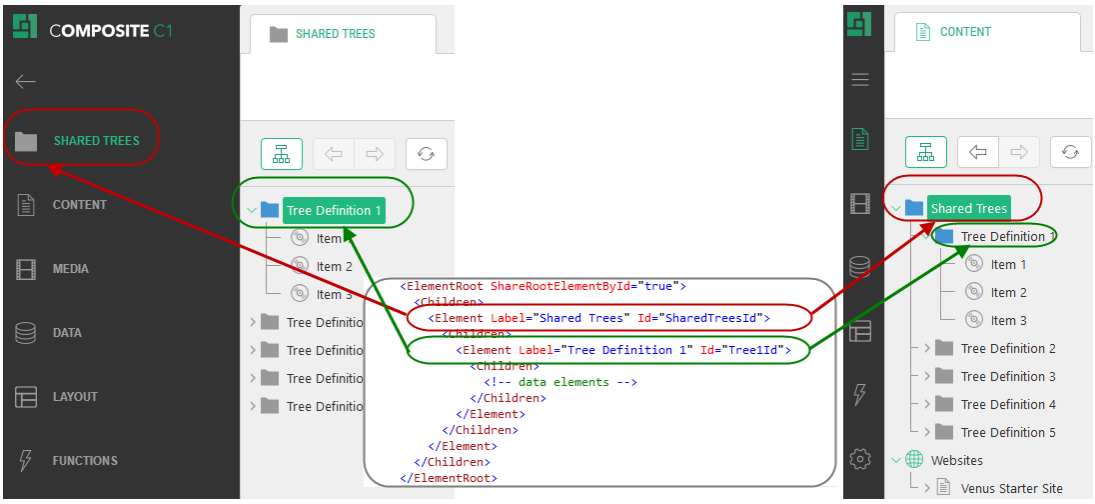
Use the standard procedures for adding elements to existing perspectives or custom perspectives when defining your trees.

Please note that it only works for trees that have a single “root” Element element (e.g. <Element Label="My Perspective" Id="ItemsRoot">). If you have multiple “root” Elements, the tree will not be grouped.

A good practice is two display tree elements under two nested Element elements. The first one provides a custom perspective with the label, tooltip and icons, or serves as the common subfolder in an existing perspective. The second one serves as a root for the tree elements from one tree definition.

```
<ElementRoot ShareRootElementById="true">
  <Children>
    <Element Label="Shared Trees" Id="SharedTreesId">
      <Children>
        <Element Label="Tree Definition 1" Id="Tree1Id">
          <Children>
             <!-- data elements etc -->
          </Children>
        </Element>
      </Children>
    </Element>
  </Children>
</ElementRoot>
```

Listing 15: Using nested Element elements in grouped tree definitions



Listing 16: Nested Element elements in the CMS Console

Please note that the first loaded tree within the same group is the one that decides the label, tooltip and icon of the new perspective or the subfolder in the existing perspective. The elements of the root child of all trees in the same group are displayed under the new perspective/folder in load order.

Changing the perspective in the first loaded tree definition will change it for all other tree definitions in the same group even though they have other perspectives explicitly defined in the NamedParent element. Changing the perspective in any of the tree definitions not loaded first will have no such effect.

5 How to Group Data Elements

In C1 CMS, you can group data items by one or more fields provided that some of the data items share the same values in those fields. Data folder elements allow you to group data elements in your console applications.

Figure 17: Data folder elements

Grouping can be hierarchical if you use more than one field to group by. (Read more on grouping data items in Field Grouping and Creating Datatypes That Reference Other DataTypes)

Normally, data folder elements of a specific type should contain data elements (DataElements) of the same type. The data folder elements can also contain other data folder elements of the same type. In this case, the top data folder elements must be distinguished from their child data folders because they might have different attributes to set.

To group data elements, you should use a DataFolderElements element:

1. Locate an element to serve as a parent.
2. Add a Children element within this parent element.
3. Add a DataFolderElements element within the Children element.
4. Specify its required attribute:
   - Type: The data interface type name
   - FieldGroupingName: The field name of the given data interface (property name) to group by

If necessary, specify its optional attributes:

- DateFormat: Date format used when grouping
- Range: Ranges for grouping data folders
- FirstLetterOnly: When set to “true” the grouping is done with the first letter only
- Display: A mode in which an element that might have child elements is displayed
- Icon: The icon of the element
- SortDirection: The sorting direction (**Note**: Composite C1 (now C1 CMS) 4.0 or later.)

```
<DataFolderElements
  Type="Composite.Community.Blog.Entries"
  DateFormat="yyyy MMMM"
  FieldGroupingName="Date"
  Display="Compact">
</DataFolderElements>
```

Listing 17: Using DataFolderElements

To have a child data folder within the above parent data folder:

5. Add a Children element within the DataFolderElements element.
6. Add another DataFolderElements element within the Children element.
7. Specify its required attribute:
    - FieldGroupingName: The field name of the given data interface (property name) to group by

If necessary, specify its optional attributes:

- DateFormat: Date format used when grouping
- Range: Ranges for grouping data folders
- FirstLetterOnly: When set to “true” the grouping is done with the first letter only
- Display: A mode in which an element that might have child elements is displayed
- Icon: The icon of the element
- ShowForeignItems: When set to “true”, data folder items not yet localized are displayed with a “Localize” action
- SortDirection: The sorting direction (**Note**: Composite C1 (now C1 CMS) 4.0 or later.)

Please note that for child data folders within a parent data folder you do not need to specify the Type attribute again. However, you can specify the ShowForeignItems attribute (unavailable for the parent data folder.)

6 How to Sort Data Elements

You can sort data elements in the ascending or descending order. For this, you need to specify the *field* to order the elements by and, optionally, the *order* (direction). The ascending order is assumed when none is specified.

Please note that only data elements (elements of the DataElements type) can be sorted. Simple elements and data folder elements cannot be sorted.

To sort data elements:

1. Locate a DataElements element you want to sort in a specific order.
2. Add an OrderBy element within the element.
3. Add the Field element within the OrderBy element.
4. Specify its required attribute:
   - FieldName: The name of the field used to order data elements by

If necessary, specify its optional attribute:

   - Direction: The order to sort data elements by

```
<OrderBy>
  <Field FieldName="Date" Direction="descending"></Field>
</OrderBy>
```

Listing 18: Sorting by Date in Descending order

Please note that the DataElements element can only contain one OrderBy element. The OrderBy element can contain as many Fields elements as you need.

7 How to Filter Data Elements

When you use data elements in your tree structures, all existing data elements are displayed. However, you can also display specific elements selectively by filtering them.

To apply a filter on data elements, you should use a Filters element that contains a specific filter. You can use the following filters:

- Parent ID filter
- Field filter
- Function filter

Please note that you can only apply filters to data elements (DataElements). Please also note that you can only use one Filters element within a DataElements element.

7.1 How to Filter Data Elements by Parent ID

The Parent ID filter allows you to select elements that share the same parent entity.

For example, if you keep comments to pages in one data type, it makes sense to only display comments relevant to a specific page. In these relations, a page serves as a specific parent entity to a number of comments and they can thus be selected by the ID of their parent (page).

To filter data elements by their parent's ID, you should use a ParentIdFilter element:

1. Locate a DataElements element you want to apply the filter to.
2. Add a Filters element within the DataElements element.
3. Add a ParentIdFilter element within the Filters element.
4. Set its required attributes:
   - ParentType: The type of the parent element (data type) to filter on
   - ReferenceFieldName: The name of the field that is the reference to the parent type

```
  <Filters>
    <ParentIdFilter ParentType="Composite.Data.Types.IPage"
ReferenceFieldName="PageId" />
  </Filters>
```

Listing 19: Filtering data elements by parent ID

Please note that a Filters element can only contain one ParentIdFilter element.

7.2 How to Filter Data Elements by Field

The Field filter allows you to select data elements if a field given contains a value that matches a specific value or a range of values.

To filter data elements by field, you should use a FieldFilter element:

1. Locate a DataElements element you want to apply the filter to.
2. Add a Filters element within the DataElements element.
3. Add a FieldFilter element within the Filters element.
4. Set its required attributes:
   - FieldName: The name of the field to filter on
   - FieldValue: The value of the field for which the elements will be shown

If necessary, set its optional attribute:

- Operator: The relation between the field and the value to select elements when filtered if they match

```
<Filters>
  <FieldFilter FieldName="Done" FieldValue="False"/>
</Filters>
```

Listing 20: Filtering data elements by field

Please note that a Filters element can contain as many FieldFilter elements as you need.

7.3 How to Filter Data Elements with CMS Functions

The Function filter allows you to use a CMS function to filter data elements. In this case, it is the function that sets the filtering rules and is quite transparent to the Function filter that uses it.

To filter data elements with a CMS function, you should use a FunctionFilter element:

1. Locate a DataElements element you want to apply the filter to.
2. Add a Filters element within the DataElements element.
3. Add a FunctionFilter element within the Filters element.
4. Add a f:function element within the FunctionFilter element.
5. Set its required attributes:
   - name: The name of the CMS function

If the function requires so:

6. Add one or more f:param elements within the f:function element.
7. Set its required attribute:
   - name: The name of the CMS function's parameter

If necessary, set its optional attribute:

- value: The value of the CMS function's parameter

```
<Filters>
  <FunctionFilter>
    <f:function name="UpcomingEventsFilter"
      xmlns:f="http://www.composite.net/ns/function/1.0">
      <f:param name="IncludeToday" value="true" />
    </f:function>
  </FunctionFilter>
</Filters>
```

Listing 21: Filtering data elements with a CMS function

(The CMS function schema is defined in a separate schema definition file located at:

~/Composite/schemas/Functions/Functions.xsd)

The Filters element can contain as many FunctionFilter elements as you need.

Please note that a CMS function can contain another CMS function.

Please also note that the filter function's parameters can be assigned dynamic field values (see the example below).

7.3.1 Example of Creating and Using a Filter Function

Let's assume that we have two datatypes: Demo.ParentType and Demo.ChildType. Both datatypes have the "Title" field of the String type. Besides, Demo.ChildType has the String field named "ParentIdList" which uses the DataIdMultiSelector widget referring to Demo.ParentType as its source datatype.

By using the DataIdMultiSelector, Demo.ChildType can have multiple parents.

When we build a tree of parent data elements of the Demo.ParentType type which nest elements of the Demo.ChildType type, we need to filter the child elements by their multiple parent items.

ParentIdFilter can only filter by one parent ID. And this is where FunctionFilter come in handy.

First we create a C# function that can filter Demo.ChildType by multiple parent IDs:

```
public static Expression<Func<Demo.ChildType, bool>> IdListFilter(Guid
ParentId)
{
  Expression<Func<Demo.ChildType, bool>> filter = f =>
f.ParentIdList.Contains(ParentId.ToString());
  return filter;
}
```

Listing 22: An example of a filter function

Then we use the function in our tree definition passing to its ParentId parameter a dynamic field value:

```
<DataElements Type="Demo.ParentType"
Label="${C1:Data:Demo.ParentType:Title}">
  <Children>
    <DataElements Type="Demo.ChildType"
Label="${C1:Data:Demo.ChildType:Title}">
      <Filters>
        <FunctionFilter>
          <f:function name="Demo.IdListFilter"
xmlns:f="http://www.composite.net/ns/function/1.0">
            <f:param name="ParentId"
value="${C1:Data:Demo.ParentType:Id}"/>
          </f:function>
        </FunctionFilter>
      </Filters>
    </DataElements>
  </Children>
</DataElements>
```

Listing 23: An example of using a filter function

8 How to Attach Actions to Tree Elements

You can add one or more actions to elements in tree structures. It includes both the existing tree structures (for example, the hierarchy of pages in the Content perspective) and custom tree structures (you create as part of your console application).

You can add actions to Element, DataElements and DataFolderElements elements as well as ElementRoot.

Normally, attaching actions to the ElementRoot is used with auto-attached applications as the way of adding custom actions to existing elements (for example, pages). The application is auto-attached to a specific data type rather than forms a tree structure of its own. Other uses of attaching actions to ElementRoot are not normally applicable.

Please note that you should always add actions to elements within an Actions element.

```
<!-- parent element: start tag -->
  <Actions>
    <!-- action 1 -->
    <!-- action 2 -->
  </Actions>
<!-- parent element: end tag -->
```

Listing 24: Using Actions when attaching actions to elements

The Actions element can contain:

- AddDataAction
- WorkflowAction
- CustomUrlAction
- ReportFunctionAction
- MessageBoxAction
- ConfirmAction

As a child element of DataElements and DataFolderElements, it can additionally contain:

- EditDataAction
- DeleteDataAction

The latter means that only the DataElements and DataFolderElement allow the edit and delete operations on its elements.

Logically, the actions can be divided into three major groups:

- The first group includes actions that execute standard C1 CMS specific data workflows such as adding, editing and deleting a data item.
- The second group consists of actions that execute custom commands in your console application. For such commands, you can use a custom workflow, a custom CMS function and a custom ASPX page.
- The third group comprises actions that display two types of dialog boxes such as message boxes and confirmation boxes.

In the following few chapters, you will learn more about the actions in each of these groups.

9 How to Execute Standard Data Workflows

When you work with data in C1 CMS, you use its standard data workflows. These data workflows allow you to add, edit or delete data items.

When you add or edit a data item, a data editing form opens specific to the data type in question. You can however customize the form in many cases by editing its form markup.

If you use data elements of a specific type in your console application, you can use these standard C1 CMS-specific data workflows to manage the data items.

Three actions are available for executing the standard data workflows:

- AddDataAction
- EditDataAction
- DeleteDataAction

Please note that EditDataAction and DeleteDataAction can be only used with DataElements and DataFolderElements.

9.1 How to Add Data

When you want to add a child element to a tree element in C1 CMS (for example, a sub page to a page, or a data item to a data type), you use its standard Add Data workflow. Normally, you select the parent element and click Add on the toolbar or in the context menu.

Figure 18: Standard Add Data action

It opens a specific “Add Data” form.

When added, the new item appears as a child element under the parent element.

To attach the Add Data workflow to an element, you should use an AddDataAction element:

1. Locate an element you want to attach the action to.
2. Add an Actions element within the element if necessary.
3. Add an AddDataAction element within the Actions element.
4. Set the required attribute:
   - Type: The data type to add an item to

If necessary, set its optional attributes:

- Label: A custom label of the action
- Tooltip: A custom tooltip of the action
- Icon: The icon of the action that appears on the toolbar and in the context menu
- CustomFormMarkupPath: The path to an alternate Form UI XML file

```
<Element Label="Blog Entries" Id="Root">
  <Actions>
    <AddDataAction
      Label="Add Blog Entry"
      Type="Composite.Community.Blog.Entries"/>
  </Actions>
</Element>
```

Listing 25: Attaching an Add Data action

Please note that the type you specify in the Type attribute (Step 4) must match that of the child elements contained in the parent element.

9.2 How to Edit Data

When you want to edit a tree element in C1 CMS (for example, a page, a datatype, or a data item), you use its standard Edit Data workflow. Normally, you select the element and click Edit on the toolbar or in the context menu.

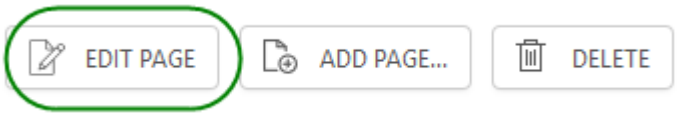


Figure 19: Standard Edit Data action

It opens an “Edit Data” form.

To attach the Edit Data workflow to a tree element, you should use an EditDataAction element:

1. Locate an element you want to attach the action to.
2. Add an Actions element within the element if necessary.
3. Add an EditDataAction element within the Actions element.

If necessary, set its optional attributes:

- Label: A custom label of the action
- Tooltip: A custom tooltip of the action
- Icon: The icon of the action that appears on the toolbar and in the context menu
- CustomFormMarkupPath: The path to an alternate Form UI XML file

```
<DataElements Type="Composite.Community.Blog.Entries"
Label="${C1:Data:Composite.Community.Blog.Entries:Title}" Display="Auto">
  <Actions>
    <EditDataAction
      Label="Edit Blog Entry" />
  </Actions>
</DataElements>
```

Listing 26: Attaching an Edit Data action

Please note that the EditDataAction is only available for DataElements and DataFolderElements.

9.3 How to Delete Data

When you want to delete a tree element in C1 CMS (for example, a page, or a data item), you use its standard Delete Data workflow. Normally, you select the element and click Delete on the toolbar or in the context menu.

Figure 20: Standard Delete Data action

To attach the Delete Data workflow to a tree element, you should use a DeleteDataAction element:

1. Locate an element you want to attach the action to.

2. Add an Actions element within the element if necessary.
3. Add a DeleteDataAction element within the Actions element.

If necessary, set its optional attributes:

- Label: A custom label of the action
- Tooltip: A custom tooltip of the action
- Icon: The icon of the action that appears on the toolbar and in the context menu

```
<DataElements Type="Composite.Community.Blog.Entries"
Label="${C1:Data:Composite.Community.Blog.Entries:Title}" Display="Auto">
  <Actions>
    <DeleteDataAction
      Label="Delete Blog Entry" />
  </Actions>
</DataElements>
```

Listing 27: Attaching a Delete Data action

Please note that the DeleteDataAction is only available for DataElements and DataFolderElements.

9.4 How to Use Custom Forms

When you use AddDataAction and EditDataAction, C1 CMS displays default editor forms.

Along with editing the Form Markup as described in Editing Form Markup (An Advanced Guide to Data Types) to customize the default forms, you can create your own forms and use them with these actions instead.

Please refer to the form schema definition files located at:

~\Composite\schemas\FormsControls

You will normally keep the form definition files on the website in a specific folder. To start using them, you should specify the relative path to them in the CustomFormMarkupPath attribute:

```
<EditDataAction
  Label="Edit Task"
  CustomFormMarkupPath="~\Frontend\Tasks\Forms\EditTask.xml" />
```

Listing 28: Using a custom form with an Edit Data action

10 How to Execute Custom Commands

When creating console applications, you have three ways of executing custom commands. Each action corresponds to the approach you can take when implementing your custom command: a workflow, a CMS function, or an ASPX page.

- WorkflowAction executes custom workflows.
- CustomUrlAction opens custom ASPX pages on the website.
- ReportFunctionAction executes CMS functions that return data in XHTML format.

10.1 How to Execute Custom Workflows

Along with the standard data workflows, you can attach custom workflows to both existing CMS elements and elements defined in your console applications. As a result, users will be able to invoke these workflows in the manner they invoke the standard ones.

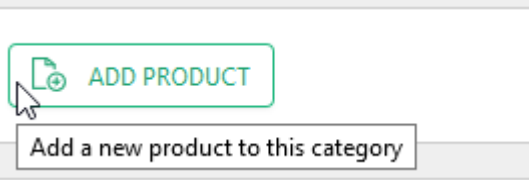


Figure 21: Custom workflow action

The workflow must be created and deployed within the website in advance. For information on creating and using workflows in console applications, please read Form Workflows.

To attach a custom workflow to a tree element, you should use the WorkflowAction element:

1. Locate an element to attach the workflow to.
2. Add an Actions element within the element if necessary.
3. Add a WorkflowAction element within the Actions element.
4. Set its required attributes:
   - WorkflowType: The type of the workflow
   - Label: The label of the workflow action

If necessary, set its optional attributes:

- Tooltip: The custom tooltip of the workflow action
- Icon: The icon of the action
- PermissionTypes: A list of permissions on the workflow action
- Location: The location of the workflow action's button on the toolbar

```
<Element Label="Products" Id="C1WorkflowDemo" Icon="pagetype-pagetype-
rootfolder" OpenedIcon="pagetype-pagetype-rootfolder-open">
  <Actions>
    <WorkflowAction
      Label="Add product..."
      ToolTip="Add a new product to this category"
      Icon="page-add-page"
      PermissionTypes="add"
      Location="Add"
      WorkflowType="C1WorkflowDemo.Workflows.AddNewProductWorkflow,
C1WorkflowDemo" />
  </Actions>
</Element>
```

Listing 29: Attaching a custom workflow

10.2 How to Open ASPX Pages

The right-pane view can display ASPX pages located on the website. All you have to do is specify its URL and select one of the views to display the ASPX page in. (For external URLs, see Opening External URLs.)

You should put your custom ASPX pages in /Composite/InstalledPackages or /Composite/InstalledPackages/content/views/<package name>, particularly, if you want to use the controls defined in ~/Composite/web.config.

The URL should be relative and start with "~" (resolved by C1 CMS to the current application path): for example, "~/HelloWorld.aspx", which means that the ASPX page "HelloWorld" is placed in the root folder of the website.

Based on your needs you can open the ASPX page in one of the four types of document view:

- Generic View
- Document View
- Page Browser
- File Download View

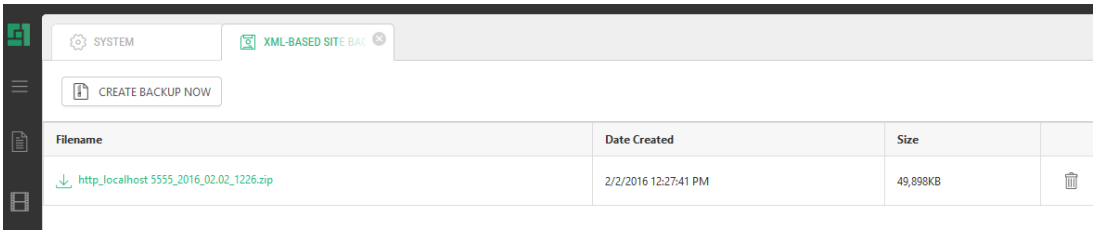


Figure 22: ASPX page opened in the generic view

When the user selects the element you have attached your custom ASPX page to, a button appears in the toolbar and in the context menu.

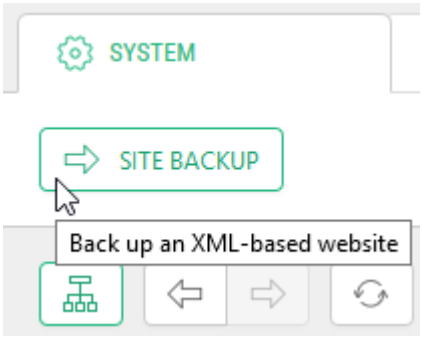


Figure 23: Custom URL action

The ASPX page must include the minimal markup (see below) to work properly. This will ensure that the page's title is displayed as the tab's title.

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://www.w3.org/1999/xhtml"
xmlns:control="http://www.composite.net/ns/uicontrol">
<control:httpheaders runat="server" />
       <head runat="server">
              <title>Page title</title>
                     <control:scriptloader type="sub" runat="server" />
       </head>
       <body>
              <ui:page>
                     …
              </ui:page>
       </body>
</html>
```

Listing 30: The mimimal markup of the ASPX page for custom URL actions

To attach this action, you should use the CustomUrlAction element:

1. Locate an element to attach the workflow to.
2. Add an Actions element within the element if necessary.
3. Add a CustomUrlAction element within the Actions element.
4. Set its required attributes:
   - Url: The URL to open
   - Label: The label of the custom URL action

If necessary, set its optional attributes.

- Tooltip: The custom tooltip of the custom URL action
- PermissionTypes: A list of permissions on the custom URL action
- ViewType: The type of the view to open the URL in
- ViewLabel: The label of the view
- ViewToolTip: The tooltip of the view
- ViewIcon: The icon of the view

```
<Actions>
  <CustomUrlAction
  Label="Calendar"
  Url="~/Composite/InstalledPackages/Calendar.aspx" />
</Actions>
```

Listing 31: Attaching an action to open a custom URL

If the URL requires one or more “post” parameters to go with:

1. Add a PostParameters element within the CustomUrlAction element.
2. Add a Parameter element within PostParameters element.
3. Specify its required attributes:
   - Key: The key part of the “post” parameter used with a custom URL
   - Value: The value part of the “post” parameter used with a custom URL
4. Repeat Steps 2-3 for as many parameters as you need.

```
<Actions>
  <CustomUrlAction Label="Calendar"
Url="~/Composite/InstalledPackages/Calendar.aspx">
    <PostParameters>
      <Parameter Key="Style" Value="Enhanced" />
      <Parameter Key="WeekStartsOnMonday" Value="True" />
      <Parameter Key="ShowWeekNumbers" Value="True" />
    </PostParameters>
  </CustomUrlAction>
</Actions>
```

Listing 32: Using post parameters with a custom URL action

Alternatively, you can specify post parameters as part of the URL.

```
<Actions>
  <CustomUrlAction Label="Calendar"
Url="~/Composite/InstalledPackages/Calendar.aspx?WeekStartsOnMonday=True&am
p;ShowWeekNumbers=True" />
</Actions>
```

Listing 33: Appending post parameters to the URL of a custom URL action

If you use post parameters in the URL itself, you can also specify the dynamic values, represented in the markup below with ${…} syntax.

```
<DataElements Type="Composite.Data.Types.IPage">
  ...
  <Actions>
    <CustomUrlAction
Url="~/WebForm1.aspx?Id=${C1:Data:Composite.Data.Types.IPage:Id}"
Label="ACTION 1!" />
    <CustomUrlAction
Url="~/WebForm1.aspx?Title=${C1:Data:Composite.Data.Types.IPage:Title}"
Label="ACTION 2!" />
    <CustomUrlAction
Url="~/WebForm1.aspx?Id=${C1:Data:Composite.Data.Types.IPage:Id}&amp;Title=
${C1:Data:Composite.Data.Types.IPage:Title}" Label="ACTION 3!" />
  </Actions>
</DataElements>
```

Listing 34: Using post parameters with dynamic values in the URL

10.2.1 Opening External URLs

You can also open external URLs in a new window with the Custom URL Action. For this, specify:

- the external URL in the Url attribute (e.g. 'http://www.contoso.com/')
- 'externalview' in the ViewType attribute

```
<CustomUrlAction Label="External URL action" Url="http://www.contoso.com"
ViewType="externalview" />
```

Note: For Internet Explorer 9, the website should be added to the "Trusted sites"; otherwise, the popup blocker may restart the website and any unsaved changes will be lost.

10.3 How to Execute CMS Functions

You can attach actions to both existing and newly-defined tree elements to display custom XHTML-based content in a document view by executing CMS functions.

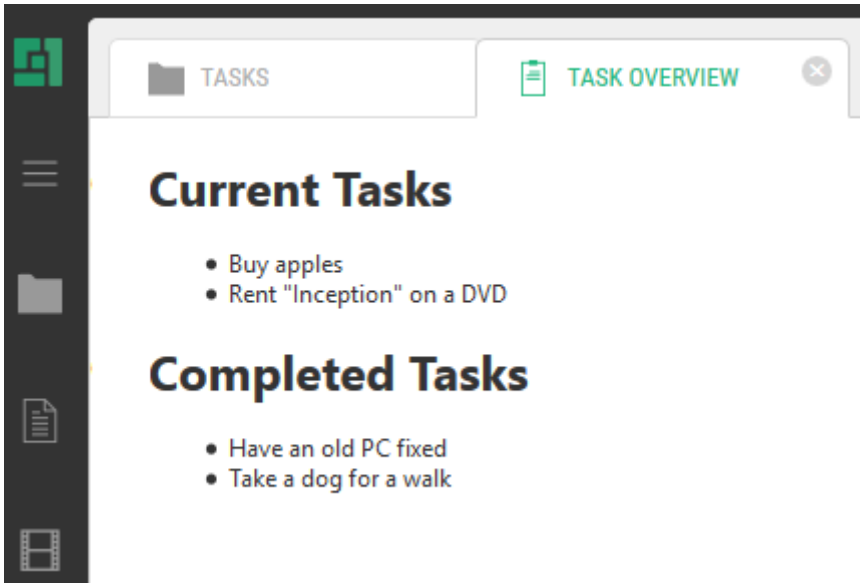


Figure 24: CMS function outputting XHTML in a document view

When the user selects the element you have attached your CMS function to, a button appears in the toolbar and in the context menu.

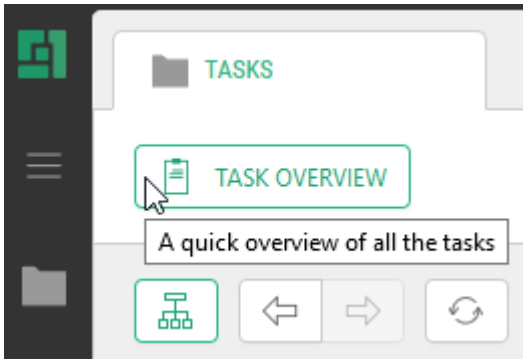


Figure 25: A report function action

The CMS function must be created in advance and output the content in XHTML format. (Read more about creating CMS functions.)

To attach such an action to an element, you should use a ReportFunctionAction element:

1. Locate an element to attach the workflow to.
2. Add an Actions element within the element if necessary.
3. Add a ReportFunctionAction element within the Actions elements.
4. Set its required attribute:
   - Label: A label of the custom URL action.
5. Add a f:function element within the ReportFunctionAction element.
6. Set its required attributes:
   - name: The name of the CMS function

If the function requires so:

7. Add one or more f:param elements within the f:function element.
8. Set its required attribute:
   - name: The name of the CMS function's parameter

If necessary, set its optional attribute:

- value: The value of the CMS function's parameter

If necessary, set the optional attributes of the ReportFunctionAction element:

- Tooltip: The custom tooltip of the report function action
- Icon: The icon of the action
- PermissionTypes: A list of permissions on the report function action
- DocumentLabel: The label of the document
- DocumentIcon: The icon of the document

```
<Actions>
  <ReportFunctionAction Label="Task Overview" Tooltip="A quick overview of
all the tasks">
    <f:function name="DisplayTasks"
      xmlns:f="http://www.composite.net/ns/function/1.0">
      <f:param name="ShowDueDates" value="true" />
    </f:function>
  </ReportFunctionAction>
</Actions>
```

Listing 35: Attaching a report function action to execute a CMS function

11 How to Display Messages

C1 CMS normally displays two types of messages to the user:

- Message boxes that inform, ask, warn the user or report an error. They serve mostly informative purposes.

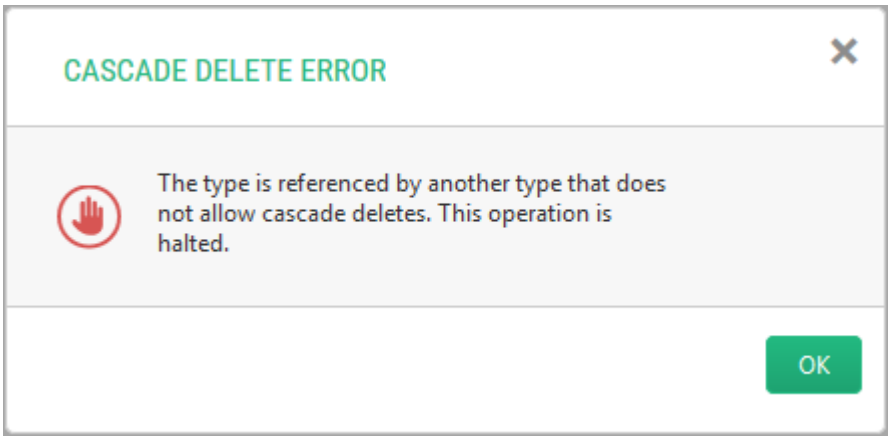


Figure 26: A message box

- Confirmation boxes that ask the user to confirm the action he or she are about to do (e.g. delete a page). If the user confirms it (by clicking “OK”) the action will be executed; otherwise, aborted.

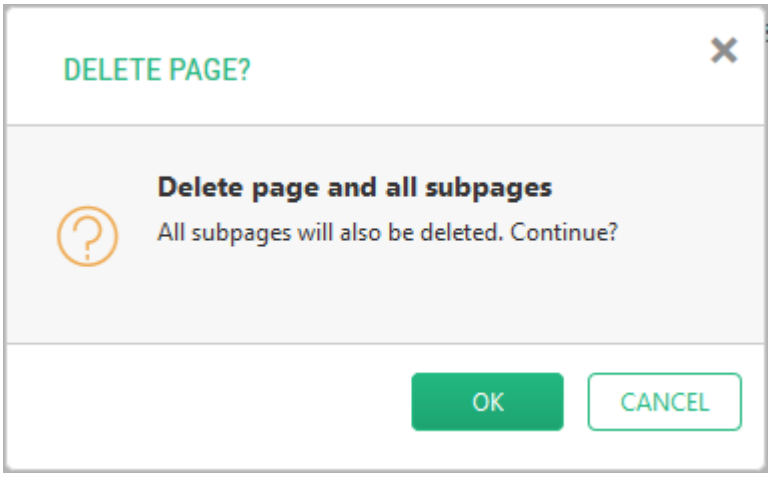


Figure 27: A confirmation box

11.1 How to Display Message Boxes

To attach an action to a tree element to display a message box, you should use the MessageBoxAction element:

1. Locate an element to attach the workflow to.
2. Add an Actions element within the element if necessary.
3. Add a MessageBoxAction element within the Actions elements.
4. Set its required attribute:
   - Label: The label of the action that shows the message box
   - MessageBoxTitle: The title of the message box
   - MessageBoxMessage: The message of the message box

If necessary, set its optional attributes:

- MessageDialogType: The type of the message box
- Icon: The icon of the action
- Tooltip: The custom tooltip of the action that shows the message box
- PermissionTypes: A list of permissions on the message box action

```
<Actions>
  <MessageBoxAction
    Label="GUID"
    MessageBoxTitle="${C1:Data:Composite.Data.Types.IPage:Title}"
    MessageBoxMessage="${C1:Data:Composite.Data.Types.IPage:Id}"/>
</Actions>
```

Listing 36: Attaching an action to display a message box

11.2 How to Display Confirmation Boxes

To attach an action to display a confirmation box, you should use a ConfirmAction element:

1. Locate an element to attach the workflow to.
2. Add an Actions element within the element if necessary.
3. Add a ConfirmAction element within the Actions elements.
4. Set its required attribute:
   - Label: The label of the action
   - ConfirmTitle: The title of the confirmation box
   - ConfirnMessage: The message of the confirmation box
5. Add a f:function element within the ConfirmAction element.
6. Set its required attributes:
   - name: The name of the CMS function

If the function requires so:

7. Add one or more f:param elements within the f:function element.
8. Set its required attribute:
   - name: The name of the CMS function's parameter

If necessary, set its optional attribute:

- value: The value of the CMS function's parameter

If necessary, set the optional attributes of the ConfirmAction element:

- RefreshTree: When set to "true", the tree refreshes if the user clicks "OK"
- Icon: The icon of the action
- ToolTip: The tooltip of the action
- PermissionTypes: A list of permissions on the custom URL action
- Location: The location of the action's button on the toolbar

```
<Actions>
  <ConfirmAction Label="Delete Completed Tasks" Tooltip="Delete all the
tasks marked completed">
    <f:function name="Demo.Tasks.DeleteCompleted"
      xmlns:f="http://www.composite.net/ns/function/1.0" />
  </ConfirmAction>
</Actions>
```

Listing 37: Attaching an action to display a confirmation box

12 Troubleshooting

In this section, you will learn to troubleshoot issues that might occur when you work with applications in C1 CMS.

12.1 My Application Won't Show Up Automatically

- Make sure that your application is configured as an "auto attachment".
- Make sure that the markup of your application is correct and you refer to existing types and objects in values.

12.2 There Is No "Add Application" Menu Command

- Make sure that your application is configured as an "allowed attachment".
- Make sure that the markup of your application is correct and you refer to existing types and objects in values.

12.3 An Application Won't Appear in Its Own Perspective

- Make sure that you set up permissions on the perspective for users/user groups that should access it.

12.4 There Are No Elements in the Tree

- Make sure that the markup of your application is correct and you refer to existing types and objects in values.
- Make sure that you have used the Children element as a parent for nested elements.

If you work with data elements:

- Make sure that you that the data type you use has data items.
- Make sure that you that you specified the data type's name correctly.
- Make sure that the filters you apply to data elements are set up correctly.
- Make sure that the Display attribute on the element is not set to "Compact" accidentally. If the data element is supposed to have child elements and, it might be hidden if it actually has none.

12.5 There Are No Action Buttons on the Toolbar / in the Menu

- Make sure that the markup of your application is correct and you refer to existing types and objects in values.
- Make sure that you have used the Actions element as a parent for actions.

12.6 I Can't Attach EditDataAction to an Element

- Make sure that the element you want to attach the action is either DataElements or DataFolderElements. You cannot attach this action to Element or ElementRoot.

12.7 I Can't Attach DeleteDataAction to an Element

- Make sure that the element you want to attach the action is either DataElements or DataFolderElements. You cannot attach this action to : Element or ElementRoot.

13 Test Your Knowledge

13.1 Task: Create an Application to Attach to Pages

1. Create the tree definition file “TestApps.xml”.
2. Set up the application so that users can attach it to pages manually (“Composite.Data.Types.IPage”).
3. Set its name to “Test Apps”.
4. Select a page in “Content” and check for “Add application” in its context menu.
5. Click “Add application” and check for “Test Apps” to appear on the list.

13.2 Task: Attach a Message Box to Pages

1. Edit “TestApps.xml”.
2. Set up the application to auto-attach to pages.
3. Add the message box action to the root element of the tree structure.
4. Set its label, title and message to “Hello World”.
5. Select a page in “Content” and check for the “Hello World” button on the toolbar.
6. Click the button and check for the message box.

13.3 Task: Attach the Application to Its Own Perspective

1. Edit “TestApps.xml”.
2. Set up the application to auto-attach to its own perspective.
3. Add a simple element to the application tree structure with the label “Contacts” and ID “ContactsPerspective”.
4. Insert another simple element in the “Contacts” element with the label “Contacts” and ID “ContactsTree”.
5. Grant permissions to the new perspective in “Users”
6. Check for the new perspective and the “Contacts” element in it.

13.4 Task: Retrieve Data Elements from a Data Type

1. Create the data type “Test.Contacts” and add fields: “Name” (string), “Company” (string) and “Country” (string).
2. Add five or more contacts to the data type only using either “CompanyA” or “CompanyB” and “US” and “Denmark”.
3. Edit “TestApps.xml”.
4. As a child of the “Contacts” element, add the element that retrieves its data items from the “Test.Contacts” data type.
5. Check for the items in the “Contacts” perspective.

13.5 Task: Group Data Elements

1. Edit “TestApps.xml”.
2. Within the “Contacts” element with the ID of ”ContactsTree”, insert the data folders element that group the “Test.Contacts” data items by the “Company” field.
3. Move the “Test.Contacts” data elements into the data folders element.
4. Check for the data folders that group data items in the “Contacts” perspective by “Company”.

13.6 Task: Sort Data Elements

1. Edit “TestApps.xml”.
2. Sort the “Test.Contacts” data elements by “Name” in descending order.

3. Check for the sorting order of data elements in the “Contacts” perspective.

13.7 Task: Filter Data Elements

1. Edit “TestApps.xml”.
2. Filter data elements by “Country” only selecting contacts from Denmark.
3. Check for the contacts now present in the “Contacts” perspective.

Hint: To hide empty parent elements (data folders) unless they have child elements (data elements), set the Display attribute on the former to “Compact”.

13.8 Task: Attach Data Actions to Elements

1. Edit “TestApps.xml”.
2. Within the simple “Contacts” elements with the ID of “ContatcsTree”, insert an action to add data items of the “Test.Contacts” type. Set its label to “Add Contact”.
3. Repeat Step 2 within the data folders element with “Company” as its grouping field.
4. Within the data elements of the “Test.Contacts” type, insert actions to edit and delete data items. Set its labels to “Edit Contact” and “Delete Contact”.
5. In the “Contacts” perspective, Check for the buttons when selecting the “Contacts” element, a grouping data folder element, a data item.
6. Add, edit and delete data items.

13.9 Task: Open an ASPX Page

1. Create an ASPX file with simple content in the root of the website.
2. Edit “TestApps.xml”.
3. Within the simple “Contacts” elements with the ID of “ContactsTree”, add an action to open this ASPX file in the view.
4. Set its label to “Open Page”.
5. In the “Contacts” perspective, check for the “Open Page” button.
6. Click the button and check the result.

13.10 Task: Execute a CMS Function

1. Create an XSLT function that outputs simple XHTML.
2. Edit “TestApps.xml”.
3. Within the simple “Contacts” elements with the ID of “ContactsTree”, add an action to execute the XSLT function.
4. Set its label to “Execute Function”.
5. In the “Contacts” perspective, check for the “Execute Function” button.
6. Click the button and check the result.