# Lab 07: Creating and Waiting on SharePoint Tasks

**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 extending what you have done in the previous lab. Previously you built the first part of the workflow that simply executed and made an entry into the history list. In this lab, you will be extending that example to include tasks. When the workflow starts, it will create a new task for a user. The workflow will then wait for the task to complete before continuing. Once the tasks are integrated you will create a custom task form that will change the user experience when editing a task from a standard task edit form to a custom form you create.

## 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\Lab07\StarterFiles\DocumentArchive Part 2.sln**.

## Exercise 1: Creating SharePoint Tasks in a Workflow

### Add a new state named **WaitingForApproval** to the workflow and change **WorkflowActivatedActivity** so it transitions to the **WaitingForApproval** state instead of the **Completed** state.

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

#### Right click on the **State Machine** designer canvas and click **Add State**.

#### Set the name of the new state to **WaitingForApproval**.

##### Right click the new state and click **Properties**.

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

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

#### Change the name of **MoveToCompleted** state to **MovingToWaitingForApproval**.

##### Right click the **MoveToCompleted** activity and click **Properties**.

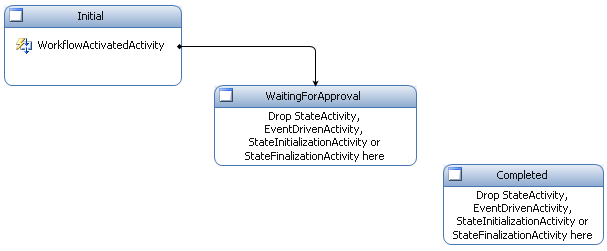
##### In the properties pane, set the **Name** property to **MoveToWaitingForApproval**.

#### Change the new **MoveToWaitingForApproval** activity’s **TargetStateName** to **WaitingForApproval**.

##### Right click the **MoveToWaitingForApproval** activity and click **Properties**.

##### In the properties pane, set the **TargetStateName** property to **WaitingForApproval**.

#### Switch back to the **Workflow** view by clicking the **Workflow** link in the upper left hand corner of the State Machine designer canvas.

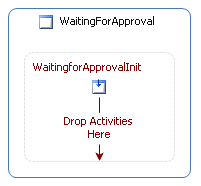


### Add a new **StateInitializationActivity** to the **WaitingForApproval** state and name if **WaitingForApprovalInit**.

#### Right click the **WaitingForApproval** state and click **Add StateInitialization**.

#### Set the name of the new **StateInitializationActivity** to **WaitingForApprovalInit**.

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



### Create a new task in the **WaitingForApprovalInit** activity.

#### Drag a new **CreateTask** activity from the Toolbox into the **WaitingForAprovalInit** activity and name it **CreateApprovalTask**

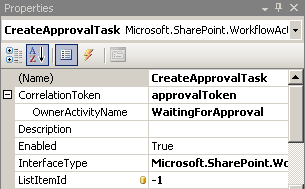
##### In the properties pane, set the **Name** property to **CreateApprovalTask**.

#### In the **CreateApprovalTask’s** properties pane, create a new correlation token scoped to the **WaitingFormApproval** state named **ApprovalToken**.

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

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

##### Set the **OwnerActivityName** property to **WaitingForApproval**

.

#### Add a **TaskId** property to the workflow class to store the Id of the new task.

##### View the code for the **Workflow** class by right clicking it in the **Solution Explorer** and selecting **View Code**.

##### Add a public property named **ApprovalTaskId** to the class with a type of **Guid**.

public Guid ApprovalTaskId { get; set; }

#### Bind the **TaskId** property of the **CreateApprovalTask** activity to the **ApprovalTaskId** property of the workflow.

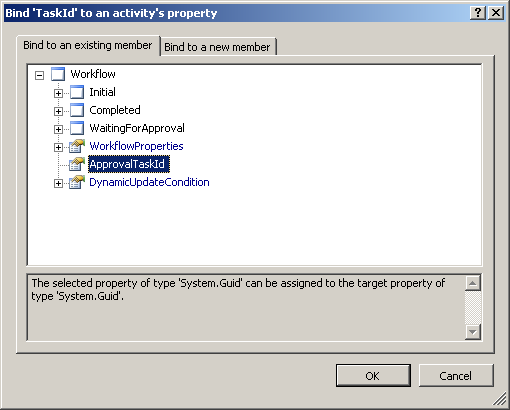
##### View the workflow designer by clicking **Workflow.cs** in the **Solution Explorer** and selecting **View Designer**.

##### Open the **WaitForApprovalInit** activity by double clicking it.

##### Right click the **CreateApprovaltask** activity and click **Properties**.

##### Open the **Binding** dialog by selecting the **TaskId** property and clicking the **…** button.

##### In the **Binding** dialog select the **ApprovalTaskId** property and click **OK**.



#### Create a handler for the **MethodInvoking** event that will set the details of the new task to create.

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

#### Initialize **ApprovalTaskId** and set the details of the **TaskProperties** property of the **CreateApprovalTask** activity.

##### In the **CreateApprovalTask\_Invoking** method, initialize the value of **ApprovalTaskId** to the results of the **Guid.NewGuid** method.

##### Create a new **SPWorkflowTaskProperties** object named **taskProperties**.

##### Set its **AssignedTo** property to **Administrator**.

##### Set its **Title** to “**Archive Approval**”.

##### Set its **Description** to “**This document was flagged to be archived. Please approve or reject this action.**”

ApprovalTaskId = Guid.NewGuid();

SPWorkflowTaskProperties taskProperties = new SPWorkflowTaskProperties();

taskProperties.AssignedTo = "LITWAREINC\\Administrator";

taskProperties.Title = "Archival Approval";

taskProperties.Description = "This document was flagged to be archived. Please approve or reject this action.";

#### Assign the **taskProperties** object to the **CreateTask** activity that is currently executing.

##### Cast the **sender** property to a **CreateTask** object and store the result in a **createApprovalTask** variable.

##### Set **createApprovalTask’s** **TaskProperties** property using the **taskProperties** variable.

CreateTask createApprovalTask = sender as CreateTask;

createApprovalTask.TaskProperties = taskProperties;

### Add a new **EventDrivenActivity** named **WaitingForApprovalTaskChanged** to the **WaitingForApproval** state.

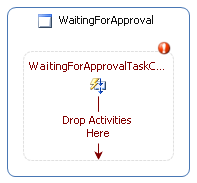
#### Right click **Workflow.cs** in the **Solution Explorer** and click **View Designer**.

#### Right click the **WaitingForApproval** state and click **Add EventDriven**.

#### Set the name of the new **EventDrivenActivity** to **WaitingForApprovalTaskChanged**.

##### Right click the new **EventDrivenActivity** and click **Properties**.

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



### Create a new **ApprovalTaskResult** property to the workflow to use when storing the result of the activity.

#### To store task results efficiently, create a new **enum** named **TaskResult** to identify the task result.

##### Right click on the project in the **Solution Explorer** and click **Add -> Class**.

##### Name the class **TaskResults.cs** and click **Add**.

##### Replace the class definition with a public **enum** definition named **TaskResults** with options for **None** and **Approved**.

public enum TaskResults

{

None,

Approved,

Rejected

}

#### Open the **Workflow.cs** code and add a new public property named **ApprovalTaskResult** of type **TaskResult**.

##### Right click on the **Workflow.cs** workflow in the **Solution Explorer** and click **View Code**.

##### Add a new property named **ApprovalTaskResult** of type **TaskResult** to the class.

public Guid ApprovalTaskId { get; set; }

public TaskResults ApprovalTaskResult { get; set; }

### Add an **OnTaskChanged** event to the **WaitingForApproval** state that waits for the previously created task to change.

#### Drag a new **OnTaskChanged** activity from the **Toolbox** into the **WaitingForApprovalTaskChanged** activity and name it **OnApprovalTaskChanged.**

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

#### In the **OnApprovalTaskChanged** properties pane, select the existing **ApprovalToken** correlation token.

##### Right click the **OnApprovalTaskChanged** activity and click **Properties**.

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

#### In the **OnApprovalTaskChanged** properties pane, bind **TaskId** to **ApprovalTaskId**.

##### Open the **Binding** dialog by selecting the **TaskId** property and clicking the **…** button.

##### In the **Binding** dialog select the **ApprovalTaskId** property and click **OK**.

### When the task changes, check if the status is **Completed**.

#### Add an **Invoked** handler to the **OnApprovalTaskChanged** activity.

##### Right click the **OnApprovalTaskChanged** activity and click **Properties**.

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

#### In the **OnApprovalTaskChanged\_Invoked** method cast the **e** parameter to **SPTaskServiceEventArgs** to access the activity specific event parameters and store it for later in the **args** variable.

SPTaskServiceEventArgs args = e as SPTaskServiceEventArgs;

#### Access the **Status** column of the task by accessing it in the **args’s** **afterProperties.ExtendedProperties** property. Use the Guid ID of the column to retrieve its value.

Guid statusFieldId = new Guid("{c15b34c3-ce7d-490a-b133-3f4de8801b76}");

string result = args.afterProperties.ExtendedProperties[statusFieldId] as string;

#### If the result of the task change is **Completed**, set the **ApprovalTaskResult** property to **TaskResults.Approved**.

if (result == "Completed")

this.ApprovalTaskResult = TaskResults.Approved;

### If the task change indicated that the task was completed, log the result and transition to the completed state.

#### Add a new **IfElse** activity to the **WaitingForApprovalTaskChanged** activity with the name of **CheckApprovalTaskResult**.

##### Drag a new **IfElse** activity from the **Toolbox** into the **WaitingForApprovalTaskChanged** following the **OnApprovalTaskChanged** activity.

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

#### Set the left branch of the **CheckApprovalTaskResult** activity’s name to **ApprovalTaskApproved** and setup its condition.

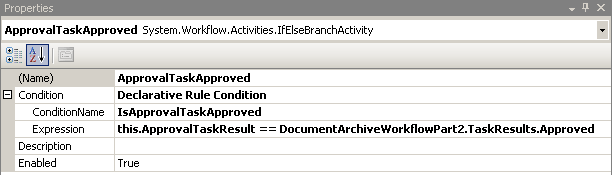
##### In the properties pane, set the **Name** property to **ApprovalTaskApproved**.

##### Set the **Condition** property to **Declarative Rule Condition** and expand the property.

##### Set the **ConditionName** property to **IsApprovalTaskApproved**.

##### Set the **Expression** property to:

this.ApprovalTaskResult == TaskResults.Approved



#### Set the right branch of the **CheckApprovalTaskResult** activity’s name to **ApprovalTaskInProgress**.

##### Right click the right branch of **CheckApprovalTaskResult** and click **Properties**.

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

#### Add a new **LogToHistoryListActivity** to the **ApprovalTaskApproved** activity and name it **LogApprovalTaskApproved**.

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

##### Set the **HistoryDescription** property to **“Approval task approved.”**

#### Add a new **SetState** activity to the **ApprovalTaskApproved** activity following the **LogApprovalTaskApproved** and name it **MoveOnApprovalTaskApproved**.

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

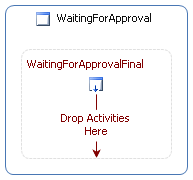
##### Set the **TargetStateName** to **Completed**.

### Add a new **StateFinalizationActivity** to the **WaitingForApproval** state and name it **WaitingForApprovalFinal**.

#### Add a **StateFinalization** activity to the **WaitingForApproval** state and name it **WaitingForApprovalFinal**

##### Right click the **WaitingForApproval** state and select **Add StateFinalization**

##### Right click the new **StateFinalization** activity and set its name to **WaitingForApprovalFinal**.



### In the **WaitingForApprovalFinal** activity, if the activity was completed, use the **CompleteTask** activity to mark the task as completed.

#### Add a new **IfElse** activity to the **WaitingForApprovalTaskFinal** activity with the name of **CheckApprovalTaskFinalAction**.

##### Drag a new **IfElse** activity from the **Toolbox** into the **WaitingForApprovalFinal**.

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

#### Set the left branch of the **CheckApprovalTaskFinalAction** activity’s name to **ApprovalTaskComplete** and setup its condition.

##### Right click the left branch of **CheckApprovalTaskFinalAction** and click **Properties**.

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

##### Set the **Condition** property to **Declarative Rule Condition** and expand the property.

##### Set the **ConditionName** property to **IsApprovalTaskComplete**.

##### Set the **Expression** property to

this.ApprovalTaskResult != TaskResults.None

#### Set the right branch of the **CheckApprovalTaskFinalAction** activity’s name to **ApprovalTaskIncomplete**.

##### Right click the right branch of **CheckApprovalTaskFinalAction** and click **Properties**.

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

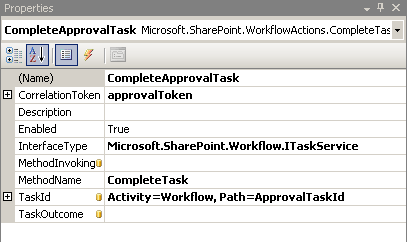
#### Mark the task as complete using the **CompleteTask** activity if the task was determined to be complete.

##### Drag a new **CompleteTask** activity into the **ApprovalTaskComplete** activity.

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

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

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



### 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.

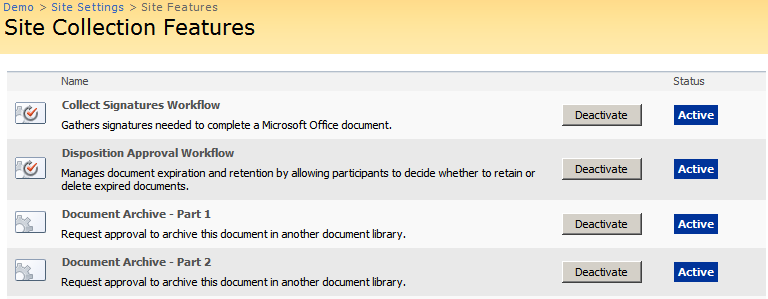
### Activate the new **DocumentArchiveWorkflow** feature.

#### 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 2** feature.



### Create an association between the **Shared Documents** document library and the new **Document Archive** **– Part 2** workflow.

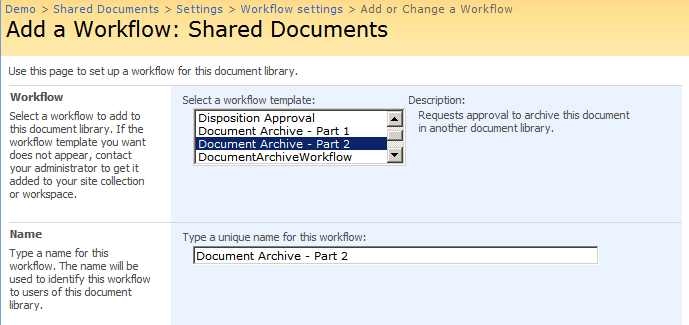
#### Navigate to the **Shared Documents** document library in the **Demos** site.

#### 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 2** workflow template and a name of **Document Archive - Part 2**.

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



### Run the **Document Archive – Part 2** workflow on a document.

#### Hover over the new document and select **Workflows** from the drop down menu.

#### In the workflows page, click the **Document Archive – Part 2** to start the workflow.

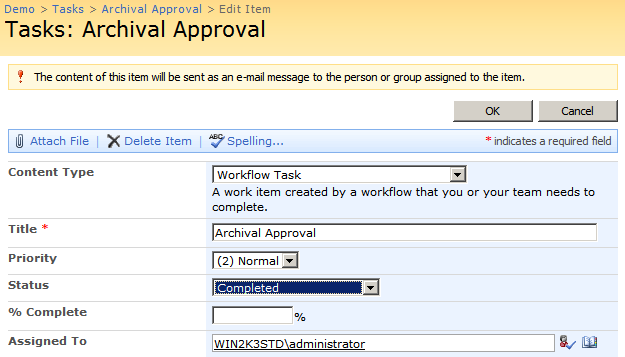
#### In the **Shared Documents** document library, verify the workflow is running.

#### 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**.

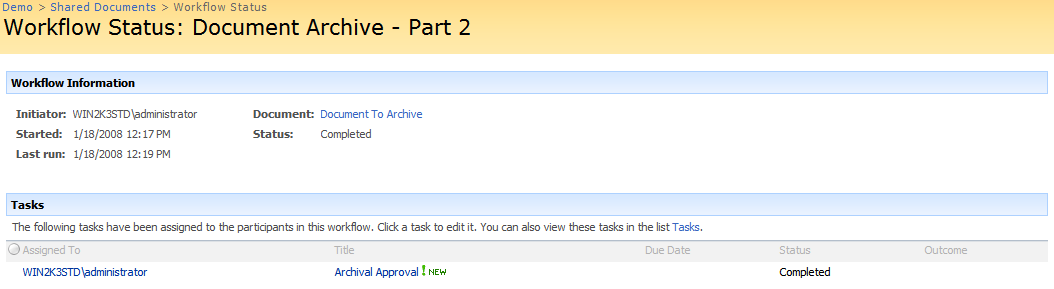
##### In the **Tasks** section, hover over the task’s title and click **Edit Item** in the drop down menu.

##### In the **Edit Task** page set its **Status** to **Completed** and click **Save**.



#### 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.



## Exercise 2: Creating custom Task Forms

### Add a new base class named **DocArchivePart2TaskForm** that will be uses as the code behind for an ASPX page.

#### Create a new folder named **UI** under the project.

##### Right click on the project in the **Solution Explorer** and click **Add -> New Folder**.

##### Right click the folder and click **Rename**.

##### Rename the folder **UI**.

#### Create a new class named **DocArchivePart2TaskForm** in the new UI folder.

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

##### Name the new file **DocArchivePart2TaskForm.cs** and click **Add**.

#### Change the **DocArchivePart2TaskForm** class into a public class that derives from **Microsoft.SharePoint.WebControls.LayoutsPageBase**.

public abstract class DocArchivePart2TaskForm : LayoutsPageBase

{ }

#### Add private fields to the class to store information about the task item, the workflow, and the list the workflow is attached to.

private SPWorkflow \_workflow;

private SPWorkflowTask \_task;

private SPList \_taskList;

private SPListItem \_taskItem;

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

#### Add a protected field named **lblListName** of type **Label** to the class.

#### Add a protected field named **lnkItem** of type **Hyperlink** to the class.

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

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

protected Label lblListName;

protected HyperLink lnkItem;

protected DropDownList lstArchiveList;

protected string TaskItemName

{

get { return \_taskItem.DisplayName; }

}

### Override the **OnLoad** method in the new **DocArchivePart2TaskForm** 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)

{

}

#### Read the **List** and **ID** url parameters and use them to find the task list and task list item to edit.

// find the tasks list and task list item

\_taskList = Web.Lists[new Guid(Request.Params["List"])];

\_taskItem = \_taskList.GetItemById(int.Parse(Request.Params["ID"]));

#### Using the \_**taskItem** object’s **WorkflowInstanceID** field, find the **SPWokflow** object representing the running workflow.

// use the task list item to find the workflow object

\_workflow = new SPWorkflow(Web,

new Guid(\_taskItem["WorkflowInstanceID"] as string));

#### Use the **SPWorkflow** object to lookup the **SPWorkflowTask** object using the **Tasks** collection.

// using the workflow object, lookup the task

\_task = \_workflow.Tasks[0];

#### If this request is not a postback, call the **PopulateControls** and **BindTaskData** methods to give the derived class a chance to populate the form.

// bind the task data

if (!this.IsPostBack)

{

// 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()));

// initialize the controls

lnkItem.Text = \_workflow.ParentItem.DisplayName;

lnkItem.NavigateUrl = \_workflow.ParentItem.Url;

lblListName.Text = \_workflow.ParentList.Title;

}

#### Call 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 method named **Cancel\_Click**.

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

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

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

SPUtility.Redirect(\_taskList.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

}

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

#### Create a **protected** event handler method named **Reject\_Click**.

#### Create a new **Hashtable** object with a single entry named Result. Set its value to **TaskResults.Rejected**.

#### Use the **SPWorkflowTask.AlterTask** to make the changes defined in the **Hashtable** to the task object.

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

protected void Reject\_Click(object sender, EventArgs e) {

// setup the task's new data

Hashtable data = new Hashtable();

data.Add("Result", TaskResults.Rejected);

// submit the changes to the task

SPWorkflowTask.AlterTask(\_taskItem, data, true);

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

SPUtility.Redirect(\_taskList.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

}

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

#### Create a **protected** event handler method named **Approve\_Click**.

#### Copy the code from **Reject\_Click** into **Approve\_Click** and change the result to **TaskResult.Approved**.

#### Add one more parameter to the **Hashtable** named **ArchiveListId** with a value of **lstArchiveList.SelectedValue**.

protected void Approve\_Click(object sender, EventArgs e) {

// setup the task's new data

Hashtable data = new Hashtable();

data.Add("Result", TaskResults.Approved);

data.Add("ArchiveListId", lstArchiveList.SelectedValue);

// submit the changes to the task

SPWorkflowTask.AlterTask(\_taskItem, data, true);

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

SPUtility.Redirect(\_taskList.DefaultViewUrl,

SPRedirectFlags.UseSource, this.Context);

}

### Create a new .ASPX page named **DocArchivePart2TaskForm.aspx** that represents the ASP.NET markup for the task page.

#### Create a new **Layouts** folder under the existing **Template** folder.

##### Right click on the **Template** folder in the solution explorer and click **Add -> New Folder**.

##### Right click the folder and click **Rename**.

##### Rename the folder **Layouts**.

#### Create a new text file named **DocArchivePart2TaskForm.aspx** in the new **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 **DocArchivePart2TaskForm.aspx** and click **Add**.

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

<%@ Assembly Name="DocumentArchiveWorkflowPart2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=15812f954569663f" %>

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

EnableSessionState="true" ValidateRequest="False"

Inherits="DocumentArchiveWorkflowPart2.UI.DocArchivePart2TaskForm" %>

#### 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">Workflow Task</asp:Content>

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

runat="server">

<%= "Tasks : " + this.TaskItemName %>

</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** inside the **table** element that will display summary information about the task. It will contain two **Label** controls with the names **lblListName** and **lnkItem**.

<%-- Hyperlink to item/document --%>

<wssuc:InputFormSection Title="Item Requiring Approval" Description="Please review this item/document."

runat="server">

<template\_inputformcontrols>

<wssuc:InputFormControl LabelText="Click on item hyperlink below to review item/document" runat="server">

<Template\_Control>

<table border="0" cellpadding="3" cellspacing="0" style="font-size:8pt;">

<tr>

<td>List:</td>

<td><asp:Label runat="server" ID="lblListName" /></td>

</tr>

<tr>

<td>Item:</td>

<td><asp:HyperLink runat="Server" ID="lnkItem" Target="\_blank" /></td>

</tr>

</table>

</Template\_Control>

</wssuc:InputFormControl>

</template\_inputformcontrols>

</wssuc:InputFormSection>

#### Implement the second **InputFormSection** that will allow the approver to override the 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 **Approve**, **Reject**, and **Cancel** buttons.

<%-- Approve, Reject 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="Approve\_Click" Text="Approve" id="btnApprove" /> &nbsp;

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

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

</asp:PlaceHolder>

</template\_buttons>

</wssuc:ButtonSection>

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

### Update the **Workflow** code that will execute when a task changes. Use the new **Result** extended property to determine the result of the workflow instead of the **Status** field.

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

#### Locate the **OnApprovaltaskChanged\_Invoked** method and replace the code processing the extended properties with code that looks from the **Result** property and converts it into a **TaskResult** enumeration.

//Guid statusFieldId = new Guid("{c15b34c3-ce7d-490a-b133-3f4de8801b76}");

//string result = args.afterProperties.ExtendedProperties[statusFieldId] as string;

//if (result == "Completed")

// this.ApprovalTaskResult = TaskResults.Approved;

string result = args.afterProperties.ExtendedProperties["Result"] as string;

this.ApprovalTaskResult = (TaskResults)Enum.Parse(typeof(TaskResults), result);

### Add another branch in the **WaitingForApprovalTaskChanged** to handle the Rejected result from the custom task form.

#### Add a new branch named **ApprovalTaskRejected**.to the **CheckApprovalTaskResult** activity.

##### Right click on the **CheckApprovalTaskResult** and click **Add Branch**.

##### Drag the new branch into the middle of the existing branches.

##### In the properties pane, set the **Name** to **ApprovalTaskRejected**.

##### Set the **Condition** property to **Declarative Rule Condition** and expand the property.

##### Set the **ConditionName** property to **IsApprovalTaskRejected**.

##### Set the **Expression** property to

this.ApprovalTaskResult == TaskResults.Rejected

#### Add a new **LogToHistoryListActivity** to the **ApprovalTaskRejected** activity and name it **LogApprovalTaskRejected**.

##### Right click the new **LogToHistoryListActivity** and click **Properties**.

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

##### Set the **HistoryDescription** property to **“Approval task rejected.”**

#### Add a new **SetState** activity to the **ApprovalTaskRejected** activity following the **LogApprovalTaskRejected** and name it **MoveOnApprovalTaskRejected**.

##### Right click the new **SetState** activity and click **Properties**.

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

##### Set the **TargetStateName** to **Completed**.

## Exercise 4: Deploying custom Task Forms

### Create a new content type for the workflow’s tasks.

#### Open the **Workflow.xml** file in the **Template\Features\DocumentArchiveWorkflowPart2** folder.

#### Following the **Workflow** element, add the following XML fragment defining the new content type.

##### The definition of the **Edit** and **Display** forms for the content type determine the form shown when the user views or edits the task.

<ContentType ID="0x01080100F16BC2E95B0E4F1D862D3773F914E20A" Name="DocumentArchiveWokflowPart2 Task" Group="DocArchiveWf Content Types" Description="Task used by the Document Archive workflow." Version="0" Hidden="False">

<FieldRefs>

</FieldRefs>

<XmlDocuments>

<XmlDocument NamespaceURI ="http://schemas.microsoft.com/sharepoint/v3/contenttype/forms/url">

<FormUrls xmlns="http://schemas.microsoft.com/sharepoint/v3/contenttype/forms/url">

<Edit>\_layouts/DocArchivePart2TaskForm.aspx</Edit>

<Display>\_layouts/DocArchivePart2TaskForm.aspx</Display>

</FormUrls>

</XmlDocument>

</XmlDocuments>

</ContentType>

### Register the new content type as the default type for the workflow by adding a **TaskListContentTypeId** attribute to the feature’s **Workflow** element.

<Workflow ...

TaskListContentTypeId="0x01080100F16BC2E95B0E4F1D862D3773F914E20A"

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.

### Run the **Document Archive – Part 2** workflow on a document.

#### Hover over the new document and select **Workflows** from the drop down menu.

#### In the workflows page, click the **Document Archive – Part 2** to start the workflow.

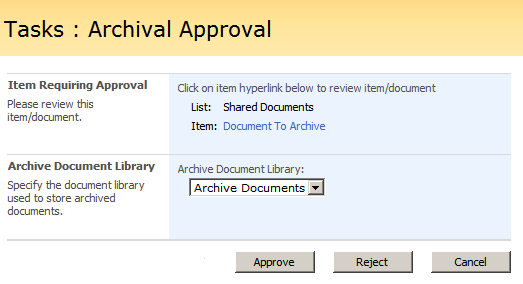
#### In the **Shared Documents** document library, verify the workflow is running.

#### 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 reject the archive process.

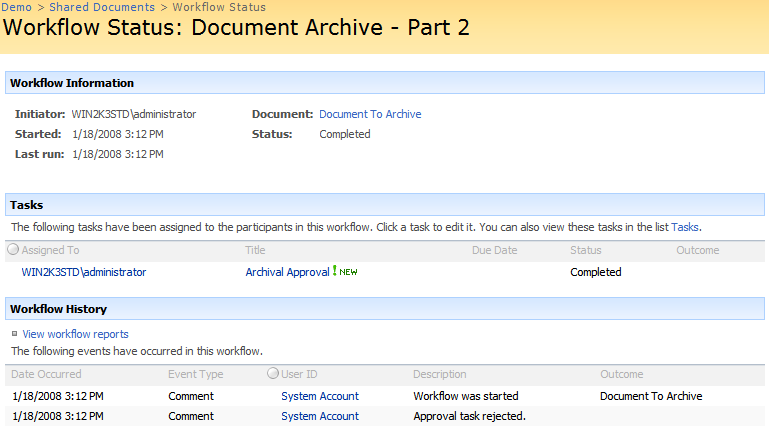
##### In the **Tasks** section, click the link to the task.

##### In the custom task form, click the **Reject** button to reject 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 Task 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 **TemplateFiles** and **TemplateFile** element immediately following the **Assemblies** element. This tells the Solution installer to install the task form when the solution is deployed.

<TemplateFiles>

<TemplateFile Location="LAYOUTS\DocArchivePart2TaskForm.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 **DocArchivePart2TaskForm.aspx** in the **LAYOUTS** folder in the .cab file.

..\..\Template\Layouts\DocArchivePart2TaskForm.aspx LAYOUTS\DocArchivePart2TaskForm.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 **DocuentArchiveWorkflowPart2** feature.

stsadm –o uninstallfeature –name DocumentArchiveWorkflowPart2 –force

#### Delete the folder at **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Features\DocumentarchiveWorkflowPart2**.

#### Delete the **DocArchivePart2TaskForm.aspx** file 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\Lab06\StarterFiles\DocumentArchiveWorkflowPart2\bin\Debug\Package\DocumentArchiveWorkflowPart2.wsp”

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

stsadm –o deploysolution –name DocumentArchiveWorkflowPart2.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 2** feature is active.