# Lab 08: Creating custom Workflow Association Forms

**Lab Overview:** Now that the management of Litware Inc. has become comfortable with the built in functionality of SharePoint workflows, they have started to request extra features. They have requested a workflow system that archives documents in another document library. When the document library manager chooses to archive a document, they manually start a workflow, which creates an approval task.

In this lab, you’ll be adding an Association form to the workflow. This will allow the user associating the workflow with the document library to choose a default approver and archive list. Once this is done, every workflow instance will run with these settings automatically.

## Exercise 0: Setup

### If you did not complete lab 4, you’ll need to create the **Demo** site collection.

#### Open SharePoint and browse to the Demo site collection

##### The url is **http://litwareinc.com/sites/Demo**.

##### If the site collection does not exist, create it using the **CreateDemo.bat** file in the **C:\Labs\Files folder**.

### Open the starter **VS 2008** solution at **\Labs\Lab08\StarterFiles\DocumentArchive Part 3.sln**.

## Exercise 1: Creating an Association Form base class

### Create the **AssociationForm** base class in the UI folder.

#### Create a new file named **AssociationForm.cs** in the UI folder.

##### Right click the **UI** folder and click **Add -> Class…**.

##### Name the new file **AssociationForm.cs** and click **Add**.

#### Modify the class definition to the **AssociationForm** class to make it **public abstract** and derive from **LayoutsPageBase**.

public abstract class AssociationForm : LayoutsPageBase

{

}

### Add the base methods to be overridden in derived classes to perform the population and extraction of values from the UI. These methods will need to be virtual or abstract so they can be overridden in derived classes.

#### Create a **protected virtual** method named **PopulateControls** to populate the controls not directly tied to the association data.

#### Create a **protected abstract** method named **BindAssociationData** that takes a single string parameter. This method populates the UI with the association data from a previously created association.

#### Create a **protected abstract** method named **UpdateAssociationData** that returns a string. This method extracts the association data from the form on the submit event.

protected virtual void PopulateControls() { }

protected abstract void BindAssociationData(string value);

protected abstract string UpdateAssociationData();

### Since a workflow can be associated to a list, content type, or a list’s content type, create an enumeration to track the type of association being managed.

#### Create a private **AssociationType** enumeration inside the **AssociationForm** class with three values: **List**, **ContentType**, or **ListContentType**.

private enum AssociationType

{

List,

ContentType,

ListContentType

}

#### Add a private field to the class named \_**associtionType** of type **AssociationType**.

private AssociationType \_associationType;

### When the page is loaded, determine the association type of the association to manage. Using the **Url** and **Form** parameters, determine if the workflow is related to a list, content type, or both and whether it’s already been created.

#### In the **AssociationForm** class, override the **OnLoad** method.

protected override void OnLoad(EventArgs e)

{

}

#### Use the **HttpRequest** parameters to find the list id, content type id, and the existing association id.

##### The existence of these values determines the type of association and its existence.

// read the form level parameters

string listId = Request.Params["List"];

string ctypeId = Request.Params["ctype"];

string guidAssocId = Request.Params["GuidAssoc"];

#### Using the existence of the request parameters, determine the association type

// determine the type of association

if (!string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))

\_associationType = AssociationType.ListContentType;

else if (!string.IsNullOrEmpty(listId) && string.IsNullOrEmpty(ctypeId))

\_associationType = AssociationType.List;

else if (string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))

\_associationType = AssociationType.ContentType;

#### If an existing workflow association id exists, convert it into a **Guid**.

// validate the workflow association id

Guid? workflowAssociationId = new Guid?();

if (!string.IsNullOrEmpty(guidAssocId))

workflowAssociationId = new Guid(guidAssocId);

### Based on the association type and the existence of the workflow association, lookup the **SPList**, **SPContentType**, and **SPWorkflowAssociation** objects.

#### Add **private** fields to the class to store the content type, list and workflow association SharePoint APIs.

private SPContentType \_contentType;

private SPList \_list;

private SPWorkflowAssociation \_workflowAssociation;

#### At the end of the **OnLoad** method, add a switch statement to differentiate between a **ContentType**, **List**, or **ListContentType** association.

// lookup the list, content type, and existing workflow associations

switch (\_associationType)

{

case AssociationType.ContentType:

break;

case AssociationType.List:

break;

case AssociationType.ListContentType:

break;

}

#### If the association type is **ContentType**, find the **SPContentType** object using the **PageLayoutBase.Web** property. If a workflow association already exists, find it as well.

case AssociationType.ContentType:

\_contentType = Web.AvailableContentTypes[new SPContentTypeId(ctypeId)];

if (workflowAssociationId.HasValue)

\_workflowAssociation =

\_contentType.WorkflowAssociations[workflowAssociationId.Value];

#### If the association type is **List**, find the **SPList** object using the **PageLayoutBase.Web** property. If a workflow association already exists, find it as well.

case AssociationType.List:

\_list = Web.Lists[new Guid(listId)];

if (workflowAssociationId.HasValue)

\_workflowAssociation =

\_list.WorkflowAssociations[workflowAssociationId.Value];

#### If the association type is **ListContentType**, find the **SPList** object and then use it to find the **SPContentType**. If a workflow association already exists, find it as well.

case AssociationType.ListContentType:

\_list = Web.Lists[new Guid(listId)];

\_contentType = \_list.ContentTypes[new SPContentTypeId(ctypeId)];

if (workflowAssociationId.HasValue)

\_workflowAssociation =

\_contentType.WorkflowAssociations[workflowAssociationId.Value];

#### Call the base implementation of **OnLoad**.

// call the base implementation

base.OnLoad(e);

### Since many of the page’s parameters are in the posted form, write them back into the page so they will be available in subsequent postbacks.

#### In the **AssociationForm** class, override the **OnPreRender** method.

protected override void OnPreRender(EventArgs e)

{ }

#### Using the **Page.ClientScript.RegisterHiddenField** method, write the values of the form parameters into hidden fields in the page.

// register parameters in the hidden field

ClientScript.RegisterHiddenField("WorkflowName",

Request.Params["WorkflowName"]);

ClientScript.RegisterHiddenField("WorkflowDefinition",

Request.Params["WorkflowDefinition"]);

ClientScript.RegisterHiddenField("AddToStatusMenu",

Request.Params["AddToStatusMenu"]);

ClientScript.RegisterHiddenField("AllowManual", Request.Params["AllowManual"]);

ClientScript.RegisterHiddenField("RoleSelect", Request.Params["RoleSelect"]);

ClientScript.RegisterHiddenField("GuidAssoc", Request.Params["GuidAssoc"]);

ClientScript.RegisterHiddenField("SetDefault", Request.Params["SetDefault"]);

ClientScript.RegisterHiddenField("HistoryList", Request.Params["HistoryList"]);

ClientScript.RegisterHiddenField("TaskList", Request.Params["TaskList"]);

ClientScript.RegisterHiddenField("UpdateLists", Request.Params["UpdateLists"]);

ClientScript.RegisterHiddenField("AutoStartCreate",

Request.Params["AutoStartCreate"]);

ClientScript.RegisterHiddenField("AutoStartChange",

Request.Params["AutoStartChange"]);

### Since the **IsPostBack** property will always return true, population of the UI will be done in the **OnPreRender** event

#### Call the virtual **PopulateControls** method.

// populate controls unrelated to the association data

this.PopulateControls();

#### If an existing workflow association was found, call the **BindAssociationData** method and use the **AssociationData** property of the association as the parameter.

// bind the association data (if any)

if (\_workflowAssociation != null)

this.BindAssociationData(\_workflowAssociation.AssociationData);

#### Call the base implementation of **OnPreRender**.

// call the base implementation

base.OnPreRender(e);

### Create a handler for the **Cancel** button that will redirect to the workflow settings page for the current list or content type.

#### Create a **private** method named **BuildRedirectString**. It will return a string that is build based on the association type and the form parameters for the list and content type.

private static string BuildRedirectString(AssociationType associationType, string paramList, string paramCtype)

{ }

#### Use a **switch** statement to generate a different string for each association type.

##### If the type is **ContentType**, add a **ctype** url query string.

##### If the type is **List**, add a list url query string.

##### If the type is **ListContentType**, add both

switch (associationType)

{

case AssociationType.ContentType:

return "WrkSetng.aspx?ctype=" + paramCtype;

case AssociationType.List:

return "WrkSetng.aspx?List=" + paramList;

case AssociationType.ListContentType:

return "WrkSetng.aspx?List=" + paramList + "&ctype=" + paramCtype;

default:

throw new InvalidOperationException();

}

#### Create a **protected** event handler named **Cancel\_Click**.

protected void Cancel\_Click(object sender, EventArgs e)

{

}

#### Use **SPUtility.Redirect** to redirect to the string generated by the **BuildRedirectString** method.

// redirect back to the workflow settings page

SPUtility.Redirect(

BuildRedirectString(\_associationType, Request.Params["List"],

Request.Params["ctype"]),

SPRedirectFlags.RelativeToLayoutsPage, this.Context);

### Create a handler for the **Submit** button that will create or update the workflow association.

#### Create a **protected** event handler named **Submit\_Click**.

protected void Submit\_Click(object sender, EventArgs e)

{

}

#### Call the abstract **UpdateAssociationData** method to extract the association data from the form.

// get the association data

string associationData = UpdateAssociationData();

### Verify the task and history lists exist. If they do not exist, create them.

#### Create a new method named **LookupOrCreateList**. It will return a **SPList** object and accept the name of the list and a **SPListTemplateType** to determine the type of list to create if it does not already exist.

private SPList LookupOrCreateList(string paramList, SPListTemplateType listType)

{

}

#### Check if the first character of the list is a lower case **z**. This indicates that the name following the **z** is the name of the list to create.

// lookup the history list

if (paramList.StartsWith("z"))

{

}

else

{ }

#### If the list name started with **z**, create a new list and return the new **SPList** object.

Guid newListId =

Web.Lists.Add(paramList.Substring(1), "Workflow Tasks", listType);

return Web.Lists[newListId];

#### If the name did not start with **z**, find the existing list. Normally the list would be looked for by guid and by name then created if it was not found. In this example we’re going to assume it’s an existing.

return Web.Lists[new Guid(paramList)];

#### In the **Submit\_Click** method, add two lines of code to get the **SPList** objects for the **Tasks** and **History** lists. Use the **LookupOrCreateList** method to help.

// create or find the lists

SPList taskList = LookupOrCreateList(

Request.Params["TaskList"], SPListTemplateType.Tasks);

SPList historyList = LookupOrCreateList(

Request.Params["HistoryList"], SPListTemplateType.WorkflowHistory);

### Check if the association already exists. If so call the update method, if not call create.

#### Back in the **Submit\_Click** method; check if derived content types should be updated by checking the **UpdateLists** form parameter.

// check whether updates to content types cascade to their lists

bool updateContentTypeLists = (Request.Params["UpdateLists"] == "TRUE");

#### Check if the workflow association already exists. If it does, call the **UpdateWorkflowAssociation** method created in a following step. If it does not exist, call **CreateWorkflowAssociation** created in another following step.

// create or update the workflow association

if (\_workflowAssociation != null)

UpdateWorkflowAssociation(

Request.Params["TaskList"], Request.Params["HistoryList"],   
 associationData, updateContentTypeLists);

else

CreateWorkflowAssociation(

Request.Params["TaskList"], Request.Params["HistoryList"],   
 associationData, updateContentTypeLists);

### If the workflow association already exists, update it.

#### Create a new method named **PopulateWorkflowAssociation** that accepts the task list, history list, and association data. This method will set the properties of the \_**workflowAssociation** field based on the form parameters.

private void PopulateWorkflowAssociation(string taskListName,   
 string historyListName, string associationData)

{ }

#### Use the form parameters to initialize several of the startup and association data parameters in the \_**workflowAssociation** field.

// assign the form data to the new workflow association

\_workflowAssociation.Name = Request.Params["WorkflowName"];

\_workflowAssociation.AutoStartCreate =

(Request.Params["AutoStartCreate"] == "ON");

\_workflowAssociation.AutoStartChange =

(Request.Params["AutoStartChange"] == "ON");

\_workflowAssociation.AllowManual = (Request.Params["AllowManual"] == "ON");

\_workflowAssociation.AssociationData = associationData;

#### Check if the task list and history lists have changed. If they have changed, update them in the \_**workflowAssociation** field. In the case of a non content type, lookup and assign the lists directly. In the case of a content type, assign the list names.

// assign/update the tasks and history lists

if (this.\_associationType != AssociationType.ContentType) {

SPList taskList = LookupOrCreateList(

taskListName, SPListTemplateType.Tasks);

SPList historyList = LookupOrCreateList(

historyListName, SPListTemplateType.WorkflowHistory);

if (\_workflowAssociation.TaskListId != taskList.ID)

\_workflowAssociation.SetTaskList(taskList);

if (\_workflowAssociation.HistoryListId != historyList.ID)

\_workflowAssociation.SetHistoryList(historyList);

}

else {

if (\_workflowAssociation.TaskListTitle != taskListName)

\_workflowAssociation.TaskListTitle = taskListName;

if (\_workflowAssociation.HistoryListTitle != historyListName)

\_workflowAssociation.HistoryListTitle = historyListName;

}

#### Create a new **UpdateWorkflowAssociation** method that accepts the task list, history list, association data and a flag indicating whether updates to a content type are applied to derived content types.

private void UpdateWorkflowAssociation(string taskListName,   
 string historyListName, string associationData, bool updateContentTypeLists)

{ }

#### Call the new **PopulateWorkflowAssociation** method to update the form parameters in the workflow association.

PopulateWorkflowAssociation(taskListName, historyListName, associationData);

#### Create a **switch** statement that will use a different update method for each association type.

switch (\_associationType)

{

case AssociationType.ContentType:

break;

case AssociationType.List:

break;

case AssociationType.ListContentType:

break;

}

#### If the association type is **ContentType**, call the content type’s **UpdateWorkflowAssociation** method. If the **updateContentTypeLists** flag is set, also call the content type’s **UpdateWorkflowAssociationsOnChildren** method.

case AssociationType.ContentType:

\_contentType.UpdateWorkflowAssociation(\_workflowAssociation);

if (updateContentTypeLists)

\_contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);

#### If the association type is **List**, call the list’s **UpdateWorkflowAssociation** method.

case AssociationType.List:

\_list.UpdateWorkflowAssociation(\_workflowAssociation);

#### If the association type is **ListContentType**, call the content type’s **UpdateWorkflowAssociation** method.

case AssociationType.ListContentType:

\_contentType.UpdateWorkflowAssociation(\_workflowAssociation);

### If the workflow association does not exist, create a new one.

#### Create a new **CreateWorkflowAssociation** method that accepts the task list, history list, association data and a flag indicating whether updates to a content type apply to derived content types.

private void CreateWorkflowAssociation(string taskListName,   
 string historyListName, string associationData, bool updateContentTypeLists)

{

}

#### Lookup the workflow name and template using the **WorkflowName** and **WorkflowDefinition** form parameters and use them to locate the **SPWorkflowTemplate** object.

// lookup the workflow template and the workflow name

string workflowName = Request.Params["WorkflowName"];

Guid workflowTemplateId = new Guid(Request.Params["WorkflowDefinition"]);

SPWorkflowTemplate workflowTemplate = Web.WorkflowTemplates[workflowTemplateId];

#### Create a **switch** statement that will use a different create method for each association type.

switch (\_associationType)

{

case AssociationType.ContentType:

break;

case AssociationType.List:

break;

case AssociationType.ListContentType:

break;

}

#### If the association type is **ContentType**, create the new workflow association, populate it using **PopulateWorkflowAssociation**, add it to the content type, and finally update any child content types.

##### Call **SPWorkflowAssociation.CreateSiteContentTypeAssociation** to create the workflow association.

##### Call the previously created method **PopulateWorkflowAssociation** to populate the new workflow association.

##### Call **SPContentType.AddWorkflowAssociation** to attach the new workflow association to the content type.

##### Optionally call **SPContentType.UpdateWorkflowAssociationsOnChildren** to update child content types.

case AssociationType.ContentType:

\_workflowAssociation =

SPWorkflowAssociation.CreateSiteContentTypeAssociation(

workflowTemplate, workflowName, taskListName, historyListName);

PopulateWorkflowAssociation(taskListName, historyListName, associationData);

\_contentType.AddWorkflowAssociation(\_workflowAssociation);

if (updateContentTypeLists)

\_contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);

#### If the association type is **List**, create the new workflow association, populate it using **PopulateWorkflowAssociation**, and add it to the list.

##### Call **SPWorkflowAssociation.CreateListAssociation** to create the workflow association.

##### Call the previously created method **PopulateWorkflowAssociation** to populate the new workflow association.

##### Call **SPList.AddWorkflowAssociation** to attach the new workflow association to the content type.

case AssociationType.List:

\_workflowAssociation =

SPWorkflowAssociation.CreateListAssociation(

workflowTemplate, workflowName,   
 LookupOrCreateList(taskListName, SPListTemplateType.Tasks),  
 LookupOrCreateList(historyListName,   
 SPListTemplateType.WorkflowHistory));

PopulateWorkflowAssociation(taskListName, historyListName, associationData);

\_list.AddWorkflowAssociation(\_workflowAssociation);

#### If the association type is **ListContentType**, create the new workflow association, populate it using **PopulateWorkflowAssociation**, and add it to the list.

##### Call **SPWorkflowAssociation.CreateListContentTypeAssociation** to create the workflow association.

##### Call the previously created method **PopulateWorkflowAssociation** to populate the new workflow association.

##### Call **SPContentType.AddWorkflowAssociation** to attach the new workflow association to the content type.

case AssociationType.ListContentType:

\_workflowAssociation =

SPWorkflowAssociation.CreateListContentTypeAssociation(

workflowTemplate, workflowName,   
 LookupOrCreateList(taskListName, SPListTemplateType.Tasks),  
 LookupOrCreateList(historyListName,   
 SPListTemplateType.WorkflowHistory));

PopulateWorkflowAssociation(taskListName, historyListName, associationData);

\_contentType.AddWorkflowAssociation(\_workflowAssociation);

### Finish the **Submit\_Click** button event handler by redirecting to the settings page for the current list or content type.

#### Use the **SPUtility.Redirect** and the **BuildRedirectString** methods to redirect to the workflow settings page.

// redirect back to the workflow settings page

SPUtility.Redirect(

BuildRedirectString(

\_associationType, Request.Params["List"], Request.Params["ctype"]),

SPRedirectFlags.RelativeToLayoutsPage, this.Context);

### Add two properties that will expose information about the current workflow association to derived association forms.

#### Create a property named **IsNewAssocition** of type **bool**. Return whether the current association is new.

protected bool IsNewAssociation

{

get { return string.IsNullOrEmpty(Request.Params["GuidAssoc"]); }

}

#### Create a property named **WorkflowAssociationName** that returns the name of the current association.

protected string WorkflowAssociationName

{

get { return Request.Params["WorkflowName"]; }

}

## Exercise 2: Creating a custom Association Form

### Add a new base class named **DocArchivePart3AssocForm** that will be uses as the code behind for an ASPX page.

#### Create a new class named **DocArchivePart3AssocForm** in the UI folder.

##### Right click the **UI** folder and click **Add -> Class…**.

##### Name the new file **DocArchivePart3AssocForm.cs** and click **Add**.

#### Change the **DocArchivePart3AssocForm** class into a public class that derives from the **AssociationForm** base class.

public class DocArchivePart3AssocForm : AssociationForm

{ }

#### Add two protected fields to the class to access the controls in the ASPX form.

protected PeopleEditor ppkApprover;

protected DropDownList lstArchiveList;

### Populate the form by overriding the **PopulateControls** and **BindAssociationData** methods.

#### Override the **PopulateControls** method.

protected override void PopulateControls()

{ }

#### Filter the list of webs and populate the **lstArchiveLists** with all of the document libraries and are not hidden.

// filter the list of lists for all document libraries that aren't hidden

IEnumerable<SPList> documentLibraries =

Web.Lists.Cast<SPList>().Where(n => n is SPDocumentLibrary && !n.Hidden);

// add each visible document library to the list

foreach (SPList documentLibrary in documentLibraries)

lstArchiveList.Items.Add(

new ListItem(documentLibrary.Title, documentLibrary.ID.ToString()));

#### Override the **BindAssociationData** method.

protected override void BindAssociationData(string value)

{ }

#### Deserialize the association data using the **AssocInitData.Deserialize** method.

##### Uses the **XmlSerializer** to convert xml to a typed class.

// deserialize the association data

AssocInitData assocInitData = AssocInitData.Deserialize(value);

#### Populate the controls using the new **AssocInitData** class.

// bind the controls to the association data

ppkApprover.CommaSeparatedAccounts = assocInitData.ApproverUserName;

lstArchiveList.SelectedValue = assocInitData.ArchiveListId.ToString();

### Extract the data from the page when the form is submitted.

#### Override the **UpdateAssociationData** class.

protected override string UpdateAssociationData()

{ }

#### Extract the data from the controls into a new instance of the **AssocInitData** class.

// populate the association data using the UI

AssocInitData assocInitData = new AssocInitData();

assocInitData.ApproverUserName = (ppkApprover.Entities[0] as PickerEntity).Key;

assocInitData.ArchiveListId = lstArchiveList.SelectedValue;

#### Return the xml representing the **AssocInitData** using the **AssocInitData.Serialize** method.

##### Uses the **XmlSerializer** to convert xml to a typed class.

// serialize the return the association data

return assocInitData.Serialize();

### Create a new .ASPX page named **DocArchivePart3AssocForm.aspx** that represents the ASP.NET markup for the task page.

#### Create a new text file named **DocArchivePart3AssocForm.aspx** in the new **Layouts** folder.

##### Right click the **Layouts** folder and click **Add -> New Item…**.

##### Select the **General** category on the left and select a template of **Text file**.

##### Name the new file **DocArchivePart3AssocForm.aspx** and click **Add**.

#### Enter the page directives that define the code behind class and the master page for the task form.

<%@ Assembly Name="DocumentArchiveWorkflowPart3, Version=1.0.0.0, Culture=neutral, PublicKeyToken=15812f954569663f" %>

<%@ Page Language="C#" MasterPageFile="~/\_layouts/application.master"

EnableSessionState="true" ValidateRequest="False"

Inherits="DocumentArchiveWorkflowPart3.UI.DocArchivePart3AssocForm" %>

#### Register the tag prefix and locations of the SharePoint WebControls and UserControls that will be used to format the page.

<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"

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

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

<%@ Register TagPrefix="wssuc" TagName="InputFormSection" Src="/\_controltemplates/InputFormSection.ascx" %>

<%@ Register TagPrefix="wssuc" TagName="InputFormControl" Src="/\_controltemplates/InputFormControl.ascx" %>

<%@ Register TagPrefix="wssuc" TagName="ButtonSection" Src="/\_controltemplates/ButtonSection.ascx" %>

#### Define the content for the page title and the form specific headers.

<asp:Content ID="PageTitle" ContentPlaceHolderID="PlaceHolderPageTitle" runat="server">

<%= this.IsNewAssociation?"Customize Workflow":"Change a Workflow" %></asp:Content>

<asp:Content ID="PageTitleInTitleArea" ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea"

runat="server">

<%= this.IsNewAssociation?"Customize Workflow: " + this.WorkflowAssociationName:"Change a Workflow: " + this.WorkflowAssociationName %></asp:Content>

<asp:Content ID="PageDescription" ContentPlaceHolderID="PlaceHolderPageDescription"

runat="server">

<%= this.IsNewAssociation?"Use this page to define the settings of a new workflow.":"Use this page to change the settings of an existing workflow." %></asp:Content>

### Define the ASPX markup to create the data entry section of the custom task form.

#### Define the **ContentPlaceHolder** control for **PlaceHolderMain** and place a table within it.

<asp:Content ID="Main" ContentPlaceHolderID="PlaceHolderMain" runat="server">

<table cellspacing="0" cellpadding="0" style="border: none; width: 100%" class="ms-propertysheet">

</table>

</asp:Content>

#### Implement the first **InputFormSection** that allow entry of the approver.

##### The special SharePoint web control called **PeopleEditor** retrieves this information.

<%-- Deafault Approver Input --%>

<wssuc:InputFormSection Title="Default Approver" Description="Specify the user who will be the default approver."

runat="server">

<template\_inputformcontrols>

<wssuc:InputFormControl runat="server" LabelText="Approver:">

<Template\_Control>

<SharePoint:PeopleEditor id="ppkApprover"

AllowEmpty="false"

ValidatorEnabled="true"

MultiSelect="false"

SelectionSet="User"

width='300px'

runat="server" />

</Template\_Control>

</wssuc:InputFormControl>

</template\_inputformcontrols>

</wssuc:InputFormSection>

#### Implement the second **InputFormSection** that will allow the approver to choose the default archive list.

<%-- Archive List Input --%>

<wssuc:InputFormSection Title="Archive Document Library"

Description="Specify the document library used to store archived documents." runat="server">

<template\_inputformcontrols>

<wssuc:InputFormControl LabelText="Archive Document Library:" runat="server">

<Template\_Control>

<asp:DropDownList ID="lstArchiveList" runat="server" />

</Template\_Control>

</wssuc:InputFormControl>

</template\_inputformcontrols>

</wssuc:InputFormSection>

#### Implement the last section containing the **Submit** and **Cancel** buttons.

<%-- Submit and Cancel Buttons --%>

<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">

<template\_buttons>

<asp:PlaceHolder runat="server">

<asp:Button UseSubmitBehavior="false" runat="server" class="ms-ButtonHeightWidth" OnClick="Submit\_Click" Text="OK" id="BtnSubmit" /> &nbsp;

<asp:Button UseSubmitBehavior="false" runat="server" class="ms-ButtonHeightWidth" OnClick="Cancel\_Click" Text="Cancel" id="cmdCancel" causesvalidation=false />

</asp:PlaceHolder>

</template\_buttons>

</wssuc:ButtonSection>

## Exercise 3: Integrating custom Association Forms into the workflow

### Handle the **Invoked** event of the **OnWorkflowActivated** and deserialize the association data.

#### Open the Workfl**o**w code by right clicking on the **Workflow** class in the **Solution Explorer** and clicking **View Code**.

#### Add a private field named **AssocInitData** of type **AssocInitData** to the **Workflow** class.

private AssocInitData AssocInitData { get; set; }

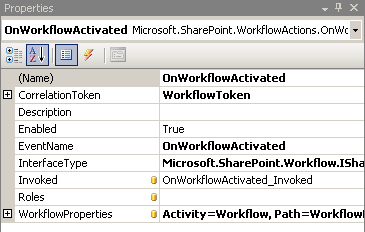
#### Open the workflow in the designer by right clicking on the Workflow class in the **Solution Explorer** and clicking **View Designer**.

#### Add an Invoked handler named **OnWorkflowActivated\_Invoked** to the **OnWorkflowActivated** activity.

##### Double click **WorkflowActivatedActivity** to open the sequential designer.

##### Right click the **OnWorkflowActivated** activity and select **Properties**.

##### In the properties pane, set Invoked to **OnWorkflowActivated\_Invoked** and press **Enter**.



#### In the **OnWorkflowActivated\_Invoked** method, deserialize the association data.

##### Deserialize the **AssocInitData** using its **Deserialize** method.

// deserialize init data, if none use defaults

this.AssocInitData = AssocInitData.Deserialize(

WorkflowProperties.AssociationData);

### Update the task creation to use the association data when initializing the task.

#### Locate the **CreateApprovalTask\_Invoking** method.

#### Change the line that sets the **taskProperties.AssignedTo** property and get its value from **AssocInitData.ApproverUserName**.

//taskProperties.AssignedTo = "LITWAREINC\\Administrator";

taskProperties.AssignedTo = AssocInitData.ApproverUserName;

#### Add an extended property to the **taskProperties** object. Use the **AssocInitData.ArchiveListId** to initialize the list selected in the task form.

taskProperties.ExtendedProperties.Add("ArchiveListId", this.AssocInitData.ArchiveListId);

### Update the task form to load the default archive list in the **OnLoad** method.

#### Open the **DocArchivePart3TaskForm.cs** file in the UI folder.

#### Locate the **OnLoad** Method.

#### In the section where controls are initialized, set the **SelectedValue** of the **lstArchiveList** control using the extended properties of the task.

##### Access the task’s extended properties using the **SPWorkflowTask.GetExtendedPropertiesAsHashtable** method.

// initialize the archive list selection

Hashtable extendedProperties =

SPWorkflowTask.GetExtendedPropertiesAsHashtable(\_taskItem);

lstArchiveList.SelectedValue = extendedProperties["ArchiveListId"] as string;

## Exercise 4: Deploying custom Association Forms

### Register the new custom Association form by assigning the **AssociationUrl** attribute in the feature’s **Workflow** element.

<Workflow ...

AssociationUrl ="\_layouts/DocArchivePart3AssocForm.aspx"

StatusUrl="\_layouts/WrkStat.aspx">

<Categories/>

<MetaData>

</MetaData>

</Workflow>

### Rebuild the workflow.

#### Right click the project in the **Solution Explorer** and click **Rebuild**.

#### In the **Output** window, verify the post build actions completed successfully.

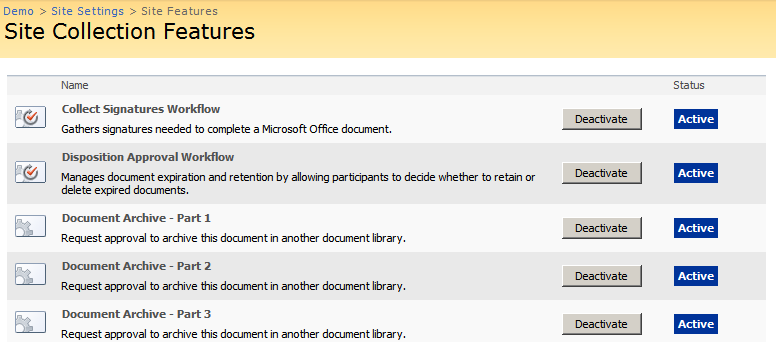
### Activate the new **DocumentArchiveWorkflowPart3** feature.

#### Using **Internet Explorer** navigate to the **Demo** site collection at **http://litwareinc.com/sites/Demo**.

#### Open the features list by clicking **Site Actions -> Site Settings**.

#### On the **Site Settings** page, click **Site collection features** in the **Site** **Collection Administration** section.

#### Click the **Activate** button next to the **Document Archive – Part 3** feature.



### Create an association between the **Shared Documents** document library and the new **Document Archive** **– Part 3** workflow.

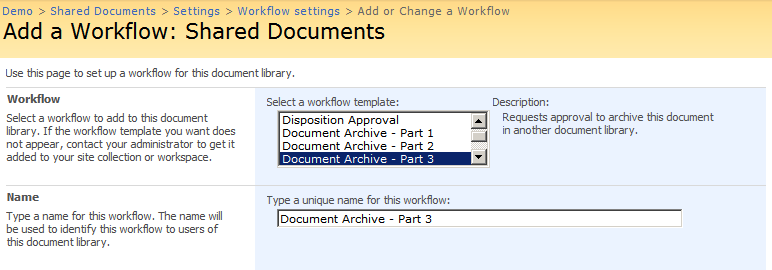
#### Navigate to the **Shared Documents** document library in the **Demos** site.

#### Click **Settings -> Document Library Settings** to load the settings page.

#### Click the **Workflow settings** link in the **Permissions and Management section**.

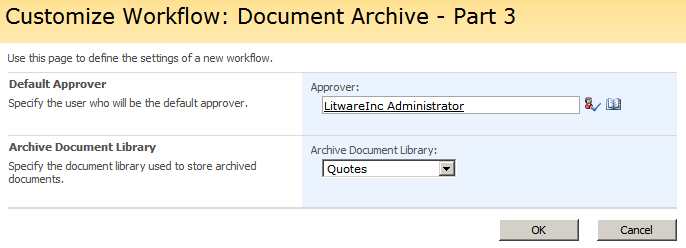
#### Create a new workflow using the **Document Archive – Part 3** workflow template and a name of **Document Archive - Part 3**.

##### Use the default values for both list and startup options.



#### In the custom association form, enter the Administrator as both the default approver and the manager. Select the **Quotes** document library as the archive location.

##### If you’d like, create an **Archive** document library to use instead.



### Run the **Document Archive – Part 3** workflow on a document.

#### Navigate to the Shared Documents document library, hover over the new document, and select **Workflows** from the drop down menu.

#### In the workflows page, click the **Document Archive – Part 3** to start the workflow.

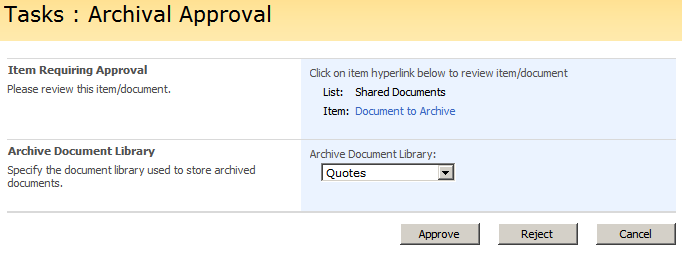
#### In the **Shared Documents** document library, verify the workflow is running.

#### Click the **In Progress** link to view the workflow status and verify the started message was logged to the workflow’s history.

#### Edit the task and set its **Status** to **Completed**.

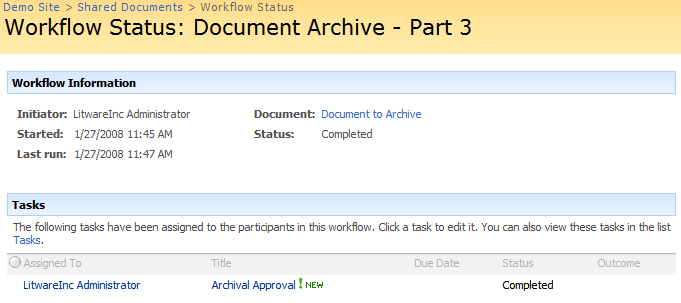
##### Notice that the **Archive Document List** defaults to the list set in the **Association Form**.

##### Click the **Approve** button to approve the task.



#### Verify that the workflow is now completed and the task is marked as completed.

##### You may have to refresh the status page to see the change.



## Challenge: Integrating Association Forms into Solution Packages

### Add the task form’s .ASPX file to the **Manifest.xml** file.

#### Open the **Manifest.xml** file in the **Solution** folder.

#### Add a **TemplateFile** element telling the solution installer to install the association form when the solution is deployed.

<TemplateFiles>

<TemplateFile Location="LAYOUTS\DocArchivePart3TaskForm.aspx"/>

<TemplateFile Location="LAYOUTS\DocArchivePart3AssocForm.aspx"/>

</TemplateFiles>

### Add the custom task form to the **Package.ddf** file.

#### Open the **Package.ddf** file in the **Solution** folder.

#### Add a line that packages the **DocArchivePart3AssocForm.aspx** and puts it in the **LAYOUTS** folder in the .cab file.

..\..\Template\Layouts\DocArchivePart3AssocForm.aspx LAYOUTS\DocArchivePart3AssocForm.aspx

#### Build the project by right clicking the project in the solution explorer and clicking **Rebuild**.

### Manually uninstall the previously deployed **Document Archive** feature.

#### Open a command window in the SharePoint **bin** directory.

##### Click **Start -> Run** and enter **cmd**.

##### Navigate to C**:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN**.

#### Execute **stsadm.exe** to uninstall the **DocuentArchiveWorkflowPart3** feature.

stsadm –o uninstallfeature –name DocumentArchiveWorkflowPart3 –force

#### Delete the folder at **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Features\DocumentarchiveWorkflowPart3**.

#### Delete the **DocArchivePart3AssocForm.aspx** and **DocArchivePart3TaskForm.aspx** files from the **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Layouts** folder.

### Install and deploy the new solution package.

#### In the same command window as the previous step, execute **stsadm** to add the feature to SharePoint.

stsadm -o addsolution –filename “C:\Labs\Lab08\StarterFiles\DocumentArchiveWorkflowPart3\bin\Debug\Package\DocumentArchiveWorkflowPart3.wsp”

#### Deploy the solution to the farm using the **deploysolution** command in **stsadm**.

stsadm –o deploysolution –name DocumentArchiveWorkflowPart3.wsp -allowgacdeployment –local

### Verify the feature is still available and activated.

#### Using **Internet Explorer** navigate to the **Demo** site collection at **http://litwareinc.com/sites/Demo**.

#### Open the features list by clicking **Site Actions -> Site Settings**.

#### On the **Site Settings** page, click **Site collection features** in the **Site** **Collection Administration** section.

#### Verify the **Document Archive – Part 3** feature is active.

## 