Customizing and Branding Web Content Management-Enabled SharePoint Sites (Part 2 of 3): Extending WCM

**Summary:** This article explains the structure of master pages, the steps to create a custom master page, the underlying mechanisms of navigation in WCM-enabled sites, the creation of custom field controls, and how to customize the Page Editing toolbar. (19 printed pages)

Patrick Tisseghem, [U2U](http://www.u2u.net/)

October 2006

**Applies to:** Microsoft Office SharePoint Server 2007

**Contents**

* [Introduction to Extending WCM](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_Introduction)
* [Understanding the Structure of Master Pages](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_UnderstandingMP)
* [Understanding the Default Master Pages](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_UnderstandingDefaultMP)
* [Building a Custom Master Page](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_BuildingCustomMP)
* [Adding Cascading Style Sheets](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_AddingCSS)
* [Customizing the Navigation](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_CustomizingNavigation)
* [Creating Custom Field Controls](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_CreatingCustomFieldControls)
* [Customizing the Edit Mode Experience](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_CustomizingEditMode)
* [Conclusion](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_Conclusion)
* [About the Author](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_AboutAuthor)
* [Additional Resources](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_AdditionalResources)

**Introduction to Extending WCM**

Although Microsoft Office SharePoint Server 2007 provides a rich site definition for delivering an Internet presence and Web content management (WCM) infrastructure, most organizations discover the need to customize and extend the default capabilities. In Office SharePoint Server 2007, all of the WCM features and the underlying artifacts are highly extensible.

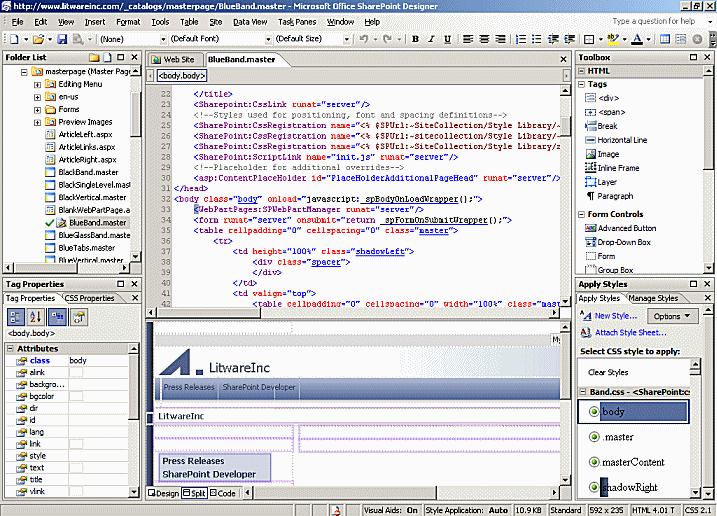
**Understanding the Structure of Master Pages**

Each page that a visitor sees in the browser when navigating to an Office SharePoint Server 2007 site is a combination of a master page and a published page based on a particular page layout (a template). The first article in this series ([Customizing and Branding Web Content Management-Enabled SharePoint Sites (Part 1 of 3): Understanding Web Content Management and the Default Features](http://msdn2.microsoft.com/en-us/library/aa830818.aspx)) provides a general overview of master pages and page layouts. Let's go further in this article.

The site master page is stored at the level of the site collection in the master page gallery. Every site has an association with a master page. The pages published in the site adopt the appearance and expose the navigation controls defined by the master page.

Office SharePoint Server 2007 is an excellent product to use to explore the various elements that make up a master page. You can use Microsoft Office SharePoint Designer 2007 to open one of your sites that uses the WCM features. For this article, I use the Publishing Portal as an example. In the folder list, expand the **\_catalogs** node and the **masterpage** node. Double-click **BlueBand.master** and confirm that you want to check out the file. Switch to **Split** mode so that you have the designer and the code editor visible, as shown in Figure 1.

**Figure 1. BlueBand.master open in SharePoint Designer**



**SharePoint Framework Controls**

Controls whose names are prefixed with "SPSWC", "SharePoint", or "WebPartPages" are the Web controls or Web Parts that are available via Microsoft.SharePoint.Portal.dll or Microsoft.SharePoint.dll. They are part of Microsoft Windows SharePoint Services or the additions provided by Office SharePoint Server 2007. You find these controls in all sites, even those that do not have the WCM features enabled.

One example is the control for delivering search in the page. The navigation controls embedded in the master page are examples of controls that come with Windows SharePoint Services (Figure 2).

The published pages can contain Web Parts. The master page contains the **SPWebPartManager** control, which coordinates the Web Parts embedded in these pages.

**Figure 2. Web controls and navigation controls in the master page**

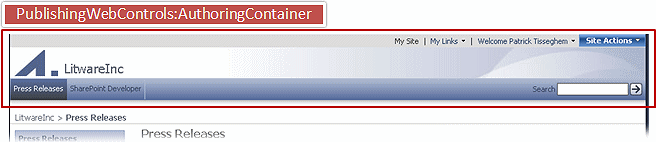


**Publishing Controls**

Controls whose names are prefixed with "PublishingWebControls" and "PublishingNavigation" are the Web controls that deliver the WCM features for the site.

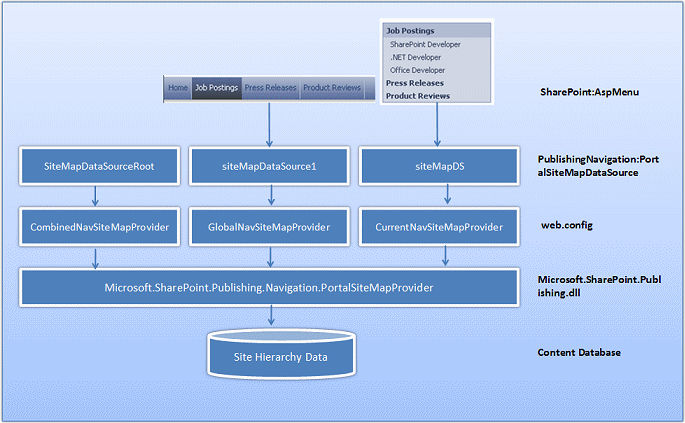
The **PublishingWebControls:AuthoringContainer** control, which is provided by Microsoft.SharePoint.Publishing.dll, creates the top part of the page (Figure 3). It is the container for the **Welcome** user control, the **Site Actions** menu, and the Microsoft ASP.NET user control that renders the authoring console.

**Figure 3. Authoring Container in the master page**



The publishing controls that are identified by the prefix "PublishingNavigation" (and also provided by Microsoft.SharePoint.Publishing.dll) are Web controls not directly visible on the page. The **PortalSiteMapDataSource** Web control adds a layer on top of the Site Map Provider model that is part of ASP.NET 2.0. The ASP.NET 2.0 Site Map Provider model gives developers a pluggable navigation model in which the user interface (UI) and the data source are separated. Windows SharePoint Services 3.0 and Office SharePoint Server 2007 add features, of which the most important is security trimming. Figure 4 shows an overview of the interaction that occurs among the various components that drive the navigation in your SharePoint sites.

**Figure 4. Components of the Site Map Provider model**



Three **PortalSiteMapDataSource** controls are embedded. Each maps to a definition for a provider within the web.config file of the site. The **SiteMapDataSourceRoot** control uses the **CombinedNavSiteMapProvider** control and is connected to the breadcrumb controls. As its name suggests, it combines the two other **PortalSiteMapDataSource** controls into one. The **siteMapDataSource1** control provides the data for the top navigation bar, and the **siteMapDS** control provides the data for the vertical navigation control.

**ASP.NET User Controls**

In addition to the ASP.NET server controls, you also find ASP.NET user controls (.ascx files) embedded in the master page:

* **Welcome.ascx** is displayed at the top of your page. You will find this control on all of your SharePoint sites (even those that do not use WCM).
* **VariationsLabelMenu.ascx** is displayed when a page is part of a variation hierarchy. The user can quickly jump to the same page in another variation by using this control (Figure 5).
* **PublishingConsole.ascx** controls all the steps within the publishing cycle of a page (Figure 6). I discuss this control in more detail later, in the section titled [Customizing the Edit Mode Experience](http://msdn2.microsoft.com/en-us/library/aa830815.aspx#Office2007SSBrandingWCMPart2_CustomizingEditMode).
* **PublishingActionMenu.ascx** is the user control that delivers the **Site Actions** menu, which is different from the user control you use on sites that do not have the WCM features enabled.

The last set of controls in the master page is the ASP.NET **ContentPlaceHolder** controls. The most important one is the **PlaceHolderMain** control, which indicates where Windows SharePoint Services inserts the actual content page.

**Understanding the Default Master Pages**

Different master pages are available with Windows SharePoint Services 3.0 and Office SharePoint Server 2007. We have looked in detail at BlueBand.master, but you can use others as starters.

To explore the differences with the others available, click **Site Settings** and then, in the **Look and Feel** section, click **Master Page**. On the Site Master Page Settings page you can change the site master page to any of the master pages stored in the master page gallery:

* **Default.master** is used for the Windows SharePoint Services team sites. You can use the same design for your site.

You might have the impression that all of the controls discussed are present in default.master, but if you open default.master in Office SharePoint Designer 2007, you notice that the page does not contain the publishing controls directly. Each control is inserted by a **DelegateControl** control. For example, the following is the definition of the Publishing Console in the master page.

Xml

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl04xml');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl04xml');)

<SharePoint:DelegateControl runat="server" ControlId="PublishingConsole" PrefixHtml="&lt;tr&gt;&lt;td colspan=&quot;4&quot;&gt;" SuffixHtml="&lt;/td&gt;&lt;/tr&gt;"> </SharePoint:DelegateControl>

Delegate controls give you a powerful yet flexible approach to determine within your Feature which control to display. A Feature can, for example, define two search controls in the manifest file, each of them sequenced with a number. At run time, the delegate control on the page selects the search control that has the lowest sequence number and displays the control to the user. Delegate controls are heavily used by Office SharePoint Server 2007 to "light-up" or enable features in areas such as navigation and search.

You should avoid using this master page as the starter for your own custom master page if you want to provide a clean WCM infrastructure.

* **BlueBand.master, BlackBand.master, and BlueGlassBand.master** belong to the same family and display both a top navigation bar and a left navigation bar. They are good candidates for your sites and good starting points for your own customizations.
* **BlueVertical.master and BlackVertical.master** are master pages that are tuned for Internet sites, with a good representation of the WCM-related controls. They offer the same look and feel as the three master pages discussed in the previous bullet but do not contain the top navigation bar.
* **BlueTabs.master** is also a valid alternative for the default master page. The difference is the tabbed representation of the top navigation bar.
* **OrangeSingleLevel.master and BlackSingleLevel.master** are variations on the BlueBand master pages but with different color combinations.

**Building a Custom Master Page**

You have various options when you want to create your own custom master page. You can make a copy of an existing master page and rearrange the controls, modifying the layout and the look and feel. Or you can start from a blank page and create your own.

Open your site in SharePoint Designer. On the **File** menu, click **New Page**. The **New** dialog box offers you many choices. Choosing **Master Page** gives you a blank master page. Be aware that starting with a blank page involves some work and might create problems because a blank file does not have all the content placeholders necessary to work with the page layouts and content pages that are included with Office SharePoint Server 2007. The following code sample represents the simplest possible master page.

Html

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl05html');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl05html');)

<%-- Identifies this page as a .master page written in C# and registers tag prefixes, namespaces, assemblies and controls. --%> **<%@ Master language="C#" %>** <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> **<%@ Import Namespace="Microsoft.SharePoint" %>** <%@ Register Tagprefix="SPSWC" Namespace="Microsoft.SharePoint.Portal.WebControls" Assembly="Microsoft.SharePoint.Portal, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls" Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages" Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@ Register Tagprefix="PublishingWebControls" Namespace="Microsoft.SharePoint.Publishing.WebControls" Assembly="Microsoft.SharePoint.Publishing, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@ Register Tagprefix="PublishingNavigation" Namespace="Microsoft.SharePoint.Publishing.Navigation" Assembly="Microsoft.SharePoint.Publishing, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@ Register TagPrefix="wssuc" TagName="Welcome" src="~/\_controltemplates/Welcome.ascx" %> <%@ Register TagPrefix="wssuc" TagName="DesignModeConsole" src="~/\_controltemplates/DesignModeConsole.ascx" %> <%@ Register TagPrefix="PublishingVariations" TagName="VariationsLabelMenu" src="~/\_controltemplates/VariationsLabelMenu.ascx" %> <%@ Register Tagprefix="PublishingConsole" TagName="Console" src="~/\_controltemplates/PublishingConsole.ascx" %> <%@ Register TagPrefix="PublishingSiteAction" TagName="SiteActionMenu" src="~/\_controltemplates/PublishingActionMenu.ascx" %> <%-- Uses the Microsoft Office namespace and schema. --%> <html> <WebPartPages:SPWebPartManager runat="server"/> <SharePoint:RobotsMetaTag runat="server"/> <%-- The head section includes a content placeholder for the page title and links to CSS and JavaScript files that run on the server. --%> <head runat="server"> <**asp:ContentPlaceHolder runat="server" id="head">** <title> <**asp:ContentPlaceHolder id="PlaceHolderPageTitle" runat="server" />** </title> </**asp:ContentPlaceHolder>** <Sharepoint:CssLink runat="server"/> <**asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server" />** </head> <%-- When loading the body of the .master page, MOSS 2007 also loads the SpBodyOnLoadWrapper class. This class handles .js calls for the master page. --%> <body onload="javascript:\_spBodyOnLoadWrapper();"> <%-- The SPWebPartManager manages all of the Web part controls, functionality, and events that occur on a Web page. --%> <form runat="server" onsubmit="return \_spFormOnSubmitWrapper();"> <wssuc:Welcome id="explitLogout" runat="server"/> <PublishingSiteAction:SiteActionMenu runat="server"/> <PublishingWebControls:AuthoringContainer id="authoringcontrols" runat="server"> <PublishingConsole:Console runat="server" /> </PublishingWebControls:AuthoringContainer> <%-- The PlaceHolderMain content placeholder defines where the page content should go for all the content from the page layout. The page layout can overwrite any content placeholder from the master page. Example: The PlaceHolderLeftNavBar can overwrite the left navigation bar. --%> <**asp:ContentPlaceHolder id="PlaceHolderMain" runat="server" />** <**asp:Panel visible="false" runat="server">** <%-- These ContentPlaceHolders are necessary only to ensure all default MOSS 2007 pages render with this master page. If the system master page is set to any default master pages, the only content placeholders required are those that are being overridden by your page layouts. --%> <**asp:ContentPlaceHolder id="PlaceHolderSearchArea" runat="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderTitleBreadcrumb runat="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderPageTitleInTitleArea" runat="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderLeftNavBar" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderPageImage" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderBodyLeftBorder" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderNavSpacer" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderTitleLeftBorder" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderTitleAreaSeparator" runat="server"/>** <**asp:ContentPlaceHolder ID="PlaceHolderMiniConsole" runat="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderCalendarNavigator" runat ="server" />** <**asp:ContentPlaceHolder id="PlaceHolderLeftActions" runat ="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderPageDescription" runat ="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" runat ="server"/>** <**asp:ContentPlaceHolder id="PlaceHolderTitleAreaClass" runat ="server"/>** </**asp:Panel>** </form> </body> </html>

This very minimal master page displays only the content of the published page that is rendered inside the **PlaceHolderMain** control and the title of the published page. The best approach to finish this master page is to open the BlueBand.master page and embed in your own master page the various controls you want to make available:

* The PortalSiteMapDataSource controls and the ASPMenu controls to handle the navigation.
* The various ASP.NET user controls to display the **Site Actions** menu and the publishing console.

**Adding Cascading Style Sheets**

Cascading style sheets (CSS) are used to define the look and feel of your pages. Office SharePoint Server 2007 delivers several CSS files that contain the various site definitions. For a WCM-enabled site, they are available through the Style Library at the site-collection level. You can review the contents by opening the **Site Content and Structure** page from the **Site Actions** menu. The Style Library contains a subfolder for each locale you want your site to support. You can add localized CSS files in the appropriate folders.

A candidate to use in your custom master page is Band.css. However, you can create your own CSS file and upload it to the Style Library. To link your master page with one of the CSS files in the Style Library, use the **Microsoft.SharePoint.WebControls.CSSRegistration** control.

Html

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl06html');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl06html');)

<SharePoint:CssRegistration name="<% $SPUrl:~sitecollection/Style Library/~language/Core Styles/Band.css %>" runat="server"/> <SharePoint:CssRegistration name="<% $SPUrl:~sitecollection/Style Library/~language/Core Styles/controls.css %>" runat="server"/>

When working in SharePoint Designer, you see the **Apply Styles** pane filled with all of the CSS classes you can associate with the various elements and controls in your page.

The administrator of a site can easily define an alternative CSS URL by using the Site Master Page Settings page. This option enables you to define new styles (and override existing ones) without having to edit the master page directly.

|  |
| --- |
| **NoteNote:** |
| You can find more documentation about customizing styles in [How to: Customize Styles](http://msdn2.microsoft.com/en-us/library/ms551040.aspx). |

**Customizing the Navigation**

Three artifacts provide the user with navigation in your pages:

* The **site map providers,** which are responsible for obtaining the navigation data stored in the content database
* **SiteMapDataSource controls,** which provide an abstraction between the navigation control itself and the underlying provider of the navigation data
* **UI navigation controls,** such as the breadcrumb and the menu displayed in the top navigation bar

You can extend and customize at any of these three levels.

**Site Map Providers**

The role of a site map provider in SharePoint Products and Technologies is the same as for ASP.NET Web applications. Office SharePoint Server 2007 uses the provider to access the navigation data stored in the content database and represents this navigation data as a tree of nodes of type **SiteMapNode**. The various navigation controls use four site map providers.

**SPSiteMapProvider**, **SPContentMapProvider**, and **SPNavigationProvider** are part of the Windows SharePoint Services classes; **PortalSiteMapProvider** is part of the Office SharePoint Server 2007 object model. Other providers are available. You can find the registration of all the providers in the web.config file that is associated with the Microsoft Internet Information Services (IIS) Web application that hosts your SharePoint sites.

In the majority of cases, these providers are sufficient to deliver the necessary data for your navigation controls. You can, however, create your own providers and define a custom way of mapping the data with a tree of nodes in memory so that the data mapping can be used by your own navigation controls. You can create your custom site map provider by creating a class that inherits either from the base **SiteMapProvider** class that is part of ASP.NET or from one of the SharePoint provider classes. Best practice is the latter approach; for example, you can subclass the **PortalSiteMapProvider**. Doing so ensures that you have the proper caching and security trimming. An example of a custom site map provider is discussed in the Office SharePoint Server 2007 SDK.

**SiteMapDataSource Controls**

**SiteMapDataSource** controls are used by some of the navigation controls such as the **TreeView** control and the **Menu** control. They function as an abstraction layer between the navigation control and the **SiteMapProvider** classes. Their primary role is to represent a view on the navigation data and allow for customization via the various properties or attributes. One use of the **SiteMapDataSource** controls is to specify where in the tree of nodes the navigation controls should start and how many levels should be displayed.

The control I use in my WCM-enabled sites is the **PortalSiteMapDataSource** control, which is part of the **System.SharePoint.Publishing.Navigation** namespace.

**UI Navigation Controls**

The most-used navigation control is **Microsoft.SharePoint.WebControls.AspMenu**. This control offers menu-based navigation. You can set numerous properties either in the HTML of the master page or programmatically via the object model. Refer to the many ASP.NET 2.0 resources for information about creating your own custom navigation controls.

**Creating Custom Field Controls**

As I discussed in the first article of this series ([Customizing and Branding Web Content Management-Enabled SharePoint Sites (Part 1 of 3): Understanding Web Content Management and the Default Features](http://msdn2.microsoft.com/en-us/library/aa830818.aspx)), you use field controls to deliver or capture the content in a publishing page. The field controls are tightly coupled with the type of field you define at the level of the content type that is associated with your page layout. For example, if you add a column named "Job Title" of type **Single Line of Text**, you see a text box field control when you are in editing mode and plain text when you are in rendering mode. Microsoft.SharePoint.Publishing.dll adds several new field types you can use when designing page layouts.

But what if all of this is not enough? The artifacts that compose the page model are very extensible. We reviewed the customization of the master pages, but you can also create extra field controls to capture and display data in a very specific way—one that is not offered as part of Windows SharePoint Services or Office SharePoint Server 2007.

**Creating an Address Field Control**

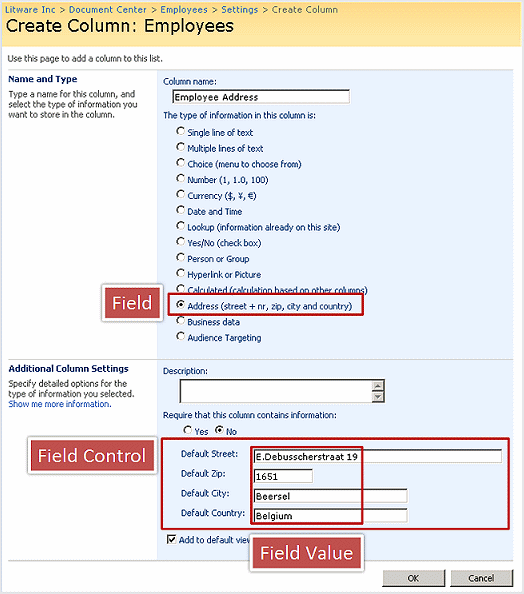
Let me walk you through the process of creating an address field type that we can use as a normal field type for our SharePoint lists and libraries and as a field control in page layouts. You can create a field control that is used only in page layouts by inheriting from the field controls that are part of Microsoft.SharePoint.Publishing.dll.

You must handle different usage modes in your custom field control. These modes include not only the display mode and the edit mode, but also the design mode. The latter is the mode in which the field control will be if you drop it on the designer in SharePoint Designer.

Your custom field must appear in the list of field types when creating a new column for a list, a document library, or a content type. The selection of the custom field renders the part in the options section you use to set the settings of the custom field and specify any default values (Figure 7).

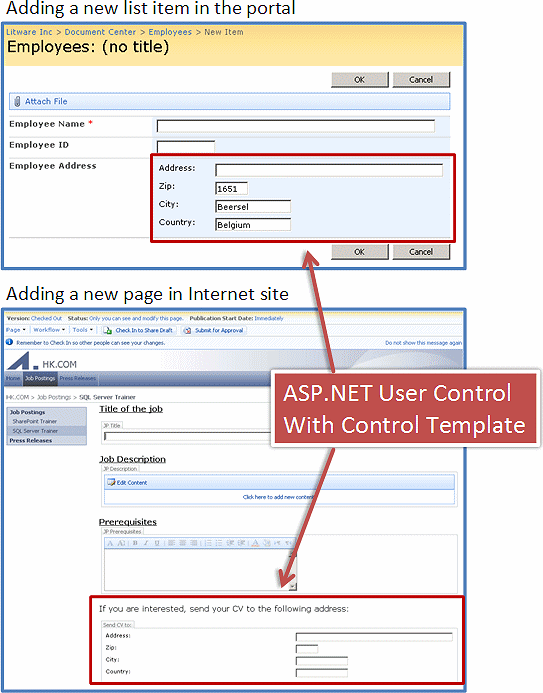
The custom address field is associated with an ASP.NET server control rendering of the default data, as it is defined by using a custom field value.

**Figure 7. Adding a column based on the custom address field type**



Your custom field also must have the capability of rendering a UI for capturing data when an item or page is created or when the user enters edit mode for the item or page. You can create an ASP.NET user control with code-behind logic, such as custom processing of the data or use of external data sources. Figure 8 shows the custom address field in edit mode in a portal site scenario, where it is used as a column of a list, and in an Internet scenario, where it is used as a field control for a specific page layout.

**Figure 8. Address field UI in a portal site and in an Internet site**



Collaborative Application Markup Language (CAML) and the **RenderPattern** element make the address data available in rendering mode. This enables users to see the address in the SharePoint list or on the published page.

Let us now look at how all of this is accomplished. (References to Microsoft.SharePoint.dll and System.Web.dll must appear at the top of the code, because the code is compiled into one Microsoft .NET Framework class library. Ensure you have the appropriate **using** statements at the top if you want to follow along with the code.)

The first class defines the value to capture. In our example, the address consists of four fields: the address (or street), the postal code, the city, and the country. Because an address is not a single value, the class derives from the **SPFieldMultiColumnValue** class. This class, named **FieldAddressValue**, stores one property per item in the structured value. The class itself resides in the **Litware.CustomFields** namespace.

C#

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl09CSharp');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl09CSharp');)

namespace Litware.CustomFields { public class FieldAddressValue : SPFieldMultiColumnValue { private const int numberOfFields = 4; public FieldAddressValue() : base(numberOfFields) { } public FieldAddressValue(string value) : base(value) { } public string Address { get { return this[0]; } set { this[0] = value; } } public string Zip { get { return this[1]; } set { this[1] = value; } } public string City { get { return this[2]; } set { this[2] = value; } } public string Country { get { return this[3]; } set { this[3] = value; } } } }

Next, create the custom field itself in the same namespace. The address field is represented by a class named **FieldAddress** that inherits from the **SPFieldMultiColumn** class. An important method to override is **FieldRenderingControl**. This method defines the ASP.NET server control to display in the options section when the column is created. Override **GetFieldValue** to return the serialized value stored in the database as type **FieldAddressValue**, breaking the value into the four pieces rendered by the control.

C#

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl10CSharp');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl10CSharp');)

public class FieldAddress : SPFieldMultiColumn { public FieldAddress(SPFieldCollection fields, string fieldName) : base(fields, fieldName) { } public FieldAddress(SPFieldCollection fields, string typeName, string displayName) : base(fields, typeName, displayName) { } public override BaseFieldControl FieldRenderingControl { get { BaseFieldControl fldControl = new AddressFieldControl(); fldControl.FieldName = InternalName; return fldControl; } } public override object GetFieldValue(string value) { if (string.IsNullOrEmpty(value)) return null; return new FieldAddressValue(value); } }

Using the same project, create a new ASP.NET server control named **AddressFieldControl**. You can derive from one of the existing ASP.NET server controls. Deriving from the **BaseFieldControl** control is another option. To make a distinction from the previous types of classes, a second namespace, **Litware.WebControls**, is used.

Notice that the ASP.NET server control is not really rendering the UI itself. It is using a control template that is defined in an ASP.NET user control. This is a new technique in ASP.NET to separate control data from control presentation. The UI for the server control is defined in a template embedded in an .ascx file. The ID of that template is connected to the **AddressFieldControl** control via an override of the **DefaultTemplateName** property. The **Value** property is overridden because the value of the address field is a composite value. Notice that **CreateChildControls** does not really create the child controls. That is the task of the ASP.NET user control. In **CreateChildControls** you connect to the UI elements—the various text boxes—via the **FindControl** method of the **TemplateContainer** class. At the end of the class, default values are retrieved from the XML file that accompanies the custom field. The three properties defined in the XML file are retrieved via the static **GetCustomProperty** method of the **Field** class.

C#

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl11CSharp');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl11CSharp');)

namespace Litware.WebControls { public class AddressFieldControl : BaseFieldControl { protected TextBox addressBox; protected TextBox zipBox; protected TextBox cityBox; protected TextBox countryBox; protected override string DefaultTemplateName { get { return "AddressFieldRendering"; } } public override object Value { get { EnsureChildControls(); FieldAddressValue fieldValue = new FieldAddressValue(); fieldValue.Address = addressBox.Text.Trim(); fieldValue.Zip = zipBox.Text.Trim(); fieldValue.City = cityBox.Text.Trim(); fieldValue.Country = countryBox.Text.Trim(); return fieldValue; } set { EnsureChildControls(); FieldAddressValue fieldValue = (FieldAddressValue)value; addressBox.Text = fieldValue.Address; zipBox.Text = fieldValue.Zip; cityBox.Text = fieldValue.City; countryBox.Text = fieldValue.Country; } } public override void Focus() { EnsureChildControls(); addressBox.Focus(); } protected override void CreateChildControls() { if (Field == null) return; base.CreateChildControls(); if (ControlMode == SPControlMode.Display) return; addressBox = (TextBox)TemplateContainer.FindControl("addressBox"); if (addressBox == null) throw new ArgumentException("Corrupted AddressFieldRendering template - missing addressBox."); addressBox.TabIndex = TabIndex; addressBox.CssClass = CssClass; addressBox.ToolTip = Field.Title + " Address"; zipBox = (TextBox)TemplateContainer.FindControl("zipBox"); if (zipBox == null) throw new ArgumentException("Corrupted AddressFieldRendering template - missing zipBox."); zipBox.TabIndex = TabIndex; zipBox.CssClass = CssClass; zipBox.ToolTip = Field.Title + " zipcode"; cityBox = (TextBox)TemplateContainer.FindControl("cityBox"); if (cityBox == null) throw new ArgumentException("Corrupted AddressFieldRendering template - missing cityBox."); cityBox.TabIndex = TabIndex; cityBox.CssClass = CssClass; cityBox.ToolTip = Field.Title + " City"; countryBox = (TextBox)TemplateContainer.FindControl("countryBox"); if (countryBox == null) throw new ArgumentException("Corrupted AddressFieldRendering template - missing countryBox."); countryBox.TabIndex = TabIndex; countryBox.CssClass = CssClass; countryBox.ToolTip = Field.Title + " Country"; if (ControlMode == SPControlMode.New) { zipBox.Text = Field.GetCustomProperty("DefaultZip").ToString(); cityBox.Text = Field.GetCustomProperty("DefaultCity").ToString(); countryBox.Text = Field.GetCustomProperty("DefaultCountry").ToString(); } } } }

As I mentioned earlier, an ASP.NET user control handles the presentation of the four controls that compose the user interface when interacting in edit mode with the custom address field. A **RenderingTemplate** control with the ID **AddressFieldRendering** displays the controls in a simple HTML table.

Xml

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl12xml');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl12xml');)

**<%@ Control Language="C#" Debug="true" %>** <%@Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %> <%@Register TagPrefix="SharePoint" Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" namespace="Microsoft.SharePoint.WebControls"%> <SharePoint:RenderingTemplate ID="AddressFieldRendering" runat="server"> <**Template>** <table class="ms-form"> <tr><td>Address:</td><td><**asp:TextBox ID="addressBox" MaxLength="255" Size="50" runat="server"/></td></tr>** <tr><td>Zip:</td><td><**asp:TextBox ID="zipBox" MaxLength="10" Size="5" runat="server"/></td></tr>** <tr><td>City:</td><td><**asp:TextBox ID="cityBox" MaxLength="50" Size="15" runat="server"/></td></tr>** <tr><td>Country:</td><td><**asp:TextBox ID="countryBox" MaxLength="50" Size="15" runat="server"/></td></tr>** </table> </**Template>** </SharePoint:RenderingTemplate>

The final part of the custom field type is the creation of the field type definition file. This is an XML file named fldtypes\*.xml—in this example, fldtypes\_Litware.xml. This field type definition file has three parts. The first part is the metadata for the custom field type. The second part is three properties made available in a declarative way so that they can be easily modified. (As discussed, the ASP.NET server control is able to access the values at run time by using the **GetCustomProperty** method of the **Field** class.) The third part is the CAML definition that handles the presentation of the values of the address field in a rendering mode. This example is fairly simple, but CAML is a rich markup language for the representation of this type of data; you can imagine more complex kinds of rendering.

Xml

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl13xml');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl13xml');)

<FieldTypes> <FieldType> <Field Name="TypeName">Address</Field> <Field Name="ParentType">MultiColumn</Field> <Field Name="TypeDisplayName">Address</Field> <Field Name="TypeShortDescription">Address (street + nr, zip, city and country)</Field> <Field Name="UserCreatable">TRUE</Field> <Field Name="ShowOnListAuthoringPages">TRUE</Field> <Field Name="ShowOnDocumentLibraryAuthoringPages">TRUE</Field> <Field Name="ShowOnSurveyAuthoringPages">TRUE</Field> <Field Name="ShowOnColumnTemplateAuthoringPages">TRUE</Field> <Field Name="FieldTypeClass">Litware.CustomFields.FieldAddress, AddressFieldType, Version=1.0.0.0, Culture=neutral, PublicKeyToken=96bdd15c5d632372</Field> <**PropertySchema>** <Fields> <Field Name="DefaultZip" DisplayName="Default Zip:" MaxLength="10" DisplaySize="10" Type="Text"> <Default>1651</Default> </Field> <Field Name="DefaultCity" DisplayName="Default City:" MaxLength="50" DisplaySize="30" Type="Text"> <Default>Beersel</Default> </Field> <Field Name="DefaultCountry" DisplayName="Default Country:" MaxLength="50" DisplaySize="30" Type="Text"> <Default>Belgium</Default> </Field> </Fields> </**PropertySchema>** <RenderPattern Name="DisplayPattern"> <Switch> <Expr><Column/></Expr> <Case Value=""> </Case> <Default> <Column SubColumnNumber="0" HTMLEncode="TRUE"/> <HTML><![CDATA[<BR>]]></HTML> <Column SubColumnNumber="1" HTMLEncode="TRUE"/> <HTML><![CDATA[&nbsp;-&nbsp;]]></HTML> <Column SubColumnNumber="2" HTMLEncode="TRUE"/> <HTML><![CDATA[<BR>]]></HTML> <Column SubColumnNumber="3" HTMLEncode="TRUE"/> </Default> </Switch> </RenderPattern> </FieldType> </FieldTypes>

**Deploying the Address Field Control**

The three parts of the custom field type—the .NET assembly, the .ascx file, and the .xml file with the field type definition—must be deployed in the following manner:

1. The .NET assembly must be deployed in the global assembly cache.
2. The fldtypes\_Litware.xml file must be copied to the path C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\XML. Several other field type definition XML files are already stored here.
3. The .ascx file must be copied to the path C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\CONTROLTEMPLATES. This folder is where you also find the various ASP.NET user controls discussed in the first article of this series.

For the custom field type to become a field control when dropped on the page layout file in SharePoint Designer, you must register the custom field in the SafeControls section of the web.config file.

Xml

[[http://msdn2.microsoft.com/msdn/Controls/CodeSnippet/en-us/copy_off.gif](javascript:CopyCode('ctl00_LibFrame_ctl14xml');)Copy Code](javascript:CopyCode('ctl00_LibFrame_ctl14xml');)

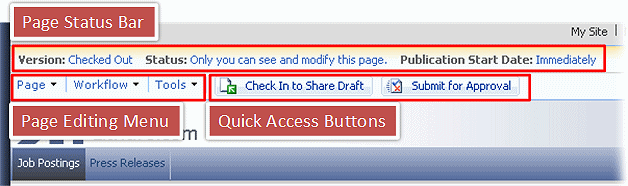
<SafeControl Assembly="AddressFieldType, Version=1.0.0.0, Culture=neutral, PublicKeyToken=96bdd15c5d632372" Namespace="Litware.WebControls" TypeName="\*" Safe="True" AllowRemoteDesigner="True" />

After the control is deployed, get it working by executing an **iisreset** command.

**Customizing the Edit Mode Experience**

The **Page Editing** toolbar plays an important role in the publishing cycle of the site pages. It offers content authors status information about the current page, menus for interacting with the page, and quick access buttons for the most popular actions given the page status and context (Figure 9).

**Figure 9. Page Editing toolbar**



As with most of the WCM-related features discussed so far, you can extend the **Page Editing** toolbar to fit the needs of your organization.

The underlying mechanism for the rendering of the Page Editing toolbar is an ASP.NET user control: **publishingconsole.ascx**. This user control is located in C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\CONTROLTEMPLATES among the many other ASP.NET user controls. The **Page Editing** toolbar is built with the concept of separating the data source and UI control; both the data source and the UI control can be modified independently.

The data sources are two XML files, EditingMenu.xml and QuickAccess.xml, located in C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\LAYOUTS\EditingMenu. For compatibility reasons, the contents of these two files should not be changed. To customize the console, you can edit two files: CustomEditingMenu.xml and CustomQuickAccess.xml, found in the Editing Menu folder under the site collection's master page gallery. The contents of these files can extend and override the settings contained in their parent files (EditingMenu.xml and QuickAccess.xml).

For documentation about the different elements you can use, refer to the XML schema named ConsoleConfigurationSchema.xsd, which you can find in C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\LAYOUTS\EditingMenu.

**Conclusion**

In this article (the second in a series of three), I focus on the various important features that make WCM happen and show the various scenarios you can try. I hope that this material, in combination with the Microsoft Windows SharePoint Services 3.0 SDK and the Microsoft Office SharePoint Server 2007 SDK, is sufficient to get you started. In the last article of the series (available in November), I concentrate on the configuration and management of WCM-enabled sites.

**About the Author**

Patrick Tisseghem (MVP, Microsoft Office SharePoint Portal Server) is managing partner at U2U, a Microsoft .NET training and services company based in Brussels, Belgium. He works most of his time as a contracted trainer for Microsoft delivering presentations, workshops, and seminars covering most of the Microsoft information worker products and technologies. Patrick is author of several articles and white papers about SharePoint Portal Server development aspects (published on MSDN) and is currently researching and writing a book about Microsoft Office SharePoint Server 2007 for Microsoft Press.