# Lab 09: Creating custom Instantiation and Modification 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 will be adding an Instantiation form and a Modification form to the workflow. This will allow the user to override the association settings when starting the workflow and even change them while the workflow is running. With these two forms added, the archive workflow will be complete.

## 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\Lab09\StarterFiles\DocumentArchive Part 4.sln**.

## Exercise 1: Creating a Custom Instantiation Form

### Add a new base class named **DocArchivePart4InitForm** that will be uses as the code behind for an ASPX page.

#### Create a new class named **DocArchivePart4InitForm** in the **UI** folder.

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

##### Name the new file **DocArchivePart4InitForm.cs** and click **Add**.

#### Change the **DocArchivePart4InitForm** class into a public class that derives from **Microsoft.SharePoint.WebControls.LayoutsPageBase**.

public abstract class DocArchivePart4InitForm : LayoutsPageBase

{

}

#### Add private fields to the class to store information about the workflow association to be started as well as the list item the workflow instance will be attached to.

private SPWorkflowAssociation \_workflowAssociation;

private SPList \_list;

private SPListItem \_listItem;

### Add a **protected** field and property that will allow interaction between the ASPX page and the code behind.

#### Add a **protected** field named **ppkApprover** of type **PeopleEditor** to the class.

#### Add a **protected** field named **lstArchiveList** of type **DropDownList** to the class.

#### Add a **protected** property named **WorkflowName** of type string to the class. Have it return the **Name** property of the \_**workflowAssociation** field.

#### Add a **protected** property named **ItemName** of type string to the class. Have it return the **DisplayName** property of the \_**listItem** field.

protected PeopleEditor ppkApprover;

protected DropDownList lstArchiveList;

protected string WorkflowName

{

get { return \_workflowAssociation.Name; }

}

protected string ItemName

{

get { return \_listItem.DisplayName; }

}

### Override the **OnLoad** method in the new **DocArchivePart4InitForm** class so it reads the **Url** parameters and creates the necessary SharePoint API objects and binds that data to the UI.

#### Override the **OnLoad** method of the base class.

protected override void OnLoad(EventArgs e)

{ }

#### Retrieve the list id, list item id, content type id, and the workflow template id from the URL parameters list.

// read the form level parameters

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

string listItemId = Request.Params["ID"];

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

string templateId = Request.Params["TemplateId"];

#### Read the **List** and **ID** url parameters and use them to find the list and list item to use when attaching the new workflow instance.

// find the list and list item

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

\_listItem = \_list.GetItemById(int.Parse(listItemId));

#### Check if the workflow association can be found in the list’s collection of workflow associations accessed using the **WorkflowAssociations** property.

// check for the workflow association in the list

\_workflowAssociation = \_list.WorkflowAssociations[new Guid(templateId)];

#### If the workflow association was not found in the list’s **WokflowAssociations** collection, find the content type the workflow association is connected to and check its **WorkflowAssociations** collection.

##### Access the content types using the **ctype** URL parameter and the **ContentTypes** property of **SPList**.

##### Once the **SPContentType** object is found, use its **WorkflowAssociations** collection to lookup the workflow association.

// if the association wasn't found, check the content type

if (\_workflowAssociation == null)

{

SPContentType \_contentType =

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

\_workflowAssociation =

\_contentType.WorkflowAssociations[new Guid(templateId)];

}

#### In the event that a workflow association could not be found in either of the locations, throw a new **SPException** notifying the user of the problem.

// make sure the workflow association could be found

if (\_workflowAssociation == null)

throw new SPException("The requested workflow could not be found.");

### If this request is not a postback, populate the **lstArchiveList** controls with all non-hidden document libraries in the current web.

#### Check if the request is a postback using the **Page.IsPostBack** property. Inside this **if** statement is where all of the UI initialization will be done.

// populate the UI

if (!this.IsPostBack)

{

}

#### Using the current site’s **Lists** collection and some filtering, populate the **lstArchiveList** drop down list with every non-hidden document library on the web.

// 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()));

#### Deserialize the association data related to workflow association using the **AssocInitData.Deserialize** method. Now attach the values in the association data to the controls to provide defaults for the form controls.

// populate the form using the association data

AssocInitData assocInitData = AssocInitData.Deserialize(\_workflowAssociation.AssociationData);

ppkApprover.CommaSeparatedAccounts = assocInitData.ApproverUserName;

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

#### Finish up the **OnLoad** method by calling the base implementation of the **OnLoad** method

// call the base implementation

base.OnLoad(e);

### Implement the event handler called after a click of the **Cancel** button.

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

#### Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list’s default view url.

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

{

// redirect to the page in the source url parameter or the default page

SPUtility.Redirect(\_list.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

}

### Implement the event handler called after a click of the **Start** button.

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

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

{

}

#### Create a new **AssocInitData** object and populate it using the values currently stored in the form controls and serialize it using the **Serialize** method.

// populate the association data using the UI

AssocInitData assocInitData = new AssocInitData();

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

assocInitData.ArchiveListId = lstArchiveList.SelectedValue;

// get the initiation data

string initiationData = assocInitData.Serialize();

#### Start a new workflow instance using the **SPWorkflowManager.StartMethod**.

##### The **SPWorkflowManager** is accessible through the current’s **SPWeb** object’s **SPSite** parent object.

// start the new workflow instance

Web.Site.WorkflowManager.StartWorkflow(\_listItem,

\_workflowAssociation, initiationData);

#### Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list’s default view url.

// redirect to the list default view

SPUtility.Redirect(\_list.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

### Create a new .ASPX page named **DocArchivePart4InitForm.aspx** that represents the ASP.NET markup for the task page.

#### Create a new text file named **DocArchivePart4InitForm.aspx** in the **Layouts** folder.

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

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

##### Name the new file **DocArchivePart4InitForm.aspx** and click **Add**.

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

<%@ Assembly Name="DocumentArchiveWorkflowPart4, Version=1.0.0.0,

Culture=neutral, PublicKeyToken=15812f954569663f" %>

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

EnableSessionState="true" ValidateRequest="False"

Inherits="DocumentArchiveWorkflowPart4.UI.DocArchivePart4InitForm" %>

#### 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">

Start Workflow</asp:Content>

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

runat="server">

<%= "Start \"" + this.WorkflowName + "\": " + this.ItemName %></asp:Content>

### Define the ASPX markup that will be used 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 primary 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"

runat="server"

SelectionSet="User"

width='300px' />

</Template\_Control>

</wssuc:InputFormControl>

</template\_inputformcontrols>

</wssuc:InputFormSection>

#### Implement the second **InputFormSection** that will allow the approver to choose the default list where the document will be archived.

<%-- 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 **Start** and **Cancel** buttons.

<%-- Start 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="Start\_Click" Text="Start" 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 2: Integration Instantiation Forms into the Workflow

### Update the workflow to use the initiation data instead of the association data.

#### Open **Workflow.cs** as code by right clicking it in the **Solution Explorer** and clicking **View Code**.

#### Locate the **OnWorkflowActivated\_Invoked** method.

#### This method uses **WorkflowProperties.AssociationData** to initialize the workflow. Change the code to use the **InitiationData** property instead.

private void OnWorkflowActivated\_Invoked(object sender, ExternalDataEventArgs e)

{

// deserialize init data, if none use defaults

this.AssocInitData = AssocInitData.Deserialize(

WorkflowProperties.InitiationData);

}

## Exercise 3: Deploying custom Instantiation Forms

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

<Workflow ...

InstantiationUrl="\_layouts/DocArchivePart4InitForm.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.

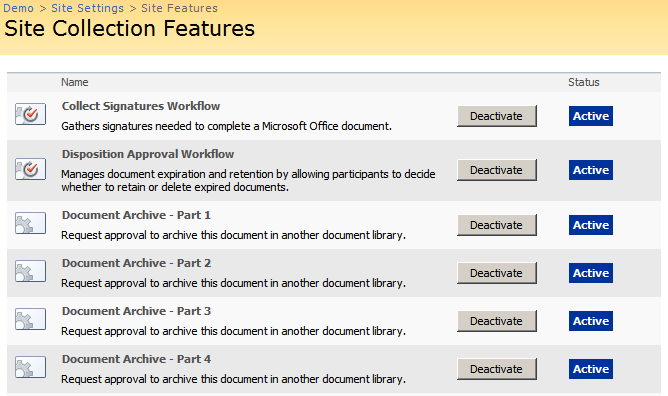
### Activate the new **DocumentArchiveWorkflowPart4** feature.

#### 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 4** feature.



### Create an association between the **Shared Documents** document library and the new **Document Archive** **– Part 4** workflow.

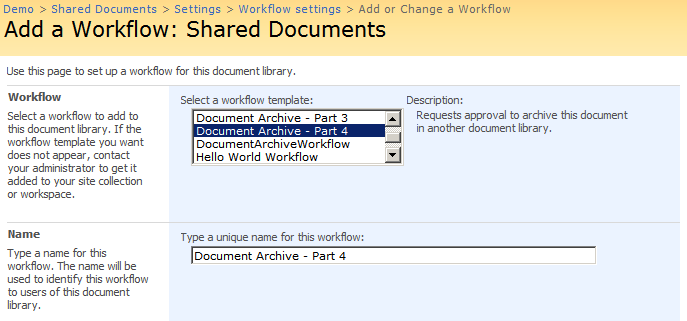
#### Navigate to the **Shared Documents** document library in the **Demos** site.

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

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

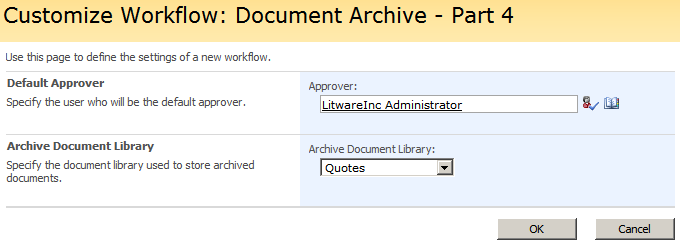
#### Create a new workflow using the **Document Archive – Part 4** workflow template and a name of **Document Archive - Part 4**.

##### 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.



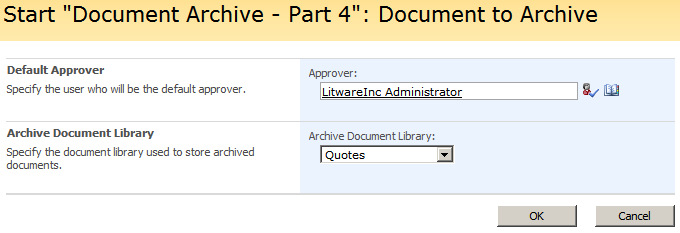
### Run the **Document Archive – Part 4** workflow on a document.

#### 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 4** to start the workflow.

#### In the custom initiation page, verify the form initialized using the data provided in the custom association form.

#### Change any values you would like and click **Start** to start a new instance of the workflow.



### Complete the workflow by approving the task in the workflow status page.

## Exercise 4: Creating a Custom Modification Form

### Add a new base class named **DocArchivePart4ModForm** that will be uses as the code behind for an ASPX page.

#### Create a new class named **DocArchivePart4ModForm** in the **UI** folder.

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

##### Name the new file **DocArchivePart4ModForm.cs** and click **Add**.

#### Change the **DocArchivePart4InitForm** class into a public class that derives from **Microsoft.SharePoint.WebControls.LayoutsPageBase**.

public abstract class DocArchivePart4ModForm : LayoutsPageBase

{ }

#### Add private fields to the class to store information about the workflow modification as well as the list item the workflow instance being modified.

private SPList \_list;

private SPListItem \_listItem;

private SPWorkflow \_workflow;

private SPWorkflowModification \_modification;

### Add a protected field and property that will allow interaction between the ASPX page and the code behind.

#### Add a **protected** field named **pplApprover** of type **PeopleEditor** to the class.

protected PeopleEditor pplApprover;

protected PeopleEditor pplManager;

### Override the **OnLoad** method in the new **DocArchivePart4InitForm** class so it reads the **Url** parameters and creates the necessary SharePoint API objects and binds that data to the UI.

#### Override the **OnLoad** method of the base class.

protected override void OnLoad(EventArgs e)

{ }

#### Retrieve the list id, list item id, workflow id, and the modification id from the URL parameters list.

// read the form level parameters

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

string listItemId = Request.Params["ID"];

string workflowId = Request.Params["WorkflowInstanceID"];

string modificationId = Request.Params["ModificationID"];

#### Use the IDs retrieved from the URL to load the related SharePoint API objects.

// find the list and workflow modification objects

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

\_listItem = \_list.GetItemById(Convert.ToInt32(listItemId));

\_workflow = \_listItem.Workflows[new Guid(workflowId)];

\_modification = \_workflow.Modifications[new Guid(modificationId)];

#### Check if the request is a postback using the **Page.IsPostBack** property. Inside this **if** statement is where all of the UI initialization will be done.

// populate the UI

if (!this.IsPostBack)

{

}

#### Deserialize the modification data related to workflow association using the **ModificationData.Deserialize** method and attach the data to the form controls.

##### Read the modification data from the **SPWorkflowModification.ContextData** property.

// deserialize the association data

ModificationData modData = ModificationData.Deserialize(\_modification.ContextData);

// bind the controls to the association data

pplApprover.CommaSeparatedAccounts = modData.ApproverUserName;

#### Finish up the **OnLoad** method by calling the base implementation of the **OnLoad** method

// call the base implementation

base.OnLoad(e);

### Implement the event handler called after a click of the **Cancel** button.

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

#### Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list’s default view url.

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

{

// redirect to the page in the source url parameter or the default page

SPUtility.Redirect(\_list.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

}

### Implement the event handler that will be called when the **Start** button is clicked.

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

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

{

}

#### Create a new **AssocInitData** object and populate it using the values currently stored in the form controls and serialize it using the **Serialize** method.

// populate the modification data using the UI

ModificationData modData = new ModificationData();

modData.ApproverUserName = (pplApprover.Entities[0] as PickerEntity).Key;

// serialize the modification data

string modificationData = modData.Serialize();

#### Modify the current workflow instance using the **SPWorkflowManager.ModifyWorkflow** method.

##### The **SPWorkflowManager** is accessible through the current’s **SPWeb** object’s **SPSite** parent object.

// apply the modification to the workflow

Web.Site.WorkflowManager.ModifyWorkflow(\_workflow,

\_modification,

modificationData);

#### Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list’s default view url.

// redirect to the page defnined in the source url parameter or the default page

SPUtility.Redirect(\_list.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

### Create a new .ASPX page named **DocArchivePart4ModForm.aspx** that represents the ASP.NET markup for the task page.

#### Create a new text file named **DocArchivePart4ModForm.aspx** in the **Layouts** folder.

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

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

##### Name the new file **DocArchivePart4ModForm.aspx** and click **Add**.

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

<%@ Assembly Name="DocumentArchiveWorkflowPart4, Version=1.0.0.0,

Culture=neutral, PublicKeyToken=15812f954569663f" %>

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

EnableSessionState="true" ValidateRequest="False"

Inherits="DocumentArchiveWorkflowPart4.UI.DocArchivePart4ModForm" %>

#### Register the tag prefix and locations of the SharePoint web and user controls 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">

Modify Workflow</asp:Content>

<asp:Content ID="PageTitleInTitleArea"

ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">

<%= "Modify" %></asp:Content>

### Define the ASPX markup that will be used 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 allows modification of the approver.

<%-- 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="pplApprover"

AllowEmpty="false"

ValidatorEnabled="true"

MultiSelect="false"

runat="server"

SelectionSet="User"

width='300px' />

</Template\_Control>

</wssuc:InputFormControl>

</template\_inputformcontrols>

</wssuc:InputFormSection>

#### Implement the last section containing the **OK** 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 5: Enabling Modifications in the Workflow

### Communicate with the SharePoint Workflow hosting environment to allow modification of this workflow.

#### Open **Workflow.cs** in the designer by right clicking it in the **Solution Explorer** and selecting **View Designer**.

#### Double click the **WorkflowActivatedActivity** activity to open its designer.

#### Drag an **EnableWorkflowModification** activity from the toolbox into the **WorkflowActivatedActivity** immediately following the **OnWorkflowActivated** activity and name it **EnableModification**..

##### In the properties pane, set the **Name** property to **EnableModification**.

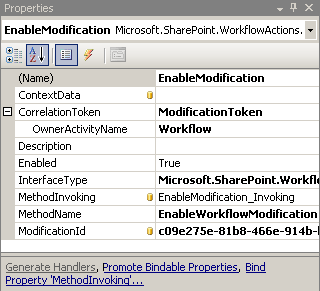
#### Assign a correlation token to the modification and assign the id of the modification to enable.

##### In the properties pane, set the **CorrelationToken** property to **ModificationToken**.

##### Expand the **CorrelationToken** property in the property pane.

##### Set the **OwnerActivityName** property to **Workflow**

##### Set the **ModificationId** property to **c09e275e-81b8-466e-914b-bae95eb27dfa**.



#### Implement the **Invoking** method and use it to serialize a **ModificationData** class into the **ContextData** of the current activity.

##### In the properties pane, set the **Invoking** property to **EnableModification\_Invoking** and press **Enter**.

private void EnableModification\_Invoking(object sender, EventArgs e)

{

// populate the modification data object

ModificationData modData = new ModificationData();

modData.ApproverUserName = AssocInitData.ApproverUserName;

// serialize and assign the context data to the modification

EnableWorkflowModification activity = sender as EnableWorkflowModification;

activity.ContextData = modData.Serialize();

}

### Add an **EventDrivenActivity** to the workflow that will receive the notification the workflow has been modified. When that happens, it will update the **AssocInitData** object and reset any tasks already created.

#### On the **State Machine** designer canvas, right click and select **Add EventDriven**.

#### Rename the new event driven activity to **WorkflowModifiedActivity**.

##### In the properties pane, set the **Name** property to **WorkflowModifiedActivity**.

#### Drag an **OnWorkflowModified** activity from the toolbox into the **WorkflowModifiedActivity activity** and name it **OnWorkflowModified**..

##### In the properties pane, set the **Name** property to **OnWorkflowModified**.

#### Update the new activity to respond to the modification previously enabled.

##### Set the **CorrelationToken** property to **ModificationToken**.

##### Set the **ModificationId** property to **c09e275e-81b8-466e-914b-bae95eb27dfa**.

#### Implement the **Invoked** event and retrieve the updated modification data for integration back into the running workflow.

##### Set the **Invoked** property to **OnWorkflowModified\_Invoked** and press **Enter**.

##### Cast the **e** parameter to **SPModificationEventArgs** to retrieve the modification data.

##### Deserialize using **ModificationData.Deserialize** and copy the new data into the **AssocInitData** object.

private void OnWorkflowModified\_Invoked(object sender, ExternalDataEventArgs e)

{

SPModificationEventArgs args = e as SPModificationEventArgs;

// deserialize the modification data and apply the changes

ModificationData modData = ModificationData.Deserialize(args.data);

AssocInitData.ApproverUserName = modData.ApproverUserName;

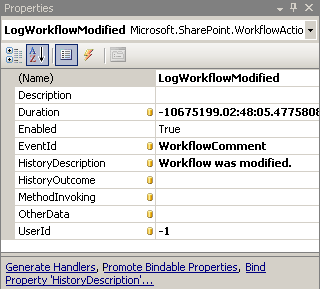
}

### Log the fact that the modification has taken place and transition back to the **WaitingForApproval** state.

#### Drag a **LogToHistoryListActivity** from the toolbox into the **WorkflowModifiedActivity** activityand name it **LogWorkflowModified**.

##### In the properties pane, set the **Name** property to **LogWorkflowModified.**

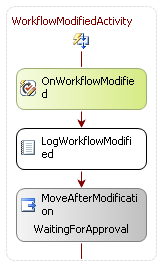
##### Set the **HistoryDescription** property to **“Workflow was modified.”**



#### Drag a **SetState** activity from the toolbox into the **WorkflowModifiedActivity** at the end of the sequence and name it **MoveAfterModification**.

##### In the properties pane, set the Name property to **MoveAfterModification**.

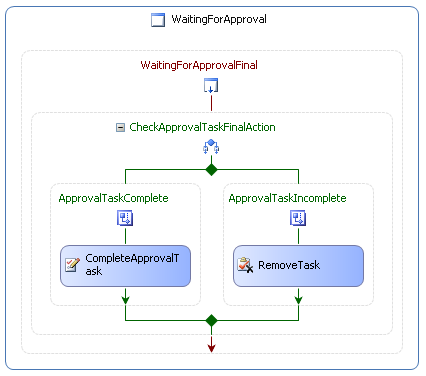
##### Set the **TargetStateName** property to **WaitingForApproval**.



### Since modifications will happen when tasks already exist, they will need to be recreated after the modification. To do this, the finalizer and initializer for the state come into play.

#### Open the **WaitingForApprovalFinal** activity in the State Machine designer canvas.

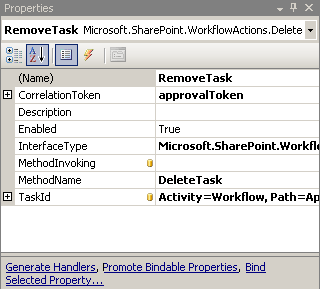
#### Drag a **DeleteTask** activity from the toolbox into the **ApprovalTaskIncomplete** branch and name it **RemoveTask**



#### Assign the correlation token and **TaskId** necessary for the activity to delete the task.

##### In the properties pane, set the **CorrelationToken** property to **approvalToken**.

##### Bind the **TaskId** property to the **ApprovalTaskId** property in the workflow class.



## Exercise 6: Deploying custom Modification Forms

### Register the new custom modification form by assigning the **ModificationUrl** attribute in the feature’s **Workflow** element and registering the modification metadata.

#### Open the **workflow.xml** file in the **Template/Features/DocumentArchiveWorkflowPart4** folder.

#### Add the **ModificationUrl** attribute to the **Workflow** element. Point it toward **\_layouts/DocArchivePart4ModForm.aspx**.

<Workflow ...

ModificationUrl="\_layouts/DocArchivePart4ModForm.aspx"

StatusUrl="\_layouts/WrkStat.aspx">

<Categories/>

<MetaData>

</MetaData>

</Workflow>

#### Add an element in the Metadata that defines the display name for the modification enabled in the workflow.

##### Use the format **Modification\_{0}\_Name** where the **{0}** token is replaced by the modification’s id.

<MetaData>

<Modification\_c09e275e-81b8-466e-914b-bae95eb27dfa\_Name>Change approvers</Modification\_c09e275e-81b8-466e-914b-bae95eb27dfa\_Name>

</MetaData>

### 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.

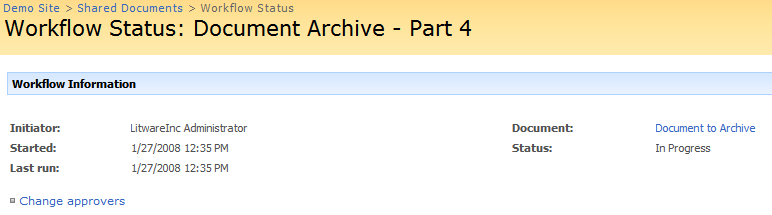
### Run the **Document Archive – Part 4** workflow on a document.

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

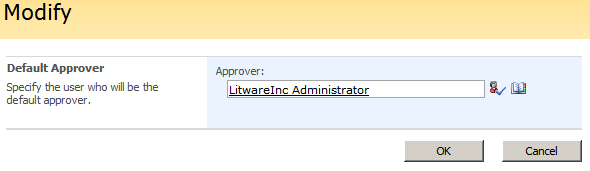
#### In the workflows page, click the **Document Archive – Part 4** to start the workflow.

#### When the custom instantiation page opens, click **Start** to create the workflow instance with the default values.

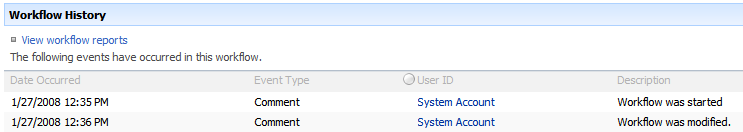
#### In the Shared Documents UI, click the In Progress link to view the status of the workflow. Click the **Change approvers** link in the Workflow Information section to modify the workflow.



#### In the modification page, do not change anything, but click **Ok** to apply the modification.



#### In the workflow status page, verify that there is a modification message in the workflow history.



### Complete the workflow by approving the task.

## Challenge: Integrate Instantiation and Modification 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\DocArchivePart4TaskForm.aspx"/>

<TemplateFile Location="LAYOUTS\DocArchivePart4AssocForm.aspx"/>

<TemplateFile Location="LAYOUTS\DocArchivePart4InitForm.aspx"/>

<TemplateFile Location="LAYOUTS\DocArchivePart4ModForm.aspx"/>

</TemplateFiles>

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

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

#### Add lines that package the **DocArchivePart4InitForm.aspx** and **DocArchivePart4ModForm.aspx** and puts it in the **LAYOUTS** folder in the .cab file.

..\..\Template\Layouts\DocArchivePart4InitForm.aspx LAYOUTS\DocArchivePart4InitForm.aspx

..\..\Template\Layouts\DocArchivePart4ModForm.aspx LAYOUTS\DocArchivePart4ModForm.aspx

### 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 **DocuentArchiveWorkflowPart4** feature.

stsadm –o uninstallfeature –name DocumentArchiveWorkflowPart4 –force

#### Delete the folder at **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Features\DocumentarchiveWorkflowPart4**.

#### Delete the **DocArchivePart4AssocForm.aspx, DocArchivePart4InitForm.aspx, DocArchivePart4TaskForm.aspx,** and **DocArchivePart4ModForm.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\Lab09\StarterFiles\DocumentArchiveWorkflowPart4\bin\Debug\Package\DocumentArchiveWorkflowPart4.wsp”

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

stsadm –o deploysolution –name DocumentArchiveWorkflowPart4.wsp -allowgacdeployment –local

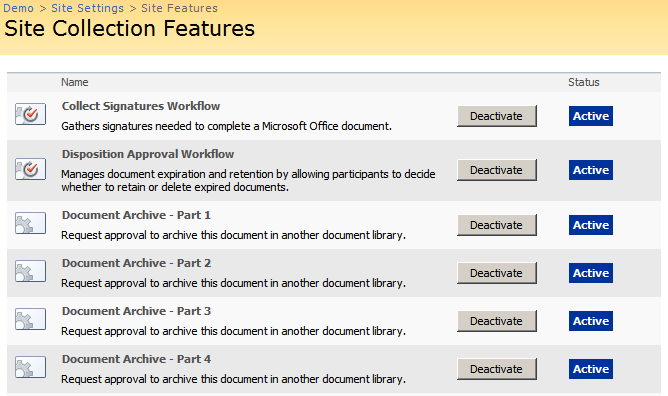
### Verify the feature is still available and activated.

#### 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 4** feature is active.



## Challenge:

This scenario implies that the file will be moved from one location to another. For the purposes of simplicity, this action is not outlined in the labs. For an additional challenge, use a **Code Activity** to add code to the workflow that will copy the file to the appropriate list. To gain access to the current item, use the **WorkflowProperties** property of the workflow, look at the **AssocInitData** property to find information about the archive folder and finally use the **SPListItem.CopyTo** method to perform the copy.