Lab 10: Understanding Workflow Foundation & Creating Custom Workflows

**Lab Time:** 120 Minutes

**Lab Overview:** In this lab, you will create a real world, practical workflow that is needed in many organizations that leverage a content management system such as Web Content Management within Office SharePoint Server 2007. Many organizations need to have a group of individuals review a piece of content before it is published for the rest of the world to see. However, within Litware, it is not important that specific people review a piece of content, rather what is important is that a majority of the reviewers approve the content before it is published. What Litware needs is a workflow where they can assign three people from the legal department to review a piece of content before it is published. However, it doesn't matter who reviews it, as long as a majority (2/3) of the reviewers approve or reject it.

The workflow that you will create in this lab will leverage InfoPath 2007 forms as the interaction vehicle between the users and workflow hosted within SharePoint. Upon initiation, it will prompt the user for three user names for the three reviewers as well as some instructions. A task will be created and assigned to each person. The reviewers then can approve or reject the workflow and enter some comments when they submit their decision. Once a majority is decided, which could be before all three decide, the workflow will terminate and approve or reject the page. To create this workflow you will do the following (7 exercises):

1. Create & configure the project
2. Create the workflow process & bind any necessary fields to the activities
3. Create & publish InfoPath 2007 forms
4. Add code to integrate the InfoPath forms and necessary logic
5. Package the workflow up for deployment
6. Implement the workflow
7. Test the workflow

Due to the nature of workflow and InfoPath form integration with workflows in SharePoint, this is a very large lab. Know that the complete solution is available to you if you want to copy & paste some code or if you need to refer to anything. Also, all steps are critical... and in the order they are presented. Workflows in SharePoint, especially those leveraging InfoPath forms are very finicky... so be astute when working through this lab!

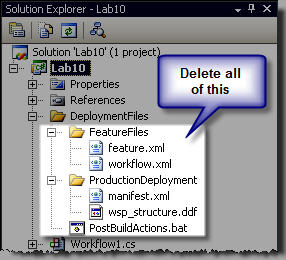
Exercise 1: Setup the sequential workflow project template

In this exercise you will create the project that will contain the workflow using a template provided with the MOSS 2007 SDK. The project that the template creates is very different than what you're used to working with in this course, so in this first exercise you will also do a few tweaks to get in a more familiar territory.

*\* Keep in mind that most of the stuff you do in this exercise is to apply consistency with this Visual Studio project and the others you've created in other labs in this course. With the exception of the project template selection, everything else is optional. However, the remainder of the lab is written assuming you did the steps in this exercise.*

*\* You can optionally skip this exercise and go straight to exercise 2 using a sample that is included in the Resources folder. If you elect this approach, note the special instructions at the beginning of Exercise 2.*

1. Open **Visual Studio** and create a new project using the template **SharePoint Server Sequential Workflow** which is found within the **Visual C# \ SharePoint** in the **Project types:** pane. Give this project a name of **Lab10**.
2. The first thing to do is to get rid of all the extra stuff included in the project template. Delete everything you see that is highlighted in the following image... what you'll be left with is a single **Workflow1.cs** file and an empty **DeploymentFiles** folder:



1. Next, copy the files **BuildSharePointPackage.ddf** & **BuildSharePointPackage.Targets** into the **DeploymentFiles** folder in the project from the following folder:

c:\Student\Labs\10\_Workflow\Resources

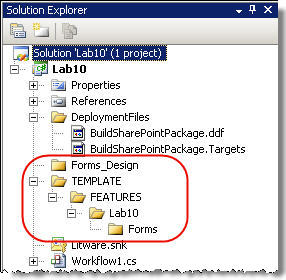
1. Now you need to configure the project to run the custom MSBuild targets file every time the project is compiled in order to automatically create the WSP file used for deployment. Unload the project by right-clicking **Lab10** in the **Solution Explorer** tool window and selecting **Unload Project**. Then right-click the unloaded project in the **Solution Explorer** tool window and select **Edit Lab10.csproj**. Scroll to the bottom of the page and add the following XML just after the existing **<Import>** nodes:

<Import Project="DeploymentFiles\BuildSharePointPackage.targets" />

<Target Name="AfterBuild">

<CallTarget Targets="BuildSharePointPackage" />

</Target>

Save your changes, right-click the unloaded project in the **Solution Explorer** tool window & select **Reload Project**. When prompted with a **Security Warning for** **Lab10** dialog, select **Load project normally**, uncheck **Ask me for every project in this solution** and click **OK**.

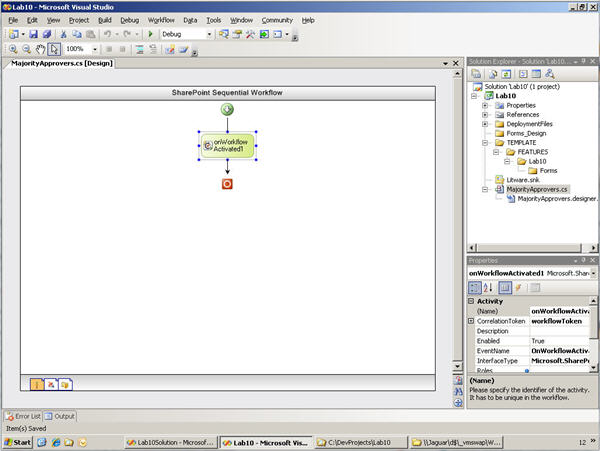
1. Next, copy the **Litware.snk** key file into the root of the project from the following folder:

c:\Student\Labs\10\_Workflow\Resources

Now, right-click the **Lab10** project in the **Solution Explorer** window and select **Properties**. Select the **Signing** tab, verify that the **Sign the assembly** checkbox is **checked**, and select **Litware.snk** from the **Choose a strong name key file** selector.

Still from within the project properties page, select the **Build Events** tab and clear the contents of the **Post-build event command line:** textbox.

1. Save all changes and close the project property page.
2. Finally, create the following folder structure within the **Lab10** project:
3. Finally, the last step is to rename the **Workflow1.cs** file to **MajorityApprovers.cs**.
4. Now, you need to clean up two issues with this renaming of the workflow refactoring. Double-click **MajorityApprovers.cs** to open the workflow designer. Notice the red circle with an exclamation point in the upper right-hand corner of the only activity on the design surface: **onWorkflowActivated1**. Select the activity and press **F4** to open the **Properties** tool window in case it isn't open already.
5. Expand the **CorrelationToken** property and select **MajorityApprovers** in the **OwnerActivityName** property.
6. Click the **ellipses** button **[...]** in the value field of the **WorkflowProperties** property in the **Properties** tool window to show the visual data binder dialog. Select **workflowProperties** on the **Bind to an existing member** tab and click **OK**.
7. There's one more place this needs to be changed. Open the **MajorityApprovers.designer.cs** file and look for the code **activitybind2.Name = "Workflow1"** ... change that to **="MajorityApprovers"**.
8. Save all your changes... your project should now look similar to the following image... notice that the red error indicator on the **onWorkflowActivated1** activity is now gone:



At this point, your project is now ready to start building the workflow! So let's get started...

Exercise 2: Create the workflow process & bind any necessary fields to the activities

In this exercise you will design your workflow process by adding activities to the design surface and connecting them together. You will also create any necessary data bound fields that will be used throughout the project.

*If you elected to skip exercise 1, you can copy the contents within the following folder into the Lab folder where you normally do the labs in this course:*

c:\Student\Labs\10\_Workflow\Resources

1. The first thing you need to do is design your workflow. To do this, you will drag activities onto the design surface. First, double-click **MajorityApprovers.cs** file to open the workflow designer.
2. Next, if it isn't already visible, select **View » Toolbox** to show the toolbox containing all the activities.
3. First, drag a **LogToHistoryListActivity** from the **SharePoint - Workflow** tab in the toolbox and attach it between the **onWorkflowActivated1** & red terminating box.

Using the **Properties** tool window, set the **(name)** property to **logWorkflowStarted**.

1. Next, select the **[...]** button in the **HistoryDescription** property to open the data binding dialog. Because you don't have any fields in your class, you need to create one. Select the **Bind to a new member** tab, enter a name of **HistoryDescription**, select **Create Field** and click **OK**.
2. Repeat the process for the **HistoryOutcome** property, creating a new field named **HistoryOutcome**.
3. Next, drag a **Parallel** activity from the **Windows Workflow** tab in the **Toolbox** and add it just after the **LogToHistoryListActivity** previously added. Change the **name** of the **Parallel** activity from **parallelActivity1** to **createTasksParallel**.
4. At this point your workflow will create three tasks. You will give the three tasks that this workflow keeps track of the unique names of Alpha, Beta & Charlie. But first, notice how only two **Sequence** activities are shown by default in the **Parallel** activity. You need to add another by right-clicking the icon just below the **createTasksParallel** activity name in the designer and selecting **Add Branch**.

Now you can add activities to each branch. In the first branch, drag a **CreateTask** activity into **sequenceActivity1** and change its name to **createAlphaApprovalTask**.

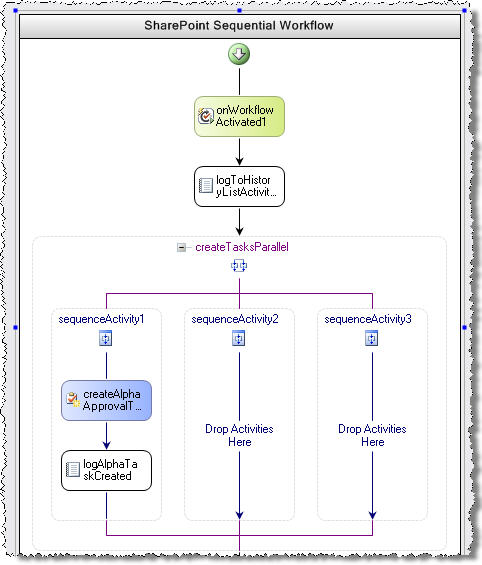
1. In order to keep track of the different tasks and activities in the workflow that will correspond to the tasks, the workflow uses correlation tokens. Enter **alphaTaskToken** in the **CorrelationToken** property for the **createAlphaApprovalTask** activity. Once you click outside of the **CorrelationToken** property, you will notice it suddenly has an expando icon to the left of the name. Click it to select the **OwnerActivityName** to **MajorityApprovers**.
2. For the **TaskId** property of the **createAlphaApprovalTask** activity, click the **[...]** button to add a new field named **AlphaTaskId** using the same process outlined previously in this exercise.

Do the same think for the **TaskProperties**, creating a new field named **AlphaTaskProperties**.

1. Now, add a **LogToHistoryActivity** just after the **createAlphaApprovalTask** activity and name it **logAlphaTaskCreated**.
2. Next, set the **HistoryDescription** & **HistoryOutcome** properties to the previously created fields **HistoryDescription** & **HistoryOutcome** by selecting them from the **Bind to an existing member** tab when you bring the data binding dialog up.

*You may be wondering "Why are we setting the same fields to two different activities?" Don't worry about it for now. We are going to use event handlers on the activities to set the values of these fields before the activity executes.*

At this point, your workflow should look like the following image:

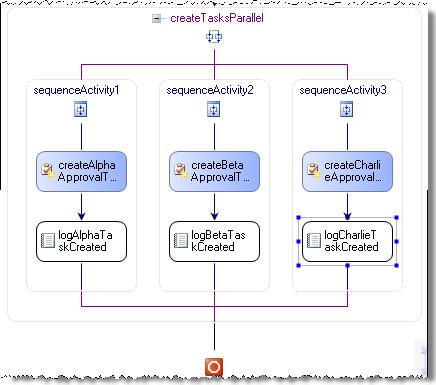


1. Now, repeat the same process for **sequenceActivity2** and **sequenceActivity3**, adding a new **CreateTask** and **LogToHistoryListActivity** and creating all the new necessary fields. The following table should help you sent the values:

|  |  |
| --- | --- |
| sequenceActivity2 |  |
| CreateTask |  |
| (Name) | createBetaApprovalTask |
| CorrelationToken | betaTaskToken |
| Owner ActivityName | MajorityApprovers |
| TaskId | BetaTaskId *(new field)* |
| TaskProperties | BetaTaskProperties (*new field*) |
| LogToHistoryListActivity |  |
| (Name) | logBetaTaskCreated |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

|  |  |
| --- | --- |
| sequenceActivity3 |  |
| CreateTask |  |
| (Name) | createCharlieApprovalTask |
| CorrelationToken | charlieTaskToken |
| Owner ActivityName | MajorityApprovers |
| TaskId | CharlieTaskId *(new field)* |
| TaskProperties | CharlieTaskProperties (*new field*) |
| LogToHistoryListActivity |  |
| (Name) | logCharlieTaskCreated |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

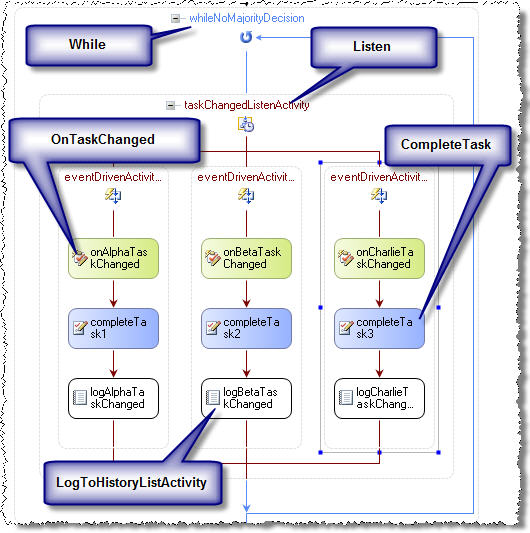
At this point your workflow should look like the following image:



1. With the task creation portion of the workflow complete, it is now time to implement the part that will listen for task changes and wait until a two-thirds (2/3) majority has responded in favor of (approve) or against (reject) publishing the item. To do this, you are going to add quite a few activities. Instead of walking through the process of every single activity, use the following image to guide the design process. The tables after the image detail the properties that need to be set. This structure should be placed just after the **createTasksParallel** activity and before the red termination icon.

*Note: You will need to add a new eventDrivenActivity branch to the Listen activity.*

*Note: Don't worry about the error that will display on the While activity. You are not at the point where you want to add the logic for this activity.*



|  |  |
| --- | --- |
| While |  |
| (Name) | whileNoMajorityDecision |

|  |  |
| --- | --- |
| Listen |  |
| (Name) | taskChangedListenActivity |

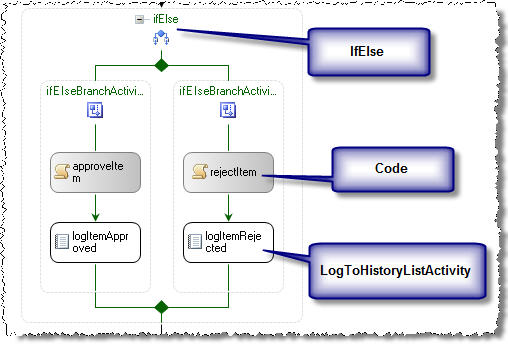
|  |  |
| --- | --- |
| eventDrivenActivity1 *(left-hand one)* | |
| OnTaskChanged |  |
| (Name) | onAlphaTaskChanged |
| CorrelationToken | alphaTaskToken |
| Owner ActivityName | MajorityApprovers |
| AfterProperties | AlphaAfterTaskProperties *(new field)* |
| TaskId | AlphaTaskId *(existing field)* |
| CompleteTask |  |
| (Name) | completeAlphaTask |
| CorrelationToken | alphaTaskToken |
| Owner ActivityName | MajorityApprovers |
| TaskId | AlphaTaskId *(existing field)* |
| LogToHistoryListActivity |  |
| (Name) | logAlphaTaskCreated |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

|  |  |
| --- | --- |
| eventDrivenActivity2 *(middle one)* | |
| OnTaskChanged |  |
| (Name) | onBetaTaskChanged |
| CorrelationToken | betaTaskToken |
| Owner ActivityName | MajorityApprovers |
| AfterProperties | BetaAfterTaskProperties *(new field)* |
| TaskId | BetaTaskId *(existing field)* |
| CompleteTask |  |
| (Name) | completeBetaTask |
| CorrelationToken | betaTaskToken |
| Owner ActivityName | MajorityApprovers |
| TaskId | BetaTaskId *(existing field)* |
| LogToHistoryListActivity |  |
| (Name) | logBetaTaskCreated |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

|  |  |
| --- | --- |
| eventDrivenActivity3 *(right-hand one)* | |
| OnTaskChanged |  |
| (Name) | onCharlieTaskChanged |
| CorrelationToken | charlieTaskToken |
| Owner ActivityName | MajorityApprovers |
| AfterProperties | CharlieAfterTaskProperties *(new field)* |
| TaskId | CharlieTaskId *(existing field)* |
| CompleteTask |  |
| (Name) | completeCharlieTask |
| CorrelationToken | charlieTaskToken |
| Owner ActivityName | MajorityApprovers |
| TaskId | CharlieTaskId *(existing field)* |
| LogToHistoryListActivity |  |
| (Name) | logCharlieTaskCreated |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

1. Next, you need to add the part of the workflow that will do the programmatic approval or rejection of the request. To do this, again, refer to the following image and tables to create the necessary activities. This section should be placed after the previously added While activity and just before the red termination icon.

*Note: Again, don't worry about the error that will display on the IfElse activity. You are not at the point where you want to add the logic for this activity.*



|  |  |
| --- | --- |
| ifElseBranchActivity *(left-hand one)* | |
| Code |  |
| (Name) | approveItem |
| LogToHistoryListActivity |  |
| (Name) | logItemApproved |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

|  |  |
| --- | --- |
| ifElseBranchActivity *(right-hand one)* | |
| Code |  |
| (Name) | rejectItem |
| LogToHistoryListActivity |  |
| (Name) | logItemRejected |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

1. Finally, add a single LogToHistoryListActivity at the very end of the workflow and set the following properties:

|  |  |
| --- | --- |
| LogToHistoryListActivity |  |
| (Name) | logWorkflowComplete |
| HistoryDescription | HistoryDescription *(existing field)* |
| HistoryOutcome | HistoryOutcome *(existing field)* |

At this point, your workflow is now completely designed. The next step, before working with the code part of the workflow, is to switch over to InfoPath 2007 and create the forms that will be used in this workflow.

Exercise 3: Create & publish InfoPath 2007 forms

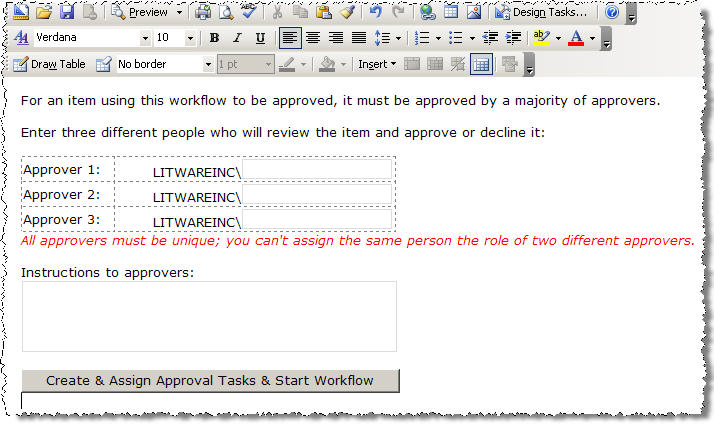
In this exercise you will create two new InfoPath 2007 forms that will be used in the initiation and task edit portion of the workflow. With MOSS 2007 installed, you should prefer InfoPath 2007 forms development over building ASPX forms because so much work is done for you by the system. However, with that added benefit comes some tricky pieces with respect to InfoPath forms. Make sure you follow the instructions in this section very closely.

You will build the initialization form first followed by the task edit form.

1. Open **Office InfoPath 2007** and in the **Getting Started** dialog, select **Design a Form Template**.
2. In the **Design a Form Template** dialog, select **Form Template**, **Based on: Blank** and check **Enable browser-compatible features only**.
3. When the new form loads, select the Task Pane **Data Source** in on the right-hand side of the application. Create a new field by right-clicking **myFields** and selecting **Add...**. Use the following table to create the new field:
   * **Name:** alphaApprover
   * **ApproverType:** Field (element)
   * **Data type:** Text (string)
   * **Cannot be blank (\*):** checked
4. Repeat the step above creating three more fields named **betaApprover**, **charlieApprover** and **instructions** with the same attributes.
5. Now, rename the top-level node **myFields** to **InitForm**.
6. Next, create a form similar to the one shown in the following image. Don't worry about the button details or large instructions textbox... you will address those in a moment.

A shortcut to get controls on the page that are linked with the fields in the data source is to right-click the root node in the data source and select Controls. InfoPath will automatically drop the controls on the design surface. You can then rearrange them at will.

*Note: You'll find the button control on the* ***Controls*** *Task Pane... you can toggle to the Controls Task Pane by clicking the down arrow in the upper right corner of the current Task Pane.*



1. Let's format the *instructions to approvers* textbox. Right-click the textbox for the instructions and select **Text Box Properties**.... Switch to the **Display** tab and check **Multi-line** then click **OK**.
2. Now you need to add some logic to the button. Right-click the button and select **Button Properties**.... On the **General** tab, change the **Label:** value to **Create & Assign Approval Tasks & Start Workflow**.
3. Then click the **Rules**... button. On the **Rules** dialog, click **Add**. On the **Rule** dialog, click **Add Action**....
4. On the **Action** dialog, select **Action: Submit using a data connection**, then click **Add**.... On the **Data Connection Wizard**, select **Create a new connection to: Submit data** and click **Next >**. Then select **To the hosting environment, such as an ASP.NET page or hosting application** and click **Next >** followed by **Finish**.
5. Now, click **OK** to get back to the **Rule** dialog. Click **Add Action...**. On the **Action** dialog, select **Close the form** and **OK** out of all the dialogs.
6. The last thing you need to do is configure the security on the form so it can run within SharePoint. Select **Tools** **» Form Options**. On the **Category** for **Security and Trust**, uncheck **Automatically determine security level (recommended)**, select **Domain** and click **OK**.

\*\*\* It is critical you follow the next two steps explicitly... this is where many errors occur.

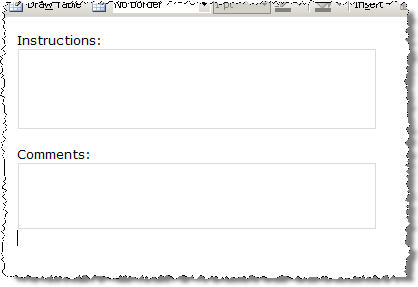
1. Now save the form to the **Forms\_Design** folder you created in the Visual Studio project containing your workflow giving it a name of **InitForm.xsn**.
2. Next, you need to publish the form. Select **File** then **Publish**. In the **Publishing Wizard**, select **To a network location** and click **Next >**. Browse to the **Forms** folder within the **Feature** folder you created in the Visual Studio project containing your workflow, giving it a name of **InitForm.xsn**. Set the **Form template name: InitForm** and click **Next >**.
3. Clear the **alternate access path** on the next step and click **Next >**. *\*\*\* If you don't do this, your form won't work.*

You should get a warning about users not being able to open the form. Ignore this and click **OK**. If you didn't get it, you didn't set the trust level to Domain!

1. Finally, click **Publish** followed by **Close**.
2. To save yourself a step later, go ahead and grab the form's URN before closing it. Select **File » Properties**. Copy the URN located in the **ID** textbox and paste it somewhere for future reference (such as creating a new text file in Notepad)... just make sure you label it with something like InitForm). Close the dialog by clicking **OK**.
3. One last thing to do with this form before you continue on... to make your life easier when you get to the code part of this lab, select **File » Save As Source Files...**, create a new directory within **Forms\_Design** called **Source Files** and save the files to that location. Then copy the **myschema.xsd** file from the **Source Files** folder to the **Forms\_Design** folder and rename it **InitFormSchema.xsd**.

At this point your initialization form is complete. Next you need to create the task form.

1. It you closed InfoPath, open Office InfoPath 2007 and in the **Getting Started** dialog, select **Design a Form Template**.
2. In the **Design a Form Template** dialog, select **Form Template, Based on: Blank** and check **Enable browser-compatible features only**.
3. When the new form loads, select the Task Pane **Data Source** in on the right hand side of the application. Create a new field by right-clicking **myFields** and selecting **Add...**. Use the following table to create the new field:
   * **Name:** instructions
   * **Type:** Field (element)
   * **Data type:** Text (string)
   * **Cannot be blank (\*):** checked
4. Repeat the step above creating three more fields named **comments** and **decision** with the same attributes (except don't make comments required).
5. Now, rename the top-level node **myFields** to **TaskForm**.
6. Next, create a form similar to the one shown in the following image. Make both the instructions & comments textboxes multiline and larger input fields:



1. Unlike the initialization form you created previously, the task form needs some extra work in order for SharePoint to pass data back and forth to the form. This is because the form can't know about any changes to the task list that may occur after deployment. The way you do this is with an XML file named **ItemMetadata.xml**. Switch over to Visual Studio for a moment and create a new XML file named **ItemMetadata.xml** at the root of project and add the following XML markup to the file:

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

<z:row xmlns:z="#RowsetSchema"

ows\_instructions="" />

1. Notice the field is prefixed with **ows\_**. This is the field that is used in the form that needs to be passed into the form. Now that you have created the ItemMetadata.xml file, you need to add the XML file to the task form as a new data source. Jump back to InfoPath and select **Tools » Data Connections...**. From the **Data Connections** dialog, click **Add**.

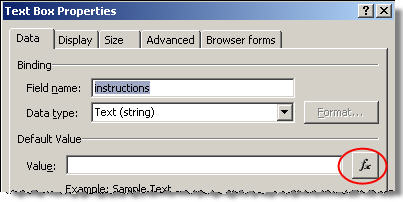
When the wizard loads, select **Create a new connection to: Receive data** and click **Next >**.

Specify you want to **receive data from an XML document** and select **Next >**. Browse to the location of the **ItemMetadata.xml** file. Then click **Resource Files...**, followed by **Add...** and select the same file.

**OK** out of the dialog back to the wizard and click **Next >**.

Accept the defaults on the next screen in the wizard (**Name =ItemMetadata** & **Automatically receive data when form is opened** is checked) and click **Finish**. Close the **Data Connections** dialog.

1. With a data source created, you now need to configure your form to pull the instructions from the XML file and insert it into the field. Right-click the **instructions** textbox and select **Properties...**.
2. Click the **function** button to the right of the **Value** textbox in the **Default Value** section.



1. Click the **Insert Field or Group...** button change the **Data source:** to **ItemMetadata (Secondary)** and se**l**ect **:ows\_instructions** and **OK** out of the function dialog.
2. On the **Display** tab, check the **Read-only** checkbox as we don't really want approvers changing their instructions and **OK** out of the dialog.
3. The last design thing you need to do to the form is you need to add two buttons to either approve or reject the task. Add two buttons at the bottom of the form.
4. Right-click the first button and select **Button Properties...**. Set the **Label:** to **Approve**.

Click the **Rules...** button to bring up the **Rules** dialog. Click the **Add...** button and then the **Add Action...** button.

Select the **Action:** to **Set a field's value**, pick the decision field from the **Main** data source and click **OK**. In the **Value** textbox, type **approve** and click **OK**.

Click **Add Action...**. On the **Action** dialog, select **Action: Submit using a data connection**, then click **Add...**. On the **Data Connection Wizard**, select **Create a new connection to: Submit data** and click **Next >**. Then select **To the hosting environment, such as an ASP.NET page or hosting application** and click **Next >** followed by **Finish**.

Now, click **OK** to get back to the **Rule** dialog. Click **Add Action...**. On the **Action** dialog, select **Close the form** and **OK** out of all the dialogs.

1. Do the same thing to the other button, except name it **Reject**, set the value of the **decision** field to **reject**, and use the existing data connection you created in the last step to submit the form to SharePoint.
2. The last thing you need to do is configure the security on the form so it can run within SharePoint. Select **Tools » Form Options**. On the **Category for Security and Trust**, uncheck **Automatically determine security level (recommended)**, select **Domain** and click **OK**.

*\*\*\* It is critical you follow the next two steps explicitly... this is where many errors occur.*

1. Now save the form to the **Forms\_Design** folder you created in the Visual Studio project containing your workflow giving it a name of **TaskForm.xsn**.
2. Next, you need to publish the form. Select **File » Publish**. In the **Publishing Wizard**, select **To** a network location and click **Next >**. Browse to the **Forms** folder within the **Feature** folder you created in the Visual Studio project containing your workflow, giving it a name of **TaskForm.xsn**. Set the **Form template name: TaskForm** and click **Next >**.

Clear the **alternate access path** on the next step and click **Next >**. \*\*\* If you don't do this, your form won't work.

1. You should get a warning about users not being able to open the form. Ignore this and click **OK**. *If you didn't get it, you didn't set the trust level to Domain!*

Finally, click **Publish** followed by **Close**.

1. To save yourself a step later, go ahead and grab the form's URN before closing it. Select **File » Properties**. Copy the URN located in the **ID** textbox and paste it somewhere for future reference (such as creating a new text file in Notepad)... just make sure you label it with something like TaskForm).

Close the dialog by clicking **OK**.

At this point you have created the two forms that will be leveraged by your workflow and published them to the Forms folder within the Feature.

Exercise 4: Add code to integrate the InfoPath forms and necessary logic

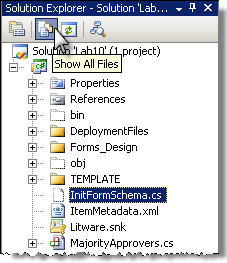
In this exercise you will integrate the InfoPath forms into the workflow as well as add all necessary logic to the code behind of the workflow. Hope you like writing code!

1. Switch back to Visual Studio where the MajorityApprovers workflow should still be open. You will now start adding code to each of the event handlers on all activities that add some logic to the workflow. You will work through the workflow from the top to the bottom. While there are quite a few class-scoped fields to create, you will add them only when necessary.
2. Before you start adding code, do one thing that was alluded to in the last exercise that would save you some headache. What you will do is create a new C# class from the InitFormSchema.xsd file you created when creating the IntiForm InfoPath form. This will save you from having to create a namespace and query the XML passed from SharePoint, rather you can just deserialize the data into the class.

Open a **Visual Studio Command Prompt**, change directory into the root of your workflow project and enter the following command:

xsd.exe [path to init form schema file]/InitFormSchema.xsd /c

1. Now, go back to Visual Studio and click the icon to **Show all files** icon, right click the **InitFormSchema.cs** file and select **Include In Project**.



1. Open the **InitFormSchema.cs** file. You'll notice the class, **InitForm**, is not in a namespace. Add the following code just after the using statement to add it to the namespace:

namespace Lab10

{

Finally, add a final closing **}** to the end of the file to close the namespace. Save you changes to the **InfoFormSchema.cs** file.

1. Right-click the **onWorkflowActivated** activity at the top of the workflow and select **Generate Handlers**. Within this method you will add code that will deserialize the data from the initialization form passed by SharePoint to the workflow in the **workflowProperties.InitiationData** property. This makes it much easier to work with the data as you don't have to write any XML XPath queries to fetch the data from the XML.
2. However, first you need to create a few class-scoped private fields. Add the following just before the method what was generated by Visual Studio:

private string \_instructions = default(string);

private string \_alphaApprover = default(string);

private string \_betaApprover = default(string);

private string \_charlieApprover = default(string);

1. Now, add the following code to the generated handler:

private void onWorkflowActivated\_Invoked (object sender, ExternalDataEventArgs e) {

// load the data from the IP form 'InitForm' into a local object

XmlSerializer serializer = new XmlSerializer(typeof(InitForm));

XmlTextReader xrInitForm = new XmlTextReader(new System.IO.StringReader(workflowProperties.InitiationData));

InitForm frmInit = serializer.Deserialize(xrInitForm) as InitForm;

// get approvers submitted

this.\_alphaApprover = @"LITWAREINC\" + frmInit.alphaApprover;

this.EnsureUserIsRecognized(this.\_alphaApprover);

this.\_betaApprover = @"LITWAREINC\" + frmInit.betaApprover;

this.EnsureUserIsRecognized(this.\_betaApprover);

this.\_charlieApprover = @"LITWAREINC\" + frmInit.charlieApprover;

this.EnsureUserIsRecognized(this.\_charlieApprover);

// get instructions

this.\_instructions = frmInit.instructions;

}

1. You may notice a method called **EnsureUserIsRecognized()** in the code above that hasn't been defined. What this method will do is check to see if the provided user has rights into the current SharePoint site where the workflow is running. If not, the user is automatically granted access to the site. This is done using the two following methods which you need to add to the **MajorityApprovers.cs** file:

private void EnsureUserIsRecognized (string userName) {

// if the user isn't found in the site, add them

if (this.GetUserIdFromUserName(workflowProperties.Web, userName) == -1)

workflowProperties.Web.SiteUsers.Add(userName, "", "", "");

}

private int GetUserIdFromUserName (SPWeb site, string userName) {

// get the user's info

Microsoft.SharePoint.Utilities.SPPrincipalInfo principalInfo =

Microsoft.SharePoint.Utilities.SPUtility.ResolvePrincipal(site,

userName,

SPPrincipalType.All,

SPPrincipalSource.All,

null,

false);

// if the user was found, return their ID, else -1

if (principalInfo != null)

return principalInfo.PrincipalId;

else

return -1;

}

1. Now you need to log a message indicating the workflow started. Right-click the **logToHistoryListActivity** at the top of the workflow and select **Generate Handlers**. Add the following code that will report the data collected from the initialization form and save it to the two local fields created when you created this activity:

private void logWorkflowStarted\_MethodInvoking (object sender, EventArgs e) {

// get reference to the item this workflow is associated with

SPListItem page = workflowProperties.Item;

// create log message & outcome

this.HistoryOutcome = "Workflow successfully initiated...";

this.HistoryDescription = String.Format("Collected three approvers: {0}, {1} & {2}. All received the following instructions: {3}",

this.\_alphaApprover,

this.\_betaApprover,

this.\_charlieApprover,

this.\_instructions);

}

1. The next step is to create the event handlers for the task creation activities and each associated log activity. Right-click the two activities in **sequenceActivity1** within the **createTasksParallel** parallel activity (**createAlphaApprovalTask** & **logAlphaTaskCreated**) and add the following code to each:

private void createAlphaApprovalTask\_MethodInvoking (object sender, EventArgs e) {

this.AlphaTaskId = Guid.NewGuid();

this.AlphaTaskProperties.Title = "Approval requested for " + workflowProperties.Item.Title;

this.AlphaTaskProperties.Description = "Please review the item, then approve or reject it.";

this.AlphaTaskProperties.AssignedTo = this.\_alphaApprover;

this.AlphaTaskProperties.PercentComplete = 0;

this.AlphaTaskProperties.StartDate = DateTime.Today;

this.AlphaTaskProperties.DueDate = DateTime.Today.AddDays(7);

this.AlphaTaskProperties.ExtendedProperties["instructions"] = this.\_instructions;

}

private void logAlphaTaskCreated\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = String.Format("Task created and assigned to first approver: {0}", this.\_alphaApprover);

this.HistoryDescription = String.Format("Approval task 'alpha' created and assigned to {0}", this.\_alphaApprover);

}

1. Repeat the previous step but for the activities within **sequenceActivity2** within the **createTasksParallel** parallel activity (**createBetaApprovalTask** & **logBetaTaskCreated**), using the following code:

private void createBetaApprovalTask\_MethodInvoking (object sender, EventArgs e) {

this.BetaTaskId = Guid.NewGuid();

this.BetaTaskProperties.Title = "Approval requested for " + workflowProperties.Item.Title;

this.BetaTaskProperties.Description = "Please review the item, then approve or reject it.";

this.BetaTaskProperties.AssignedTo = this.\_betaApprover;

this.BetaTaskProperties.PercentComplete = 0;

this.BetaTaskProperties.StartDate = DateTime.Today;

this.BetaTaskProperties.DueDate = DateTime.Today.AddDays(7);

this.BetaTaskProperties.ExtendedProperties["instructions"] = this.\_instructions;

}

private void logBetaTaskCreated\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = String.Format("Task created and assigned to second approver: {0}", this.\_betaApprover);

this.HistoryDescription = String.Format("Approval task 'beta' created and assigned to {0}", this.\_betaApprover);

}

1. Repeat the previous step but for the activities within **sequenceActivity3** within the **createTasksParallel** parallel activity (**createCharlieApprovalTask** & **logCharlieTaskCreated**), using the following code:

private void createCharlieApprovalTask\_MethodInvoking (object sender, EventArgs e) {

this.CharlieTaskId = Guid.NewGuid();

this.CharlieTaskProperties.Title = "Approval requested for " + workflowProperties.Item.Title;

this.CharlieTaskProperties.Description = "Please review the item, then approve or reject it.";

this.CharlieTaskProperties.AssignedTo = this.\_charlieApprover;

this.CharlieTaskProperties.PercentComplete = 0;

this.CharlieTaskProperties.StartDate = DateTime.Today;

this.CharlieTaskProperties.DueDate = DateTime.Today.AddDays(7);

this.CharlieTaskProperties.ExtendedProperties["instructions"] = this.\_instructions;

}

private void logCharlieTaskCreated\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = String.Format("Task created and assigned to third approver: {0}", this.\_charlieApprover);

this.HistoryDescription = String.Format("Approval task 'charlie' created and assigned to {0}", this.\_charlieApprover);

}

With all the tasks created you can now move onto the section where the workflow will go to sleep waiting for some activity on the tasks. However, first there are a few things you need to create before moving on.

1. First, you are going to need an enumeration called **ApproverDecision** that will be used instead of strings (much safer!). Create a new **C#** class file named **ApproverDecision.cs** and add the following code to it:

using System;

namespace Lab10

{

public enum ApprovalDecision

{

Approved,

Rejected,

NoAnswer

}

}

1. Second, there are a few new class-scoped fields that are needed. Some will keep track of the different tasks answers, others keep track of how many approvals & rejections there have been, and one is used to determine if the majority of approvers have responded one way or another. Add the following field declarations to the **MajorityApprovers** class:

public ApprovalDecision AlphaTaskAnswer = ApprovalDecision.NoAnswer;

public ApprovalDecision BetaTaskAnswer = ApprovalDecision.NoAnswer;

public ApprovalDecision CharlieTaskAnswer = ApprovalDecision.NoAnswer;

public int ApproveAnswerCount = 0;

public int RejectAnswerCount = 0;

public bool MajorityHasAnswered = false;

1. Third, when you add code to the event handlers for the **OnTaskChanged** activities within the while activity, you are going to add a call to a method called **UpdateMajorityDecision**. This method is called to total up the approve/reject score into a single value that will be used to determine if the while loop should continue or not. Add the following method to the **MajorityApprovers.cs** file:

private void UpdateMajorityDecision () {

int countApproved = 0;

int countRejected = 0;

// check alpha approver

if (this.AlphaTaskAnswer == ApprovalDecision.Approved)

countApproved++;

else if (this.AlphaTaskAnswer == ApprovalDecision.Rejected)

countRejected++;

// check beta approver

if (this.BetaTaskAnswer == ApprovalDecision.Approved)

countApproved++;

else if (this.BetaTaskAnswer == ApprovalDecision.Rejected)

countRejected++;

// check charlie approver

if (this.CharlieTaskAnswer == ApprovalDecision.Approved)

countApproved++;

else if (this.CharlieTaskAnswer == ApprovalDecision.Rejected)

countRejected++;

// set results

this.ApproveAnswerCount = countApproved;

this.RejectAnswerCount = countRejected;

// if three respondants, no need to check futher

if (countApproved + countRejected == 3) {

this.MajorityHasAnswered = true;

}// else if two of EITHER answer have responded, majority rules

else if (countApproved == 2 || countRejected == 2)

this.MajorityHasAnswered = true;

}

*Now you can to work on creating the event handlers that are waiting for the tasks to update...*

1. There is some work to do on the **whileNoMajorityDecision** while activity, but you will address that at the end. For now, just generate handlers for the two of the three activities in the eventDrivenActivity1 (**onAlphaTaskChanged** & **logAlphaTaskChanged**... no need to create a handler for the **completeAlphaTask** activity) and add the following code to the created handlers:

private void onAlphaTaskChanged\_Invoked (object sender, ExternalDataEventArgs e) {

// check if the task was approved

string taskResult = this.AlphaTaskAfterProperties.ExtendedProperties["decision"].ToString();

if (taskResult.ToLower() == "approve")

this.AlphaTaskAnswer = ApprovalDecision.Approved;

else if (taskResult.ToLower() == "reject")

this.AlphaTaskAnswer = ApprovalDecision.Rejected;

// now, need to check all the other tasks to see 2/3 are approved

// b/c if so, need to mark whole thing as complete

UpdateMajorityDecision();

}

private void logAlphaTaskChanged\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = ("First approver task answered.");

this.HistoryDescription = String.Format("Approval task 'alpha' status is now: {0}. Current score: {1}-{2}. Approver entered the following comments: '{3}'.",

this.AlphaTaskAnswer.ToString(),

this.ApproveAnswerCount.ToString(),

this.RejectAnswerCount.ToString(),

this.AlphaTaskAfterProperties.ExtendedProperties["comments"].ToString());

}

1. Repeat the previous step but for the activities within **evenDrivenActivity2** within the **taskChangedListenActivity** listen activity (**onBetaTaskChanged** & **logBetaTaskChanged**), using the following code:

private void onBetaTaskChanged\_Invoked (object sender, ExternalDataEventArgs e) {

// check if the task was approved

string taskResult = this.BetaTaskAfterProperties.ExtendedProperties["decision"].ToString();

if (taskResult.ToLower() == "approve")

this.BetaTaskAnswer = ApprovalDecision.Approved;

else if (taskResult.ToLower() == "reject")

this.BetaTaskAnswer = ApprovalDecision.Rejected;

// now, need to check all the other tasks to see 2/3 are approved

// b/c if so, need to mark whole thing as complete

UpdateMajorityDecision();

}

private void logBetaTaskChanged\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = ("Second approver task answered.");

this.HistoryDescription = String.Format("Approval task 'beta' status is now: {0}. Current score: {1}-{2}. Approver entered the following comments: '{3}'.",

this.BetaTaskAnswer.ToString(),

this.ApproveAnswerCount.ToString(),

this.RejectAnswerCount.ToString(),

this.BetaTaskAfterProperties.ExtendedProperties["comments"].ToString());

}

1. Repeat the previous step but for the activities within **evenDrivenActivity3** within the **taskChangedListenActivity** listen activity (**onCharlieTaskChanged** & **logCharlieTaskChanged**), using the following code:

private void onCharlieTaskChanged\_Invoked (object sender, ExternalDataEventArgs e) {

// check if the task was approved

string taskResult = this.CharlieTaskAfterProperties.ExtendedProperties["decision"].ToString();

if (taskResult.ToLower() == "approve")

this.CharlieTaskAnswer = ApprovalDecision.Approved;

else if (taskResult.ToLower() == "reject")

this.CharlieTaskAnswer = ApprovalDecision.Rejected;

// now, need to check all the other tasks to see 2/3 are approved

// b/c if so, need to mark whole thing as complete

UpdateMajorityDecision();

}

private void logCharlieTaskChanged\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = ("Third approver task answered.");

this.HistoryDescription = String.Format("Approval task 'charlie' status is now: {0}. Current score: {1}-{2}. Approver entered the following comments: '{3}'.",

this.CharlieTaskAnswer.ToString(),

this.ApproveAnswerCount.ToString(),

this.RejectAnswerCount.ToString(),

this.CharlieTaskAfterProperties.ExtendedProperties["comments"].ToString());

}

1. Before you move on, let's go back and configure the while activity's condition for when it will terminate. You want the while loop to continue until a majority has responded one way or another. For example, you want it to stop when two approve or reject the task, but if the score is 1-1 after two responses, it waits for the third response.

Select the **whileNoMajorityDecision** while activity and take a look at the **Properties** tool window. Notice there is an error for the **Condition** property. Change the **Condition** to a **Declarative Rule Condition**, click the expando icon and then click the ellipses **[...]** button in the **ConditionName** property.

Create a new condition by clicking **New...**. Then, in the **Rule Condition Editor**, enter the following code and click **OK**:

!this.MajorityHasAnswered

After the condition has been created, rename it from **Condition1** to **Majority has not answered**. Select it and click **OK**.

1. With the workflow tasks answered and a result determined, you now need to configure the code activities that will either approve or reject the item that kicked off the workflow.

Select the **ifElseBranchActivity1**, the one on the left that contains the **approveItem** code activity. Change the **Condition to a Declarative Rule Condition**, click the expando icon and then click the ellipses **[...]** button in the **ConditionName** property.

Create a new condition by clicking **New...**. Then enter the following code in the **Rule Condition Editor** and click **OK**:

this.ApproveAnswerCount >=2

After the condition has been created, rename it from **Condition1** to **Majority approved**.

1. Right-click the **approveItem** code activity, select **Generate Handlers** and add the following code to the handler:

private void approveItem\_ExecuteCode (object sender, EventArgs e) {

// get the item...

SPListItem item = workflowProperties.Item;

// approve the item

item.File.Approve("Approved by a majority.");

}

1. Right-click the **logItemApproved** activity, select **Generate Handlers** and add the following code to the handler:

private void logItemApproved\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = "Item approved by a majority.";

this.HistoryDescription = String.Format("The final score was {0} approvers & {1} rejecters. Once a majority of three approvers is reached, the workflow terminates.",

this.ApproveAnswerCount.ToString(),

this.RejectAnswerCount.ToString());

}

1. Select the **ifElseBranchActivity2**, the one on the right that contains the **rejectItem** code activity. Change the **Condition** to a **Declarative Rule Condition**, click the expando icon and then click the ellipses **[...]** button in the **ConditionName** property.

Create a new condition by clicking **New...**. Then enter the following code in the **Rule Condition Editor** and click **OK**:

this.RejectAnswerCount >=2

After the condition has been created, rename it from **Condition1** to **Majority rejected**.

1. Right-click the **rejectItem** code activity, select **Generate Handlers** and add the following code to the handler:

private void rejectItem\_ExecuteCode (object sender, EventArgs e) {

SPListItem item = workflowProperties.Item;

item.File.Deny("Rejected by a majority.");

}

1. Right-click the **logItemRejected** activity, select **Generate Handlers** and add the following code to the handler:

private void logItemRejected\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = "Item rejected by a majority.";

this.HistoryDescription = String.Format("The final score was {0} approvers & {1} rejecters. Once a majority of three approvers is reached, the workflow terminates.",

this.ApproveAnswerCount.ToString(),

this.RejectAnswerCount.ToString());

}

Finally... right-click the **logWorkflowComplete** activity, select the **Generate Handlers** and add the following code to the handler:

private void logWorkflowComplete\_MethodInvoking (object sender, EventArgs e) {

this.HistoryOutcome = "Workflow complete.";

this.HistoryDescription = "Workflow finished... majority always wins (except in hand grenades & horseshoes).";

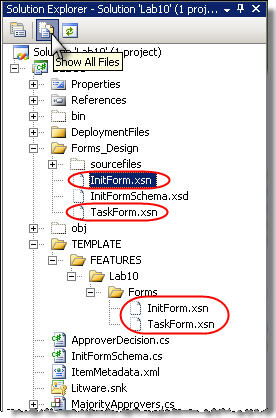
}

That's it! You made it through all that code!!!!In this exercise, you added logic and integrated the InfoPath forms into the workflow.

Exercise 5: Package the workflow up for deployment

In this exercise you will configure the Feature that will be used to install the workflow as well as the WSS solution package that will be used for deploying the Feature and associated files.

1. There are a few files that you've added to the folder structure of the workflow project in Visual Studio, but you've yet to add them to the project itself... let's take care of that now. If hidden items aren't being shown currently, click the **Show all files** icon at the top of the **Solution Explorer** tool window. Take note of two InfoPath published form files you need to add to the project (within the **Feature** folder) and the two InfoPath design form files saved within the **Forms\_Design** folder. Right-click each of these **\*.XSN** files and select **Include In Project**:



1. Now, the first thing you need to do is create the necessary Feature files. First, create a new XML file named **feature.xml** in the **Lab10** folder in the project and add the following XML markup:

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

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

Id="39513EA7-A8A3-4D92-9028-14DF5690B532"

Title="Lab 10 - Majority Rules Workflow for Publishing Sites"

Description="Deploys a workflow template that will assign one task to three people, and then wait for a majority to approve/reject before completing the workflow."

Scope="Site"

Hidden="False"

ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"

ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"

Version="1.0.0.0">

<ElementManifests>

<ElementManifest Location="workflow.xml" />

<ElementFile Location="Forms\InitForm.xsn" />

<ElementFile Location="Forms\TaskForm.xsn" />

</ElementManifests>

<Properties>

<Property Key="GloballyAvailable" Value="true" />

<Property Key="RegisterForms" Value="Forms\\*.xsn" />

</Properties>

</Feature>

Parts of this file deserve a bit of explanation:

* + **<Feature Site="">**: all Features that deploy workflow templates to SharePoint must be scoped at the site collection level.
  + **<Feature ReceiverAssembly="" & ReceiverClass="">**: this is boilerplate code... when your workflow leverages InfoPath forms, you need to use this Feature receiver as it will do the work of uploading the InfoPath forms to SharePoint.
  + **<Property Key="GloballyAvailable">**: this property tells the Feature receiver that the InfoPath forms should be uploaded to Central Administration for all sites in the farm. This is so the same workflow added to site collections across your organization doesn't create hundreds if not thousands of instances of your forms.
  + **<Property Key="RegisterForms">**: this property tells the Feature receiver which InfoPath forms to upload & register as well as where it will find these forms.

1. Now, create a new XML file named **workflow.xml** in the **Lab10** folder in the project and add the following XML markup:

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

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

<Workflow Name="Majority Approvers"

Description="Assigns three different people approval tasks. When a majority approves/rejects their tasks, the workflow approves/rejects the item."

Id="D5613F31-FA46-4E45-A264-BDB5CC8061EC"

CodeBesideAssembly="Lab10, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d4e5777b16a5749f"

CodeBesideClass="Lab10.MajorityApprovers"

TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"

AssociationUrl="\_layouts/CstWrkflIP.aspx"

InstantiationUrl="\_layouts/IniWrkflIP.aspx"

ModificationUrl="\_layouts/ModWrkflIP.aspx"

StatusUrl="\_layouts/WrkStat.aspx">

<Categories />

<MetaData>

<Instantiation\_FormURN>[[ your form's URN ]]</Instantiation\_FormURN>

<Task0\_FormURN>[[ your form's URN ]]</Task0\_FormURN>

</MetaData>

</Workflow>

</Elements>

Parts of this file deserve a bit of explanation:

* + **<Workflow Name="" & Description="">**: these values are seen by site owners/admins who associate this workflow template with a list or content type.
  + **<Workflow CodeBesideAssembly="" & CodeBesideClass="">**: these tell SharePoint where the class that contains the workflow, and then assembly containing the class, can be found. Note: the assembly will be added to the GAC which you'll see in a moment.
  + **<Workflow TaskListContentTypeId="">**: your workflow used the OOTB task list so this is the content type ID for the task content type, but if you created a custom content type used by your workflow, you would specify its content type ID here.
  + **<Workflow [Association|Instantiation|Modification|Status]Url="">**: these attributes tell SharePoint what pages should be used for the different forms; the nice thing here is this is typically boilerplate code as each of these provided pages contains a Web Part which will load the appropriate InfoPath form based on what has been defined in the <MetaData> section.
  + **<MetaData>**: this is where you need to specify the InfoPath form's URN that you copied to a text file; put your form's URN's in the appropriate XML node.

1. With the Feature created, you now need to package everything up into a WSS solution package. Open the **BuildSharePointPackage.ddf** file and add the following code between the comments:

DeploymentFiles\manifest.xml

bin\debug\Lab10.dll

.Set DestinationDir=Lab10

TEMPLATE\FEATURES\Lab10\feature.xml

TEMPLATE\FEATURES\Lab10\workflow.xml

.Set DestinationDir=Lab10\Forms

TEMPLATE\FEATURES\Lab10\Forms\InitForm.xsn

TEMPLATE\FEATURES\Lab10\Forms\TaskForm.xsn

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

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

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

SolutionId="85EF6F81-4B4A-44EB-A71E-12605A24B7D9"

DeploymentServerType="WebFrontEnd"

ResetWebServer="FALSE">

<Assemblies>

<Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab10.dll" />

</Assemblies>

<FeatureManifests>

<FeatureManifest Location="Lab10\feature.xml" />

</FeatureManifests>

</Solution>

1. Build the project to create the WSP file... now it's time to deploy and see this guy in action!

In this exercise you finished creating the Feature that is used to deploy the workflow and custom InfoPath 2007 forms to SharePoint as well as the final configuration tasks necessary to package everything up in a WSS solution package.

Exercise 6: Implement the workflow

At last... we can finally see if this thing works! In this exercise you will install the WSS solution package, deploy the workflow to a site collection, and associate it with an existing Publishing site's Pages list.

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

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

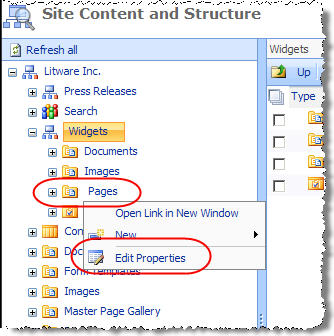
1. Enter the following command into the command line window and hit Enter:

stsadm -o addsolution -filename c:\Student\Labs\10\_Workflow\Lab\wsp\Debug\Lab10.wsp

1. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
2. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
3. On the **Solution Management** page, click the link on **lab10.wsp**.
4. On the **Solution Properties** page, select **Deploy Solution**.
5. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.
6. Browse to the **http://wcm.litwareinc.com/** site and select **Site Actions » Site Settings » Modify All Site Settings**.
7. Under the **Site Collection Administration** section, select **Site collection features**.
8. On the **Site Collection Features** page, click the **Activate** button for the **Lab 10 - ... Feature**.
9. Now you need to associate the workflow with a Pages list to see it work. You will do this to the Pages list within the Widgets sample site that was created using the dummy content creator tool in the Web Parts.

*If you didn't run the tool to create the dummy content, refer back to the Web Parts lab for instructions on how to create the dummy content using the provided utility.*

The easiest way to do this is to jump to the **Site Content and Structure** page by selecting **Site Actions » Manage Content and Structure**. Then, open the **Widgets** subsite, followed by selecting **Edit Properties** from the **Pages** list's ECB menu to jump straight to the Pages' settings page:



1. On the **Customize Pages** page, select **Workflow settings** under the **Permissions and Management** section.
2. Now, what you want to do is not only add your custom workflow, but also remove or deactivate the Parallel Approval workflow so it doesn't get in the way. Click **Remove a workflow**, then set the **Parallel Approval** workflow to **No New Instances** and click **OK**.
3. Next, click **Add a workflow** and use the following values to complete the **Add a Workflow: Pages** page... leave all default settings where not specified below and click **Next**:
   * **Workflow:** Majority Approvers
   * **Name:** Majority of Three Approvers
4. Now we're ready to test!

In this exercise you deployed and associated the workflow with an existing list.

Exercise 7: Test the workflow

Now the part you've worked so hard to get to... to see this guy in action! In this exercise you will test your workflow on some existing pages.

Note that the buttons in the Page Editing Toolbar's Quick Access Button area are configured to work with the Parallel Approval workflow, so you'll have to manually fire it off and check the status. This is a good thing so you can see all parts of this workflow.

1. Navigate to **Widget Product 1** page (**http://wcm.litwareinc.com/Widgets/Pages/WidgetProduct1.aspx**)... the page should not be checked out, it should be published. If it is, pick another page that is published.
2. Select **Site Actions » Edit Page**. No need to make any changes, just click **Check In To Share Draft** on the Quick Access Button area. Then from the **Page Actions** area, select **Workflow » Start a Workflow...**.
3. On the **Workflows: WidgetProduct1** page, select **Majority of Three Approvers** as shown in the following image:



1. On the **Start "Majority of Three Approvers": WidgetProduct1** page, use the following information in the initiation form... notice this is your **InitForm.xsn** InfoPath 2007 form!
   * **Approver1:** LITWAREINC\brianc
   * **Approver2:** LITWAREINC\angelab
   * **Approver3:** LITWAREINC\jayh
   * **Instructions:** Please review this page and either approve or reject it.
2. After the workflow starts, from the **Page Actions** area, select **Workflow » View Workflow Status...**. Notice the tasks have been created and assigned and there is a lot of information that's been logged to the history as shown in the following image:



1. Now, open each of the tasks and try different combinations of selecting Approve or Reject and watch how the workflow reacts!

In this exercise you have successfully tested your workflow.