Lab 06: Custom Field Types & Controls

**Lab Time:** 60 minutes

**Lab Overview:** In this lab you will practice creating a custom field type and associated field control for use within a MOSS Publishing site. The field type will store four data elements as one item that make up a linked string of text with an associated icon. This new field type is called "Resource with Icon". The field control associated with this field type is used to provide a custom editing experience for content authors. Here's what the field will look like when you are finished with this lab:

|  |  |
| --- | --- |
| **Edit Mode**  C:\Development\Writing\Courses\WCM401\trunk\Student Download\Labs\08_FieldTypesControls\Figures\PreviewAuth.jpg | **Display Mode**  C:\Development\Writing\Courses\WCM401\trunk\Student Download\Labs\08_FieldTypesControls\Figures\PreviewDisplay.jpg |

Exercise 1: Creating a custom field type

In this exercise you will create a custom field type. At the end of this exercise the field type will not be complete and usable until it has an associated field control, but this lab is broken into two exercises for readability and future reference.

Similar to previous labs, since you have already created Visual Studio projects using the automated process of creating WSS solution packages, we have given you a project to get started with.

1. In Visual Studio, open the **Lab7** solution located in the following directory:

c:\Student\Labs\06\_FieldTypesControls\Lab\Lab7.sln

1. The first thing you need to do is to add a few references to the project. Add the following references to the project by right-clicking the **References** folder within the project in the **Solution Explorer** tool window:
   * **Microsoft.SharePoint** (hint: Component Name = Windows SharePoint Services)
   * **Microsoft.SharePoint.Publishing** (hint: Component Name = Microsoft Content Publishing and Management)
   * **System.Configuration**
   * **System.Web**
2. Now, with the necessary references, create a new class named **FieldResourceIcon.cs** in the root of the project. This class will be the "hub" of the custom field type... it is where you will later tell SharePoint to get all information about the field type.
3. This custom field type will store four pieces of data as a single data element. To do this, you will need to inherit from the **Microsoft.SharePoint.SPFieldMultiColumn** class. This class has two constructors that you will need to implement... add the following code to the **FieldResourceIcon.cs** class:

using System;

using System.Web.UI;

using Microsoft.SharePoint;

using Microsoft.SharePoint.WebControls;

namespace Lab7 {

public class FieldResourceIcon : SPFieldMultiColumn {

public FieldResourceIcon (SPFieldCollection fields, string fieldName)

: base(fields, fieldName) { }

public FieldResourceIcon (SPFieldCollection fields, string typeName, string displayName)

: base(fields, typeName, displayName) { }

}

}

1. With the field created, you now need to create a new class that will serialize/deserialize the value to/from the SharePoint content database. Create a new class named **FieldResourceIconValue.cs** in the root of the project and add the following code to it:

using System;

using Microsoft.SharePoint;

namespace Lab7 {

public class FieldResourceIconValue : SPFieldMultiColumnValue {

private const int NUM\_OF\_FIELDS = 4;

public FieldResourceIconValue ()

: base(NUM\_OF\_FIELDS) { }

public FieldResourceIconValue (string value)

: base(value) { }

}

}

1. SharePoint passes multi column fields back and forth using strings with the delimiter **;#**. Thankfully, the **Microsoft.SharePoint.SPFieldMultiColumnValue** class handles the parsing of this delimited string for you, as long as you provide the information where each data element is found in the string. Add the following public properties to the class:

public string Title {

get {

return this[0];

}

set {

this[0] = value;

}

}

public string Description {

get {

return this[1];

}

set {

this[1] = value;

}

}

public string UrlTarget {

get {

return this[2];

}

set {

this[2] = value;

}

}

public string Icon {

get {

return this[3];

}

set {

this[3] = value;

}

}

1. With the value class created, you now need to wire it up to the field type class. Add the following overridden method to the **FieldResourceIcon.cs** class:

public override object GetFieldValue (string value) {

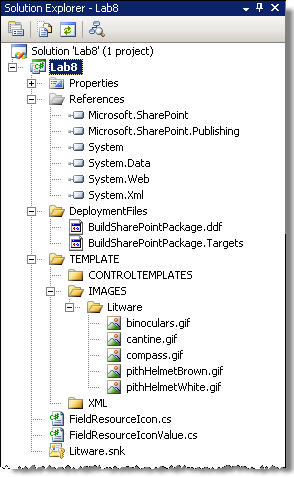
if (string.IsNullOrEmpty(value))

return null;

return new FieldResourceIconValue(value);

}

1. At this point the code part of the field type is complete. Now, you need to add the images to the project, create the field type definition and package everything up.
2. Copy the images in the **Resources** folder within this lab (**c:\Student\Labs\06\_FieldTypesControls\Resources\\*.gif**) into the **TEMPLATE\IMAGES\Litware** folder within the project, as shown in the following image:



1. Next, you need to create the field type definition file. This file will, when deployed, makes SharePoint aware of the new field type, and also specifies how the field should be rendered. Create a new **XML** file within the **TEMPLATE\XML** folder within the project named **fldtypes\_Litware.xml**. This file is broken down into three pieces: (1) the definition and meta information about the field, (2) any default settings for the field and (3) the rendering information. For this lab, you will only specify the meta and rendering information. Add the following metadata to the **fldtypes\_Litware.xml** file:

<?xml version="1.0" encoding="utf-8" ?>

<FieldTypes>

<FieldType>

<Field Name="TypeName">FieldResourceIcon</Field>

<Field Name="ParentType">MultiColumn</Field>

<Field Name="TypeDisplayName">Resource with Icon</Field>

<Field Name="TypeShortDescription">Resource with Icon</Field>

<Field Name="UserCreatable">TRUE</Field>

<Field Name="ShowInListCreate">TRUE</Field>

<Field Name="ShowInSurveyCreate">TRUE</Field>

<Field Name="ShowInDocumentLibraryCreate">TRUE</Field>

<Field Name="ShowInColumnTemplateCreate">TRUE</Field>

<Field Name="FieldTypeClass">Lab7.FieldResourceIcon, Lab7, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d4e5777b16a5749f</Field>

</FieldType>

</FieldTypes>

1. This metadata tells SharePoint a few things about the field type. First, it specifies the type name, the type it is derived from, and the name of the type. Then it specifies the different conditions under which the field can be created. Finally, it specifies the fully qualified five-part-name of the class (and containing assembly) where the field type can be found.
2. Now you need to specify the rendering information. There are many different rendering patters you can specify, including one to use for mobile devices, one to use when editing the value of the field, etc…. You will now implement a single rendering pattern that will be used when the field is being displayed (you do not need an edit pattern as it is addressed with a control in exercise 2). The pattern is defined using CAML... add the following CAML code into the **fldtypes\_Litware.xml** file just before the closing **</FieldType>** node:

<RenderPattern Name="DisplayPattern">

<Switch>

<Expr><Column /></Expr>

<Case Value="" />

<Default>

<HTML><![CDATA[<img src="/\_layouts/images/Litware/]]></HTML>

<Column SubColumnNumber="3" HTMLEncode="TRUE" />

<HTML><![CDATA[.gif" />&nbsp;<a href="]]></HTML>

<Column SubColumnNumber="2" HTMLEncode="TRUE" />

<HTML><![CDATA[" title="]]></HTML>

<Column SubColumnNumber="1" HTMLEncode="TRUE" />

<HTML><![CDATA[">]]></HTML>

<Column SubColumnNumber="0" HTMLEncode="TRUE" />

<HTML><![CDATA[</a>]]></HTML>

</Default>

</Switch>

</RenderPattern>

1. The next thing to do is to update the BuildSharePointPackage.ddf file to include all the necessary files in the WSS solution package. Add the following to the **BuildSharePointPackage.ddf** file between the comments:

DeploymentFiles\manifest.xml

bin\debug\Lab7.dll

.Set DestinationDir=IMAGES\Litware

TEMPLATE\IMAGES\Litware\binoculars.gif

TEMPLATE\IMAGES\Litware\cantine.gif

TEMPLATE\IMAGES\Litware\compass.gif

TEMPLATE\IMAGES\Litware\pithHelmetBrown.gif

TEMPLATE\IMAGES\Litware\pithHelmetWhite.gif

.Set DestinationDir=XML

TEMPLATE\XML\fldtypes\_Litware.xml

1. Finally, the last thing to do is to create the WSS solution package manifest. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

<Solution xmlns="http://schemas.microsoft.com/sharepoint/"

SolutionId="733641DA-95D9-446E-812E-6070947171B2"

DeploymentServerType="WebFrontEnd"

ResetWebServer="TRUE">

<Assemblies>

<Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab7.dll">

<SafeControls>

<SafeControl Namespace="Lab7" TypeName="\*" Safe="True" />

</SafeControls>

</Assembly>

</Assemblies>

<TemplateFiles>

<TemplateFile Location="XML\fldtypes\_Litware.xml"/>

<TemplateFile Location="IMAGES\Litware\binoculars.gif" />

<TemplateFile Location="IMAGES\Litware\cantine.gif" />

<TemplateFile Location="IMAGES\Litware\compass.gif" />

<TemplateFile Location="IMAGES\Litware\pithHelmetBrown.gif" />

<TemplateFile Location="IMAGES\Litware\pithHelmetWhite.gif" />

</TemplateFiles>

</Solution>

1. Save all changes... that's it!

In this exercise you created a working field type. Unfortunately it is not very useful as you have not created an editing experience.

Exercise 2: Creating a custom editing experience (field control) for a field type

In this exercise you will create a field control that will provide a customized editing experience for the field type you just created.

1. Just like the display rendering (which you defined as a rendering pattern in the field type definition file), you can specify the editing experience using CAML in the field type definition. However, the markup can quickly get to be quite messy and pose a significant maintenance challenge. Another option is to use a standard ASP.NET 2.0 user control (\*.ASCX) file... and link it up with a code behind. This approach is much easier for readability as well as maintenance moving forward.

Because the Visual Studio C# Class Library project template does not support creating an ASP.NET user control, you need to trick it a bit. Create a new text file with the name of **FieldResourceIconControl.ascx** in the **TEMPLATE\CONTROLTEMPLATES** directory within the project.

1. Next, you need to add the control directive to the page by adding the following code to the **FieldResourceIconControl.ascx** file:

<%@ Control Language="C#" %>

1. Now you need a reference to the **Microsoft.SharePoint.dll** assembly by adding the following code to the **FieldResourceIconControl.ascx** file:

<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>

1. In order to specify a rendering template, you need to add a register directive to a specific namespace within the **Microsoft.SharePoint.dll** assembly by adding the following code to the **FieldResourceIconControl.ascx** file:

<%@ Register Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" Namespace="Microsoft.SharePoint.WebControls" TagPrefix="SharePoint" %>

1. Finally, you can now specify the rendering template for the editing experience. Add the following markup to the **FieldResourceIconControl.ascx** file:

<SharePoint:RenderingTemplate ID="FieldResourceIconControl" runat="server">

<Template>

<table class="ms-form">

<tr>

<td align="right">Title:</td>

<td><asp:TextBox ID="ResourceTitle" runat="server" Size="30" /></td>

</tr>

<tr>

<td align="right">Description:</td>

<td><asp:TextBox ID="ResourceDescription" runat="server" Size="30" /></td>

</tr>

<tr>

<td align="right">Url:</td>

<td><asp:TextBox ID="ResourceUrl" runat="server" Size="30" /></td>

</tr>

<tr>

<td align="right">Icon:</td>

<td><asp:DropDownList ID="ResourceIcon" runat="server" Size="5" /></td>

</tr>

</table>

</Template>

</SharePoint:RenderingTemplate>

1. Now that the rendering template has been created, you now need to create the code file that will handle the server-size logic for the control as well as wire it up to the field type. Create a new class in the root of the project named **FieldResourceIconControl.cs** and add the following code to it:

using System;

using System.Configuration;

using System.Web.UI.WebControls;

using Microsoft.SharePoint.WebControls;

namespace Lab7 {

public class FieldResourceIconControl : BaseFieldControl {

}

}

1. First add a constant containing the name of the rendering template and some internal fields that will be used as references to the ASP.NET controls in the rendering template. Add the following code to the **FieldResourceIconControl.cs** file:

private const string RENDERING\_TEMPLATE\_NAME = "FieldResourceIconControl";

protected TextBox \_txbTitle;

protected TextBox \_txbDescription;

protected TextBox \_txbUrlTarget;

protected DropDownList \_ddlIcon;

1. Next, override the **DefaultTemplateName** property on the **BaseFieldControl** class and return the name of the rendering template by adding the following code just after the fields you just added to the **FieldResourceIconControl.cs** file:

protected override string DefaultTemplateName {

get {

return RENDERING\_TEMPLATE\_NAME;

}

}

1. Like any ASP.NET server control, much of the user interface work is done in the **CreateChildControls()** method... and this server control is no different. You now need to override the **CreateChildControls()** method to (1) verify that there is data in the field, (2) that the current mode of the control is the desired mode (the rendering template you created is only for the new/edit experience, so you want to exclude any other modes), (3) obtain references to the ASP.NET controls in the rendering template and finally (4) initialize any controls needed initialization. Add the following code to the **FieldResourceIconControl.cs** file:

protected override void CreateChildControls () {

// don't do anything if this is display mode or if the field has nothing assigned to it

if (this.Field == null || this.ControlMode == SPControlMode.Display || this.ControlMode == SPControlMode.Invalid)

return;

base.CreateChildControls();

// get reference to the control

this.\_txbTitle = TemplateContainer.FindControl("ResourceTitle") as TextBox;

if (this.\_txbTitle == null)

throw new ConfigurationErrorsException("ResourceTitle TextBox not found. Corrupt control template.");

this.\_txbDescription = TemplateContainer.FindControl("ResourceDescription") as TextBox;

if (this.\_txbDescription == null)

throw new ConfigurationErrorsException("ResourceDescription TextBox not found. Corrupt control template.");

this.\_txbUrlTarget = TemplateContainer.FindControl("ResourceUrl") as TextBox;

if (this.\_txbUrlTarget == null)

throw new ConfigurationErrorsException("ResourceUrl TextBox not found. Corrupt control template.");

this.\_ddlIcon = TemplateContainer.FindControl("ResourceIcon") as DropDownList;

if (this.\_ddlIcon == null)

throw new ConfigurationErrorsException("ResourceIcon DropDownList not found. Corrupt control template.");

// init the drop down list

this.\_ddlIcon.Items.Add(new ListItem("Binoculars", "binoculars"));

this.\_ddlIcon.Items.Add(new ListItem("Cantine", "cantine"));

this.\_ddlIcon.Items.Add(new ListItem("Compass", "compass"));

this.\_ddlIcon.Items.Add(new ListItem("Brown pith helmet", "pithHelmetBrown"));

this.\_ddlIcon.Items.Add(new ListItem("White pith helmet", "pithHelmetWhite"));

}

1. To finish off the server-side part of the rendering control, you need to override the **Value** property to set the ASP.NET controls when the control is loaded as well as fetch the data from the controls when the property is read. Add the following code to the **FieldResourceIconControl.cs** file:

public override object Value {

get {

EnsureChildControls();

// fetch values from rendering control return back as an object

FieldResourceIconValue field = new FieldResourceIconValue();

field.Title = this.\_txbTitle.Text.Trim();

field.Description = this.\_txbDescription.Text.Trim();

field.UrlTarget = this.\_txbUrlTarget.Text.Trim();

field.Icon = this.\_ddlIcon.SelectedValue;

return field;

}

set {

EnsureChildControls();

// if something stored in the value, init controls in rendering control

if (value != null && !string.IsNullOrEmpty(value.ToString())) {

FieldResourceIconValue field = new FieldResourceIconValue(value.ToString());

this.\_txbTitle.Text = field.Title;

this.\_txbDescription.Text = field.Description;

this.\_txbUrlTarget.Text = field.UrlTarget;

this.\_ddlIcon.Items.FindByValue(field.Icon).Selected = true;

}

}

}

1. Finally, you can now wire the control up to the field type created in exercise 1. Add the following code to override the **FieldRenderingControl()** property on the field type found within the **FieldResourceIcon.cs** file:

public override BaseFieldControl FieldRenderingControl {

get {

BaseFieldControl fieldControl = new FieldResourceIconControl();

fieldControl.FieldName = this.InternalName;

return fieldControl;

}

}

1. Now all that's left to do is include this file in the packaging of the WSS solution package! Add the following lines to the **BuildSharePointPackage.ddf** file to add the rendering template to the package:

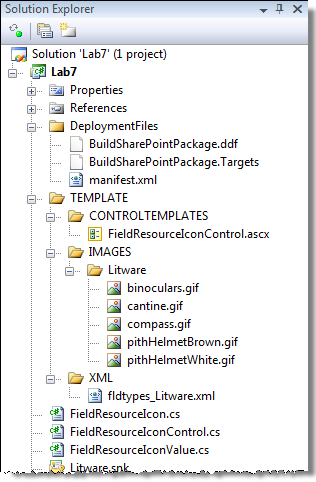
.Set DestinationDir=CONTROLTEMPLATES

TEMPLATE\CONTROLTEMPLATES\FieldResourceIconControl.ascx

1. And lastly, add the following line to the **manifest.xml** file to tell SharePoint what to do with this file (this line should go with the other existing **<TemplateFile>** nodes):

<TemplateFile Location="CONTROLTEMPLATES\FieldResourceIconControl.ascx" />

1. When you save everything, your project should look like the following image:



Now it's time to deploy the WSS solution package.

1. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN

1. Enter the following command into the command line window and hit Enter:

stsadm -o addsolution -filename c:\Student\Labs\06\_FieldTypesControls\Lab\wsp\Debug\Lab7.wsp

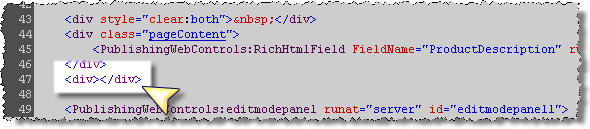
1. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
2. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
3. On the **Solution Management** page, click the link on **lab7.wsp**.
4. On the **Solution Properties** page, select **Deploy Solution**.
5. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.

With the solution deployed, you now need to use the field type & control. To do this, you will add the field to an existing content type and page layout. Specifically, you'll add it to the page layout created by the utility you used in a previous lab.

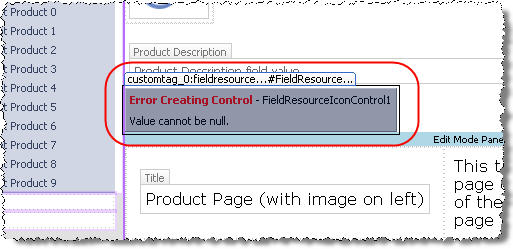
1. Browse to the **http://wcm.litwareinc.com/** site and select **Site Actions » Site Settings » Modify All Site Settings**.
2. Select **Site content types** from the **Galleries** section.
3. On the **Site Content Type Gallery** page, scroll to the bottom and select **Widget Product Page** under the **Widget Content Builder** group.

*Note: The* ***Widget Content Builder*** *is included in the* Student\Resources\Sample Data\WidgetContentBuilder *folder. You can run that Feature (it is a hidden Feature so activation must be done via STSADM.EXE, or just run the batch file) or you can use the* ***Article Page*** *content type in the* ***Page Layout Content Types*** *group*.

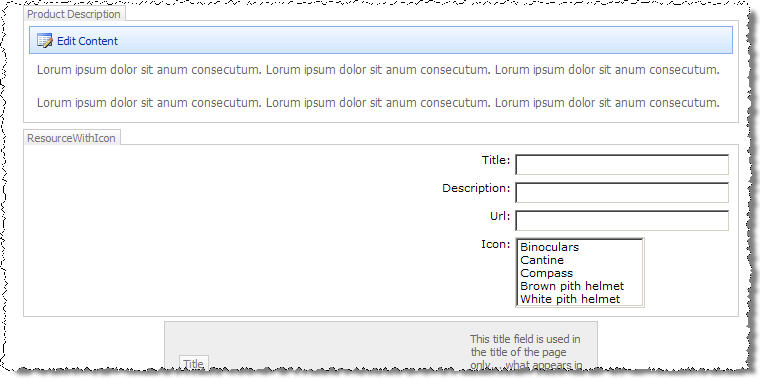
1. On the **Site Content Type: Widget Product Page** page, scroll to the bottom and select **Add from new site column**.
2. Use the following information to create a new site column and click **OK**:
   * **Column name:** ResourceWithIcon
   * **The type of information in this column is:** Resource with Icon
   * **Put this site column into:** Existing group: Widget Content Builder
3. With the content type updated, now you need to update an associated page layout. Check out and open the page layout **ProductPageLeft.aspx** (or whichever page layout matches the selected content type) in **SharePoint Designer** and switch to the **Code** view.
4. Scroll down to the **<PublishingWebControls:RichHtmlField>** field control used for the **ProductDescription** field is used (~ line 45). After the closing **<div>** add another opening & closing **<div>**, like the following image shows:



1. Switch back to **Design** view. Grab the **ResourceWithIcon** field from the content type and drop it into the **<div>** you just created as shown in the following image (ignore the error message):



1. Save your changes, go back to your browser and navigate to one of the Widget product pages such as **http://wcm.litwareinc.com/Widgets/Pages/WidgetProduct0.aspx**. Select **Site Actions » Edit Page**.
2. Scroll down and you should see the new field control, as shown in the following image:



1. Enter some information into the field control and then select **Page » Save and Stop Editing** to see the display mode of the field control, as shown in the following image:



In this exercise you created a custom field control and associated it with the custom field type you created in the first exercise.