

Sitecore Foundry Developers Guide

Author: Sitecore Corporation

Date: February 27, 2008

Release: Rev. 0.9

Language: English

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders.

*The contents of this document are the property of Sitecore.
Copyright © 2001-2008 Sitecore. All rights reserved.*

Table of Contents

Chapter 1	Introduction	4
1.1	Features	4
1.2	Standard Sitecore functionality	4
Chapter 2	Concepts	6
2.1	The Site Type	6
2.2	Standard Sitecore Components	6
2.3	Sitecore Foundry Specific Settings	6
2.4	Website Layouts and Site types	7
2.5	Security Template	8
2.6	Site Template	9
2.7	Administration Menu	9
2.8	Global Menu	9
2.9	Dictionaries	9
2.9.1	Dictionary Locations	10
2.9.2	Example	11
2.10	Skins and Wizards	12
2.11	Modules and Module Settings	12
2.12	Site Settings	13
Chapter 3	Site Modification Tutorials	14
3.1	What does the wizard generate?	14
3.2	Default styles	16
3.3	System CSS styles	19
3.4	Form style	19
3.5	List style	21
3.6	XSL helper functions	21
3.7	Tutorials	23
3.7.1	Creating a top menu rendering	23
3.8	Using dictionary lookup in XSL controls	25
3.8.1	Built-in dictionary controls	26
3.9	Creating a new module	29
3.10	Adding a new Site Wizard step	34
3.11	Modifying the local administration menu	39
Chapter 4	Reference	42
4.1	Sitecore Foundry content structure	42
4.1.1	Foundry Content Items	42
4.1.2	Foundry Administration Items	43
4.1.3	Sitecore Foundry Modules Items	43
4.1.4	Sitecore Lookup tables items	45
4.1.5	Site Creation Wizards Items	46
4.1.6	Modules MSS items	47
4.1.7	Template Sites items	48
4.2	Site Type Components	48
4.3	Site Type Fields	49
4.3.1	Templates	50
4.3.2	Masters	51
4.3.3	Renderings	51
4.3.4	Layouts	53
4.3.5	Sub-layouts	53
4.4	The Runtime Engine	54
4.4.1	The Server class.	57
4.4.2	SiteContextManager and SiteContext	58
4.4.3	SiteTypeManager and SiteTypes	59
4.4.4	Dictionary and DictionaryManager.	60
4.4.5	ModuleManager and Module	61
4.4.6	SkinPackageManager, SkinPackage and Skin	61



4.5	Security	62
4.5.1	The Sitecore domain	62
4.5.2	The Extranet domain	62
4.5.3	Foundry domains	62
4.5.4	Users, Groups and Roles	63
4.5.5	Setting up the Local Admin Rights	63
4.6	The structure of a site	65
4.7	Sitecore Foundry Maintenance Service	67
4.7.1	Creating a new site	68
4.7.2	Deleting a site	68
4.7.3	Checking to see if a site exists	69
4.7.4	Backup and restore a site	69
4.7.5	Work with packages	69
<hr/>		
Chapter 5	The wizard and skin packages	71
5.1	Introduction	71
5.2	Skin Packages	72
5.3	Skin packages and site types	74
5.4	Skin Fields	74
5.4.1	Skin package data items	74
5.4.2	Skin package components	77
5.5	Preview images	81
5.6	Skin package files	81
5.7	Creating your own skin package	81
5.8	Customizing the wizard	82
5.8.1	Hiding or Removing Unwanted Steps	82
5.8.2	Adding Steps	83
5.9	Wizard generated CSS classes	84
<hr/>		
Chapter 6	Appendix A: Configuration	86
6.1	Configuration	86
6.1.1	MSS.config	86
6.2	Sitecore Foundry specific additions to web.config	87
6.2.1	<AppSettings>	87
6.2.2	Item:saved Event Handlers	87
6.2.3	Item:deleted event handlers	87
6.2.4	Publish:end event handlers	88
6.2.5	Pipeline processors	88
6.2.6	Database changes	88
6.2.7	Indexes	89
6.2.8	Domains	89
6.2.9	Processors	89
6.2.10	xslExtensions	89
6.2.11	xslControls	89
6.2.12	controlSources	90
6.2.13	References	90
6.2.14	Settings	90
6.2.15	log4net	90
6.2.16	http Modules	91
<hr/>		
Chapter 7	Appendix B: Wizard styling output	92
7.1	Link Example	92
<hr/>		
Chapter 8	Appendix C: Creating a new site type manually	94
8.1	Required components	94
8.1.1	Making A New Site Type From Scratch	96
<hr/>		
Chapter 9	Appendix D: Standard website sizes	98
9.1	Standard Sizes	98

Chapter 1

Introduction

Sitecore Foundry provides a framework based on the Sitecore CMS for running multiple websites within one solution. The sites can be based on multiple different website designs, hence the term “Multi Site Solution”. The framework is designed to isolate the individual sites from each other and also provide individual site and global functionality. Sitecore Foundry has a runtime engine that keeps track of the information necessary for a site. Sitecore Foundry, in addition, consists of several web controls, user controls and Xsl controls to help develop new sites and modules. It is necessary to understand that when implementing new site design, most of the work is done using Standard Sitecore functionality and the Sitecore client. This developer guide does not go into Sitecore specific implementation issues, but explains how Sitecore Foundry is build on top of Sitecore.

The internal code name for the Sitecore Foundry is **Multi Site Solution** abbreviated to **MSS**, which can be seen in many places such as in the web.config file, the naming of system dll files, in the API namespaces and in Sitecore content.

The abbreviation **MSCC** is short for **Multi Site Control Center**, which is the component of Sitecore foundry that manages sites and site types.

1.1 Features

This developer’s guide is divided into three parts:

1. **Concepts.** This starts the guide by giving you an introduction to the central concepts of Sitecore Foundry.
2. **A how-to guide.** Describes how to implement a new site design by looking at how the default site type accompanying the product is composed.
3. **References.** A more thorough explanation of the different aspects of the system.

1.2 Standard Sitecore functionality

A large part of Sitecore Foundry uses standard Sitecore functionality to produce the end result. The following features are part of standard Sitecore functionality used in Foundry:

- The Sitecore layout engine is used to render the sites using layouts, sublayouts and renderings.
- WebEdit mode for editing website content.

- Layout-groups to render the website in normal website view and printer-friendly view.
- Templates for defining storage for document data.
- Masters for adding templates and documents and security settings.
- Layout and rendering information stored on templates and masters for directing the layout engine.
- Both extranet and Sitecore security, through a separate security layer build on-top.
- Media library for storing media used on the sites.
- Language support.

Chapter 2

Concepts

In the following chapter the concepts of a site type, site template, dictionary, wizard, skins, modules and site settings will be explained.

2.1 The Site Type

The Site type is a central concept in Sitecore Foundry as every site is created using a specific site type. A Site type is a combination of a unique collection of standard Sitecore components and Sitecore Foundry specific components and settings. These are used along with Site templates to create the sites within the Foundry. A site type, along with its content template can then be used to quickly create a multitude of sites all along the same theme. These themes, set using the various settings and the website wizard can then be tailored for each site type to give the groups of sites their own unique look and feel.

2.2 Standard Sitecore Components

These are the standard Sitecore components within the Foundry model.

1. Templates
2. Masters
3. XSLT renderings
4. Web Controls
5. Sub Layouts
6. Layouts
7. Security
8. Media

2.3 Sitecore Foundry Specific Settings

Sitecore Foundry has several components and settings which are unique to the Foundry and not part of the standard Sitecore model. These are;

1. **Security** - An initial default user and three default groups are created each time a new site is generated by the Foundry. For more information about Security see the concept “Security Template”, on page 8.
2. **Site Template** – This is where the initial content for new sites is stored. When a new site is created using the Multi Site Control Center (MSCC) the initial content

for the new site is taken from here. For more information about the site template see the concept “2.6 Site Template”, on page 9.

3. **Administration Menu** – This is where the definitions of the Local Administration Menu are held. Each Site type has its own local administration menu, so each different site based on a different site type can have a different local administration menu. For more information about the Administration Menu see the concept “2.7 Administration Menu”, on page 9.
4. **Global Menu** – You can have different predefined menu structures for specific site types. For more information about Global Menu’s see the concept “2.8 Global Menu”, on page 9.
5. **Dictionary items** – Each site and site type has its own dictionary that overrides the text defined in the system dictionary. For more information about site dictionaries see the concept “2.9 Dictionaries”, on page 9.
6. **Wizard type** – a predefined and user definable wizards at the site type level to customize a site. For more information about wizard types see the concept “2.10 Skins and Wizards”, on page 12.
7. **Modules** – This defines which modules are available for use with specific site types. For more information about modules see the concept “2.11 Modules and Module Settings”, on page 12”.
8. **Module settings** – The site specific settings override the default module settings. For more information about Module settings see the concept “2.11 Modules and Module Settings”, on page 12.
9. **Site settings** – Default settings for either site types or individual sites. For more information about Site settings see the concept “2.12 Site Settings”, on page 13.

2.4 Website Layouts and Site types

There is a fixed one to one relationship between a website layout and a Site type. For a unique website layout there is a unique Site type. Layout groups are used in the Foundry model therefore there is always a normal and a print layout. A Site type, therefore, is related to at least two physical aspx files, but can consists of as many layout files as necessary.

If two types of sites are similar it is possible to base them on the same Site type. You can then use settings so that layouts and renderings can differentiate between the two when rendered to the web site. This possibility should be used with caution, however, since the task of maintaining two slightly different sites this way can become difficult. On the other hand the number of Site types should also be kept to a minimum, since every type requires Sitecore resources.

Even though two sites differ in layout they could still use the same skin.

When the Sitecore Foundry product is installed only one Site type is installed which is the default “Standard” site type. In all cases a new site type is needed to accommodate the customer’s unique layouts.

Important: This site type should not be altered since updates to the product could update this site type and overwrite any customizations.

In the Multi Site Control Center (MSCC) it is possible to create new Site types by cloning the default site type and altering it to meet specific site needs. For more information on

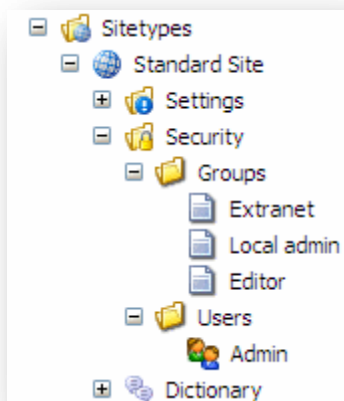
creating site types see the section “Chapter 8 Appendix C: Creating a new site type manually”, on page 94

When creating the new html, layout and design for a unique new site and site type, the possibilities are almost the same as in the full Sitecore product, with only a few restrictions.

There are no restraints as to how the layout should look and it is optional as to whether you use items such as the sitemap, contact formula, spots etc. provided by Sitecore Foundry.

2.5 Security Template

Creating a new site based on a given site type means creating a set of predefined users and groups with corresponding relationships. The product comes with a predefined user named **Admin** and three predefined groups named **Local Admin**, **Editor** and **Extranet** that are set for the Standard site type.

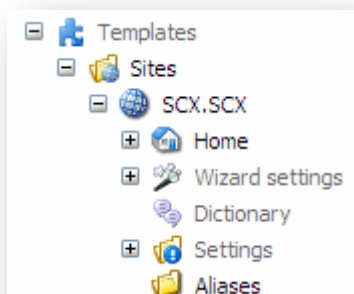


These predefined users and groups can be defined on the site type. This means that whenever you create a site all the predefined users and groups on that site type will automatically be created on the new site, saving time.

Important: It is not advisable to change the permissions and settings for these defaults as their purpose is very specific and changing the function of the defaults will affect the way they perform on the website. It is recommended that you create new users and groups if you wish to create specific user and group settings.

2.6 Site Template

Every site type is committed to using a specific Site template which defines the basic document content structure. The Site template is equivalent to a standard Sitecore website.



This consists of a root document (website front page normally called Home), default documents (menu items), global documents, a dictionary, and a variety of global settings. For more information about the site template see the section 4.6 “The structure of a site”, on page 65.

2.7 Administration Menu

The standard local administrator menu items are mandatory to facilitate the correct functioning of site administration. The Local Administration menus are stored at:
`Sitecore/content/Sitecore Foundry/Site Administrator Menus/`

The local admin menu is linked directly to each site type. There is also the facility to add custom menu items giving the ability to create custom Local Administration Menu's for each site type

2.8 Global Menu

A global menu is a common menu structure for all sites of a given site type. Using the site template these can be designed individually for each site type. It is possible, therefore, to force all related sites in the same company to have a common menu structure added by defining the global menu for the related site type.

2.9 Dictionaries

A dictionary containing text items in different languages is used in Sitecore Foundry for the many labels, buttons, dialogs and input boxes.

Text that is displayed in back-end dialogs to the administrator of the site is stored in the standard system dictionary. All other dictionary items can be defined at the following levels:

1. **Site** – Every local site holds its own dictionary.
2. **Site type** – Each Site type also contains a dictionary.
3. **Global** – At the global level a further dictionary is held.
4. **System** – Finally, there is the Foundry System dictionary.

Important: The System dictionary should not be altered since updates to the product could update this dictionary and overwrite any customizations. If custom texts are to be used on the system these should be placed on the global, site type or on the individual site dictionaries as needed.

Sitecore recommend using the following rules when deciding where to put a dictionary entry. If the item is only needed by one site, then it should be stored at the local site level. If the entry may be need by more than one similar site then it can be stored at the site type level. If the entry is needed by more than one site outside of site types or by all sites then it should be stored at the Global level.

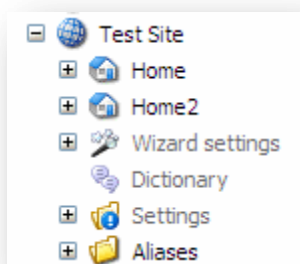
Sitecore Foundry uses a four step process in resolving the dictionary item's in the following order:

1. **Local Site dictionary** - The requested item is first searched for in the local site dictionary.
2. **Site type dictionary** - If the dictionary item is not present in the local site dictionary, the Site type dictionary is then searched.
3. **Global dictionary** - If the entry is neither defined in the site or the Site type dictionaries, then the Global dictionary is used.
4. **System dictionary** – Finally, if it is neither defined in the site, the Site type or the Global dictionaries, then the default system dictionary is used.

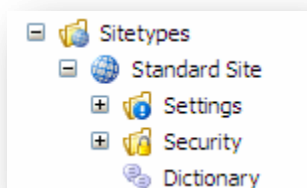
Tip: It is recommended that your local, site type and Global dictionaries are kept as up to date as possible to prevent exceptions occurring when dictionary entries are not found, or are found, but are blank.

2.9.1 Dictionary Locations

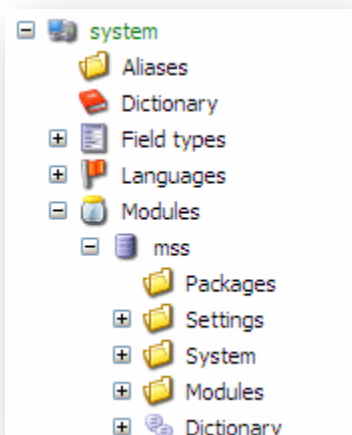
A site level dictionary is located under a site's root item and is named **Dictionary**.



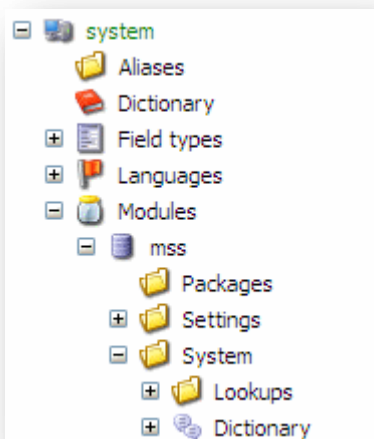
A site type level dictionary is located under a site type's root item and is also named **Dictionary**.



The global level dictionary is located at:
`/sitecore/system/Modules/mss.`



The system level dictionary is located at:
`/sitecore/system/Modules/mss/System.`



2.9.2 Example

To override the dictionary items used for the contact form, copy the dictionary items (shown by the path below) to the Site type dictionary where you wish to override the default texts.

`/Sitecore/system/modules/mss/system/dictionary/system/contactform`

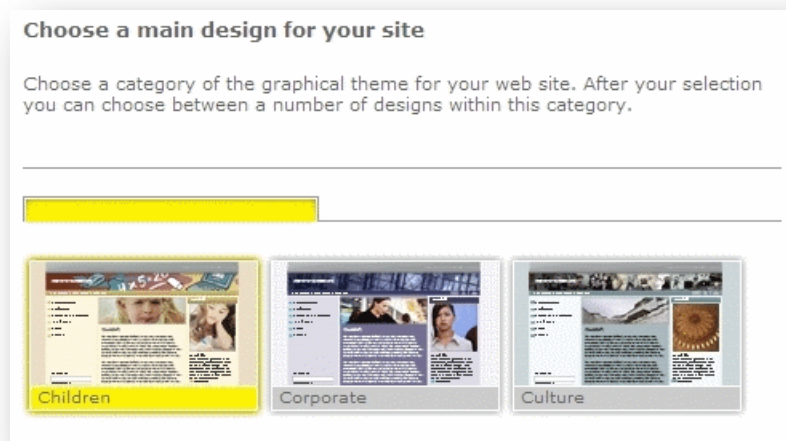
Copy to:

`/content/... ../Site type's/Test Site type/dictionary/contactform`

This will override the text used for the default system contact form on the Site type **Test Site type**.

2.10 Skins and Wizards

Sitecore Foundry comes with a customizable wizard and a default of 18 (eighteen) predefined skins divided into three Skin packages: **Children**, **Corporate** and **Culture** serving three completely different design branches.



It is possible to add complete new skin packages as required. Depending on requirements you are able to configure the wizard in a variety of ways within the given site type (e.g. to add, edit and hide Wizard steps).

The default wizard comes with a wide range of steps for setting background colors and images, buttons, contact data and settings.

Site Wizards are stored at:

`/sitecore/content/Sitecore Foundry/Site Creation/Wizards`

The wizards are linked with a Site type so if a partner or customer should wish to alter the design, it is possible to build a completely customized layout and design, so customers can see it as their own unique corporate wizard. For more information see section Chapter 5 "The wizard and skin packages", on page 71.

2.11 Modules and Module Settings

Sitecore Foundry compliant modules can be used on sites of a given site type. The site type defines which modules are available and these modules can be turned on or off in the Website Wizard. Sitecore Foundry comes with the following built-in modules:

- **Activity calendar** – A flexible events calendar that means that all news events of any importance can be effectively displayed for users to see.
- **Newsletters** - Newsletters created by local administrators can be subscribed to by users. This gives a business a great way to make sure that a whole group of interested users can receive product news and other information at the click of a button. Newsletters are very flexible and can be delivered via SMS or E-mail.
- **Image collections** - An effective way to collect and display sets of images.
- **News Archive** - The two parts of the News modules give local administrators the ability to display a variety of news articles either globally or attached to individual parts of the site.

- **Mini Forum** - The Mini Forum module gives site users a great way to discuss events. Users can create threads and post replies to other threads. Overall control of the Forum, threads and individual postings is done by local administrators.

2.12 Site Settings

There are three levels of settings for the sites within the Foundry. There are settings stored at local site level, site type level and system level.

Defaults settings for the system and modules are located at:

`/sitecore/system/Modules/mss/Settings`

The settings are retrieved in the following order.

1. **Local Site Settings** - Foundry first looks for the settings at a local site level.
2. **Site type Settings** - If it cannot find them it then moves to the settings for the site type.
3. **System Settings** - Finally it retrieves the settings from the system level.

Important: The system settings should not be altered since updates to the product could update these settings and overwrite any customizations. If custom settings need to be used on the system these should be placed on the site type or the individual site settings as needed.

Chapter 3

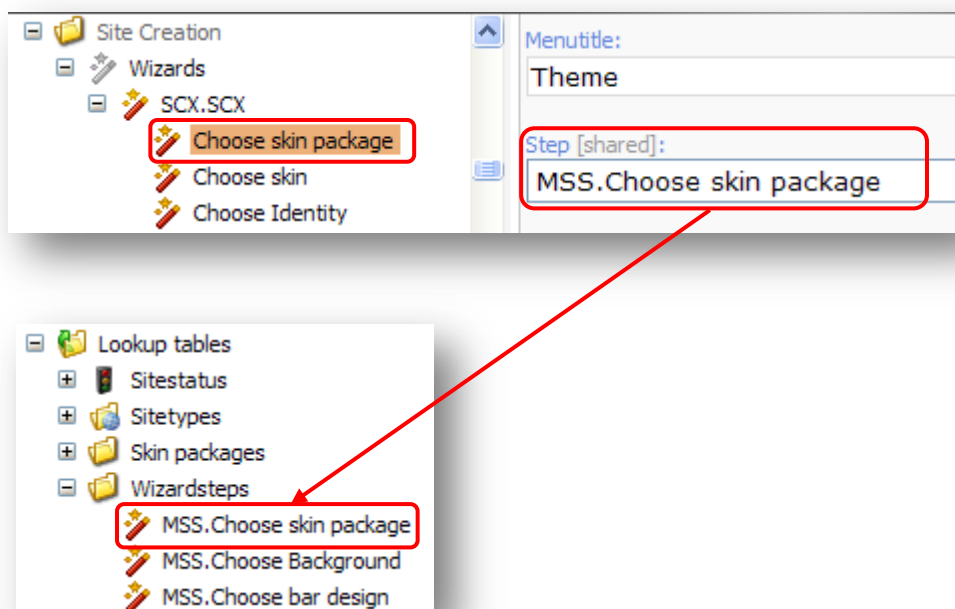
Site Modification Tutorials

In order to understand how a site type is implemented we will examine the default demo site type implementation. A basic knowledge of HTML, JavaScript, CSS styling, .NET, C# & Sitecore is required.

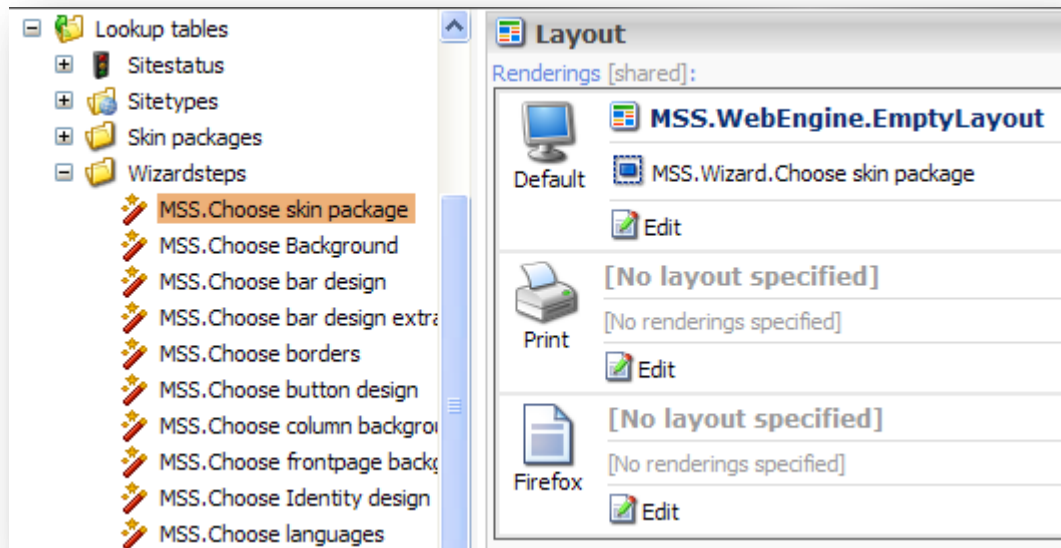
3.1 What does the wizard generate?

Let's take a look at how the presentation of a site is affected by the site wizard when a user with appropriate rights makes changes to the Website Wizard.

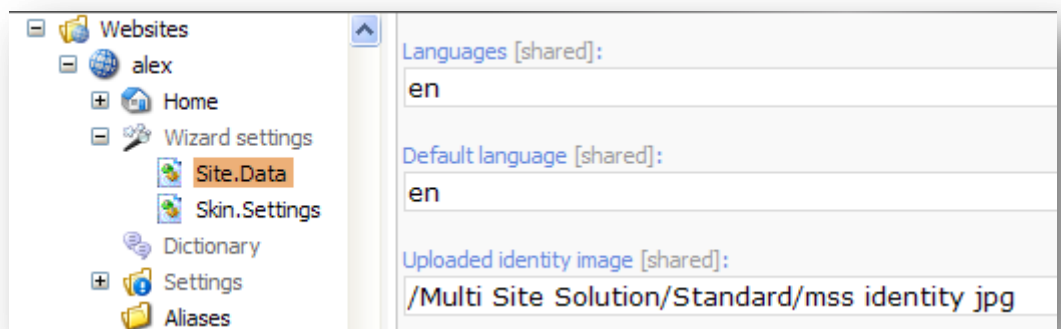
- Each step of a wizard under **/Sitecore Foundry/Site Creation/Wizards/** contains the **Step** field which references the predefined wizard step located at; **/Sitecore Foundry/Administration/Lookup tables/Wizardsteps**.



- Each wizard step has a layout assigned, which provides the functionality for the step.



The layout defined for the wizard step calls methods, which render the appropriate fields (for example, the drop-down menu with the list of skins or the color selector) and update the items **Site.Data** and **Skin.Settings** according to the selections made by the user.



The **Site.Data** and **Skin.Settings** items contain the settings defined by the Website Wizard. These settings are used by the **SkinCss.aspx** page to generate the css stylesheet dynamically as described in the following paragraph.

- The main layout contains the following stylesheet definition:

```
<link rel="stylesheet" type="text/css"
href="/sites/scx.scx/css/mss.skin.css" />
<link id="SkinCss" rel="stylesheet" type="text/css" href="/sitecore
modules/MSS/Wizard/Skins/SkinCss.aspx" />
```

The **SkinCss.aspx** page calls the **MSS.Wizard.Skins.SkinCssPage** class, which parses the **Site.Data** and the **Skin.Settings** items and outputs the css stylesheet. The **SkinCssPage** class in turn implements the **CssFactory** class which contains all settings for css generation.

So, if you want to create your own styles, you should implement the **ICssFactory** interface.

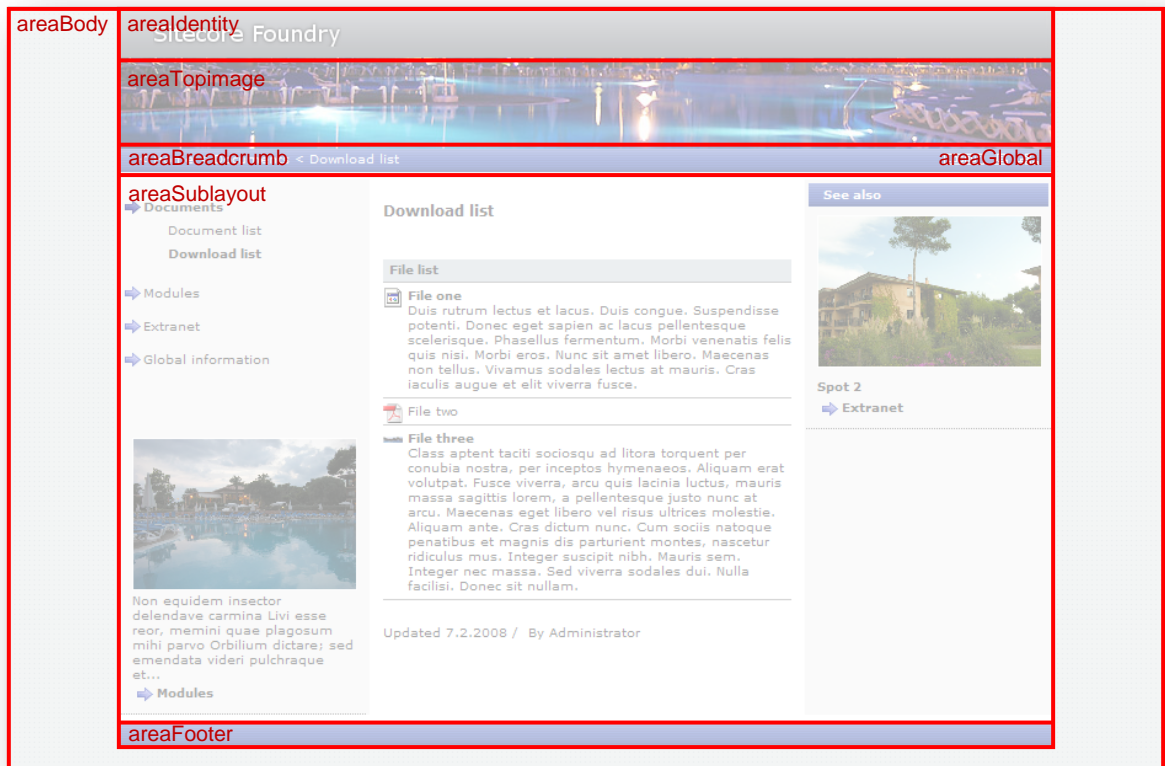
3.2 Default styles

This section describes the styles that are used on a default Sitecore Foundry site.

The following styles are used on the main layout of the demo site.

- **body.areaBody**
Used in the body HTML tag to set the background color of the site.
- **div.areaIdentity**
Used on the main layout to set the height and background image for the identity area.
- **div.areaTopImage**
Used on the main layout to set the height and background image for the top bar area. Padding for the top bar text is also set here.
- **div.areaBreadcrumb**
Used on the main layout to set the background image, the font color and the height of the breadcrumb area.
- **div.areaFooter**
Used on the main layout to set the background of the footer area.
- **div.areaGlobal**
Used on the main layout to set the padding and the “float” attribute of the global area. **Note:** Not defined by the default wizard.

- **div.areaSublayout**
Used on the main layout to set the background of the columns and the content area.

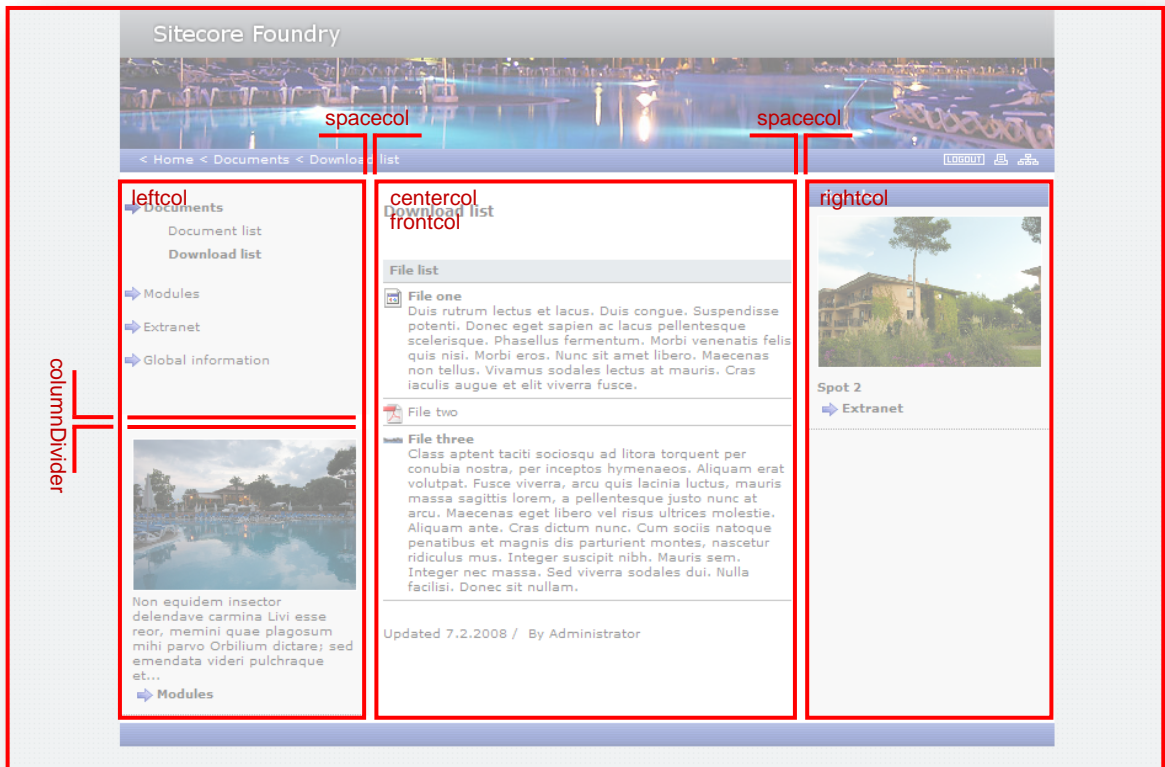


The following styles are used on sublayouts of the demo site.

- **table.sublayoutTable td.spacecol**
Used on sublayouts to set the spacing between the columns.
Note: Not defined by the default wizard.
- **table.sublayoutTable td.leftcol**
Used on sublayouts to set the background color of the left column area.
- **table.sublayoutTable td.columnDivider**
Used on sublayouts to set the spacing between columns.
Note: Not defined by the default wizard.
- **table.sublayoutTable td.centercol**
Used on sublayouts to set the background color of the center column area.
- **table.sublayoutTable td.frontcol**
Used on the front page sublayout to set the background color of the front page center column area.



- **table.sublayoutTable td.rightcol**
Used on sublayouts to set the background color of the right column area.



- **div.MssNavigation div.navButtons input**
Used on forms to set the background images for buttons.
- **img.MssListArrow**
Used to set icons for bullet points.



3.3 System CSS styles

All Sitecore Foundry layouts should inherit the **MSS.Web.UI.Pages.SitePage** class. For instance, the following line should be located at the beginning of a layout.

```
<%@ Page language="c#" Inherits="MSS.Web.UI.Pages.SitePage, MSS.WebEngine"
Codepage="65001" AutoEventWireup="false"%>
```

The **SitePage** class adds the system CSS classes to the page in the same way as if they were added at the beginning of the **<head>** html tag. These system CSS classes are used in modules, forms and lists. The classes are stored in three files:

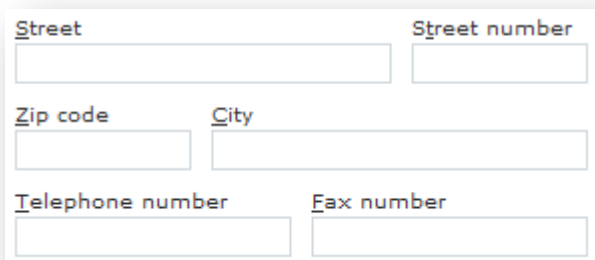
- /sitecore modules/MSS/WebEngine/CSS/MSS.Layout.css
- /sitecore modules/MSS/WebEngine/CSS/MSS.Forms.css
- /sitecore modules/MSS/WebEngine/CSS/MSS.Lines.Arrows.css

If some of the default CSS styling should be changed on a site, just override the styling by including a css file on the layouts of a site type with the changed styling.

3.4 Form style

The **MSS.Forms.css** stylesheet, which is located at **/sitecore modules/MSS/WebEngine/CSS/** provides the styles for creating forms with all necessary form elements.

For instance, input fields can be formatted using four predefined widths: **Full**, **Half**, **Third** and **Quarter**. Take a look at the following screenshot:



The screenshot shows a form with six input fields arranged in three rows. The first row has 'Street' and 'Street number'. The second row has 'Zip code' and 'City'. The third row has 'Telephone number' and 'Fax number'. Each field is a simple rectangular box with a thin border.

The classes for the input fields in this form look like this:

```
class="TwoThird", class="Third Last"
class="Third", class="TwoThird Last"
class="Half", class="Half Last".
```

The word **Last** should be added in the end of the class name to remove the **padding-right** attribute.



MSS.Forms.css stylesheet also provides classes for the navigation area of a form, for instance, the upper and the lower dividers and the submit button. For example, the dividers and the submit button of the following form

Contact form

If you want to send us an e-mail, please fill out the contact form and press "Send".

First name * Surname *

Enter first name Surname

Phone E-mail

Phone E-mail

Message *

Message

Fields marked * are required

Send

are defined by the following code:

```
<div class="MssNavigation">
  <!-- Encapsulates the form buttons, is required -->
  <div class="navLine">
    </div>
  <!-- Divider line -->
  <div class="navButtons">
    <!-- Group buttons on one row and make them float to right -->
    <scx:button id="btnSend" tabindex="15" runat="server"
causesvalidation="True" validationgroup="Subscribe"></scx:button></div>
  <div class="navLine">
    </div>
  <!-- Divider line -->
</div>
```

The complete code for the Contact form can be found in the **mss.website.contactform.ascx** sublayout located at;

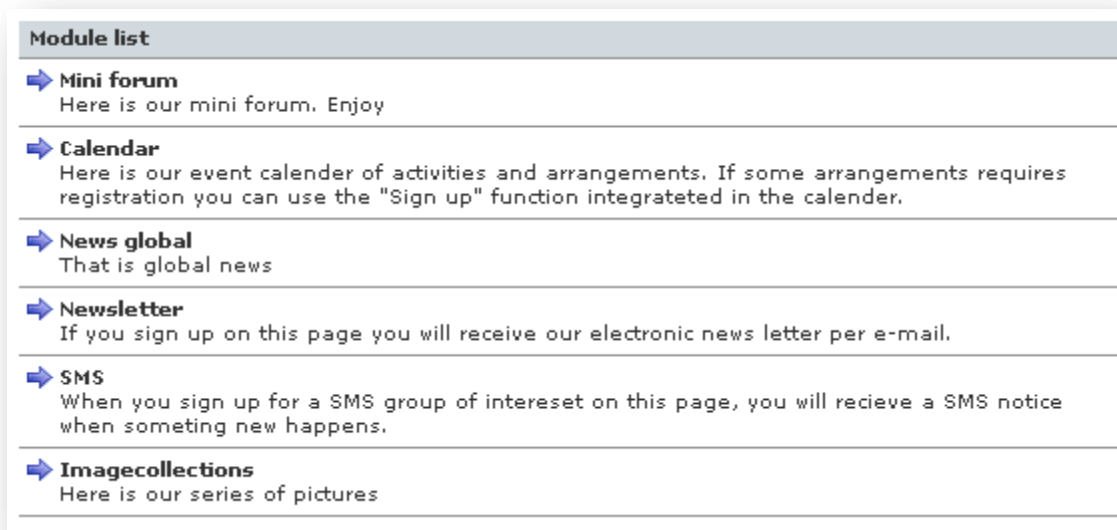
/sitecore modules/MSS/website/components/contactform/.

If any modifications to styling are required, they should be done in the following file:

/sites/_your_website_/css/MSS.Skin.css

and not in the default stylesheet **MSS.Forms.css**.

3.5 List style



A list like the one shown above is rendered using the following stylesheets:

- **/mss/WebEngine/css/MSS.Layout.css**
- **/mss/WebEngine/css/MSS.Lines.Arrows.css**

If any modifications to the styling are required, they should be done in the following file:

/sites/_your_website_/css/MSS.Skin.css

and not in the default system stylesheets **MSS.Layout.css** and **MSS.Lines.Arrows.css**.

The following code can be used to render a list like the one shown in this section:

```
<div class="sysList wizBottomLine">
  <div class="listHead">Module list</div>
  <ul>
    <xsl:for-each select="item">
      <li class="wizLine">
        <scx:link select="." class="wizListItemIcon" title="{@name}">
          <span class="sysTitle"><xsl:value-of select="@name"/></span>
          <xsl:value-of select="sc:clip(sc:striptags(mss:fld('text',
            .)), 256, 1)" disable-output-escaping="yes"/>
          </scx:link>
        </li>
      </xsl:for-each>
    </ul>
  </div>
```

3.6 XSL helper functions

Sitecore Foundry provides a set of XSL extension functions. The appropriate assembly is called **MSS.XslHelper**. This is how the assembly is defined in **web.config** file:

```
<extension mode="on" type="MSS.XslHelper.XslHelper, MSS.XslHelper"
  namespace="http://www.sitecore.net/mss" singleInstance="true"/>
```

Some of the extension functions are described in this section.

- XPathNodelterator GetHome()
Purpose: returns site's home page.
Use example: `<xsl:variable name="home" select="mss:GetHome()" />`
- XPathNodelterator GetGlobalMenuRoot()
Purpose: returns the root of the site's global menu.
Use example: `<xsl:value-of select="mss:GetGlobalMenuRoot()"/>`
- XPathNodelterator GetGlobalDocument(string docName)
Purpose: returns a document under the Global item of the site.
Use example: `<xsl:variable name="calendaritem" select="mss:GetGlobalDocument('Event calendar')"/>`
- bool IsLocalDocument(XPathNodelterator ni)
Purpose: is true if the current document is local for the current site.
Use example:

```

      <xsl:if test="mss:IsLocalDocument(.)">
        <sc:dot/>
      </xsl:if>
    
```
- string Path(XPathNodelterator ni)
Purpose: returns user friendly URL of a page. If the UseFriendlyURL setting in mss.config is enabled, the function returns an address without the ".aspx" extension.
Use example: `Read more`
- string Replace(string str, string oldValue, string newValue)
Purpose: replaces all occurrences of oldValue in str with newValue.
Use example:

```

      replace all "%20" strings with the space characters (" ")
      <xsl:value-of select="mss:Replace($path, '%20', ' ')/>
    
```
- bool IsWebEdit()
Purpose: is true if web edit mode is enabled.
Use example:

```

      <xsl:if test="mss:IsWebEdit()">
        Web Edit Mode.
      </xsl:if>
    
```
- bool IsLoggedIn()
Purpose: is true if the user is logged in.
Use example:

```

      <xsl:if test="mss:IsLoggedIn()">
        <a href="/admin">Change settings</a>
      </xsl:if>
    
```
- void LoggedIn()
Purpose: checks if the user is logged in. If the user is not logged in, redirects to the login page.

Use example: `<xsl:value-of select="mss:LoggenIn()"/>`

- XPathNodeIterator GetItem(string id)
Purpose: returns a item from the current database by id.
Use example: `<xsl:variable name="eventItem" select="mss:GetItem(sc:qs('eventID'))" />`
- string fld(string fieldName, XPathNodeIterator item)
 string fld(string fieldName, XPathNodeIterator item, string subFieldName)
Purpose: return a field value. If the value is empty in the current language, return the value of the item in the site's default language.
Use example: `<xsl:value-of select="mss:fld('file', ., 'mediaid')"/>`
- string GetFileExtension(string fileName)
Purpose: returns file extension.
- string SiteSettings(string section, string key)
 string SiteSettingsLowerCase(string section, string key)
Purpose: return site setting by the key and the section.
Use example: `<xsl:param name="templates" select="mss:SiteSettingsLowerCase('templates', 'documents')" />`
- string SiteData(string key)
Purpose: returns wizard generated settings.
Use example: `<xsl:value-of select="mss:SiteData('Name on top')"/>`
- string SiteModulesSetting(string section, string key)
Purpose: returns the module setting in the lower case.
- string SiteModulesSetting(string key)
Purpose: returns the module setting in the lower case. The key is searched for in all sections.
Use example: `<xsl:param name="templates" select="mss:SiteModulesSetting('Calendar', 'EventTemplate')"/>`

3.7 Tutorials

3.7.1 Creating a top menu rendering

This tutorial shows how to add a simple top menu rendering like the one shown in the following image to the Sitecore Foundry demo site.



1. Start **Developer Center (Sitecore » Developer Center)**.
2. Create a new XSLT rendering. For instance, the **"MSS.MainMenu"** rendering with the following code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sc="http://www.sitecore.net/sc"
  xmlns:dot="http://www.sitecore.net/dot"
  xmlns:mss="http://www.sitecore.net/mss"
  xmlns:dic="http://www.sitecore.net/dic"
  exclude-result-prefixes="dot sc dic mss">

  <!-- output directives -->
  <xsl:output method="html" indent="no" encoding="UTF-8" />

  <!-- parameters -->
  <xsl:param name="lang" select="'en'"/>
  <xsl:param name="id" select="''"/>
  <xsl:param name="sc_item"/>
  <xsl:param name="sc_currentitem"/>

  <!-- variables -->
  <xsl:variable name="home" select="mss:GetHome()" />
  <xsl:param name="templates"
  select="mss:SiteSettingsLowerCase('templates', 'documents')" />

  <!-- entry point -->
  <xsl:template match="*"
    <xsl:apply-templates select="$home" mode="main"/>
  </xsl:template>

  <!--=====
  <!-- main P -->
  <!--=====
  <xsl:template match="*" mode="main">

    <style>
      span.menuItem { border-right: 1px solid yellow; padding-right: 5px;
padding-left: 5px;}
      span.menuItem a { color: white; text-decoration:none;}
    </style>

    <div class="areaBreadcrumb">

      <xsl:for-each select="item[contains($templates,@template) and
mss:IsVisible(.)]">
        <xsl:sort select="@sortorder"/>
        <xsl:sort select="sc:fld('title',.)"/>

        <span class="menuItem">
          <xsl:variable name="MenuTitle" select="sc:fld('menutitle',.)"/>
          <xsl:variable name="Title" select="sc:fld('title',.)"/>

          <sc:link>
            <xsl:choose>
              <xsl:when test="$MenuTitle != ''">
                <xsl:value-of select="$MenuTitle"/>
              </xsl:when>
              <xsl:when test="$Title != ''">
                <xsl:value-of select="$Title"/>
              </xsl:when>
            </xsl:choose>
          </sc:link>
        </span>
      </xsl:for-each>
    </div>
  </xsl:template>
  </xsl:stylesheet>
```



```

    <xsl:otherwise>
      <xsl:value-of select="@name" />
    </xsl:otherwise>
  </xsl:choose>
</sc:link>
</span>

</xsl:for-each>

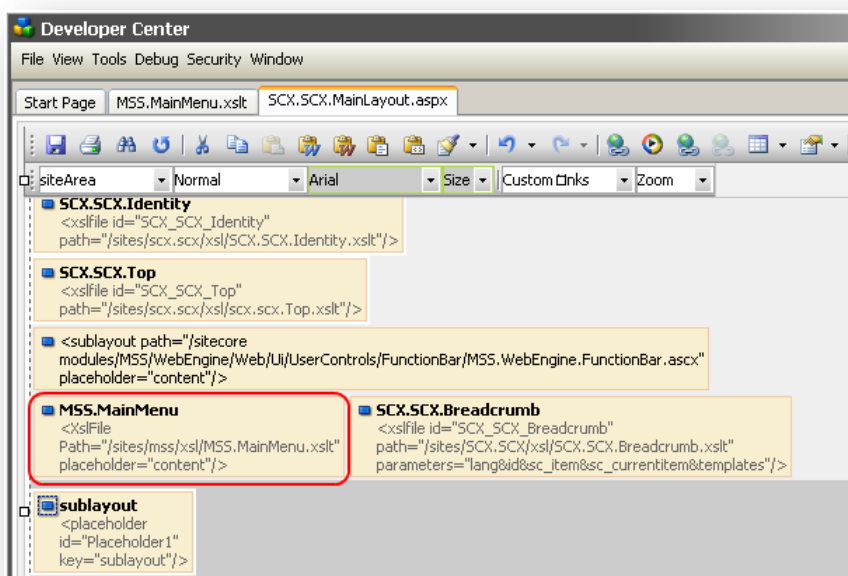
</div>

</xsl:template>

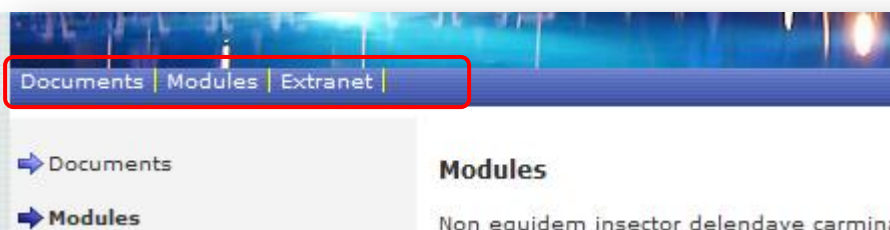
</xsl:stylesheet>

```

3. Add the rendering to the main layout.



4. Refresh the front page of the site and you will see a menu like this:



3.8 Using dictionary lookup in XSL controls

The following code snippet lists the public methods of the Dictionary class **MSS.Dictionaries namespace**. The **key** parameter is the name of the dictionary item to look for and the **path** is the relative path under the dictionary's root.

```

namespace MSS.Dictionaries
{

```

```

public class Dictionary
{
    ...
    public string GetText(string key)
    public string GetText(string key, string language)
    public string GetText(string key, string language, string path)
    public string GetHint(string key)
    public string GetHint(string key, string language)
    public string GetHint(string key, string language, string path)
    public string GetAccessKey(string key)
    public string GetAccessKey(string key, string language)
    public string GetTabIndex(string key)
    public string GetTabIndex(string key, string language)
    ...
    ...
}
}

```

The following line should be added in the xslt stylesheet declaration to make the dictionary functionality available.

```
xmlns:dic="http://www.sitecore.net/dic"
```

The following lines should be present in the **web.config** file for the dictionary functionality to work.

```

<control mode="on" tag="scx:label" type="MSS.Web.UI.XslControls.Label"
assembly="MSS.WebEngine"></control>

<extension mode="on" type="MSS.XslHelper.XslDictionaryHelper, MSS.XslHelper"
namespace="http://www.sitecore.net/dic" singleInstance="true"/>

```

Dictionary-aware controls are described in the following section. The SiteType and SiteContext classes also provide access to the dictionary. For more information see “4.4 The Runtime Engine”, on page 54.

3.8.1 Built-in dictionary controls

The dictionary related custom controls are derived from the common .NET controls which belong to the **System.Web.UI.WebControls** namespace; the names of the controls are the same as in the .NET namespace. The difference is that the Sitecore Foundry controls use the Foundry dictionary to look for the text values for the attributes **text**, **alt** (hint), Access key (shortcut), tab index and initial input values where available.

As for the Validator classes (for instance, RegularExpressionValidator, RequiredFieldValidator), the error text is supplied by the Foundry dictionary. The RequiredFieldValidator also checks that the initial text of an input control has been altered before displaying the error message.

The following dictionary related custom controls are provided.

Name	Description
Label	Sets the accesskey, tabindex, text and alt attribute (hint), retrieved from the dictionary, on the label control
TextBox	Sets the accesskey, tabindex, initial text (value) and alt attribute (hint), retrieved from the dictionary, on the text control



Name	Description
Button	Sets the accesskey, tabindex, text (value) and alt attribute (hint), retrieved from the dictionary, on the button control
Checkbox	Sets the accesskey, tabindex and alt attribute (hint), retrieved from the dictionary, on the checkbox control
Imagebutton	Sets the accesskey, tabindex and alt attribute (hint), retrieved from the dictionary, on the imagebutton control
Form	Form control is derived from the System.Web.UI.WebControls.Panel control and is used to encapsulate other dictionary custom controls. The Form control provides the path, below the dictionary's root, to look for the dictionary item. Can be overwritten on the child controls by providing their own path.
Link	Sets the accesskey, tabindex, text and alt attribute (hint), retrieved from the dictionary, on the link control
Radiobutton	Sets the accesskey, tabindex, alt attribute (hint), retrieved from the dictionary, on the radiobutton control
RegularExpressionValidator	Sets the accesskey, tabindex, errortext and alt attribute (hint), retrieved from the dictionary, on the control
RequiredFieldValidator	Sets the accesskey, tabindex, errortext and alt attribute (hint), retrieved from the dictionary, on the control. Takes the dictionary text control's initial value into account.

Common read/write properties on the controls are:

Name	Description
Formname	Name of the path, below the dictionary's root, to look for the dictionary item. All dictionary items are located in subfolders beneath the dictionary. If no formname is given the control search for a parent control of the type Form (custom control).



Name	Description
DictionaryItem	If dictionaryitem is not set, the name of the control (ID) is used as key to lookup the dictionary item in the selected path given by formname (see above). Dictionaryitem makes it possible to have an ID and look for a dictionary item with another name.
Language	Language to use when extracting the values from the dictionary item.
Text	Value of Text field on dictionary item
Alt	Value of Hint field on dictionary item for use on alt attributes as hints
Accesskey	Value of Accesskey field on dictionary item. Used for shortcut key (Accesskey attribute)
TabIndex	Value of Tabindex field on dictionary item. Used for setting tab order.

To enable the controls on a layout, add the following .NET page directive at the top of the layout:

```
<%@ Register TagPrefix="scx" Namespace="MSS.Web.UI.WebControls"
Assembly="MSS.WebEngine" %>
```

For an example, take a look at the code snippet taken from a form rendering (**scx:Label** and **scx:TextBox** are used):

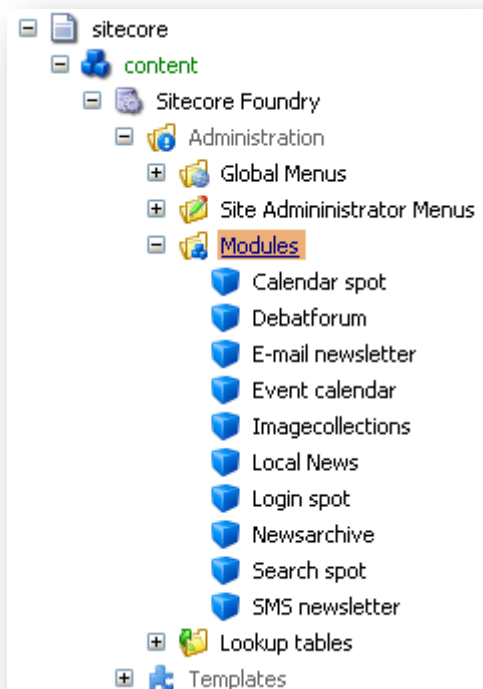
```
<div class="fldGroup">
  <div class="field Full">
    <scx:Label id="lblName" runat="server"></scx:Label><br>
    <scx:TextBox id="txtName" tabIndex="1" runat="server" CssClass="inpFull"
      Defaultbutton="btnSubscribe"></scx:TextBox>
  </div>
</div>
```

3.9 Creating a new module

This section describes how to create your own module.

All modules are stored at;

/sitecore/content/mss content/Administration/Modules.



Let's create a module which will show the number of the running sites on the server.

1. Create a new sublayout and call it **SiteCount**.

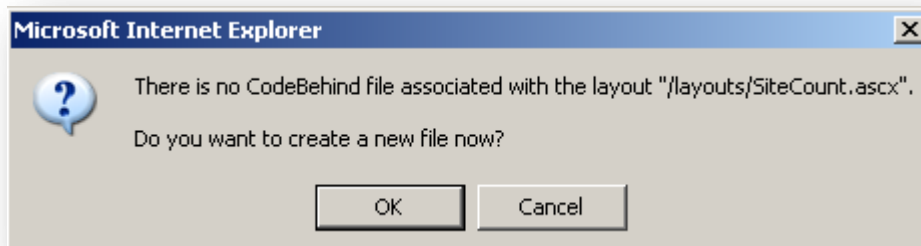
2. Enter the Sublayout code. Switch to the HTML mode by clicking HTML button.



Enter control markup

```
<%@ Control Language="c#" AutoEventWireup="true"
Inherits="Layouts.Sitecount.SitecountSublayout"
Src="/layouts/SiteCount.ascx.cs" %>
<div style="border:solid 1px gray;">
  <div style="font-weight:bold;background-color:#f0f0f0;">Server
  info</div>
  <asp:label id="txtInfoMessage" runat="server"></asp:label>
</div>
<br/>
```

3. Create the code behind file. Switch to Edit code mode; the dialog which asks you about creating a code behind file will appear (if the code behind file already exists, open it).



Change code to the following:

```
using System;
using System.Web.UI.WebControls;
using MSS.Sites.Contexts;

namespace Layouts.Sitecount {

    public partial class SitecountSublayout : System.Web.UI.UserControl
```

```

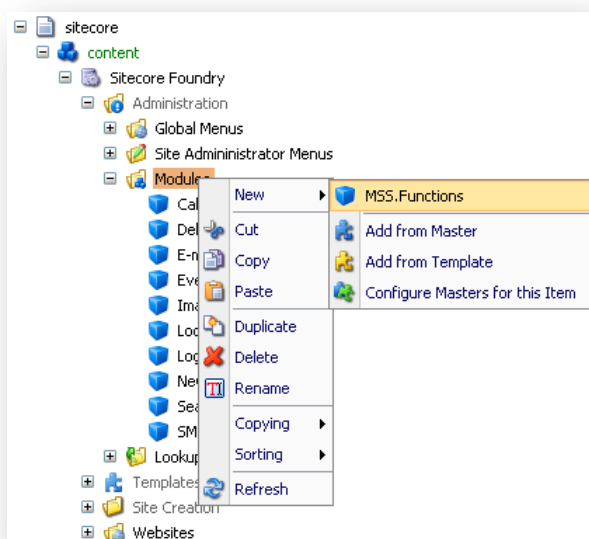
{
    protected Label txtInfoMessage;
    protected override void OnLoad(EventArgs e) {
        if(!IsPostBack)
        {
            int count = 0;
            foreach(SiteContext site in
MSS.Context.Server.Sites.GetSites(MSS.Context.CurrentDatabase))
            {
                if(site.Status == SiteContext.SiteStatus.Running)
                {
                    ++count;
                }
            }

            txtInfoMessage.Text = string.Format("The server has {0}
running site(1).", count, (count > 1) ? "s" : string.Empty);
        }
    }
}
}

```

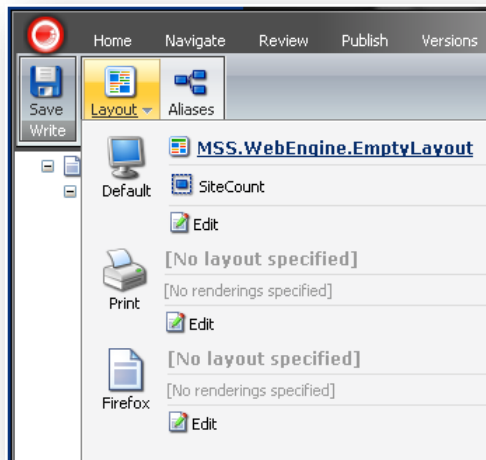
Save changes.

4. Add a new module to the module list. The new module item should be based on the **MSS.Functions** template (or created from the **MSS.Functions** master).

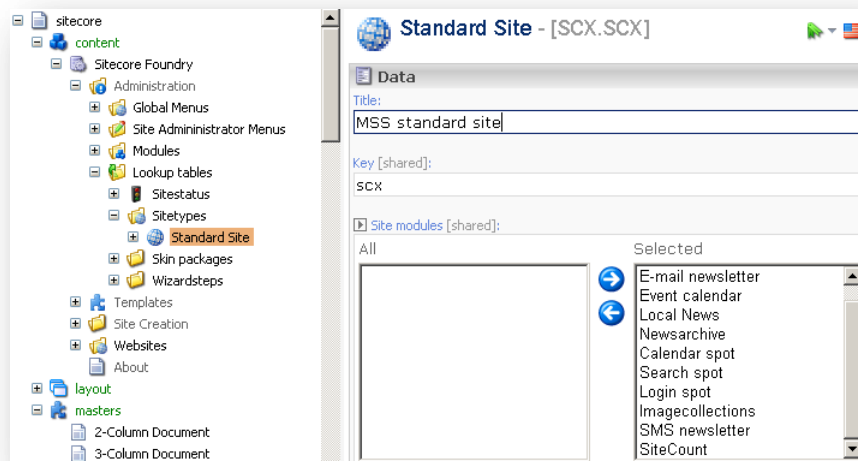


Give the SiteCount name to the new module, fill the Title and Text fields and add the SiteCount sublayout to the module. Sitecore CMS does not allow to have a sublayout without a layout on an item. So you should specify a layout for the

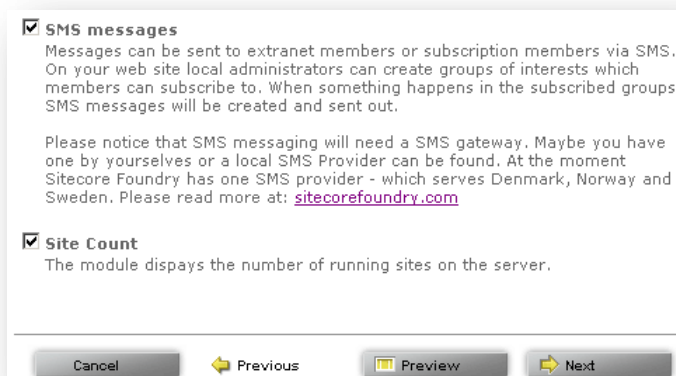
module. You can use the **MSS.WebEngine.EmptyLayout** for this purpose.



5. Enable the module on a site type. Move the SiteCount module to the Selected list in the Site modules field of a site type.

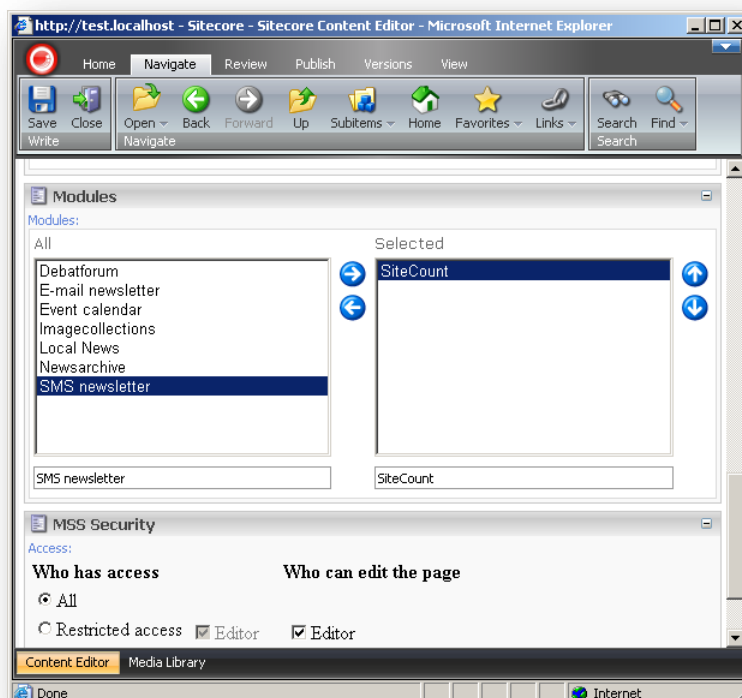


6. Enable the module on the site.
Open the site's front-end, login to the site as a local admin, start the Site Wizard, navigate to the Modules page, find the SiteCount module in the list and enable it (check the checkbox).



Go to the Finish page and click **Save**. Now the module can be used on the site.

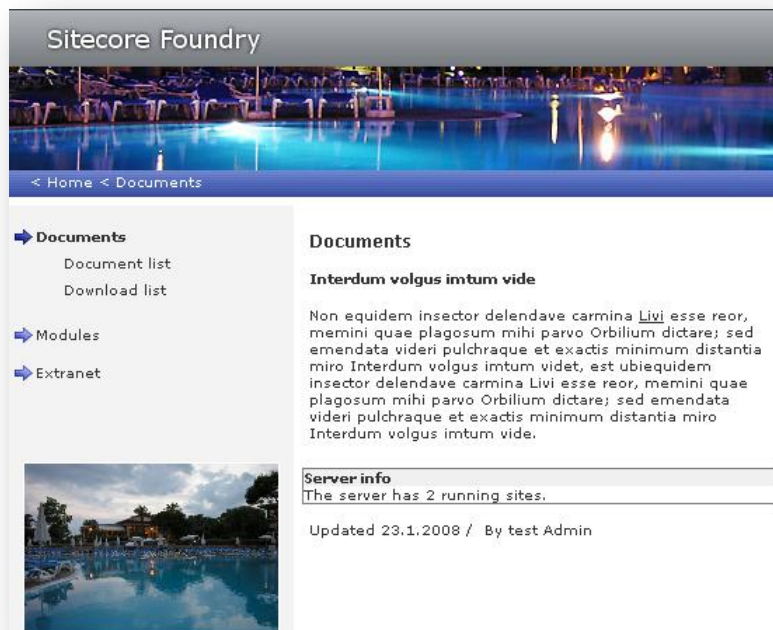
7. Add the module to a page.
Enter edit mode (by clicking on the **Edit Content** menu item from the Administration menu). Go to the page where you want to add the module. Click the green content marker to edit the document. Add the Site Count module to the page by moving the Site Count module to the selected group in the Modules field in the navigate tab.



Click **Save** and close the window.



8. The module, which displays the number of the running sites is present on the page now.



3.10 Adding a new Site Wizard step

This section describes how to add a new Site Wizard step which can be used to enter the meta tag information.

1. Create a sublayout called **MMS.ChooseMetatags** which will be used to enter meta tag information. Enter the following sublayout code:

```
<%@ Control Language="c#" AutoEventWireup="true"
Inherits="Layouts.Wizard.ChooseMetatags"
Src="/layouts/MMS.ChooseMetatags.ascx.cs" %>
<div class="fldGroup">
  <div class="field Half">
    Keywords
    <br />
    <asp:TextBox ID="Keywords" runat="server" Rows="2"
    TextMode="MultiLine" class="inputshort"/>
  </div>
  <div class="field Half Last">
    Description<br />
    <asp:TextBox ID="Description" runat="server" Rows="2"
    TextMode="MultiLine" class="inputshort"/>
  </div>
</div>
```

2. Create the code behind file with the following code:

```
using System;
using System.Web.UI.WebControls;
using MSS.Wizard.UserControls;

namespace Layouts.Wizard
{
  public partial class ChooseMetatags : WizardControl
```

```

{
    const string sectionName = "Metatags";

    protected TextBox Keywords;
    protected TextBox Description;

    protected override void OnLoad(EventArgs e)
    {
        // Subscribe to save event
        WPage.SaveData += new EventHandler(Page_SaveData);

        if (!IsPostBack)
        {
            Keywords.Text = Settings.Setting(sectionName, "Keywords");
            Description.Text = Settings.Setting(sectionName,
"Description");
        }
    }

    void Page_SaveData(object sender, EventArgs e)
    {
        Settings.SetSetting(sectionName, "Keywords", Keywords.Text);
        Settings.SetSetting(sectionName, "Description",
Description.Text);
    }
}

```

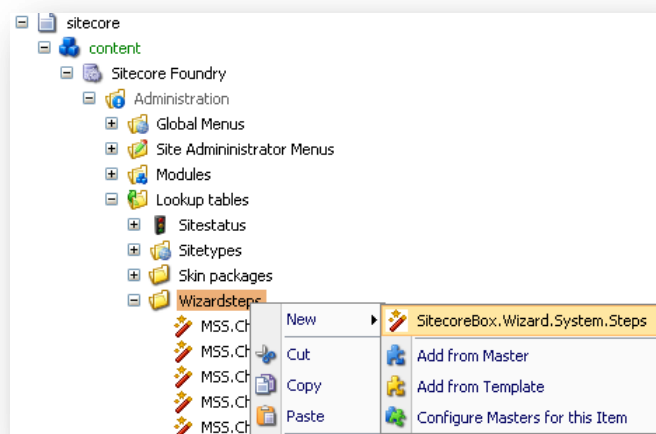
3. Add a wizard step.

All wizard steps are located at;

/sitecore/content/mss content/Administration/Lookup tables/Wizardsteps

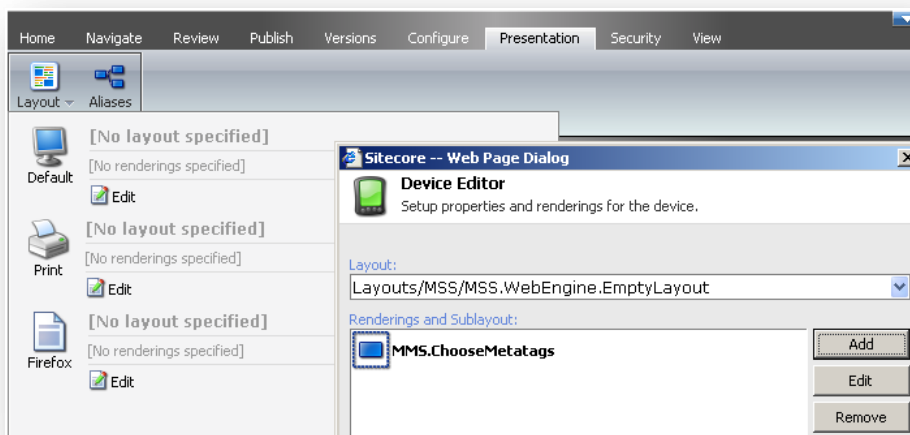
For more information about the wizard steps, refer to 5.8 Customizing the wizard”, on page 82.

Add the **MSS.Metatags** step.

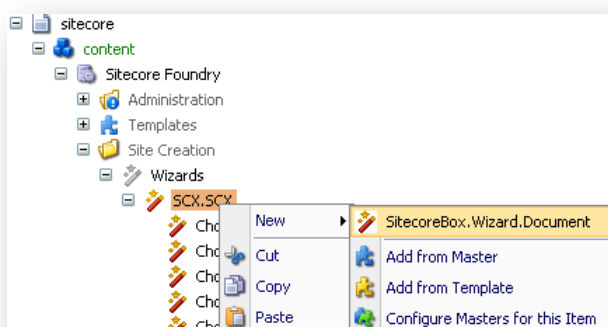


4. Assign the created sublayout to the **MSS.Metatags** step item. Sitecore CMS does not allow to have a sublayout without a layout on an item. So you should specify a layout for the module. You can use the **MSS.WebEngine.EmptyLayout**

for this purpose.



- Add the step to the wizard.
Select a wizard and add a page with the **ConfigureMetaTags** name.



Fill the item fields as shown below.

Title - Configure meta tags.

Menutitle – Metatags

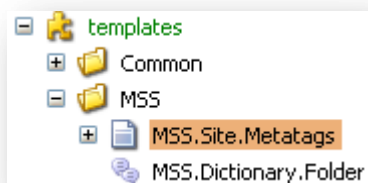
Step – select the step created in this tutorial.

Text – enter the description to this page.

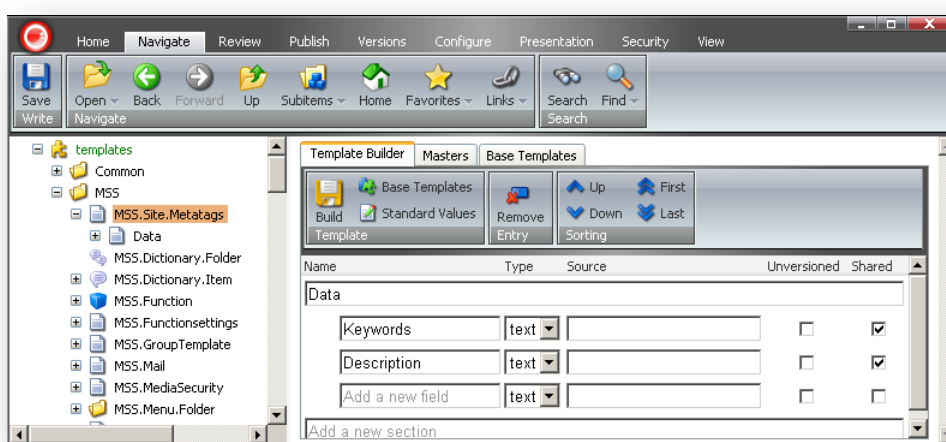
PreviewPostback – enable this checkbox. If you enable this checkbox, the HTML form will be submitted when we click preview or select a menu item.

5. Add an item which will store the new information.
In order to save the new information you should add an item to site settings which will contain the meta data information (keywords and description). For this purpose we will create a new template with two fields: Keywords and Description. Then we will add an item based on this template to the Site Wizard settings.

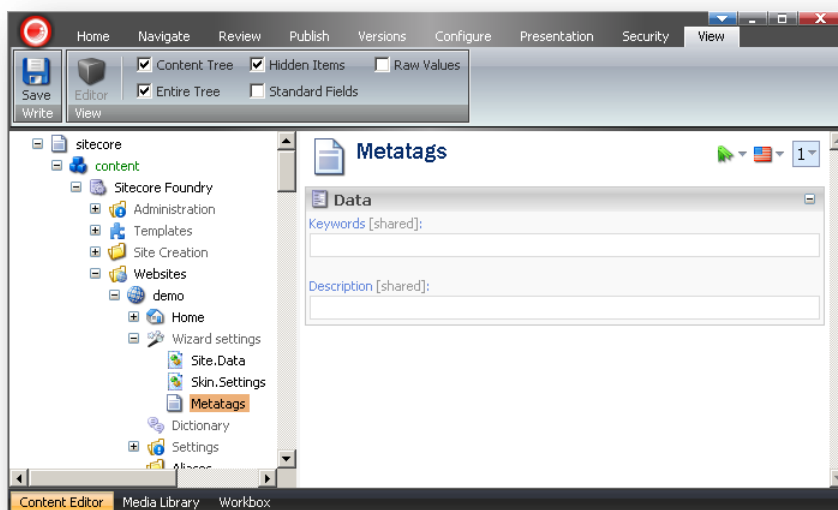
Using the Template Manger create the **MSS.Site.Metatags** template under the **/templates/mss/** folder.



Add two text fields to the template called Keywords and Description respectively.

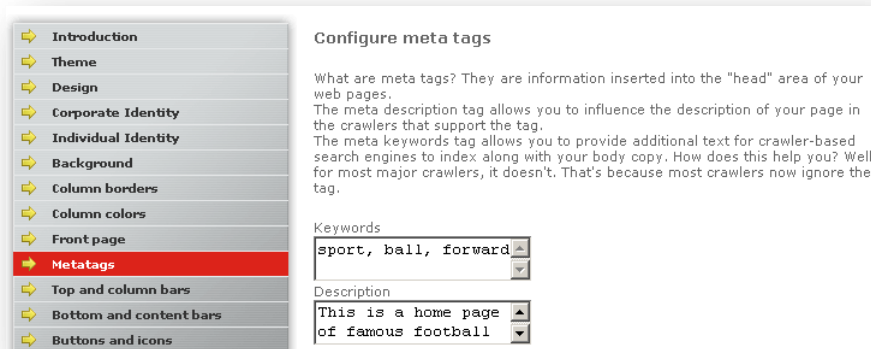


Add an item called Metatags based on this template to the site wizard settings.





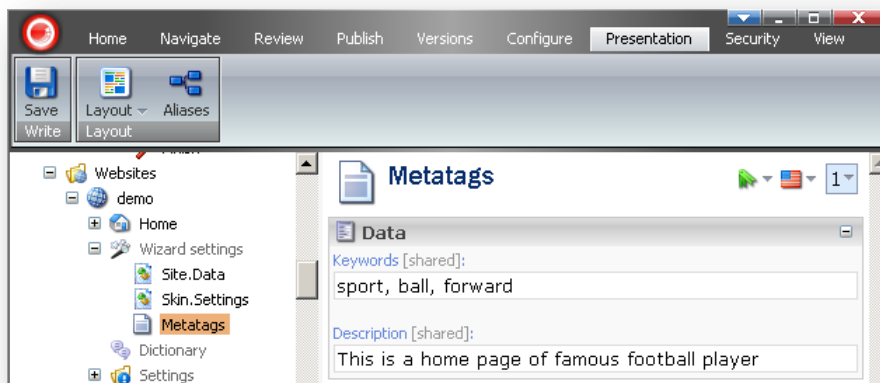
Open the front-end, go to the Site Wizard, and to the Metatags page. Fill the appropriate



fields.

Go to the **Finish** wizard page and click **Save**.

The keyword and description information will be saved in the site settings item.



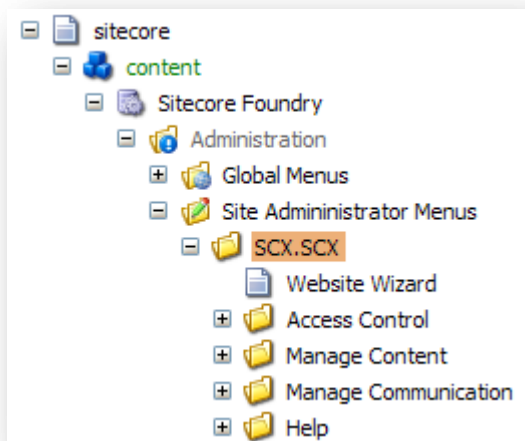
Use the following code to access this information:

```
string keywords = MSS.Context.Site.SiteData.Setting("Metatags", "Keywords");  
string desc = MSS.Context.Site.SiteData.Setting("Metatags", "Description");
```

3.11 Modifying the local administration menu

The local administration menu items are stored under the following item:

/sitecore/content/Sitecore Foundry/Administration/Site Administrator Menus



The “Site Administrator Menus” folder contains subfolders which contain the actual administration menus. A root item (for instance, the SCX.SCX item) has two masters assigned: MSS.Menu.Folder and MSS.Menu.Item. MSS.Menu.Folder is used to create folders in the administration menu and MSS.Menu.Item is used to create the menu items themselves.

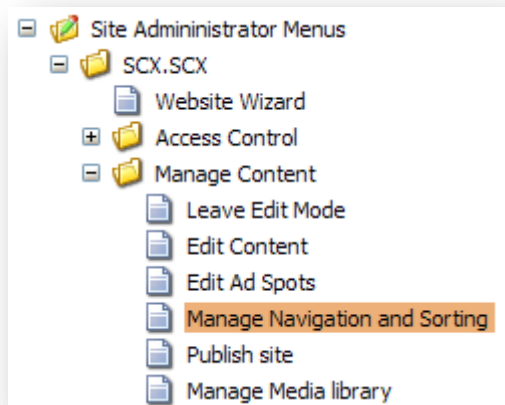
The MSS.Menu.Item template contains the following fields:

- **Image**
The image which is displayed in the content area of the site layout when the menu link is selected.
- **Imagelink**
The page where a user will be redirected when one clicks on the image.
- **Title**
The title which is displayed in the content area of the site layout below the image when the menu link is selected.
- **Abstract**
The text which is displayed in the content area of the site layout below the title when the menu link is selected.
- **Text**
The text which is displayed in the content area of the site layout below the abstract when the menu link is selected.
- **Menu title**
The title which is shown in the menu
- **Menulink**
The link which is opened when a user selects the menu item. The layout settings of the menu item are ignored when this field is not empty. If this field is empty, the layout assigned to the menu item will be rendered.
- **Webedit mode**
The mode in which the given menu item will be visible. The valid values are:
 - **Empty** (no value) – default value. The item is visible in all modes.

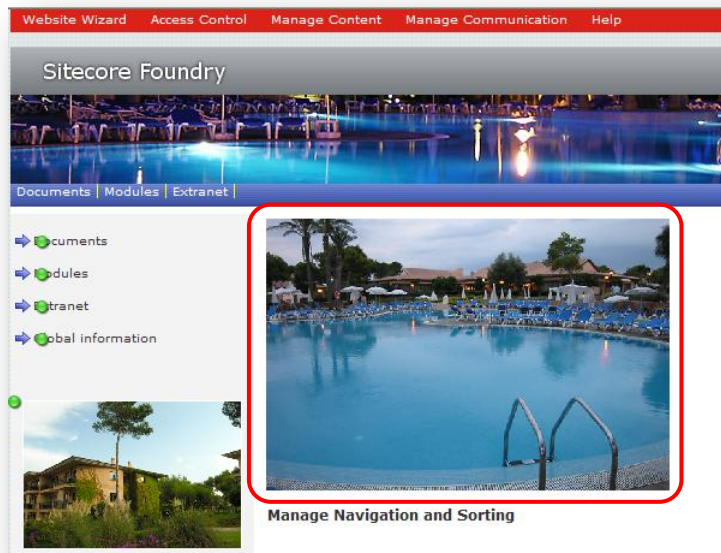
- **None** – the item is visible in all modes except Webedit.
- **Preview** – the item is visible only in the Preview mode.
- **Webedit** – the item is visible only in Webedit mode.
- **Required modules**
The menu item will only be visible when one of the modules listed in this field is active. If no module is selected, the field is ignored.
- **MssSecurityRight**
The security role which has access to this item.

For example, consider the menu item with the following field values entered:

- **Item name** – Manage Navigation and Sorting.



- **Image** - /Downloaded picture series/Default pictures/Image01

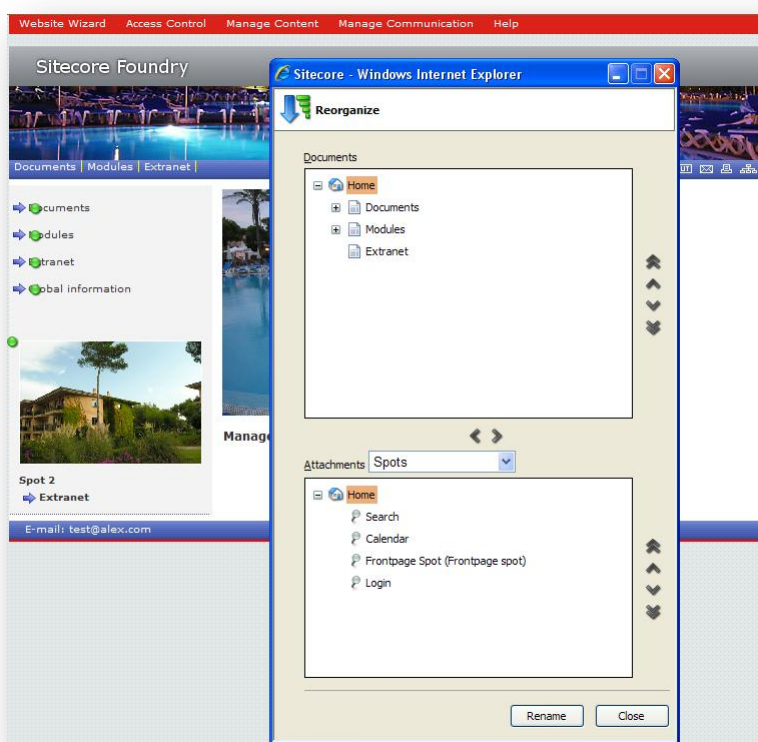


- **Imagelink** – empty
- **Title** - Manage Navigation and Sorting
- **Abstract** – empty
- **Text** – empty

- **Menu title - Manage Navigation**



- **Menulink – empty (the MSS.WebEngine.Reorganize layout is rendered)**



- **Webedit mode - Webedit**
- **Required modules - empty**

MssSecurityRight - ContentEditing

Chapter 4

Reference

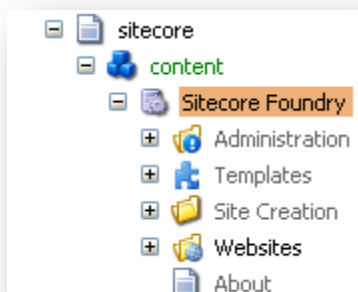
This chapter deals with various references within the Foundry and their structure and function.

4.1 Sitecore Foundry content structure

In this section we show the key paths within the Foundry that are critical to its functionality.

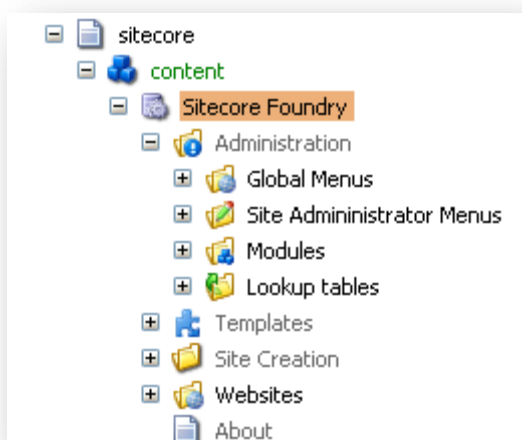
Note: The following paths within Sitecore Foundry are key to the product and should not be changed.

4.1.1 Foundry Content Items



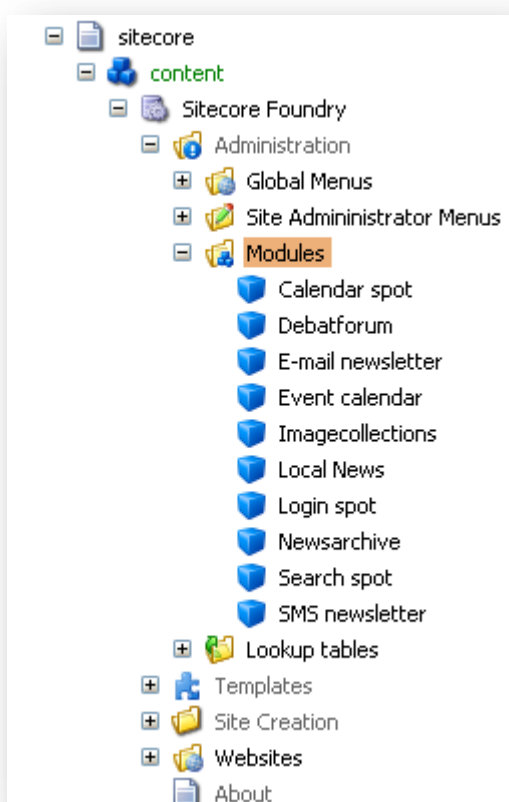
- /sitecore/content/mss content
All Sitecore Foundry related content is stored here, except for the system specific content.
- /sitecore/content/mss content/administration
Central administration information.
- /sitecore/content/mss content/websites
All websites are stored here when duplicated from a site template.

4.1.2 Foundry Administration Items



- /mss content/Administration/Global menus
Global menus that can be appended to a site's normal content are stored here.
- /mss content/Administration/Site Administrator Menus
Local administrator menus are stored here.

4.1.3 Sitecore Foundry Modules Items



- /mss content/Administration/Modules

All Sitecore Foundry modules are stored here. Each module contains the following fields:

- Title – defines the module title;
- Text – defines the short description of the module. You can see this description in the Site Wizard.
- Is spot – defines whether the module is a spot. If the module is a spot it will be rendered in a left or right column. If not, the module will be rendered in the central column of the site.
- Type – defines which class this module implements. The class should implement the IModule interface. If the field is empty, the MSS.Modules.Module type is used. The interface IModule is shown below.

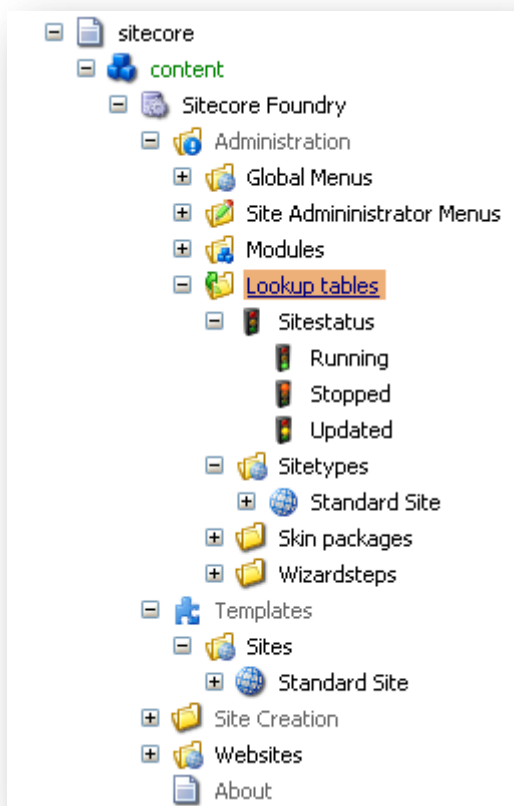
```
public interface IModule
{
    string Title { get; }
    string Description { get; }
    string LicenseKey { get; }
    bool IsSpot { get; }
    ID ID { get; }
    Item Item { get; }

    Control[] GetRenderings(Page page);

    void CreateSite(SiteContext site);
    void DeleteSite(SiteContext site);

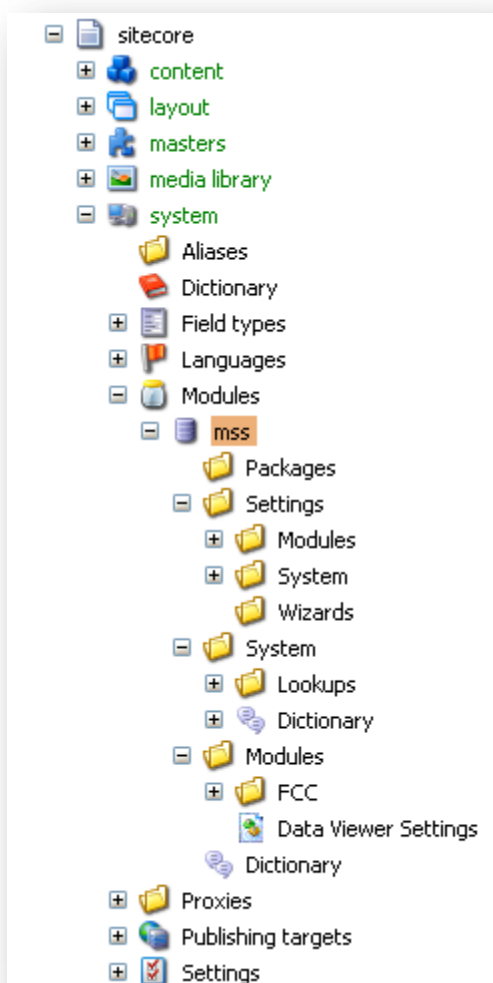
    XmlNode Backup(SiteContext site);
    void Restore(SiteContext site, XmlNode moduleData);
}
```

4.1.4 Sitecore Lookup tables items



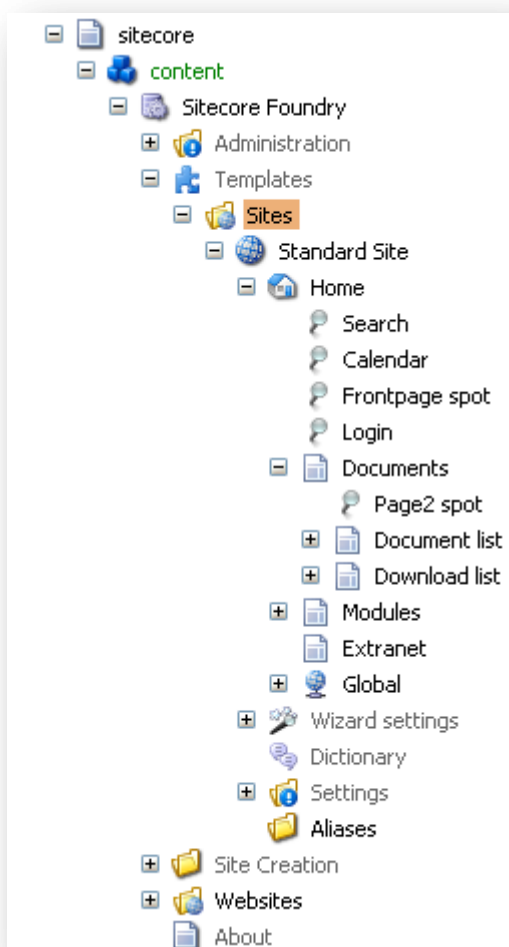
- /mss content/Administration/Lookup tables/Sitestatus
Lookup tables for system use. Site type information is stored here at “**Sitetypes**”.
- /mss content/Administration/Lookup tables/Sitetypes
All Site types are stored here.
- /mss content/Templates/Sites
All site templates are stored here.

4.1.6 Modules MSS items



- /sitecore/system/modules/mss
Sitecore Foundry system related information is held here.
- /mss content/Templates/Sites
Site templates used to duplicate when a new site is created. Which site template is used depends on the site type.

4.1.7 Template Sites items



- /mss content/Templates/Sites
Site templates used to duplicate when a new site is created. Which site template is used depends on the site type.

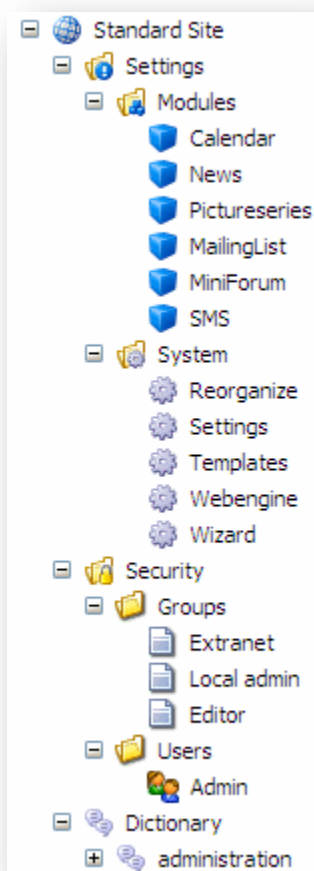
4.2 Site Type Components

A site type is given a unique name that is used to prefix the names of all the components the site type consists of. This is to help the developer differentiate between all the system components and the individual site type components.

Information about all Site types is stored at:

/sitecore/content/mss content/Administration/Lookup tables/Sitetypes

Before creating a new layout and design for a new site type for the first time, take a look at the layout of the demo website accompanying the product.



Take note of how the **Wizard** settings and **/system** components are assembled to form the website.

If the standard website functionality with sitemap, contact form, search and module functionality is required, the structure of the content on the following path in a local site must be preserved.

/sitecore/content/mss/Templates/Sites/sitetypeName/Global/

The standard Sitecore Foundry website consists of the components listed in the following tables. The HTML contained in the renderings and layouts can be modified as much as needed, and so can the templates and masters to suit the needs of the site type. However, care must be taken when altering the renderings, layouts, templates and masters as it can affect the way the modules are displayed.

4.3 Site Type Fields

The site type contains the following fields:

Field	Purpose	Related item path
Title	The name of the site type in different languages	-
Key	For internal use	-



Field	Purpose	Related item path
Site modules	Selects which modules are available for the sites based on this site type.	/sitecore/content/mss content/Administration/Modules
Site globalmenu	Selects the global menu items to append to the site's menu for all sites of this site type.	/sitecore/content/mss content/Administration/Global Menu
Site Administrator Menu	Selects the back-end administration menu to use. Some of the items on the menu (e.g. "reorganize" and "help/manual") rely on the normal standard Sitecore layout to render content into the running website. In future versions this content will be put into popup dialogs.	/sitecore/content/mss content/Administration/Site Administrator Menu
Site template	Selects the site content to be duplicated and used when new sites are created. The use of the site template and Administrator menu are connected through the use of the masters, templates, renderings and layouts on these items.	/sitecore/content/mss content/Templates/Sites
Site Wizard	Selects the wizard to use	/sitecore/content/mss content/Site Creation/Wizards

4.3.1 Templates

The following table shows the templates used in a site type. Additional templates can be added to a site type, but remember to register them on the site type settings in order for them to be displayed in the sitemaps and menus. Sitecore stores layouts and renderings on templates and masters. Therefore, each **site** has its own templates and masters for items that are rendered by the Sitecore layout engine.

Name	Purpose
xxx.DocRef	Document appendix. Text documents that can be attached to a standard document as an appendix. It is not shown in the menu but listed on the main document. Fields: Image, Imagelink, title abstract and text.
xxx.Document	Standard document template containing fields: Image, Imagelink, title abstract, text, menu title, menulink, modules (Multilist showing which modules are attached to the document), Hide update information ("HideUpdateInfo" which hides the document footer that says who updated the document last) and the number of random spots shown to the right.
xxx.Event	Calendar event item used when creating new events on the calendar

Name	Purpose
xxx.FileRef	File appendix used to append files to a document.
xxx.News	News article item used when creating news articles using the news module
xxx.PictureSeries.Picture	Picture series picture template used when creating picture series
xxx.PictureSeries	Picture series template used when creating picture series
xxx.Spot	Spot template used for adding spots to a document.

4.3.2 Masters

The following table shows the masters that make up a site type. Additional masters can be added to a site type.

Name	Purpose
xxx.DocRef	Document appendix. Text documents that can be attached to a standard document as an appendix. It is not shown in the menu but listed on the main document. Fields: Image, Imagelink, title abstract and text.
xxx.Document.SingleColumn	Document master using layout settings to display only one column that are stretched and no spots/highlights. Uses the same xxx.document template as master xxx.TwoColumns
xxx.Document.TwoColumns	Document master using the template's layout settings to display the default two column layout.
xxx.Event	Calendar event item used when creating new events on the calendar
xxx.FileRef	File appendix used to append files to a document.
xxx.News	News article master used when creating news articles using the news module
xxx.PictureSeries.Picture	Picture series picture master used when creating picture series
xxx.PictureSeries	Picture series master used when creating picture series
xxx.Spot	Spot master used for adding spots to a document.

4.3.3 Renderings

The following table shows the renderings that make up a site type. Additional renderings can be added to a site type

Name	Purpose	File
xxx.bottombar	Displays the contact information at the bottom of the page.	"/sites/xxx/xsl/xxx.bottombar.xslt"



Name	Purpose	File
xxx.Breadcrumb	Used to display the breadcrumb path into the site content	"/sites/xxx/xsl/xxx.breadcrumb.xslt"
xxx.Document	Used to display the content of a document with both single and double columns.	"/sites/xxx/xsl/xxx.document.xslt"
xxx.DocumentFooter	Used to display the name and time of the last the person that edited the current document.	"/sites/xxx/xsl/xxx.Documentfooter.xslt"
xxx.DocumentList	Displays the document and file appendices. This rendering is not used by default. Instead a generic system document list rendering is used. If not satisfactory use this rendering instead and modify it.	"/sites/xxx/xsl/xxx.DocumentList.xslt"
xxx.Frontpage	Displays the front page center column, which is often different from the normal document display.	"/sites/xxx/xsl/xxx.Frontpage.xslt"
xxx.Highlight	Displays the spots in the right column on two column pages.	"/sites/xxx/xsl/xxx.Frontpage.xslt"
xxx.Leftmenu	Displays the menu in the left side.	"/sites/xxx/xsl/xxx.Leftmenu.xslt"
xxx.SearchResults	Displays the search results. This rendering is not used by default. Instead a generic system search results rendering is used. If not satisfactory use this rendering instead and modify it.	"/sites/xxx/xsl/xxx.Searchresults.xslt"

Name	Purpose	File
xxx.Sitemap	Displays the sitemap. This rendering is not used by default. Instead a generic system sitemap rendering is used. If not satisfactory use this rendering instead and modify it.	"/sites/xxx/xsl/xxx.sitemap.xslt"
xxx.Top	Displays the top bar containing the wizard generated image and name for the site.	"/sites/xxx/xsl/xxx.Top.xslt"

As stated in the above table renderings **DocumentFooter**, **SearchResults** and **Sitemap** are not used by default. Instead generic system renderings are used. The three renderings are included in case the default renderings are unsatisfactory. You can then use these renderings instead, modify them and set them on the respective items under global on the document template and masters.

4.3.4 Layouts

The following table shows the layout of a site type. Additional layouts can be added to a site type. When creating new layouts be sure to inherit the page from the "**MSS.Web.UI.Pages.SitePage**" class.

Name	Purpose	Files
xxx.Mainlayout (MainLayout.Normal, MainLayout.Print)	A standard Foundry site uses only one main layout and two or more sub-layouts containing specific renderings and placeholders for documents, modules etc.	"/sites/xxx/layouts/xxx.MainLayout.aspx" and "print" layout for printing content. "/sites/xxx/layouts/xxx.PrintLayout.aspx"

4.3.5 Sub-layouts

The following table shows the sub-layouts a site type consists of. Additional sub-layouts can be added to a site type. When creating new sub-layouts be sure to inherit the page from the "**MSS.Web.UI.UserControls.SiteControl**" class.

Name	Purpose	File
xxx.SingleColumn	Used for single column layout	/sites/xxx/layouts/xxx.SingleColumn.ascx
xxx.TwoColumns	Used for standard two column layout	/sites/xxx/layouts/xxx.TwoColumns.ascx



Name	Purpose	File
xxx.ThreeColumn	Used for standard three column layout	/sites/xxx/layouts/xxx.ThreeColumns.ascx
xxx.Frontpage	Used on the front page of the site	/sites/xxx/layouts/xxx.FrontPage.ascx

4.4 The Runtime Engine

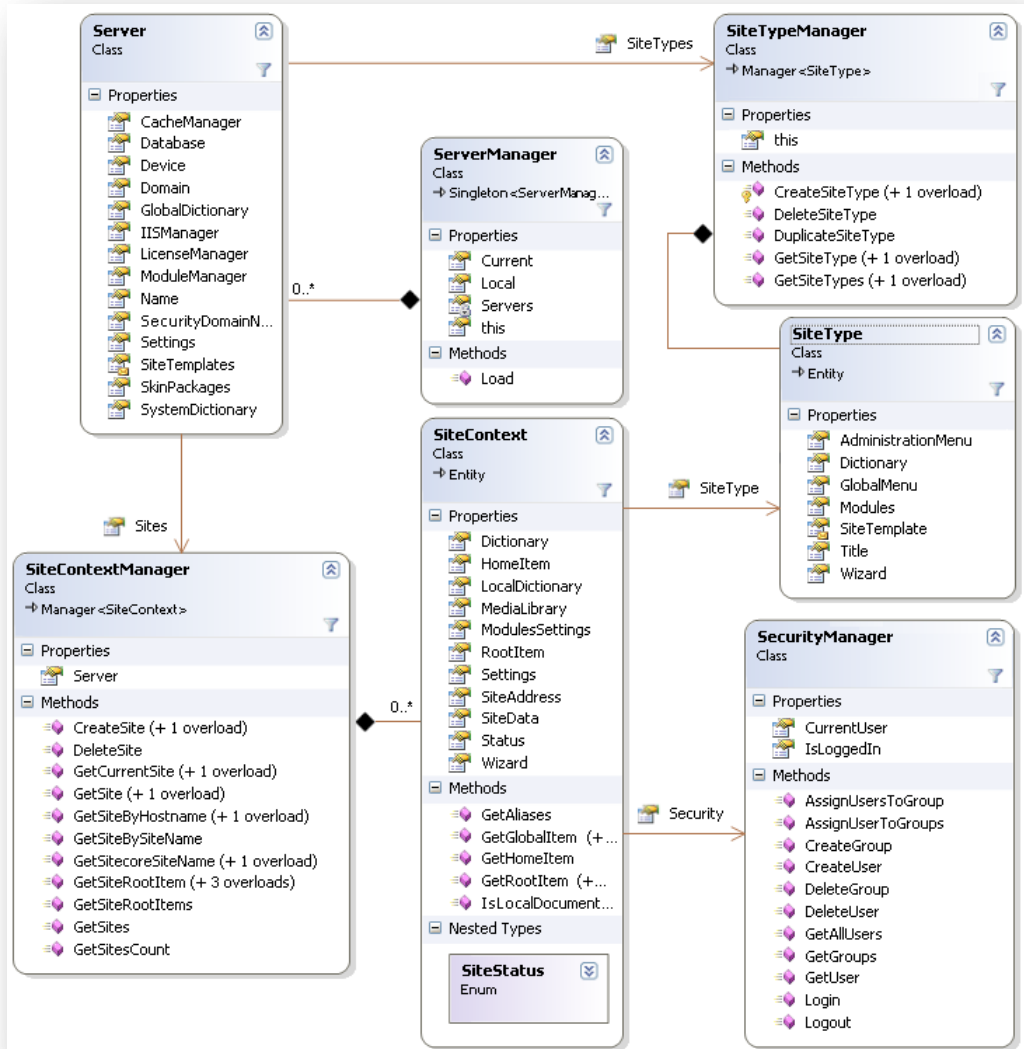
The Foundry Runtime Engine is designed to work with multiple servers, but in the current version it only has functionality implemented for one local server. You can get access to all runtime objects from a server object. To get a local server use the current property from the instance of the `ServerManager` object you can use the following:

```
Server local = ServerManager.Instance.Current;
```

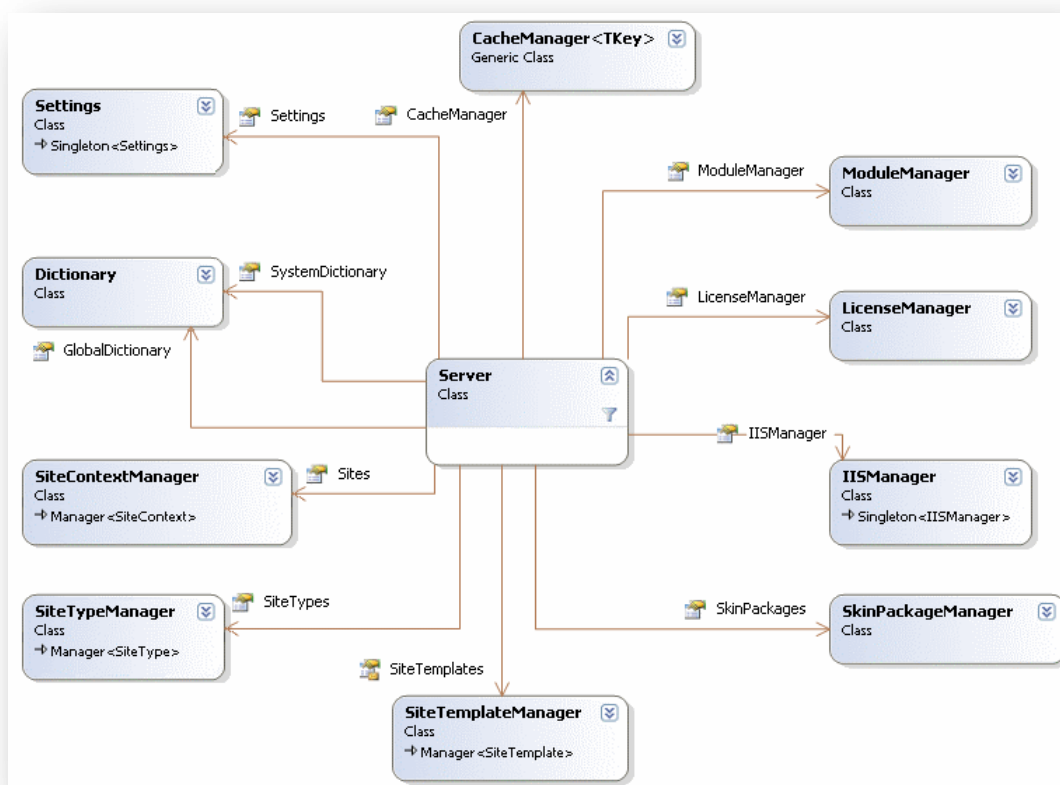
Or

```
Server local = MSS.Context.Server;
```

The Server object contains the cache manager, global and system dictionary, IIS manager, site, site type, site template, skin package and module managers. See the class diagrams below.



The following image is a representation of the server components.



This table introduces developers to the most common Sitecore Foundry classes

Class	Description
Context	Holds information about the current state.
ServerManager	Manages all sitecore foundry servers.
Server	Represents a sitecore foundry server. You can get access to all runtime object from a sitecore foundry server using this object.
SiteContextManager	Represents a manager of sitecore foundry sites.
SiteContext	Represents a sitecore foundry site. All sites is located under the /sitecore/content/Sitecore Foundry/Websites item.
LicenseManager	Represents a License manager which gets access to user license information.
SiteTypeManager	Represent a site type manager which give access to all site types.
SiteType	Represent a site type. Provides access to site type properties.
SecurityManager	Represents a site security manager. Provides methods for work with site users and groups.

Class	Description
Dictionary	Represents one level of the dictionary. Read more about four dictionary levels in the Administration guide document.
Settings	Provides access to setting from mss.config and web.config file.
ModuleManager	Represents a manager of sitecore foundry modules. Gives access to all modules from server.
Module	Represents a module. Provide access to module properties.
SkinPackageManager	Provides access to skin packages and skins.

You can get access to all Runtime object you can through MSS.Context class.

The table described below show some methods of the **MSS.Context** class.

Class	Description
Server	Gets the current server.
Language	Gets the current language.
Item	Gets the requested item.
CurrentDatabase	Gets the current database.
ContentDatabase	Gets the content database. It is the database where is stored all content of sites. All settings and content is stored in this database. Usually it is the database "master".
Site	Get the current site. Site is computed by host name.
Dictionary	Get the instance of the DictionaryManager class. You can get dictionary texts use this object.

4.4.1 The Server class.

The server class contains all objects from the selected server. In future versions sitecore foundry API will allow you to work with different sitecore foundry servers from different computers. But for now it enables you to work only with one server. Use the MSS.Context.Server static method to get the current server object.

The methods from the Server object are described below.

Class	Description
Sites	Gets the server site context manager as a MSS.Sites.Contexts.SiteContextManager. This class is used to work with sitecore foundry sites.
SiteTypes	Gets the server site type manager as a MSS.Sites.Types.SiteTypeManager object. This class is used to work with site types.
Domain	Gets the security domain as a Sitecore.SecurityModel.Domain object. Sitecore Foundry use one security domain for all sites, you can get this domain through this property.



Class	Description
IISManager	Gets the instance of the server iis manager as a MSS.IIS.IISManager object. You can work with iis from the selected server through this object.
LicenseManager	Gets the server license manager as a MSS.Licenses.LicenseManager object.
SystemDictionary	Gets the system dictionary as a MSS.Dictionaries.Dictionary object. It is dictionary which contains system level of dictionary. Read more about dictionary in the Administration Guide document.
GlobalDictionary	Gets the global dictionary as a MSS.Dictionaries.Dictionary object. It is dictionary which contains global level of dictionary.
ModuleManager	Gets the server module manager as a MSS.Modules.ModuleManager object.
Settings	Gets the server settings as a MSS.Configuration.Settings object.
SkinPackages	Gets the server skin package manager as a MSS.SkinPackages.SkinPackageManager object.

4.4.2 SiteContextManager and SiteContext

The site manager enables a user to work with Sitecore Foundry sites. The user can create, delete and manage sites using this class.

Examples:

get all Sitecore Foundry sites.

```
List<SiteContext> sites =
MSS.Context.Server.Sites.GetSites(MSS.Context.CurrentDatabase);
```

get the current site:

```
MSS.Context.Server.Sites.GetCurrentSite();
```

SiteContext represents a Sitecore Foundry site.

Some of the **SiteContextManager** methods are described below.

Class	Description
CreateSite	Creates a new site.
DeleteSite	Deletes a site.
GetSiteByHostName	Gets a site by host name.
GetSiteBySiteName	Gets a site by site name.
GetCurrentSite	Gets a current executed site.
GetSites	Gets all sites.
GetSiteRootItem	Gets site root item.

Some of the **SiteContext** methods are described below.

Class	Description
SiteId	Get unique id of the site.
HomeItem	Gets site home item.
MediaLibrary	Gets access to the site media library.
LocalDictionary	Gets site dictionary. It is the fourth level dictionary.
Settings	Gets access to site settings.
SiteData	Gets access to wizard saved settings.
Status	Gets a site status. It can be Running, Stopped, Updated or UnInitialized.
Wizard	Gets a site wizard as a MSS.Sites.Wizards.Wizard object.
GetAliases	Gets site aliases.

SecurityManager

The Securitymanager class provides access to site security. It contains the methods described below.

Class	Description
CreateGroup	Creates a new security group for a site.
CreateUser	Creates a new site user.
DeleteGroup	Deletes a group.
DeleteUser	Deletes a user.
GetGroups	Gets all site security groups.
GetAllUsers	Gets all site user.
Login	Login a user to a site.
SetRestrictAccess	Sets restricted access to an item. After than not all user will have access to the item.
RemoveRestrictAccess	Removes restricted access from an item.
UserIsExists	Checks whether user is exist.
GroupsExists	Checks whether group is exist.

4.4.3 SiteTypeManager and SiteTypes

Each site is based on a site type. Site types hold information about the site template, the site wizard, site administration and global menus, available modules, initial security settings for a new site, and the site type level of the Dictionary and Settings.

Some methods from **SiteTypeManager** are described below.

Class	Description
DuplicateSiteType	Creates a new site type by coping from another site type.
DeleteSiteType	Deletes a site type.



Class	Description
GetSiteTypes	Gets all site types.
GetSiteType	Gets a site type by site type name or site type id.

Some methods from **SiteType** are described below.

Class	Description
Title	Gets a site type title.
SiteTemplate	Gets a site template as a MSS.Sites.Templates.SiteTemplate object.
AdministrationMenu	Gets a site type administration menu.
GlobalMenu	Gets a site type global menu.
Wizard	Gets a site type wizard. All sites based on the site type will use this site wizard to configure site.
Modules	Gets list of available modules.
Dictionary	Gets dictionary of site type level.

4.4.4 Dictionary and DictionaryManager.

Sitecore Foundry has a four-level dictionary. At the first level the system tries to get the text from the local site dictionary. If it does not exist, the system tries to get the text from the site type dictionary, then the global dictionary and finally the system dictionary.

The Dictionary class represents one level of these dictionaries. **DictionaryManager** uses four dictionary levels to translate text.

Each dictionary item has four fields:

- **Text** – The text.
- **Hint** – – This is used in all dictionary web controls. It represents the tool tip text.
- **TabIndex** – This is used in all dictionary web controls. It represents a tab index for the control.
- **Accesskey** – This is used in all dictionary web controls. It represents an access key to the control.

To translate the text use an instance of the **DictionaryManager** object. A dictionary manager uses all four dictionaries when translating text – **site**, **site type**, **global** and **system** dictionary.

To get the dictionary manager use the static method **MSS.Context.Dictionary**.

Methods from the **DictionaryManager** are described below.

Class	Description
GetText	Gets a text by a key and a language.
GetHint	Gets a hint text by a key and a language.
GetTabIndex	Gets a tab index by a key and a language.
GetAccessKey	Gets a access key by a key and a language.

4.4.5 ModuleManager and Module

To work with modules use the module manager.

Methods from the **ModuleManager** are described below.

Class	Description
GetModules	Gets a list of modules which is on a item.
GetModules	Gets a list of available modules for a site type.
IsAvailable	Returns whether module is available for a site.

In a Module class there is only one important method – **GetRendering**. This method returns a list of controls which will render the module on a page.

4.4.6 SkinPackageManager, SkinPackage and Skin

Skin packages are used to set a quantity of available settings for a site. The skin is built from a set using these settings. Use **SkinPackageManager** to work with skins and skin packages.

Methods from the **SkinPackageManager** are described below.

Class	Description
GetSkinPackage	Gets selected skin package for a site.
GetSkinPackage	Gets skin package by id.
SetSkinPackage	Sets a skin package to a site.
SetSkin	Applies skin settings to a site.
GetAllSkinPackages	Gets all skin packages in the solution.
GetSkinPackages	Gets all available skin packages for a site.

Some methods from the **SkinPackage** are described below.

Class	Description
Title	Gets skin package title.
Group	Gets skin package group.
DefaultSkin	Gets default skin.
Skins	Gets all skins for the skin package.
Identities	Gets all identity images.
Tops	Gets all top images.
Buttons	Gets all button images.
ColumnBars	Gets all columns bar images.
TopBars	Gets all top bar images.
ContentBars	Gets all content bar images.
BottomBars	Gets all bottom bar images.

4.5 Security

Sitecore CMS by default is configured with two security domains: **Sitecore** and **Extranet**.

4.5.1 The Sitecore domain

This domain is primarily for internal security. It handles security for the Sitecore client. This domain stores information about content editors, administrators, developers and other members of the team who build and maintain the site.

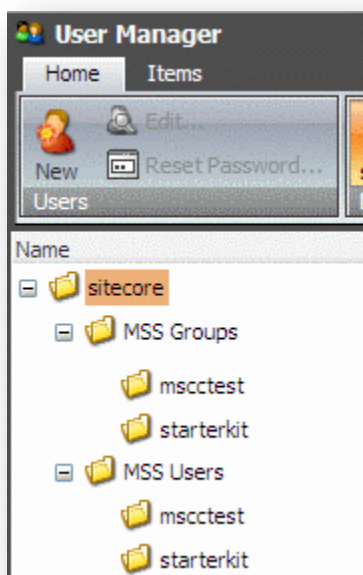
4.5.2 The Extranet domain

This domain is used for site security. It defines who can access the information published on website.

4.5.3 Foundry domains

Sitecore Foundry, however, is configured with only **one** domain – **Sitecore**. All users and group are stored in this domain.

Two folders are created for each site in the security database. When a site is created, folders for the site roles and for the site users are created.



`/sitecore/MSS Groups/site_name`

Here are stored all the roles for the site with name **site_name**.

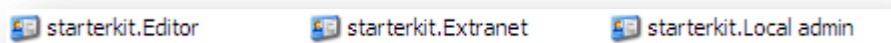
`/sitecore/MSS Users/site_name`

Here are stored all the users for the site with name **site_name**.

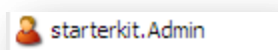
4.5.4 Users, Groups and Roles

Site roles and users are stored with the following naming format:

site_name.role_name



site_name.user_name



Example: for the Admin user in the site MyNewBlog the site user will be store in

/sitecore/MSS Users/MyNewBlog

As the user **MyNewBlog.Admin**

The Editor role will be stored in

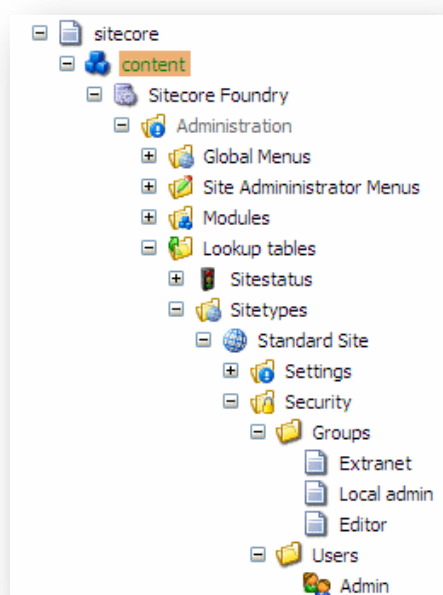
/sitecore/MSS Groups/MyNewBlog

As the role **MyNewBlog.Editor**

To login on the site the user should enter the username **Admin**, but to log in to the client interface a user should enter user **MyNewBlog.Admin**.

4.5.5 Setting up the Local Admin Rights

Each site type specifies a list of the default users and roles which will be created when a new site is created.

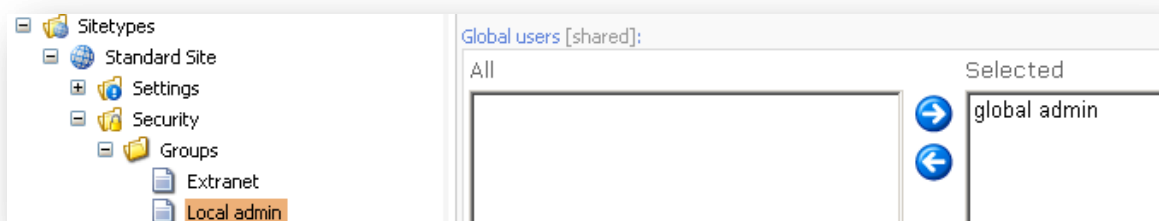


Under the Groups item are stored the default roles which will be created for each site based on the site type and under the User item are stored the default users, which will be created for each site based on the site type.

A group item has the following fields:

Global users

This specifies the list of the global users who will be a member of the created role when a new site has been created.



In the image above the **global admin** user will have the local admin roles for each site created using this role.

CopyRightsFrom

This field specifies the roles on the site templates, rights from these roles will be copied to a new role when a site is created.

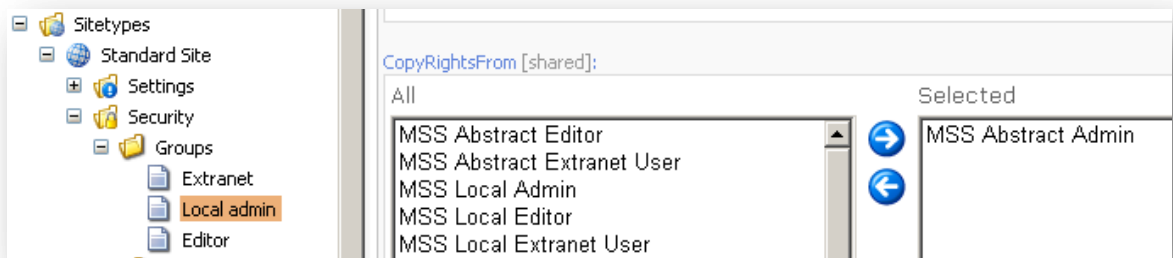
AdditionalRoles

This field specifies a list of additional roles, which will get the new role. The new role will be a member of these roles.

For example: If a site type has a site template. Each local admin of the site, based on this site type should have administrator rights to his site root item and should not have administrator rights to any other site's root items. In this case we add to the local site root in the site template administrator rights for the **Abstract Admin** role.

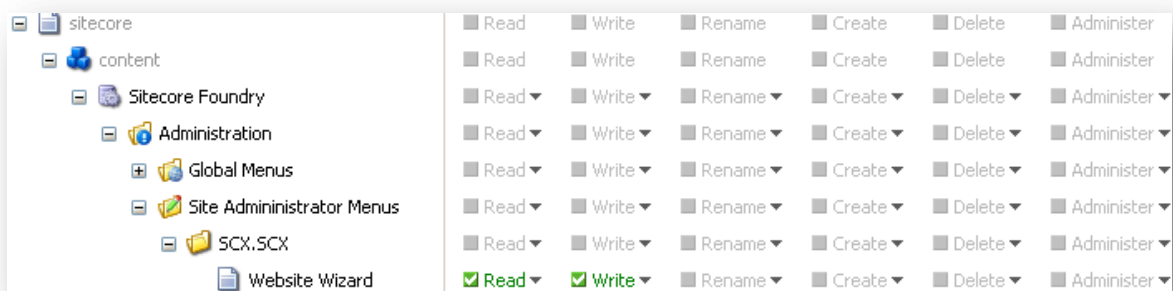


Then we add to the Admin group in the **CopyRightsFrom** field the **Abstract Admin** role

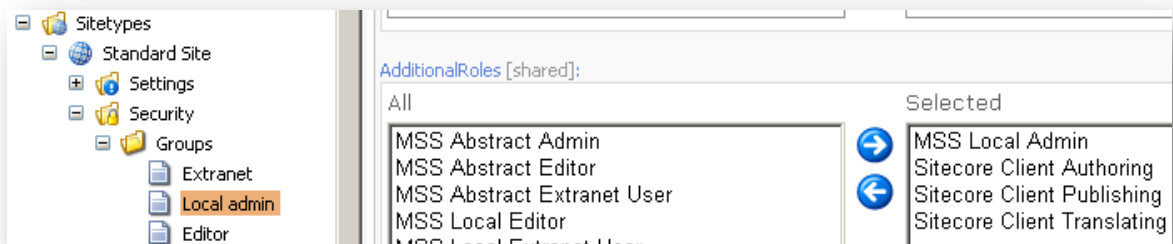


For each new site that will be created the Local admin role and this role will have Administration right to own site root, this right will be copied for the role.

In the second case all users of the Local admin role should have read/write right to some item in the Administration menu. We add the read/write right to the item for the MSS Local Admin user



And add the MSS Local Admin user to the **AdditionalRoles** field for the Local admin.



All new local admin roles will then be members of the MSS Local Admin roles and the users will have write access to the item in the administration menu.

4.6 The structure of a site

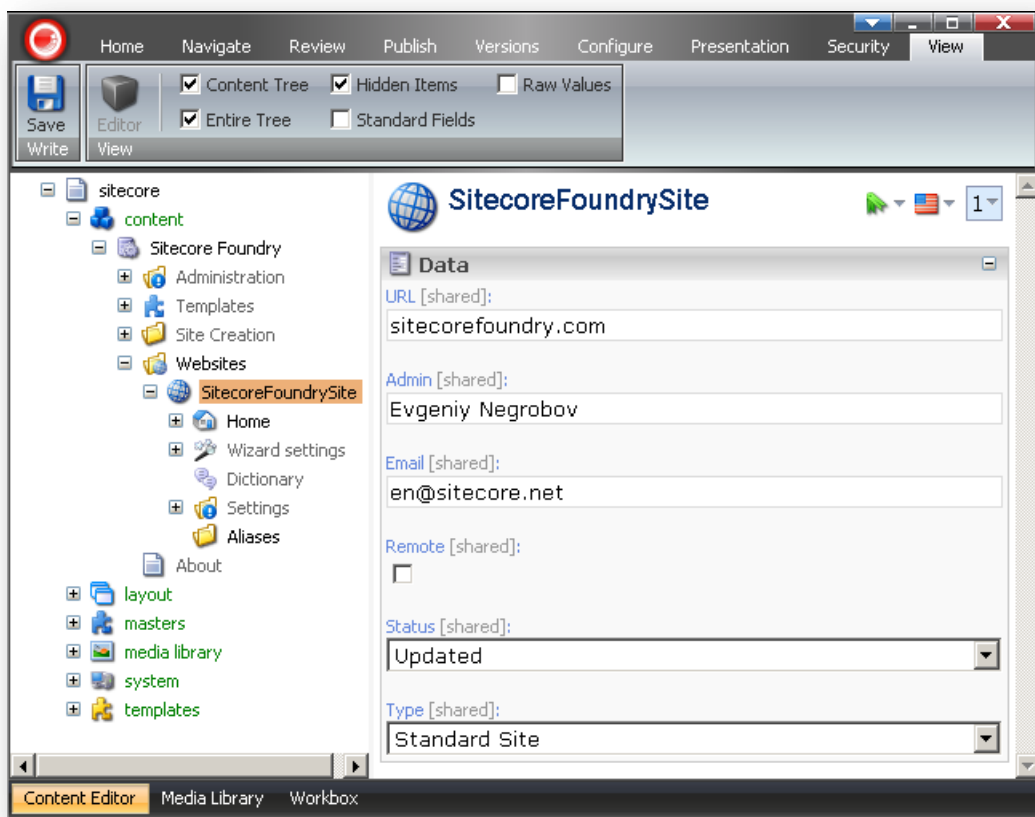
This section deals with the basic structure of a Foundry site.

All individual sites are located at:

/sitecore/content/mss content/Websites

The **site** item has the following fields:

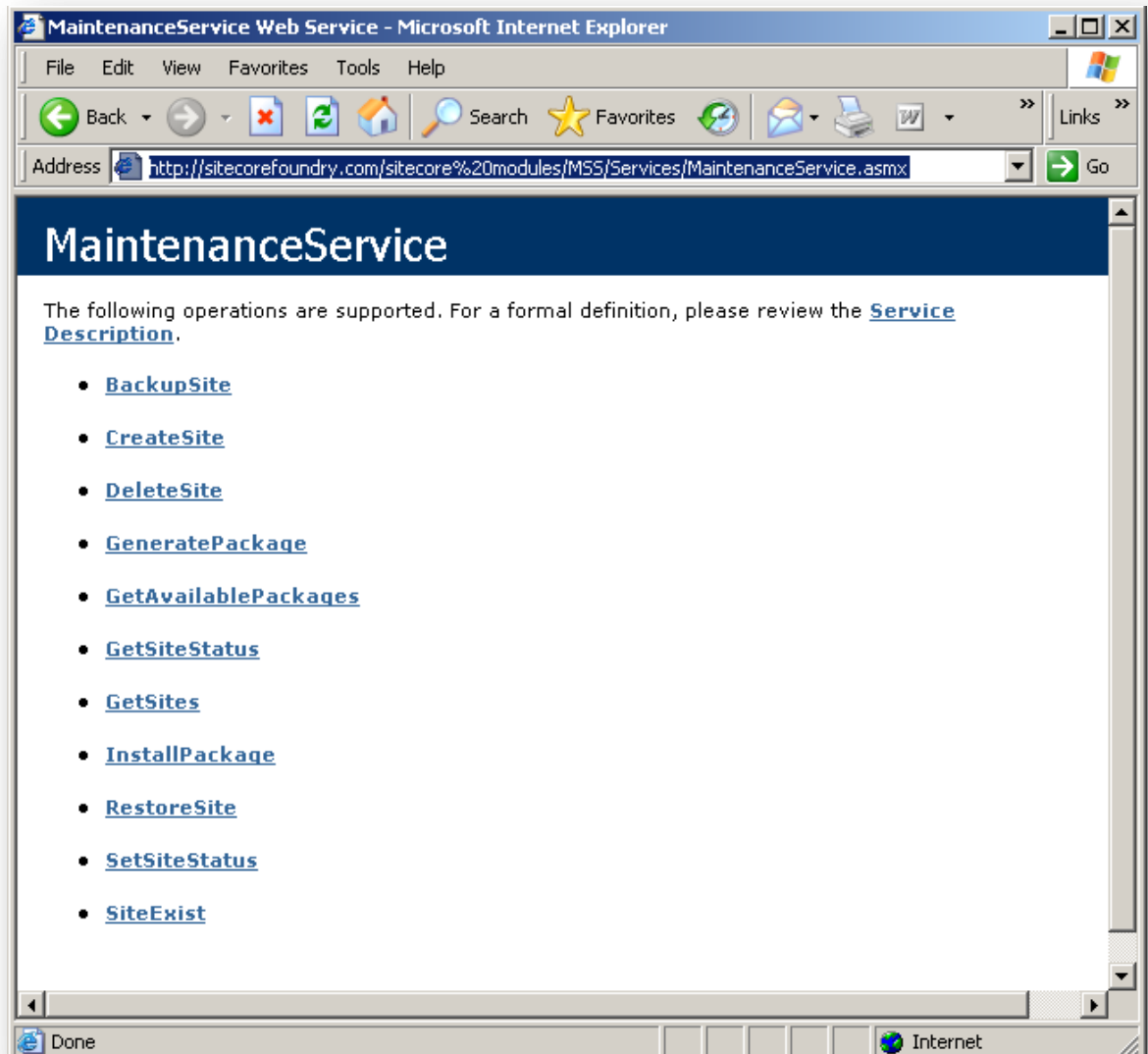
Name	Purpose
URL	The domain name used to match and resolve which site is shown. See “Sitecore Foundry installation” whitepaper for more detail.
Admin	Name or initials for the local website administrator
E-mail	Local website administrator’s e-mail
Remote	For internal use
Status	Determines site status. Can be Running, Stopped, Updated.
Type	Specifies which site type the site is running on



4.7 Sitecore Foundry Maintenance Service

Sitecore Foundry provides end user with web service which enables to user manage sites. The web service is located at

/sitecore 20modules/MSS/Services/MaintenanceService.asmx.



You are able to create, delete, backup and restore a site, read and set a sites status, check whether a site exists, read all sites and install packages on the server.

All service's methods require a valid Sitecore user to perform operations on the server. You should pass an instance of the **Sitecore.SecurityModel.Credentials** class with a sitecore administrator user to these methods.

Below is an example of the use of an instance of the Credentials object.

```
MaintenanceService service = new MaintenanceService();
Credentials credentials = new Credentials();
credentials.UserName = "Administrator";
credentials.Password = "Password";
```

```
string []sites = service.GetSites("master", credentials);
```

4.7.1 Creating a new site

Use the **CreateSite** function to create a new site, as in the example below.

```
string CreateSite(string siteTypeID,  
                 string siteName,  
                 string hostname,  
                 string administratorInitials,  
                 string administratorEmail,  
                 string defaultLanguage,  
                 bool waitForPublish,  
                 Credentials credentials)
```

where:

siteTypeID – The ID of a site type (The ID or name of the site type item),

siteName – The web site name,

hostname – The hostname for the new site,

administratorInitials – The local site administrator's initials,

administratorEmail – The local site administrator's e-mail,

defaultLanguage – The site's default language,

waitForPublish – Whether to wait for the publish or return control without waiting for the end of the publish action. If you use the server in live mode, the publish will not be performed, so the argument is unimportant,

credentials – The user's credentials.

The function returns a site creation report. If the creation process has failed the function throws the System.Exception.

4.7.1.1 *Example;*

The code used to create a new site is;

```
MaintenanceService service = new MaintenanceService();  
Credentials credentials = new Credentials();  
credentials.UserName = "Administrator";  
credentials.Password = "Password";  
service.CreateSite("{D18E2C18-B94F-44CF-98FE-1B9945C9EE7C}", "news",  
"news.sitecorefoundry.com", "john smith", "email@provider.domain", "en", false,  
credentials);
```

4.7.2 Deleting a site

Use the **DeleteSite** method to delete a site.

```
void DeleteSite(string siteName, Credentials credentials)
```

where;

siteName – The web site name,

credentials – The user's credentials.

If the site does not exist the method throws the System.Exception.

4.7.3 Checking to see if a site exists

Use the **SiteExist** function to check whether a site exist or not.

```
bool SiteExist(string siteName, Credentials credentials)
```

where;

siteName – The web site name

credentials – The user's credentials

4.7.4 Backup and restore a site

Use the **BackupSite** method to backup a site and the **RestoreSite** to restore a site from a site backup.

```
void BackupSite(string siteName, string backupName, Credentials credentials)
```

```
void RestoreSite(string backupName, Credentials credentials)
```

where:

siteName – The web site name

backupName – The name of the new backup or the name of the backup to be restored.

credentials – The user's credentials

4.7.4.1 *Example;*

Backup a site:

```
service.BackupSite("news", "news backup 070115 18.00", credentials);
```

Restore a site;

```
Service.RestoreSite("news backup 070115 18:00", credentials);
```

4.7.5 Work with packages

Sitecore Foundry maintenance web service provides you with a few methods for working with packages. Using these methods you can generate a new package from the package project, install a package and get all the available packages from the server.

- To return a list of packages from the package folder;

```
string[] GetAvailablePackages(Credentials credentials)
```

- To install a package on the server.

```
string InstallPackage(string packageFileName, Credentials credentials)
```

This function enable you install a package from any place on the server and not just from the packages folder. The function returns an install log. If the function fails it throws a exception.

4.7.5.1 *Examples:*

Installing a package

```
service.InstallPackage("/data/packages/items.zip", credentials);
```

Generating a new package

```
void GeneratePackage(string solutionFile, string packageFile, Credentials  
credentials)  
service.GeneratePackage("/data/packages/mss.layouts.xml",  
"/data/packages/mss.layouts.zip", credentials);
```

where;

solutionFile – The file to be packaged

packageFile – The name for the new package file

credentials – The user's credentials

This code generates a new package from a package project file.

Chapter 5

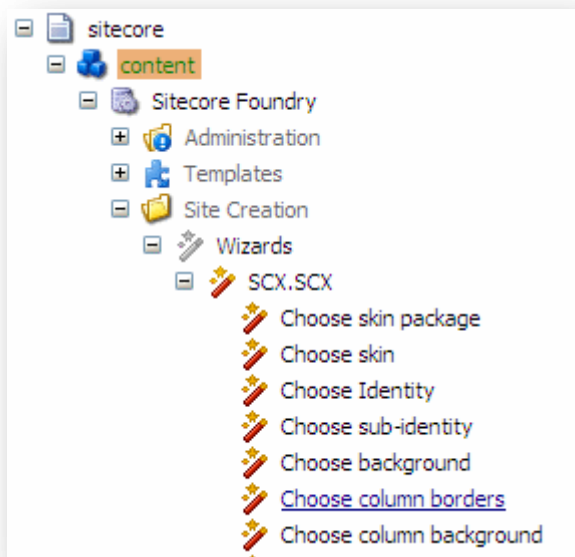
The wizard and skin packages

The Website Wizard is the part of a Foundry site that allows the Local Administrator to customize his sites appearance to give it that unique look and feel. However, the wizard itself is also customizable and can vary a great deal from one site type to another.

5.1 Introduction

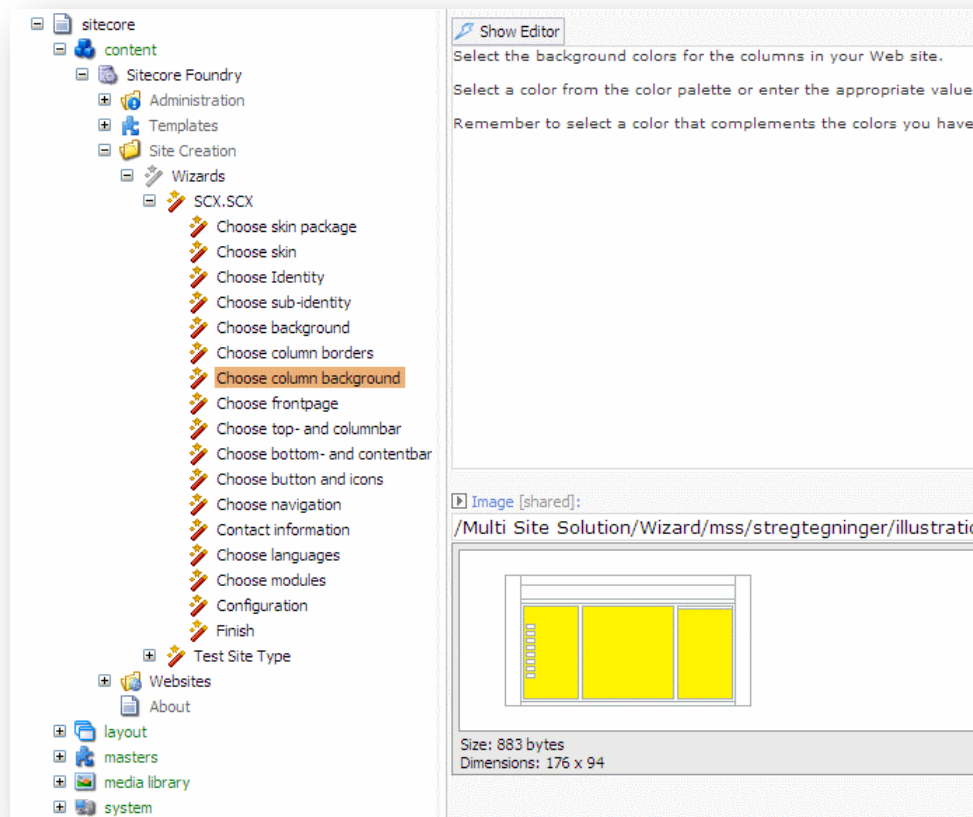
A Website Wizard consists of a number of steps and each step in the Wizard contains information displayed in the user version of the Wizard. The Wizard definition for the Standard site type is located at

`/sitecore/content/mss content/Site Creation/Wizards/SCX.SCX`





A step also keeps a reference to a page that is the actual wizard page containing the forms, color pickers and image pickers displayed on the individual steps.



Having the wizard steps with layout and text defined in one place and the functionality defined in another place, makes it possible to change or customize the layout and design of the wizard as well as change the text of the steps in the wizard between two different Site types.

The Wizard delivered with the Standard Sitecore Foundry site uses three standard skin packages **Children**, **Corporate** and **Cultural**, which are all free designs. Each of the default skin package has six skins defined for it.

The Wizard uses the definition of the skin packages and skins to select from on the individual wizard steps. For more information about adding new wizard steps see “5.8 Customizing the wizard”, on page 82.

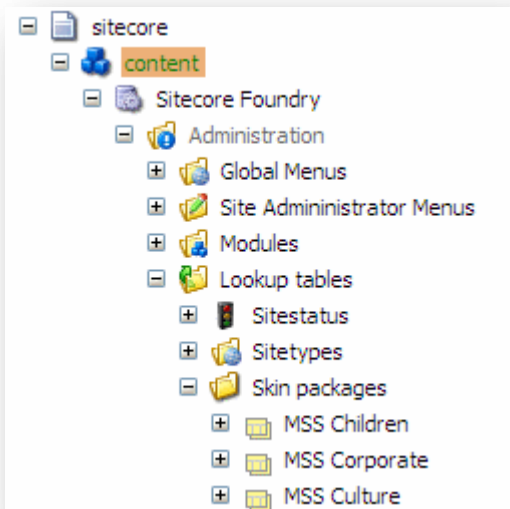
5.2 Skin Packages

In Sitecore Foundry a web-site consists primarily of three elementary components: The layouts, the renderings and the skins. Layouts and renderings are standard Sitecore components and the skins are a part of Sitecore Foundry. Layouts define the size and position of the different elements on the web-page, while the renderings define the individual elements, by generating the HTML sent to the clients.

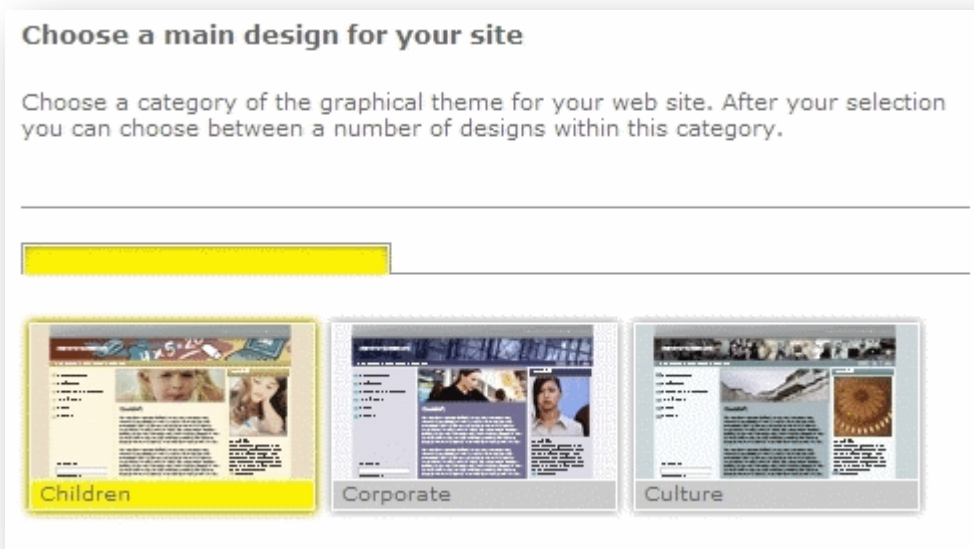
A skin package consists of a number of graphic elements, colors palettes and skins. A skin is a predefined selection of graphic elements for background, top bar and coloring. The graphic elements are picked from the elements in the package. There are no limits

to how many skins that can be defined from a package, but usually a number of skins are predefined by the designer. The skin packages are located at:

/sitecore/content/mss content/Administration/Lookup tables/Skin packages



The skins have two kinds of interfaces. The developer uses the Sitecore client to configure what skins are possible. The end-users use the Sitecore Foundry Wizard to configure what skin elements to use on the web-site.

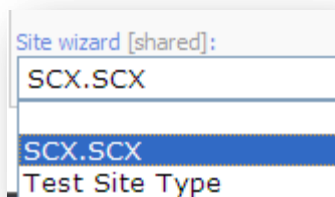


A skin package can be used on different layouts, e.g. a web-site with a level 1 top menu and left menu for levels 2+, could easily use the same skin as one with only a left menu. In theory the visual differences could be larger than that; the layouts and the skin just have to be designed with that in mind.

5.3 Skin packages and site types

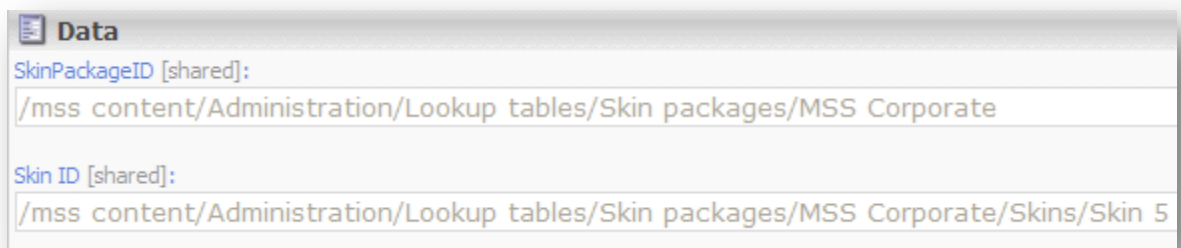
Since skin packages may differ a great deal in design, it is possible, to have skin packages that are not compatible with the same layouts. There could be a layout that is designed for a widescreen high resolution monitor (1680x1050 pixels) and another layout designed for a basic 800x600 pixel monitor. These would most probably require different skin packages, since image sizes may vary in both size and shape.

On a Site type it is possible to select which of the installed skin packages is compatible with the Site type. The definition is held in a field on the Site Type root item in the field **Site wizard**.



The wizard will, in its standard form, show all compatible skin packages. A single site has a reference to the Skin package that it is defined from. This data is stored against the item

/sitecore/content/mss content/Websites/Site Name/Wizard Settings/Skin.Settings



The skin is defined within the package itself. When a new site is created the skin is taken from the defaults for the site type.

5.4 Skin Fields

This section describes the fields defined for the Skin items located at:

sitecore/content/mss content/Administration/Lookup tables/Skin packages/

5.4.1 Skin package data items

The fields for this item are as follows:

5.4.1.1 Title

This is the name of the skin package.

Title:
Corporate

It will be displayed under the image of the skin package on the Design page of the Website Wizard

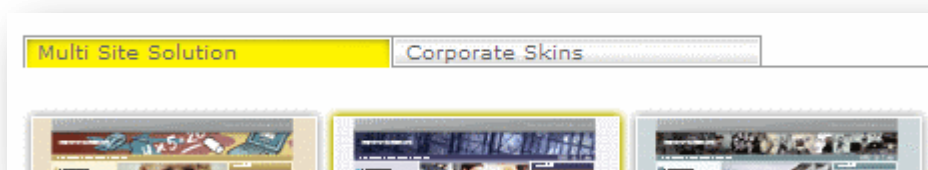


5.4.1.2 Group

This field is used to group the skin packages together on tabs with each set of related skins under a common group name.

Group [shared]:
Multi Site Solution

This name is used for the title of the tab which holds the skins.



5.4.1.3 Default skin

This is the default skin selected on the Wizard page following the selection of the skin package.

Default skin [shared]:
Skin 1



If no default skin is selected the package will automatically default to the first available skin.



5.4.1.4 Stylesheet

This is a relative path to a CSS Style file containing styling corrections for the skin display.

```
Stylesheet [shared]:  
/skin packages/mss/cooperate/css/cooperate.skin.css
```

Using the same skin package on different site types may require corrections in order to be displayed correctly. This stylesheet is where those corrections would be placed. This allows you to use the same skin package on different site types, giving a further layer of flexibility.

5.4.1.5 Wizard stylesheet

This is a relative path to a CSS stylesheet file containing styling corrections for the display of the skin package on the Website Wizard.

```
Wizard stylesheet [shared]:  
[ ]
```

Different wizard designs might require corrections in order for the skin package content to be displayed as required in the wizard. Skin packages might differ slightly and may need small corrections to be displayed correctly.

5.4.1.6 Sitetypes

This is where you can associate the site types that are compatible with this wizard.



5.4.2 Skin package components

Each skin package contains a number of folders which contain the data selectable in each step of the Website Wizard. These steps are not fixed and alterations and additions can be made by developers. For more information see "5.8 Customizing the wizard", on page 82.

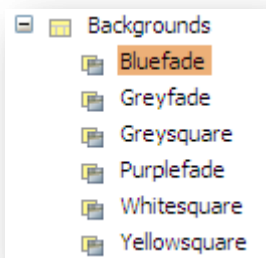
These skin package components are located at:

/sitecore/content/mss content/Administration/Lookup tables/Skin packages/skin name/

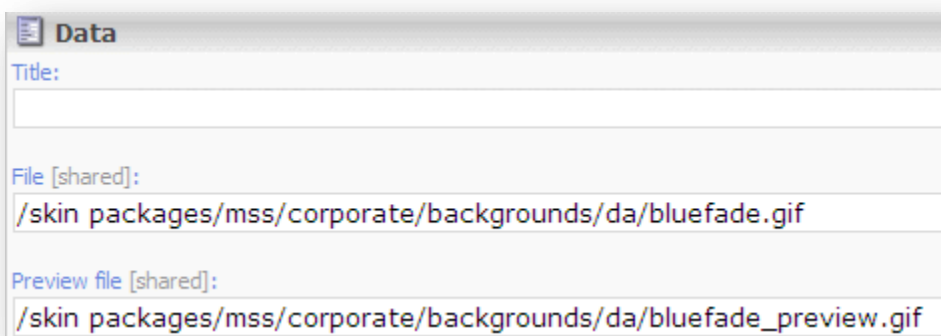
The components of the standard skin package are as follows.

5.4.2.1 Backgrounds

This folder contains a set of background images used on the standard wizard for the entire site background and column backgrounds.



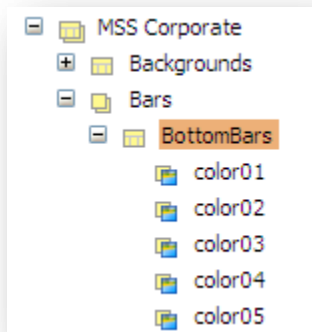
Each item contains three fields.



These are **Title**, **File** and **Preview file**. **Title** is the title of the image. File and Preview file are relative pointers to the image linked with this item.

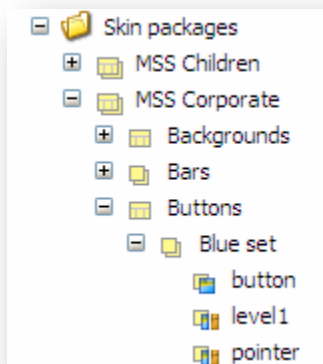
5.4.2.2 Bars

This folder stores sets of images used for the bottom bar, column bars, content bars and the top bar. Each of these is a folder item containing color locations as in the backgrounds folder.



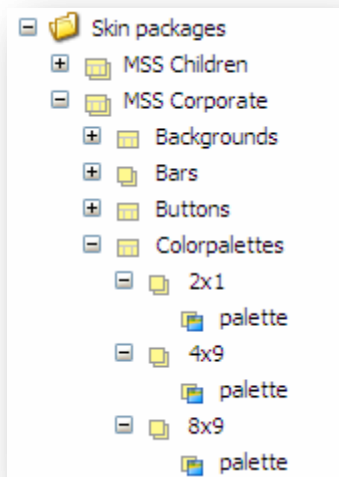
5.4.2.3 Buttons

This folder consists of a set of folders containing color sets for images used as backgrounds on buttons, menu bullets and pointers.



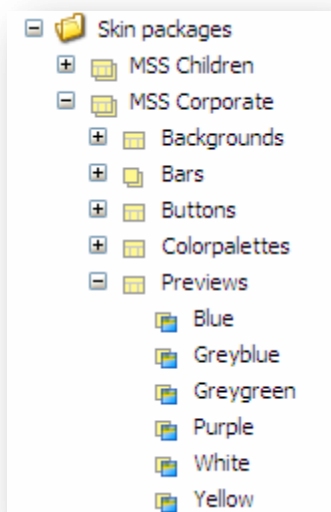
5.4.2.4 Colorpalettes

This folder consists of color palettes of different sizes and number of colors used in choosing colors for the various background and text colors on a website.



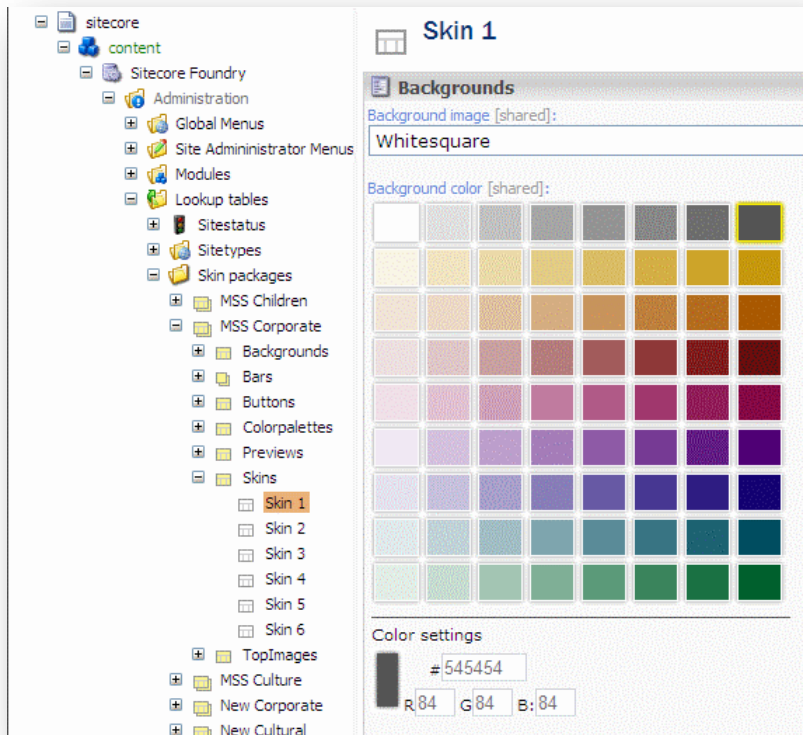
5.4.2.5 Previews

This folder consists of a set of thumbnail snapshots of a site. One for each skin.



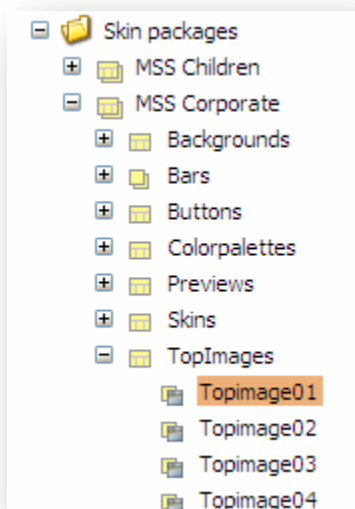
5.4.2.6 Skins

This folder contains a set of predefined settings for each skin for the selected skin package. In the Website Wizard these settings are used along with a preview thumbnail to allow the user to see how the site will look if they choose the selected skin



5.4.2.7 TopImages

This folder consists of a set of images which can be used for the top bar.

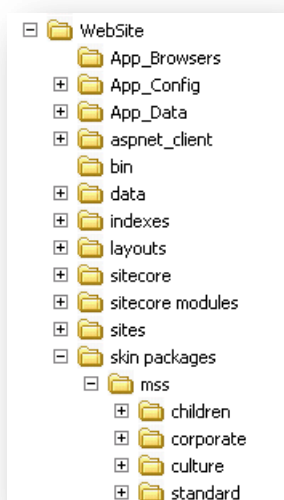


5.5 Preview images

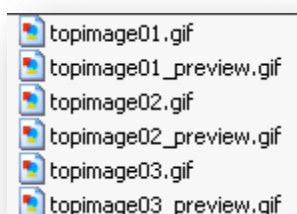
What is displayed on the resulting website and what is shown in the wizard are two different things. Therefore some of the images also have a preview image associated with them that is used in the wizard to better illustrate how it looks. For example, background images can be small images with dimensions of 2x4 pixels or 1x1200 pixels. Instead of stretching these images to make them more easily visible in the wizard, a preview image with dimensions of 143x53 pixels is supplied for each image for easy viewing.

5.6 Skin package files

The images and color palettes defined in the skin packages all have a physical file associated and located on a local hard drive in the directory **/skin packages**.



In the case of the standard skin packages they are located in the directory **/skin packages/mss**. The directory structure resembles the structure of the skin packages in Sitecore content. A suffix **_preview** is used as a naming convention for preview images for instance the image **Image001.jpg** would have an associated preview image of **Image001_preview.jpg**. All images are referenced by the relative path from the root of the site.



5.7 Creating your own skin package

Sitecore foundry also supports the creation of complete custom skin packages. For more information on creating your own skin package see the document **How to create a skin package**.

5.8 Customizing the wizard

The wizard can be customized in several ways:

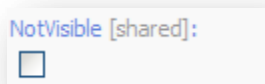
- It is possible to hide or remove steps not needed.
- It is possible to add new steps configuring settings not supplied by the default Foundry site.
- It is possible to change the CSS styling output used on the site.
- It is possible to change the CSS styling used in the wizard

5.8.1 Hiding or Removing Unwanted Steps

To hide or remove unwanted steps you can take the following action.

5.8.1.1 *Hiding Steps*

To hide a Wizard step you just need to click the **NotVisible** check box at the bottom of the wizard step.



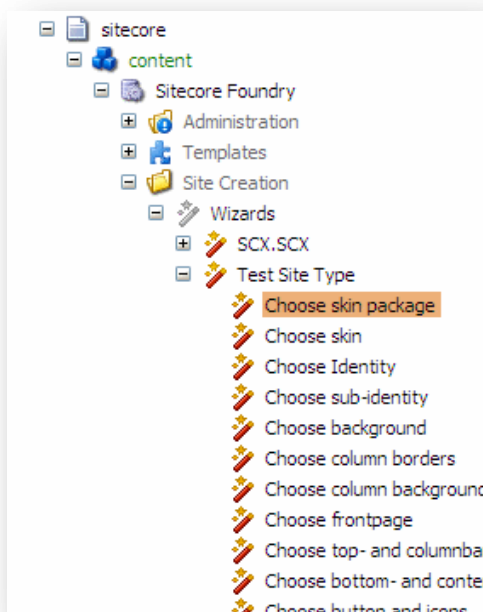
Then save, and publish the item and it will not be visible in the users Website Wizard.

Note: It is not advisable to hide the finish page of the wizard as users will then have no way to save any alterations they make to the skin.

5.8.1.2 Removing Steps

Wizards are all stored globally at:

/sitecore/content/mss/Site Creation/Wizards/



To remove a Wizard step you can just delete the step from the definition. However, this leaves the step definition, which is stored at:

/sitecore/content/mss/Administration/Lookup Tables/Wizardsteps/

Note: This is the safest way to remove unwanted wizard steps. If you remove the step definition then you risk corrupting the steps in other wizards.

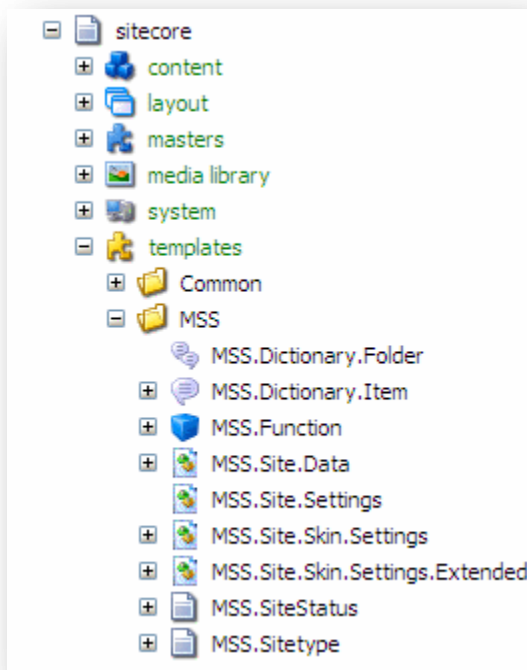
5.8.2 Adding Steps

In order to add a new step to the wizard, follow these steps:

1. Create the required sublayout in Sitecore.
2. Create a new step document.
3. Add the sublayout to the new step document.
4. Add the sublayout to the standard layout **WizardMainLayout**.
5. Add the sublayout to the sublayout **WizardMainLayout**.
6. Add the sublayout to the rendering **MSS.Wizard.Document**.

For the new settings to be stored, make a duplicate of the system template **MSS.Site.Data** or **MSS.Site.Skin.Settings** and replace it on the path **home/wizard settings** under the site template. Both of these templates can be found at:

/sitecore/templates/MSS/



The difference between the two is that **MSS.Site.Skin.Settings** is used for the skin settings read from the predefined skins every time a new skin is chosen.

Whereas, **MSS.Site.Data** is used to store input from the user such as contact information and system settings.

The wizard uses a .NET class to implement the interface **ICssFactory**. The used class is found by instantiating it from the assembly name and type name given on the wizard's front page item.

5.9 Wizard generated CSS classes

By default the wizard generates the following css classes.

Class name	Purpose	Wizard step	Using fields
body.areaBody	Sets the browser background color or image	step "Background"	"Background color", "Background image".
div.areaSublayout	Sets the column background color or image	step "Column borders"	"Columns background color", "Columns background image".
table.sublayoutTable td.leftcol	Sets the left column background color	step "Column colors"	"Column left color"
table.sublayoutTable td.centercol	Sets the center column background	step "Column colors"	"Column center color"



table.sublayoutTable td.rightcol	Sets the right column background color	step "Column colors"	"Column right color"
table.sublayoutTable td.frontcol	Sets the frontpage center column background color depending on	step "Front page"	"Frontpage Column color"
div.areaIdentity	Sets corporate identity image or background color	step "Corporate Identity"	"Identity color", "Identity image", "Uploaded identity image"
div.areaTopImage	Sets topbar image or background color	Step "Individual Identity"	"Top image", "Top color", "Uploaded top image"
div.MssNavigation div.navButtons input	Sets button image background and width	Step "Buttons and icons"	"Button set"
img.MssListArrow	Sets bullet image on menu items	Step "Buttons and icons"	"Button set"
div.MssSubMenu	Sets bullet image on menu items	Step "Buttons and icons"	"Button set"
div.areaBreadcru mb	Sets image used for top bars	Step "Choose bar design"	"Topbar image"
div.areaFooter	Sets image used for bottom bars	Step "Bottom and content bars"	"Bottombar image"

Important: These class names are subject to change in the future, you can use your own names by implementing the **ICssFactory** interface and setting the assembly name and the type name on the wizard. For more information, see the section "Chapter 5 The wizard and skin packages", on page 71.

Chapter 6

Appendix A: Configuration

This Appendix deals with the configuration of Foundry using the standard Sitecore web.config and the Foundry file mss.config.

6.1 Configuration

Sitecore Foundry has its own configuration file called mss.config. However, it still ties in some functionality to web.config.

6.1.1 MSS.config

A new file similar to the .net **web.config** file is supplied by Sitecore Foundry entitled **mss.config**. Here is an example of how it looks.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <mss>
    <settings>
      <setting name="Sites.SingleDatabase" value="true" />
      <setting name="UseFriendlyURL" value="true" />
      <setting name="SMTPServer" value="mail.server.net" />
      <setting name="MainSiteAddress" value="localhost" />
      <setting name="MainSiteDescription" value="MSS" />
      <setting name="SiteTemplate" value="mss.site" />
      <setting name="ExceptionMailAddress" value="support@sitecorefoundry.com" />
    </settings>
    <setting name="PasswordLength" value="8" />
    <setting name="GlobalUsers" value="global admin|sitecorefoundry" />
    <setting name="UpdateIISAlias" value="true" />
    <setting name="IndexUpdateInterval" value="00:10:00" />
    <setting name="DefaultLanguage" value="en" />
    <setting name="SecurityDomainName" value="sitecore" />
    <setting name="SiteUpdatedPage" value="/Global/SiteUpdated" />
    <setting name="DomainNameRegEx" value="^[A-Za-z]([A-Za-z0-9])*((\.-)[A-Za-z]([A-Za-z0-9])*)*$" />
    <setting name="DeniedSiteNames" value="sitecorefoundry|files|images|secure" />
    <setting name="BackupFolder" value="backups" />
    <!-- Initial site status - one of the next: Running, Stopped, Updated -->
    <setting name="InitialSiteStatus" value="Updated" />
  </settings>
  <securitySettings>
    <right name="ManageSecurity" allow="MSS Local Admin" deny="" />
    <right name="ManageWizard" allow="MSS Local Admin" deny="" />
  </securitySettings>
</mss>
</configuration>
```

```

        <right name="ContentEditing" allow="MSS Local Admin|MSS Local Editor"
deny="" />
        <right name="CanSeeBackend" allow="MSS Local Admin|MSS Local Editor"
deny="" />
        <!-- Managing modules -->
        <right name="CanSeeDataviewer" allow="MSS Local Admin|MSS Local Editor"
deny="" />
        <right name="ManagingNewsletterModule" allow="MSS Local Admin|MSS Local
Editor" deny="" />
        <right name="ManagingSMSModule" allow="MSS Local Admin|MSS Local Editor"
deny="" />
    </securitySettings>
</mss>
</configuration>

```

Note: For changes to web.config pertaining to the accompanying modules see the documentation for each of these modules.

6.2 Sitecore Foundry specific additions to web.config

Though Sitecore Foundry has its own mss.config file with Foundry specific settings, there are still a number of places where the system hooks into the Sitecore environment. These places in the web.config file are listed below:

6.2.1 <AppSettings>

- <add key="configfile" value="/mss.config"/>

This tells Sitecore foundry where to find the Sitecore Foundry settings file.

6.2.2 Item:saved Event Handlers

In the **Item:saved** the following event handlers are used.

- <handler type="MSS.Modules.EventCalendar.ItemHandlers.ItemMover, MSS.Modules" method="OnItemSaved" />
Moves calendar event documents to the correct year and month folder based on the date field value.
- <handler type="MSS.Modules.News.ItemHandlers.ItemHandler, MSS.Modules" method="OnItemSaved" />
Moves a news documents to the correct year and month folder based on the date field value.
- <handler type="MSS.Caching.CacheEventHandler, MSS.Kernel" method="OnItemSaved" />
Clears the mss caches
- <handler type="MSS.Sites.Contexts.ItemEventHandler, MSS.Kernel" method="OnItemSaved" />
Handles event for a site root item and changes sites list – adds or removes the site to site list.

6.2.3 Item:deleted event handlers

- <handler type="MSS.Modules.EventCalendar.ItemHandlers.ItemMover, MSS.Modules" method="OnItemDeleted" />
After an event is deleted this handler cleans the month and year folder
- <handler type="MSS.Modules.PictureSeries.ItemEventHandler, MSS.Modules" method="OnItemDeleted" />



When a user has deleted a picture series document with images this handler deletes the images from the media library.

- `<handler type="MSS.Caching.CacheEventHandler, MSS.Kernel" method="OnItemDeleted" />`
Removes the entity based on the deleted item from the MSS caches.
- `<handler type="MSS.Caching.CacheEventHandler, MSS.Kernel" method="OnItemDeleted" />`
Removes the entity based on the deleted item from the MSS caches.
- `<handler type="MSS.Sites.Contexts.ItemEventHandler, MSS.Kernel" method="OnItemDeleted" />`
Handles site root items and removes the site from site list.

6.2.4 Publish:end event handlers

- `<handler type="MSS.Publishing.HtmlCacheClearer, MSS.Kernel" method="ClearHtmlCache"/>`
Clears html cache when publish has finished.

6.2.5 Pipeline processors

- `<processor type="MSS.WebSite.Components.Search.Native.InitializeNativeSearchIndexer, MSS.WebSite" />`
Initialize the search indexer that rebuilds the search indexes.
- `<processor type="MSS.Publishing.PublishingWatcher, MSS.Kernel" />`
Hides/shows some items for a user when the user change the **Sites.SingleDatabase** setting (e.g. the Publish site menu)
- `<processor type="MSS.SitecoreExtensions.Pipelines.SiteResolver, MSS.Kernel" />`
MSS site resolver.
- `<processor type="MSS.SitecoreExtensions.Pipelines.SecurityResolver, MSS.Kernel" />`
Does not allow a user without the CanSeeBackend rights to enter the Sitecore client.
- `<processor type="MSS.SitecoreExtensions.Pipelines.ItemResolver, MSS.Kernel" />`
Represents the item resolver which hides the site from user when the site has the Updated status.

6.2.6 Database changes

- `<Engines.HistoryEngine.Storage>`
`<obj type="Sitecore.Data.$(database).$(database)HistoryStorage,`
`Sitecore.$(database)">`
...
`</obj>`
`</Engines.HistoryEngine.Storage>`
Adds the history engine to the web database. The engine is used to update the search indexes.

6.2.7 Indexes

- `<index id="mssIndex" singleInstance="true" type="Sitecore.Data.Indexing.Index, Sitecore.Kernel">`
`...`
`</index>`
 Adds the search index which is used to search on the web site.

6.2.8 Domains

Removes the Extranet security domain and changes the type of the Sitecore domain.

6.2.9 Processors

- `<processor mode="on" type="MSS.SitecoreExtensions.Pipelines.Login, MSS.Kernel" />`
 Does not allow login to the Sitecore client for a user with the CanSeeBackend rights.

6.2.10 xslExtensions

- `<extension mode="on" type="MSS.XslHelper.XslHelper, MSS.XslHelper" namespace="http://www.sitecore.net/mss" singleInstance="true"/>`
 Makes the Sitecore Foundry specific help functions available in XSLT files
- `<extension mode="on" type="MSS.XslHelper.XslDictionaryHelper, MSS.XslHelper" namespace="http://www.sitecore.net/dic" singleInstance="true"/>`
 Makes dictionary functionality available in XSLT files
- `<extension mode="on" type="MSS.Wizard.API.WizardXslHelper, MSS.Wizard" namespace="http://www.sitecore.net/wizard" singleInstance="true"></extension>`
 Makes Sitecore Foundry specific help functions available in XSLT files for the wizard

6.2.11 xslControls

- `<control mode="on" tag="scx:image" type="MSS.Web.UI.XslControls.Image" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:label" type="MSS.Web.UI.XslControls.Label" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:link" type="MSS.Web.UI.XslControls.Link" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:flash" type="MSS.Web.UI.XslControls.Flash" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:DictionaryLink" type="MSS.Web.UI.XslControls.DictionaryLink" assembly="MSS.WebEngine"></control>`
- `<control mode="on" tag="scx:Breadcrumb" type="MSS.Web.UI.XslControls.Breadcrumb" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:text" type="MSS.Web.UI.XslControls.Text" assembly="MSS.WebEngine"/>`
- `<control mode="on" tag="scx:html" type="MSS.Web.UI.XslControls.Html" assembly="MSS.WebEngine"/>`

Adds various xsl controls to xslt

6.2.12 controlSources

- `<source mode="on" namespace="MSS.WebSite.Components.RecycleBin" assembly="MSS.WebSite" />`
- `<source mode="on" namespace="MSS.Modules.Common.DataViewer" folder="/sitecore modules/MSS/Modules/Common/DataViewer/Xaml" deep="true" />`
- `<source mode="on" namespace="MSS.Modules.Common.DataViewer" folder="/sitecore modules/MSS/Modules/NewsLetter/Xaml" deep="true" />`
- `<source mode="on" namespace="MSS.Modules.Common.DataViewer" folder="/sitecore modules/MSS/Modules/SMS/Xaml" deep="true" />`
- `<source mode="on" namespace="MSS.WebSite.Components.Reorganize" folder="/sitecore modules/MSS/WebSite/Components/Reorganize" deep="false" />`
- `<source mode="on" namespace="MSS.WebSite.Components.SecurityEditor.Codebeside" folder="/sitecore modules/MSS/WebSite/Components/SecurityEditor/Xaml" deep="true" />`
- `<source mode="on" namespace="Sitecore.Web.UI.XmlControls" folder="/sitecore modules/MSS/Administration/Application" deep="true"/>`
- `<source mode="on" namespace="MSS.Web.UI.XamlControls" assembly="MSS.WebEngine" />`
- `<source mode="on" namespace="MSS.Web.UI.WebControls" assembly="MSS.WebEngine" />`

Adds various xaml controls

6.2.13 References

- `<reference>/bin/MSS.Modules.dll</reference>`
- `<reference>/bin/MSS.Administration.dll</reference>`
- `<reference>/bin/MSS.WebSite.dll</reference>`
- `<reference>/bin/MSS.WebEngine.dll</reference>`
- `<reference>/bin/Sitecore.AsyncUI.dll</reference>`
- `<reference>/bin/MSS.StatCenter.dll</reference>`

Add assemblies to the xaml controls

6.2.14 Settings

- `<setting name="WebStylesheet" value="/Sites/SCX.SCX/CSS/Design.css" />`
Adds mss styles to the Rich Text Editor

6.2.15 log4net

- `<appender name="MSS" type="log4net.Appender.SitecoreLogFileAppender">`
...
`</appender>`
`<logger name="MSS.Logging.Log" additivity="false">`

...
</logger>
Add the mss logger in log4net module.

6.2.16 http Modules

- <add type="MSS.Web.HttpModule, MSS.Kernel" name="SitecoreFoundryHttpModule" />
Adds mss http module.

Chapter 7

Appendix B: Wizard styling output

The output from the wizard is generated by the aspx file “/sitecore modules/MSS/wizard/ skins/ skincss.aspx”. A normal stylesheet link is placed in the main layout file:

7.1 Link Example

An example of the code used in the stylesheet link is;

```
<html>
  <head>
    <link id="SkinCss" rel="stylesheet" type="text/css"
href="/sitecore modules/MSS/Wizard/Skins/SkinCss.aspx" />
  </head>
</html>
```

The sample output looks like this (generated by the default wizard ICSSFactory implementation, see below):

```
body.areaBody {color: rgb(51,51,51);background-image: url(/Sitewizards/Sitecore
Express/Images/background/1/cust_background01.gif)}
div.areaIdentity {}
div.areaTopImage {font-size: ; color: ;background-image:
url(/Sitewizards/Sitecore Express/Images/topbar/1/cust_topbar01.gif)}
div.MssNavigation div.navButtons input {background-image:
url(/Sitewizards/Sitecore Express/Images/buttons/1/cust_btn03.gif); width: 104;
height: 17; font-weight: bold; font-size: 10px; color: white; font-family:
Verdana; cursor: hand; BORDER: 0px; background-color: Transparent;}
img.MssListArrow {background-image: url(/Sitewizards/Sitecore
Express/Images/buttons/1/cust_menupil03.gif); }
div.areaBreadcrumb {background-image: url(/Sitewizards/Sitecore
Express/Images/bars/1/cust_menubar01.gif)}
div.areaGlobal {}
div.areaSublayout {background-color: rgb(226,231,232)}
div.areaFooter {background-image: url(/Sitewizards/Sitecore
Express/Images/bars/1/cust_menubar01.gif)}
table.sublayoutTable td.leftcol {background-color: rgb(190,205,209)}
table.sublayoutTable td.centercol {background-color: rgb(190,205,209)}
table.sublayoutTable td.spacecol {}
table.sublayoutTable td.columnDivider {}
table.sublayoutTable td.frontcol {color: rgb(255,255,255);background-color:
rgb(104,131,138)}
table.sublayoutTable td.rightcol {background-color: rgb(190,205,209)}
table.sublayoutTable td.rightcol div.spotHead {background-image:
url(/Sitewizards/Sitecore Express/Images/bars/1/cust_menubar01.gif)}
```

```
div.menuArrow {background-image: url(/Sitewizards/Sitecore  
Express/Images/buttons/1/cust_menupil03.gif); }  
div.menuArrowOn {background-image: url(/Sitewizards/Sitecore  
Express/Images/buttons/1/cust_menupilselected03.gif); }  
.SCXWizardFrontspot {background-image: url(/Sitewizards/Sitecore  
Express/Images/images/1/spotA_01.jpg)}  
.SCXWizardPage2Spot {background-image: url(/Sitewizards/Sitecore  
Express/Images/images/1/spotB_01.jpg)}  
.SCXWizardFrontImage {background-image: url(/Sitewizards/Sitecore  
Express/Images/images/1/Front_01.jpg);width: 300; height: 100px}  
.SCXWizardFrontImage img {width: 300; height: 104px;}
```

Chapter 8

Appendix C: Creating a new site type manually

This Appendix deals with the process of creating a new site type manually. If an existing site type layout exists that resembles the new design to be implemented, it might be more feasible to duplicate the site type and then make any necessary changes.

8.1 Required components

The following components must be duplicated as part of the site creation process:

- Templates
- Masters
- Layouts
- Sublayouts
- renderings
- Administration menu, see page **Error! Bookmark not defined.**
- Site template, see page **Error! Bookmark not defined.**
- Site type and make the changes to the settings and security setup, see page **Error! Bookmark not defined.**

All the existing elements of a site type should be prefixed with the site type name.

As an example the template

MSS.DocRef

Should be renamed to

SiteTypeName.DocRef

Remember to copy the files for layouts, sublayouts and renderings or make them from scratch.

One of the most difficult areas when dealing with duplicating Sitecore components is updating the Sitecore information on each of the components.

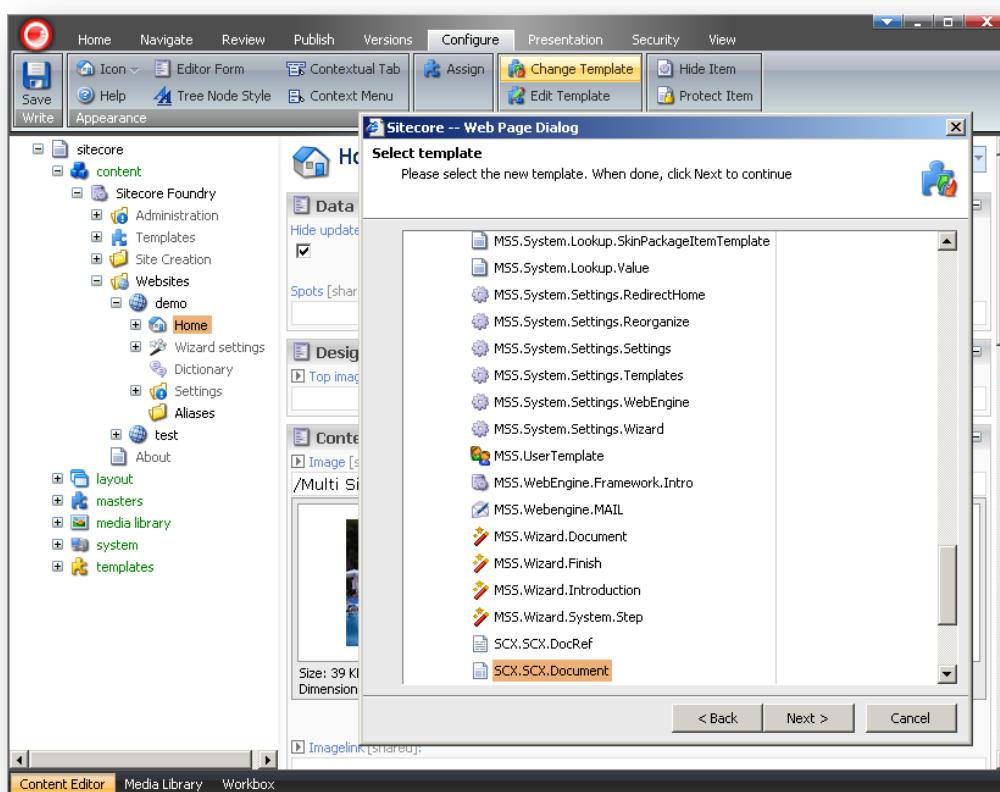
The following settings:

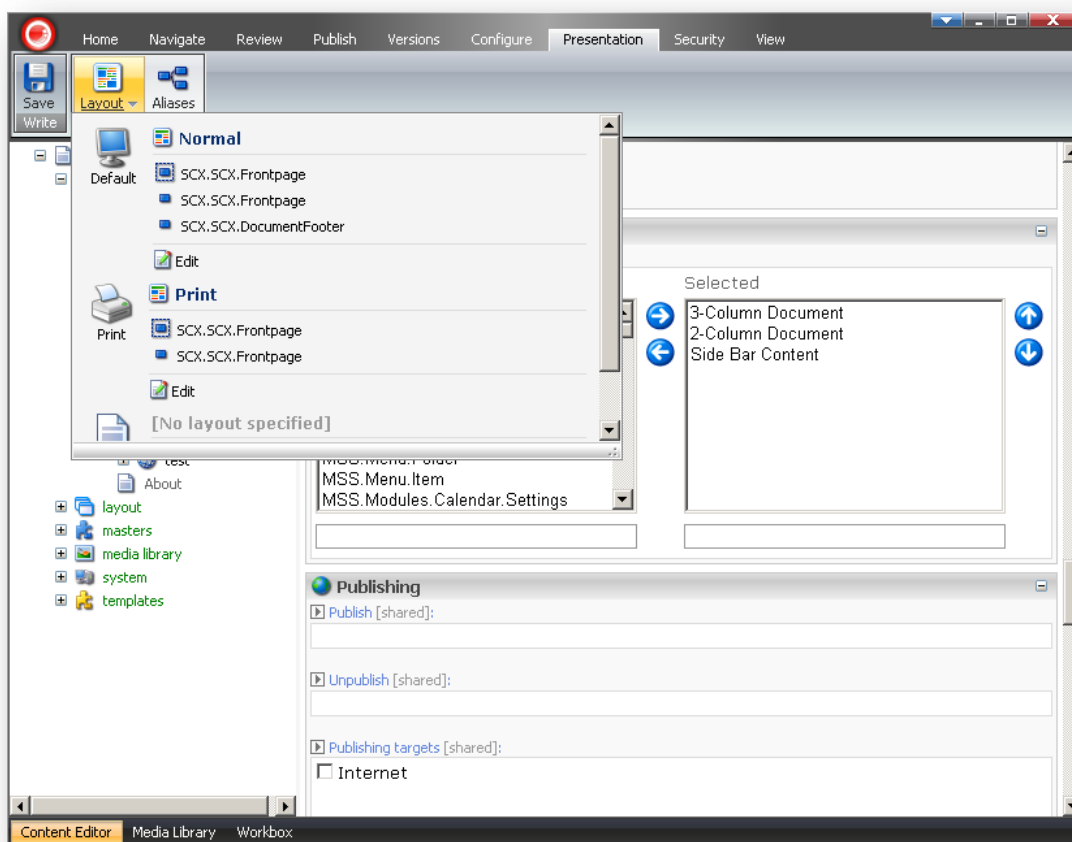
- Templates
- Masters and Layouts

- Sublayouts
- Rendering

Should be replaced with the new duplicated settings on the Sitecore items in the following places:

- The new duplicated Administration menu
- The new duplicated Site template
- The new duplicated master items
- The new duplicated templates items.





In addition the following needs to be checked:

- The new duplicated renderings and sublayouts have to replace the old ones which are explicitly specified on the duplicated layouts and sublayouts.
- Binding to Code-behind files has to be checked and updated.
- CSS links and image files specified on layouts, sublayouts and renderings have to be replaced.

Use:

- The dictionary for all language dependent texts shown on the site that is not stored on the documents. For that purpose use the xsl-helper extension or the label xsl-control. On layouts and sublayouts use the custom controls: text, label, button etc.
- The classes SitePage and SiteControl to make any new layouts and sublayouts inherit from these.
- Xsl helper functions when using xsl renderings and you need some site information, e.g. the home node.

8.1.1 Making A New Site Type From Scratch

To make a new site type Sitecore strongly recommends that you duplicate an existing site type and then make necessary changes. However, it is possible to create one from

scratch. You need to start by breaking the site up into individual and logical components. The following are standard Sitecore guidelines for building a site:

- Create layouts. Remember to use the layout groups functionality.
- Create sublayouts.
- Create renderings.
- Place the sublayouts and renderings on the layouts and sublayouts.
- Create templates.
- Create masters based on the templates and set the available masters on them.
- Setup the layout and rendering information on the templates and masters.
- Create a site template. Remember the content item “global” with its global items used by built-in modules. It might be easier to copy from an existing site template and change the Sitecore information such as masters, layout and renderings. Also remember the content item “Wizard settings” with the wizard related settings documents.
- Create a new administration menu. Copy it from an existing one. Then make the changes to masters, layout and rendering information on the menu items if needed.
- Create the site type including the system and module settings and configure the site type properties, see section “The wizard ”, page 71.
- If needed create a new wizard configuration and skinpackage, see section “The wizard ”, page 71 .
- If necessary create a new implementation of the ICSSFactory for the wizard to use.

Important: When creating the new Sitecore components, remember to prefix the names with the new site type name, in order to isolate the components from the existing ones.

Chapter 9

Appendix D: Standard website sizes

This appendix gives the standard sizes used for columns and spacing in Sitecore Foundry.

9.1 Standard Sizes

The following are the standard sizes used in Sitecore Foundry for various parts of the web site display.

Website width: 760 pixels

Left column width: 201px

Center column width:

Right column width: 201px

Center image width: 340px

Space between columns: 6 px

Spot image width: 181px

Center image width (two column layout): 540

Top image: 760 x 74px

Identity image: 760 x 66px