## Creating Fields, Site Columns and Content Types

**Lab Time:** 60 minutes

**Lab Folder**: C:\Student\Labs\ContentTypes

**Lab Overview:** In this lab you will work with some of the new capabilities Microsoft added to the Visual Studio 2010 SharePoint Tools to create site columns & content types. In addition you will also learn how to create a custom field type with an associated field control.

Lab Setup Requirements

* Before you begin this lab, you must run the batch file named **SetupLab.bat** located in the folder for this lab. This batch file creates a new SharePoint site collection with a top-level Blank site at the location **http://intranet.wingtip.com/sites/ColumnsContentTypes**.

### Exercise 1: Creating Site Columns & Content Types

In this exercise you will create a few new site columns and content types that leverage these site columns using the Visual Studio 2010 SharePoint Tools. You will create a content type for managing contracts.

1. Open Visual Studio 2010 and create a new SharePoint project of type **Content Type**. Give it the name **ContractContentType**.
2. Complete the SharePoint Customization wizard that appears using the following information:

**Debugging site:** http://intranet.wingtip.com/sites/ColumnsContentTypes

**Deploy as a farm solution**: selected

1. It will take a few seconds before the next screen in the wizard is displayed because Visual Studio must connect to the SharePoint site and run some queries. When the next screen appears, it contains a dropdown with all existing content types. Select the **Item** content type and click the **Finish** button.
2. Locate the **ContentType1** folder in the Solution Explorer. Right-click it to rename it to **ContractContentType**.
3. The **ContractContentType** folder contains a file with name elements.xml. Double-click it to open it in the Visual Studio XML editor. The XML contains the starting point for the definition of the content type.

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

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

<ContentType ID="0x0100fb1ad12faa9b4834ad4d590f0f030151"

Name="ContractContentType – ContractContentType"

Group="Custom Content Types"

Version="0">

<FieldRefs>

</FieldRefs>

</ContentType>

</Elements>

Take notice of the ID attribute. This is the unique identifier for this content type. A content type ID consists of the following parts:

1. 0x01: This is the unique identifier of the parent content type, which is the **Item** content type in this exercise.
2. 00: The separator.
3. [Guid]: A generated GUID that makes the new content type unique.

**Note:** The GUID you see will be different than the one shown here.

1. Inside the XML editor, change the Name of the content type to **Contract Content Type and change the** Group **to Wingtip Content Types**.

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

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

<ContentType ID="0x0100fb1ad12faa9b4834ad4d590f0f030151"

Name="Contract Content Type"

Group="Wingtip Content Types"

Version="0">

<FieldRefs>

</FieldRefs>

</ContentType>

</Elements>

1. A content type consists of a number of site columns. Some of the columns already exist in SharePoint but other columns will need to be defined first. The contract content type in this exercise consists of the following columns:
2. Contract number
3. Company name
4. Start Date
5. Duration

The best practice for creating site columns is to create them in a separate XML file. Right click the project in the Solution Explorer and choose to add an **Empty Element** from the SharePoint 2010 templates. Give it the name **Contract Site Columns**.

1. Open the Elements.xml file which only contains the <Elements> root element:

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

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

</Elements>

1. The **Item** content type contains the **Title** column. You will use this column for the company name, so you don’t have to define an extra column for that. Add the following code within the Elements element to define a site column for the **Contract Number**.

A contract number must be unique so you have to set the attributes Indexed and EnforceUniqueValues to true. You should note that SharePoint requires you to define the Indexed attribute above the EnforceUniqueValues attribute.

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

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

<Field

SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{C6BA9DDC-307E-4bbe-98DC-19421D843924}"

Name="ContractNumber"

DisplayName="Contract Number"

Group="Wingtip Contract Columns"

Type="Text"

Indexed=“TRUE”

EnforceUniqueValues=“TRUE”

DisplaceOnUpgrade="TRUE" />

</Elements>

**Note:** You don’t need to use the same GUIDs as shown here. You can generate your own GUIDs using the guidgen.exe tool that comes with the Windows SDK which has been added as a shortcut in Visual Studio 2010 (**Tools » Create GUID**). Just make sure you track the ones you use because you’ll need to reference them in future steps. Also ensure the **ID** attribute is in CAPS. IntelliSense tries to use the invalid Id="" format.

1. Other important attributes of a field definition are:
2. ID: A GUID that is the unique identifier for the site column
3. Name: The internal name
4. DisplayName: The display name
5. Group: The group name to group the site columns together
6. Type: The data type of the column
7. DisplaceOnUpgrade: If a field definition already exists for the field, specifying TRUE will force updates to field properties with the values that are specified in this field definition.
8. Enter the following CAML to specify the definition for the start date:

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

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

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{C6BA9DDC-307E-4bbe-98DC-19421D843924}"

Name="ContractNumber"

DisplayName="Contract Number"

Group="Wingtip Contract Columns"

Type="Text"

Indexed=“TRUE”

EnforceUniqueValues=“TRUE”

DisplaceOnUpgrade="TRUE" />

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{AE270EED-EF9A-41ca-A303-962F2ABDE05B}"

Name="ContractStartDate"

DisplayName="Start Date"

Group="Wingtip Contract Columns"

Type="DateTime"

Format= "DateOnly"

DisplaceOnUpgrade="TRUE" />

</Elements>

Notice the **DateTime** data type. If you only need the date part, you can specify Format=”DateOnly”.

1. The **Duration** column is a choice field to give the user the possibility to choose from a limited list of values. Enter the following CAML to specify the definition for the duration:

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

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

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{C6BA9DDC-307E-4bbe-98DC-19421D843924}"

Name="ContractNumber"

DisplayName="Contract Number"

Group="Wingtip Contract Columns"

Type="Text"

Indexed=“TRUE”

EnforceUniqueValues=“TRUE”

DisplaceOnUpgrade="TRUE" />

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{AE270EED-EF9A-41ca-A303-962F2ABDE05B}"

Name="ContractStartDate"

DisplayName="Start Date"

Group="Wingtip Contract Columns"

Type="DateTime"

Format= "DateOnly"

DisplaceOnUpgrade="TRUE" />

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{2B851289-08F1-4f17-BAEC-19A84A5C1BFC}"

Name="ContractDuration"

DisplayName="Duration"

Group="Wingtip Contract Columns"

Type="Choice"

DisplaceOnUpgrade="TRUE">

<CHOICES>

<CHOICE>1 Month</CHOICE>

<CHOICE>1 Year</CHOICE>

<CHOICE>3 Years</CHOICE>

<CHOICE>5 Years</CHOICE>

</CHOICES>

</Field>

</Elements>

1. The definition of the site columns is complete. Save your work.
2. Return to the elements.xml file in the **ContractContentType** folder to complete the definition of the content type.
3. The content type definition contains an empty <FieldRefs> element. You will have to specify the fields that make up the content type here. Define a <FieldRef> element for each column in your content type, starting with the contract number. As already mentioned, you are going to use the existing **Title** site column for this. Add the following <FieldRef> element within the <FieldRefs> element:

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

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

<ContentType ID="0x0100fb1ad12faa9b4834ad4d590f0f030151"

Name="Contract Content Type"

Group="Wingtip Content Types"

Version="0">

<FieldRefs>

<FieldRef ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}"

Name="Title"

DisplayName="Company Name" />

</FieldRefs>

</ContentType>

</Elements>

1. The three other fields can be added as follows, using the GUIDs you earlier defined for each field:

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

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

<ContentType ID="0x0100fb1ad12faa9b4834ad4d590f0f030151"

Name="Contract Content Type"

Group="Custom Content Types"

Version="0">

<FieldRefs>

<FieldRef ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}"

Name="Title"

DisplayName="Company Name" />

<FieldRef ID="{C6BA9DDC-307E-4bbe-98DC-19421D843924}"

Name="ContractNumber"

DisplayName="Contract Number" />

<FieldRef ID="{AE270EED-EF9A-41ca-A303-962F2ABDE05B}"

Name="ContractStartDate"

DisplayName="Start Date" />

<FieldRef ID="{2B851289-08F1-4f17-BAEC-19A84A5C1BFC}"

Name="ContractDuration"

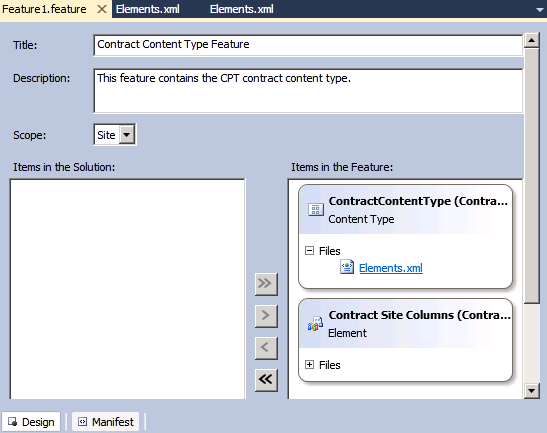
DisplayName="Duration" />

</FieldRefs>

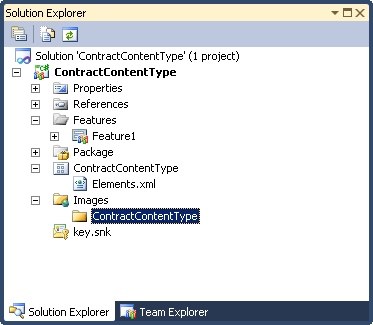
</ContentType>

</Elements>

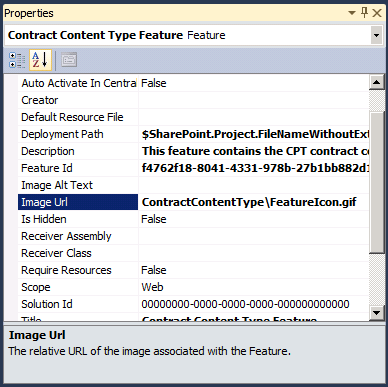
1. Your site columns and content type are now ready to deploy. To give your feature a more personalized touch double-click the **Feature1.feature** file to open the feature in the **Feature Designer**. Change the following attributes:
2. Title: Contract Content Type Feature
3. Description: This feature contains the Wingtip Contract content type
4. Scope: Site
5. Note the different parts that make up the feature.



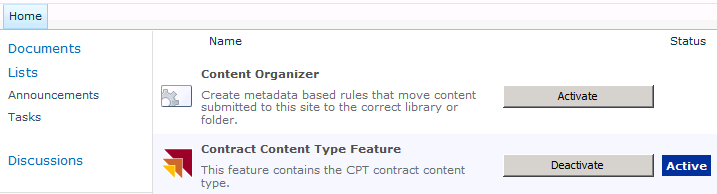
1. You can also add your own icon to the feature. Right click the **ContractContentType** project in the Solution Explorer and choose **Add » SharePoint “Images” Mapped Folder**. This adds the **Images** folder to the project, containing a sub folder with the name of the project. It is a best practice to deploy custom images to a sub folder of the SharePoint **Images** folder.



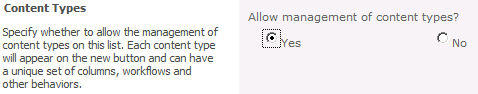
1. Right-click the **ContractContentType** folder inside the **Images** folder, choose **Add » Existing Item** and browse to the FeatureIcon.gif image in the **[[LAB FILES]]\Starter Files** folder for this lab.
2. Inside the **Solution Explorer**, double click on the **Feature1.feature** node to open the Feature Designer. Once the Feature Designer becomes the active windows, the main Visual Studio windows should display the **Properties** window for the feature in the lower right-hand corner. Inside the **Properties** window you should be able to find a property named ImageUrl. Since the ImageUrl property is relative to the **Images** folder, you should set its value to ContractContentType\FeatureIcon.gif.



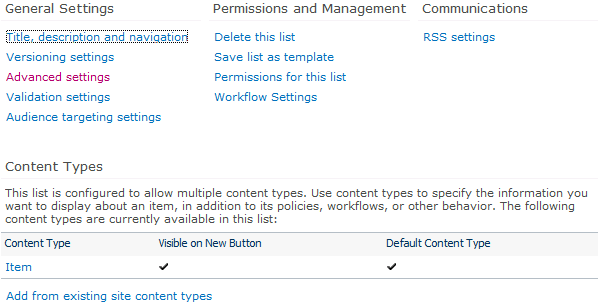
1. Right-click the project in the Solution Explorer and choose **Deploy**.
2. Navigate to the SharePoint site located at http://intranet.wingtip.com/sites/ColumnsContentTypes.
3. Go to the **Site Settings** page and click the hyperlink **Site Collections Features** to verify that your feature deployed correctly and that it is activated.



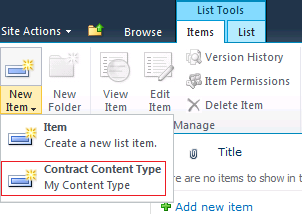
1. Navigate to the site columns page by clicking the **Site Columns** hyperlink in the **Gallery** section of the **Site Settings** page. Scroll down and locate the **Wingtip Contract Columns** group. Verify that your columns are there.
2. Navigate to the content types page by clicking the **Sit e** **Content Types** hyperlink in the **Gallery** section of the **Site Settings** page. Scroll down and locate the **Wingtip Content Types** group.
3. Click the **Contract Content Type** hyperlink and take your time to inspect the settings page for your content type. Verify that all columns are added to the content type.
4. Create a new list based on the **Custom List** template and give it the name **Contracts**. You can accomplish this by selecting **More Options…** from the **Site Actions** menu and then choosing the **Custom List** template in the resulting dialog that appears.
5. In the ribbon select the **List »** **List Settings**.
6. On the **List Setting** page, click on the **Advanced Settings** link.
7. In the **Content Types** section change to setting for **Allow the management of content types** to **Yes** and click the **OK** button at the bottom of the page to save your changes.



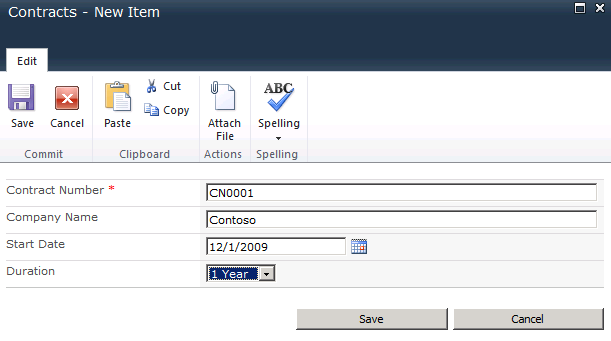
1. Notice the **Content Types** section that has been added to the list settings page. Click the **Add from existing site content types** hyperlink to add your content type to the list.



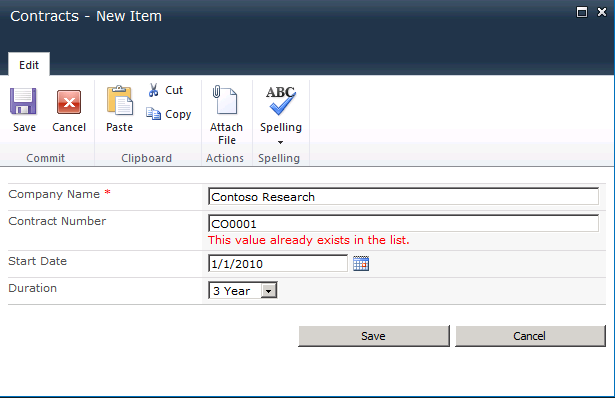
1. Choose the **Wingtip Content Types** group from the first dropdown. Your **Contract** content type should become visible. Click the **Add** button and then the **OK** button to save your changes.
2. Navigate back to the list by clicking the **Contracts** hyperlink in the breadcrumb.
3. Create a new contract using the ribbon and by selecting **Items »** **New Item**. Click the **Contract Content Type** option.



1. Fill out some data to create a new contract:



1. Save your contract item.
2. Try to add another company with the same contract number. You will get a message indicating that you cannot use the same contract number twice.



In this exercise you created a few new site columns and content types that leveraged these site columns using the Visual Studio 2010 SharePoint Tools.

### Exercise 2: Creating Custom Field Types & Controls

In this exercise you will create a new SharePoint field type for entering an address and define the rendering in an associated field control.

1. Open Visual Studio 2010 and create a new project using the **Empty SharePoint Project** template. Give it the name **AddressFieldType**.
2. Complete the wizard that appears using the following information.

**Debugging site:** http://intranet.wingtip.com/sites/ColumnsContentTypes

**Deploy as a farm solution**: selected

1. Add a new item of type **Empty Element** and give it the name **Address Field**. Click the **Add** button.
2. Add a new class to the project and give it the name AddressField.cs. Add an extra using statement:

using Microsoft.SharePoint;

The first step in creating a custom field type and field control is to create the field type class. A field type inherits from Microsoft.SharePoint.SPFieldor from a class that is derived from it. As an address consists of different parts like street number, street name, city, zip code and state, your Address field type must inherit from Microsoft.SharePoint.SPFieldMultiColumn. Don’t forget to change the scope of the class to public.

public class AddressField: SPFieldMultiColumn

{

}

1. All fields inheriting from Microsoft.SharePoint.SPFieldMultiColumn need to have two constructors:

public class AddressField: SPFieldMultiColumn

{

public AddressField(SPFieldCollection fields, string fieldName)

: base(fields, fieldName) { }

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

: base(fields, typeName, displayName) { }

}

1. Because you are creating a field type that consists of four different columns, the next step is to create a custom field value. While the data in the field is stored as a delimited string using the ;# delimiter, the field value class will take care of the different parts of the data. Add another class to the project and give it the name AddressFieldValue.
2. Make the following changes:
3. Add a using statement to the code for the Microsoft.SharePoint namespace.
4. Change the scope of the class to public.
5. Make the class inherit from SPFieldMultiColumnValue.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.SharePoint;

namespace AddressFieldType

{

public class AddressFieldValue: SPFieldMultiColumnValue

{

}

}

1. The address field type consists of four parts: street, city, state code and zip code. Define a constant containing the number of columns:

public class AddressFieldValue: SPFieldMultiColumnValue

{

private const int NumberOfColumns = 4;

}

1. Implement two constructors, one that passes the number of columns and one that passes the incoming data:

public class AddressFieldValue: SPFieldMultiColumnValue

{

private const int NumberOfColumns = 4;

public AddressFieldValue() : base(NumberOfColumns) { }

public AddressFieldValue(string value) : base(value) { }

}

1. Each column of the field value should be exposed by a property. Add a property for the Street property:

public class AddressFieldValue: SPFieldMultiColumnValue

{

private const int NumberOfColumns = 4;

public AddressFieldValue() : base(NumberOfColumns) { }

public AddressFieldValue(string value) : base(value) { }

public string Street

{

get { return this[0]; }

set { this[0] = value; }

}

}

1. Add also the necessary properties for the other columns like City, StateCode and ZipCode:

public class AddressFieldValue: SPFieldMultiColumnValue

{

private const int NumberOfColumns = 4;

public AddressFieldValue() : base(NumberOfColumns) { }

public AddressFieldValue(string value) : base(value) { }

public string Street

{

get { return this[0]; }

set { this[0] = value; }

}

public string City

{

get { return this[1]; }

set { this[1] = value; }

}

public string StateCode

{

get { return this[2]; }

set { this[2] = value; }

}

public string ZipCode

{

get { return this[3]; }

set { this[3] = value; }

}

}

1. Now that the field value class is ready, you have to associate it with the field type class. To make the field type class aware of the field value class, you have to override the SPField.GetFieldValue() method, returning a value of type AddressFieldValue:

public class AddressField: SPFieldMultiColumn

{

public AddressField(SPFieldCollection fields, string fieldName)

: base(fields, fieldName) { }

public AddressField(SPFieldCollection fields, string typeName,

string displayName)

: base(fields, typeName, displayName) { }

public override object GetFieldValue(string value)

{

if (string.IsNullOrEmpty(value))

return null;

else

return new AddressFieldValue(value);

}

}

1. Users of your custom field type can add columns of this type to their custom lists. At that time they can decide whether this is a required field or not. In case of a required field, a value must be entered when creating a list item. You can build in this type of validation by overriding the GetValidatedString() method.

public override string GetValidatedString(object value)

{

if (value == null && this.Required)

{

throw new SPFieldValidationException(

"This is a required field. Please fill out a value.");

}

else

{

// enter your custom validations here

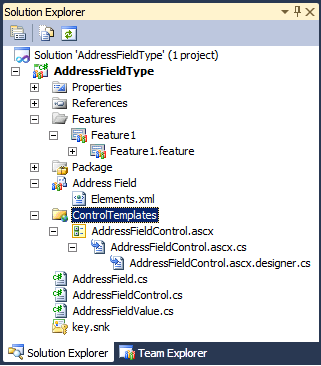
return value.ToString();

}

}

In case additional validations are needed, you must implement them in this method.

1. The next step is to define the user interface for the field type. While this can be achieved using server controls, it can best be done using a RenderingTemplate in a user control. These \*.ascx controls are always deployed to the [..]\14\TEMPLATE\CONTROLTEMPLATES folder. Right-click the project in the **Solution Explorer** and choose **Add » New Item.** In the **SharePoint 2010** category select the **User Control** item template and give it the name AddressFieldControl.ascx. Click the **Add** button.
2. Notice that the \*.ascx control is added to a sub folder of the CONTROLTEMPLATES folder. By design rendering templates for custom field types must reside in the root of the CONTROLTEMPLATES folder, not in sub folders of that CONTROLTEMPLATES folder. Select the AddressFieldControl.ascx file and drag it into the CONTROLTEMPLATES folder. Remove the AddressFieldType sub folder.



1. Define a rendering template with id of AddressFieldControlTemplate. This template will later be loaded from another class that also will be part of this field type.

<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>

<%@ Assembly Name="Microsoft.Web.CommandUI, Version=14.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>

<%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls" Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>

<%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities" Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>

<%@ Register Tagprefix="asp" Namespace="System.Web.UI" Assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" %>

<%@ Import Namespace="Microsoft.SharePoint" %>

<%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages" Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>

<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="AddressFieldControl.ascx.cs" Inherits="AddressFieldType.ControlTemplates.AddressFieldType.AddressFieldControl" %>

<SharePoint:RenderingTemplate id="AddressFieldControlTemplate" runat="server">

<Template>

</Template>

</SharePoint:RenderingTemplate>

1. Within this RenderingTemplate you will have to define the necessary controls to make the custom field render as follows:



The user interface is defined using HTML and ASP.NET controls. Add the following code within the <SharePoint:RenderingTemplate> element.

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

<Template>

<table class="ms-form">

<tr>

<td>Street:</td>

<td colspan=”5”>

<asp:TextBox ID="StreetTextBox" runat="server" Width="300px" />

</td>

</tr>

<tr>

<td>City:</td>

<td><asp:TextBox ID="CityTextBox" runat="server" Width="300px" /></td>

<td>State:</td>

<td><asp:TextBox ID="StateTextBox" runat="server" Width="50px" /></td>

<td>Zip:</td>

<td><asp:TextBox ID="ZipTextBox" runat="server" Width="50px" /></td>

</tr>

</table>

</Template>

</SharePoint:RenderingTemplate>

1. To make this rendering template fully functional, you have to create another new class that derives from BaseFieldControl. Don’t forget to change the scope of the class to public.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.SharePoint;

using Microsoft.SharePoint.WebControls;

namespace AddressFieldType

{

public class AddressFieldControl: BaseFieldControl

{

}

}

1. Define a constant for the rendering template and override the DefaultTemplateName property. This property will make SharePoint aware of which rendering template to use when rendering the field type

public class AddressFieldControl: BaseFieldControl

{

private const string RENDERING\_TEMPLATE = "AddressFieldControlTemplate";

protected override string DefaultTemplateName

{

get

{

return RENDERING\_TEMPLATE;

}

}

}

1. Then define class level variables for each TextBox on the control. ASP.NET controls reside in the System.Web.UI.WebControls namespace so first add an extra using statement for this namespace:

using System.Web.UI.WebControls;

1. Then define the class level variables:

public class AddressFieldControl: BaseFieldControl

{

private const string RENDERING\_TEMPLATE = "AddressFieldControl";

protected TextBox StreetTextBox;

protected TextBox CityTextBox;

protected TextBox StateTextBox;

protected TextBox ZipTextBox;

protected override string DefaultTemplateName

{

get

{

return RENDERING\_TEMPLATE;

}

}

}

1. Add a CreateChildControls() method to the AddressFieldControl class. The main purpose of this method is to wire up the TextBox variables with the ones defined in the RenderingTemplate**.** Throw an exception if one of the text boxes is not found.

protected override void CreateChildControls()

{

if (this.Field == null

|| this.ControlMode == SPControlMode.Display

|| this.ControlMode == SPControlMode.Invalid)

return;

base.CreateChildControls();

StreetTextBox =

TemplateContainer.FindControl("StreetTextBox") as TextBox;

if (StreetTextBox == null)

throw new ArgumentNullException(

"Street textbox not found. Please check the control template.");

CityTextBox = TemplateContainer.FindControl("CityTextBox") as TextBox;

if (CityTextBox == null)

throw new ArgumentNullException(

"City textbox not found. Please check the control template.");

StateTextBox = TemplateContainer.FindControl("StateTextBox") as TextBox;

if (StateTextBox == null)

throw new ArgumentNullException(

"State textbox not found. Please check the control template.");

ZipTextBox = TemplateContainer.FindControl("ZipTextBox") as TextBox;

if (ZipTextBox == null)

throw new ArgumentNullException(

"zip textbox not found. Please check the control template.");

}

1. The next property to override is the Value property. This property is used by SharePoint to set the value of the control when loading it in Edit mode, as well as to retrieve the values from the ASP.NET controls in the rendering template upon postbacks. Note that you have to call the EnsureChildControls() method within the getter and the setter of the Value property to make sure the controls are properly loaded.

public override object Value

{

get

{

EnsureChildControls();

}

set

{

EnsureChildControls();

}

}

1. In the getter instantiate a variable of type AddressFieldValue and populate it with the values entered in the TextBox controls:

public override object Value

{

get

{

EnsureChildControls();

AddressFieldValue fieldValue = new AddressFieldValue();

fieldValue.Street = StreetTextBox.Text;

fieldValue.City = CityTextBox.Text;

fieldValue.StateCode = StateTextBox.Text;

fieldValue.ZipCode = ZipTextBox.Text;

return fieldValue;

}

set

{

EnsureChildControls();

}

}

1. The setter will display the values from the field value in the TextBox controls.

public override object Value

{

get

{

// getter code

}

set

{

EnsureChildControls();

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

{

AddressFieldValue fieldValue = new AddressFieldValue(value.ToString());

StreetTextBox.Text = fieldValue.Street;

CityTextBox.Text = fieldValue.City;

StateTextBox.Text = fieldValue.StateCode;

ZipTextBox.Text = fieldValue.ZipCode;

}

}

}

1. The field control is finished now. The last step is to wire it up to the field type. Return to the AddressField.cs class and add an additional using statement:

Using Microsoft.SharePoint.WebControls;

1. Override now the FieldRenderingControl property:

public override BaseFieldControl FieldRenderingControl

{

get

{

BaseFieldControl control = new AddressFieldControl();

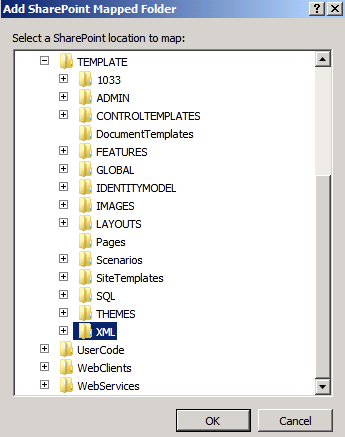
control.FieldName = this.InternalName;

return control;

}

}

1. A custom field type also needs a definition file to make SharePoint aware of the field. Such a definition must be deployed to the [..]\14\TEMPLATE\XML folder. Right-click the project in Solution Explorer and choose **Add » SharePoint Mapped Folders..**. A dialog containing the SharePoint folders hierarchy appears. Expand the TEMPLATE folder and choose the XML folder. Click the **OK** button.



1. In Solution Explorer notice the AddressFieldType sub folder created within the XML folder. Remove this sub folder because field definition files need to be deployed in the root of the XML folder.
2. Right-click the XML folder and add an XML file with name fldtypes\_address.xml to it. An XML file containing a field definition must always be prefixed by fldtypes\_ for SharePoint to recognize it and load it when SharePoint starts up.
3. Within the XML file, a field types are defined within the <FieldTypes> element. This parent element can contain one or more <FieldType> elements, each one defining a field type. Each field type must specify certain information like the type name, the parent type, the display name and the assembly and class name that contain the implementation of the field type. You also have to specify how the field will be rendered in a view and this is done using a RenderPattern element.

Add the following XML markup to the fldtypes\_address.xml file:

<FieldTypes>

<FieldType>

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

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

<Field Name="TypeDisplayName">Mail Address</Field>

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

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

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

<Field Name="FieldTypeClass">

AddressFieldType.AddressField, $SharePoint.Project.AssemblyFullName$</Field>

<RenderPattern Name="DisplayPattern">

<Switch>

<Expr>

<Column/>

</Expr>

<Case Value=""></Case>

<Default>

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

<HTML><![CDATA[,&nbsp;]]></HTML>

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

<HTML><![CDATA[,&nbsp;]]></HTML>

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

<HTML><![CDATA[&nbsp;]]></HTML>

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

</Default>

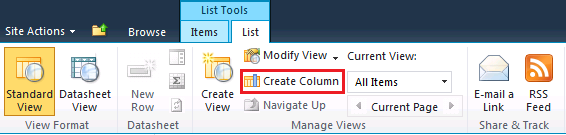
</Switch>

</RenderPattern>

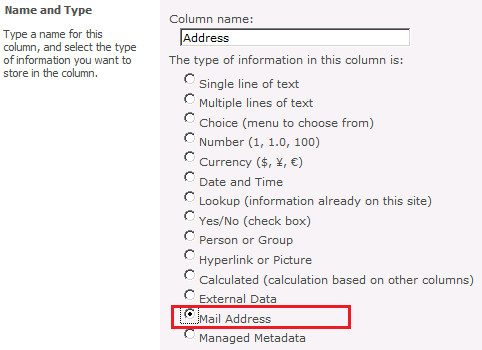
</FieldType>

</FieldTypes>

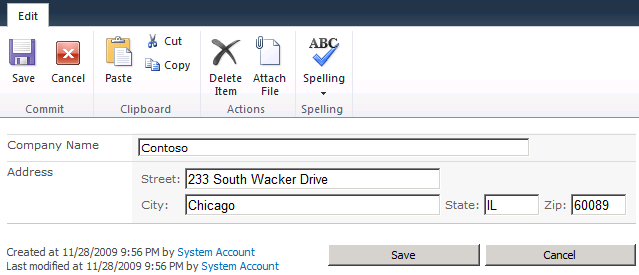
1. Your custom field type is ready to deploy. Right-click the project in Solution Explorer and choose **Deploy**.
2. Next, recycle IIS to get SharePoint to pickup the new field type by running IISRESET.EXE from a command prompt.
3. Navigate to the SharePoint site located at **http://intranet.wingtip.com/sites/ColumnsContentTypes**.
4. Create a new list of type custom list and give it the name **Companies**.
5. In **List** ribbon tab, click the **Create Column** button.



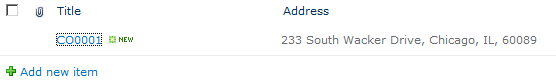
1. Give the new column the name **Address**. Choose to add a column of type **Mail Address** and click the OK button.



1. Create a new list item and populate it with data.



1. The address is rendered as follows in the **AllItems** view:



In this exercise you created a new SharePoint field type and define the rendering in an associated field control.

### Exercise 3: Creating Columns Based off Custom Field Types

In this exercise you will add another site column to the previously created content type that leverages the new field type created in the previous exercise. You will learn how you can upgrade an already deployed feature that adds a new site column to an existing content type that is already in use.

1. Go back to Visual Studio 2010 and open the project **ContractContentType** you created in Exercise 1.
2. Open the elements.xml that is associated with the **Contract Site Columns** folder.
3. Add a new column for the company address. This column is of the type AddressFieldyou created in Exercise 2.

<Field SourceID="http://schemas.microsoft.com/sharepoint/v3"

ID="{F5702212-9BF9-4c2d-8520-1B68016F0819}"

Name="CompanyAddress"

DisplayName="Address"

Group="Wingtip Contract Columns"

Type="AddressField"

DisplaceOnUpgrade="TRUE" />

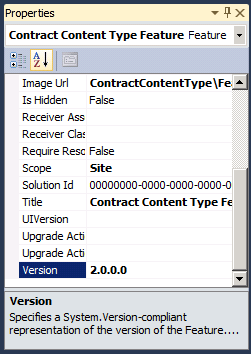
1. This new site column must also be added to the **Contract** content type. Open the elements.xml that is associated with the ContractContentType folder.
2. Add the new column at the bottom of the content type:

<FieldRef ID="{F5702212-9BF9-4c2d-8520-1B68016F0819}"

Name="CompanyAddress"

DisplayName="Address" />

1. The feature.xml file itself need to be modified to be able to inform SharePoint about the changes in the content type. Double-click the **Feature1.Feature** file to open the feature in design view. In the **Properties** window, set the Version property to 2.0.0.0.



1. The XML definition of the feature needs some changes to. Click the **Manifest** button at the bottom to display the XML. Click the **Edit Options** button to make the XML editable.
2. After the <Properties> element, add an additional <UpgradeActions> element. If you want to make SharePoint aware of an extra column in a content type, you have to use the <AddContentTypeField> element. You have to specify the id of the content type and the id of the field you want to add.

<UpgradeActions>

<VersionRange BeginVersion=”1.0.0.0” EndVersion=”2.0.0.0”>

<AddContentTypeField

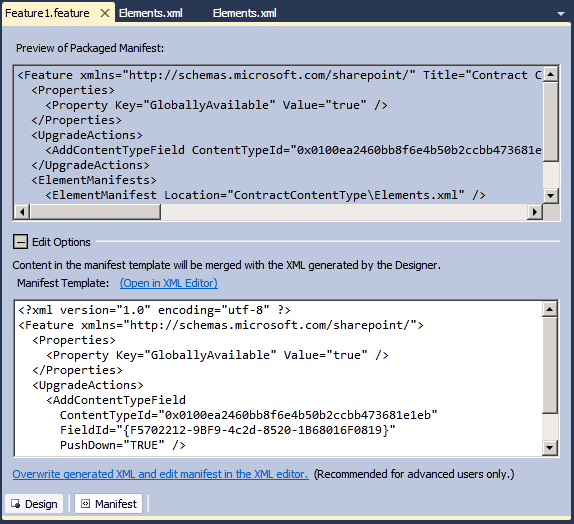
ContentTypeId="0x0100ea2460bb8f6e4b50b2ccbb473681e1eb"

FieldId="{F5702212-9BF9-4c2d-8520-1B68016F0819}"

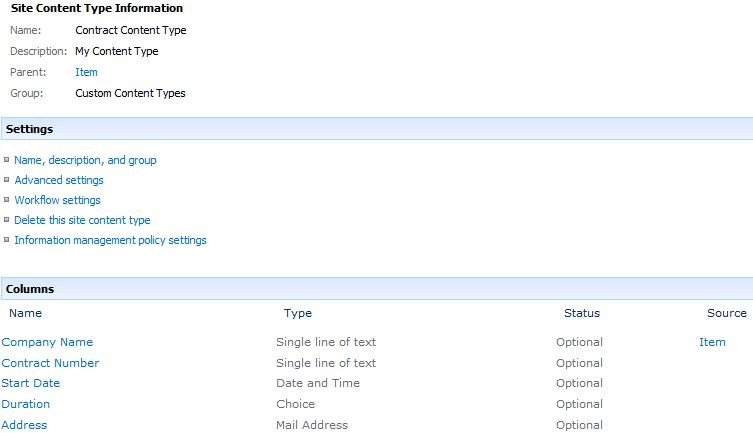
PushDown="TRUE" />

</VersionRange>

</UpgradeActions>



1. Save your changes and right-click the project in **Solution Explorer** and select **Deploy**.
2. Navigate to the **Site Columns Gallery** and notice that the new site column is added to the **Wingtip Contract Columns** group.
3. Navigate to the **Content Type Gallery** and notice that the **Contract Content Type** contains the **Address** column.



In this exercise you added another site column to the previously created content type that leverages the new field type created in the previous exercise. You learned how you can upgrade an already deployed feature that adds a new site column to an existing content type that is already in use.