# Lab 05: Programming with SharePoint Workflow API

**Lab Overview:** Litware Inc. needs to extend the capabilities of SharePoint’s workflow system. To do this you will need to create your own workflows with custom forms. To do this you will need to be familiar with the SharePoint APIs, specifically those that represent the workflow objects.

To do this you will write a simple Windows Forms application that will use the SharePoint APIs to create workflow associations, start workflow instances, and monitor running and completed workflow instances.

## Exercise 0: Setup

### If you didn’t complete lab 4, you’ll need to setup the **Quotes** list in 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**.

#### Click the **Site Actions -> Create** menu option.

#### On the **Create** page, click the **Document Library** link under the **Libraries** section header.

#### Enter a name of **Quotes** and click the **Create** button to create the new document library.

#### Add a new **Quote** to the document library by clicking the **New -> New Document** button.

#### In the warning dialog, click **OK** to open the new document.

#### In Word 2007, enter some text into the document and click **Save**. Save the document using the name **Sample Quote.docx** to the **Quotes** library.

## Exercise 1: Creating a Workflow Association

### Connect the **Association** user control to the **MainForm** object.

#### Open the code for the **MainForm.cs** class by right clicking it in the **Solution Explorer** and choosing **View Code**.

#### In the **MainForm** class, add two private fields to store the **SPWeb** and **SPSite** objects.

private SPSite m\_site;

private SPWeb m\_web;

#### In the **MainForm\_Load** method, add the code to connect to the WSS Site.

m\_site = new SPSite("http://litwareinc.com/sites/Demo");

m\_web = m\_site.RootWeb;

#### Next, add the code that will provide the **ctlAssociation** object with the **SPWeb** object just created.

ctlAssociation.Initialize(m\_web);

#### Cleanup by adding code to dispose of the **SPSite** and **SPWeb** objects in the **MainForm\_FormClosed** method.

m\_web.Dispose();

m\_site.Dispose();

### Use the **SPWeb** object to initialize the UI of the association user control

#### Open the code for the **Association.cs** class by right clicking it in the **Solution Explorer** and choosing **View Code**.

#### In the **Association** class, add two private fields to store the **SPWeb** and **SPWorkflowAssociation** objects needed to store the state of the user interface.

private SPWeb m\_web;

private SPWorkflowAssociation m\_current;

#### Locate the **Initialize** method and store the **SPWeb** parameter in the **m\_web** private field.

m\_web = web;

#### Populate the **Lists** combo box using the **SPWeb.Lists** collection and the **PopulateComboBox** method.

Look at the **PopulateComboBox** method to understand how it works.

PopulateComboBox(m\_web.Lists, lstWebLists);

#### Populate the **WorkflowTemplates** combo box using the **SPWeb.WorkflowTemplates** collection and the **PopulateComboBox** helper method.

PopulateComboBox(m\_web.WorkflowTemplates, lstWorkflowTemplates);

#### Populate the **Tasks Lists** combo box using a filtered **SPWeb.Lists** collection and a filtering version of the **PopulateComboBox** method.

Only add **SPList** objects that have a **BaseTemplate** of **SPListTemplateType.Tasks**.

Look at the **PopulateComboBox** method to understand how filtering is performed with a lambda function.

PopulateComboBox(m\_web.Lists.Cast<SPList>(),

lstTaskList, n => n.BaseTemplate == SPListTemplateType.Tasks);

#### Populate the **History Lists** combo box using a filtered **SPWeb.Lists** collection and a filtering version of the **PopulateComboBox** method.

Only add **SPList** objects that have a **BaseTemplate** of **SPListTemplateType.WorkflowHistory**.

PopulateComboBox(m\_web.Lists.Cast<SPList>(),   
 lstHistoryList, n => n.BaseTemplate == SPListTemplateType.WorkflowHistory);

### Implement the **DisplayAssociation** method that will populate the UI based on an existing **SPWorkflowAssociation** object.

#### Locate the **DisplayAssociation** method and add the code to set the **txtName** control’s text to the name of the workflow association.

// initialize the name and workflow template

txtName.Text = association.Name;

#### Add code to set the selected item of **lstWbrkflowTemplates** to the workflow template defined in the **SPWorkflowAssociation’s BaseTemplate.Id** property.

LINQ is used here to perform a query using the items in the combo box.

lstWorkflowTemplates.SelectedItem =

lstWorkflowTemplates.Items.Cast<SPWorkflowTemplate>().First(

n => n.Id == association.BaseTemplate.Id);

#### Using the same technique used to select the workflow template, select the task and history lists from their respective combo boxes. Compare the **ID** of the **SPList** to the **SPWorkflowAssociatoins** **TaskListId** and **HistoryListId**.

// find the task and history list in the drop down lists

lstTaskList.SelectedItem =

lstTaskList.Items.Cast<SPList>().First(

n => n.ID == association.TaskListId);

lstHistoryList.SelectedItem =

lstHistoryList.Items.Cast<SPList>().First(

n => n.ID == association.HistoryListId);

#### Update the state of the UI to allow the user to change some information, but not change the selected workflow template.

// enable editing controls

lstWorkflowTemplates.Enabled = false; // not enabled since it is already chosen

grpAssociation.Enabled = true;

btnDelete.Enabled = false;

### Implement the **DisplayNewAssociation** method that will populate the UI to allow creation of a new **SPWorkflowAssociation**

#### Locate the **DisplayNewAssociation** method and add code to set the **txtName** control’s text to an empty string and select the first item in the **lstWorkflowTemplates** control.

// choose a default name and workflow template

txtName.Text = string.Empty;

lstWorkflowTemplates.SelectedIndex = 0;

#### Add code to set the selected item of **lstTaskList** and **lstHistoryList** to the first item in the list.

// select the first task or history list available

lstTaskList.SelectedItem = 0;

lstHistoryList.SelectedItem = 0;

#### Update the status of the UI to allow the user to enter all the information they need.

// enable editing controls

lstWorkflowTemplates.Enabled = true;

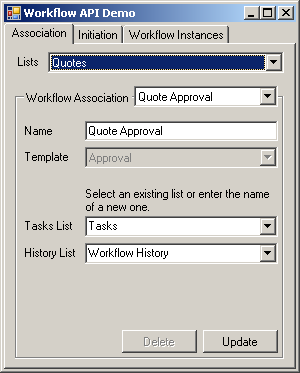
grpAssociation.Enabled = true;

btnDelete.Enabled = false;

### Test the user interface.

#### Start the application using the debugger by selecting **Debug -> Start** **Debugging** or pressing **F5**.

#### Switch to the **Association** tab and explore the lists and their workflow associations.



### Implement the **CreateWorkflowAssociation** method to allow creation of new **SPWorkflowAssociation** objects.

#### Locate the **CreateWorkflowAssociation** method and replace the existing code with the code to lookup the selected **SPList** and **SPWorkflowTemplate** objects from their combo box controls.

SPList list = lstWebLists.SelectedItem as SPList;

SPWorkflowTemplate template =

lstWorkflowTemplates.SelectedItem as SPWorkflowTemplate;

#### Check if the selected history list already exists by comparing **lstHistoryList.SelectedItem** to **null**. If not, create it using the **CreateList** method. If so, cast the **SelectedItem** property to **SPList** store it for later.

// create the history list if it doesn't exist

SPList historyList = (lstHistoryList.SelectedItem == null) ?

CreateList(lstHistoryList.Text, SPListTemplateType.WorkflowHistory) :

lstHistoryList.SelectedItem as SPList;

#### Find the selected tasks list, creating it if necessary and store it for later.

// create the task list if it doesn't exist

SPList taskList = (lstTaskList.SelectedItem == null) ?

CreateList(lstTaskList.Text, SPListTemplateType.Tasks) :

lstTaskList.SelectedItem as SPList;

#### Create the new workflow association using the **SPWorkflowAssociation.CreateListAssociation** static method. Provide the **SPWokflowTemplate**, name, task list, and history list retrieved previously and store the new **SPWorkflowAssociation** for later.

// create the association

SPWorkflowAssociation result =

SPWorkflowAssociation.CreateListAssociation(template, txtName.Text,

taskList, historyList);

#### Using the **AssociationData** property of the **SPWorkflowTemplate**, assign the **AssociationData** property of the new **SPWorkflowAssociation**.

result.AssociationData = template.AssociationData;

#### Attach the new **SPWorkflowAssociation** object to the selected list using the **SPList.AddWorkflowAssociation** method. Do not forget to call the list’s **Update** method.

// add the association to the list

list.AddWorkflowAssociation(result);

list.Update();

#### Return the new **SPWorkflowAssociation** object.

return result;

### Implement the **UpdateWorkflowAssocition** method to allow updating existing **SPWorkflowAssociation** objects.

#### Update the name of the **SPWorkflowAssociation** object to the value in the **txtName** control.

// update the association name

association.Name = txtName.Text;

#### Find the selected history and tasks lists, creating them if necessary and storing them for later.

// create the history list if it doesn't exist

SPList historyList = (lstHistoryList.SelectedItem == null) ?

CreateList(lstHistoryList.Text, SPListTemplateType.WorkflowHistory) :

lstHistoryList.SelectedItem as SPList;

// create the task list if it doesn't exist

SPList taskList = (lstTaskList.SelectedItem == null) ?

CreateList(lstTaskList.Text, SPListTemplateType.Tasks) :

lstTaskList.SelectedItem as SPList;

#### If the selected history list’s **ID** does not match the **SPWorkflowAssociation’s** current **HistoryListId’s** value, update the list using the **SPWorkflowAssociation.SetHistoryList** method.

// update the history list if it's changed

if (association.HistoryListId != historyList.ID)

association.SetHistoryList(historyList);

#### Check and update the tasks list if necessary using the **SPWorkflowAssociation.SetTasksList** method.

// update the task list if it's changed

if (association.TaskListId != taskList.ID)

association.SetTaskList(taskList);

#### Update the **SPWorkflowAssociation** object using its **ParentLists’s UpdateWorkflowAssociaiton** method.

// update the association

association.ParentList.UpdateWorkflowAssociation(association);

### Implement the **DeleteWorkflowAssociation** method to allow deletion of existing **SPWorkflowAssociation** objects.

#### Find the **SPWorkflowAssociation’s** parent list using the **ParentList** property and store it for later.

SPList parentList = association.ParentList;

#### Remove the workflow association from the parent list using the **SPList.RemoveWorkflowAssociation** method. Do not forget to call the list’s **Update** method.

// remove the workflow association

parentList.RemoveWorkflowAssociation(association);

parentList.Update();

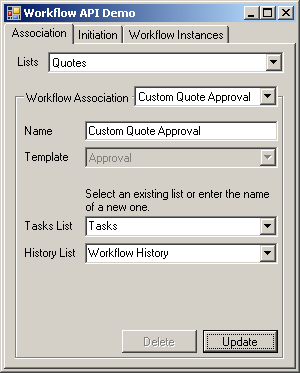
### Test the application by adding, updating, and deleting a workflow association.

#### Start the application using the debugger by selecting **Debug -> Start** **Debugging** or pressing **F5**.

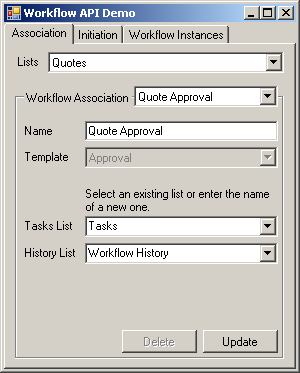
#### Switch to the **Association** tab and select the **Quotes** list in the **Lists** dropdown.

#### In the **Workflow Association** dropdown, verify selection of the **Quote Approval** workflow association and verify the selected tasks and history lists that.

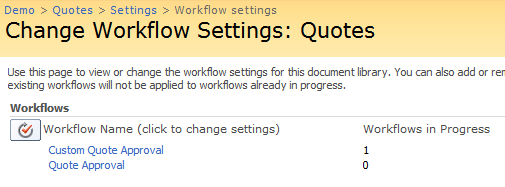
#### Change the name of the **Quote Approval** workflow association to **Custom Quote Approval** and click **Update** to change its name.



#### Select **New Item…** in the **Workflow Association** drop down and enter a name of **Quote Approval**. Also, select a template of type **Approval**. When done, click **Update** to commit the changes.



#### In **Internet Explorer**, browse to the **Quotes** list and click **Settings -> Document Library Settings**. Next click the **Workflow** **settings** link in the **Permissions and Management** section and verify the listing of **Custom Quote Approval** and **Quote Approval**.



## Exercise 2: Starting a Workflow

### Connect the **Initiation** user control to the **MainForm** object.

#### Open the code for the **MainForm.cs** class by right clicking it in the **Solution Explorer** and choosing **View Code**.

#### In the **MainForm\_Load** method, add the code that will provide the **ctlInitiation** object with the **SPWeb** object just created.

ctlInitiation.Initialize(m\_web);

### Use the **SPWeb** object to initialize the UI of the initiation user control.

#### Open the code for the **Initiation.cs** class by right clicking it in the **Solution Explorer** and choosing **View Code**.

#### In the **Initiaiton** class, add a private field to store the **SPWeb** object to store the state of the user interface.

private SPWeb m\_web;

#### Locate the **Initialize** method and store the **SPWeb** parameter in the **m\_web** private field.

m\_web = web;

#### Populate the **Lists** combo box using the **SPWeb.Lists** collection and the **PopulateComboBox** method.

PopulateComboBox(m\_web.Lists, lstWebLists);

### Implement the **PopulateListDetails** method that will populate the list items and workflow associations drop down lists based on the currently selected list.

#### Locate the **PopulateListDetails** method and add the code to populate the list items drop down list with the list items in the currently selected list. Use the **SPList.Items** collection to access a list’s list items.

// populate the combo box containing the list items

// and workflow associations for the list

PopulateComboBox(list.Items, lstListItems);

#### Add the code to populate the list of workflow associations using the currently selected list’s **WorkflowAssociations** collection.

PopulateComboBox(list.WorkflowAssociations, lstAssociations);

### Implement the **ContainsRunningWorkflow** method that will determine if a specific list item has a running instance of a specific workflow association. This disallows the starting of a workflow if one is already running.

#### Locate the **ContainsRunningWorkflow** method and add convert the **SPListItem.Workflows** collection to an **IEnumerable<SPWorkflow>** object by using the **Cast<SPWorkflow>** method.

##### This will be used later to check if any workflow’s of a specific type are currently running.

IEnumerable<SPWorkflow> workflows = listItem.Workflows.Cast<SPWorkflow>();

#### Add code that will search through the list of workflows and find any workflows of the selected type. If any exist, find those with an **InternalState** of **Running**.

##### The method **FirstOrDefault** will return the first **SPWorkflow** object that matches the criteria in the lambda expression. Return **null** if no **SPWorkflow** object exists.

// find the first item in the list that matches the condition

// FirstOrDefault returns null if no item is found

SPWorkflow workflow = workflows.FirstOrDefault(

n => n.AssociationId == association.Id &&

n.InternalState == SPWorkflowState.Running);

#### Return **true** if a **SPWorkflow** was found, **false** otherwise.

// return true if a workflow was found

return (workflow != null);

### Implement the **StartWorkflow** method that will use the current UI information to start a new instance of a workflow.

#### Locate the **StartWorkflow** method and add code that locates the currently selected list item and workflow association. Cast them to a **SPListItem** and **SPWorkflowAssociation** and store them for later.

SPListItem listItem = lstListItems.SelectedItem as SPListItem;

SPWorkflowAssociation association =

lstAssociations.SelectedItem as SPWorkflowAssociation;

#### Use the **SPWorkflowManager.StartWorkflow** method to start the workflow. Use the text in **txtInitiationData.Text** as the **eventData** to use when starting the workflow.

##### The last parameter of **isAutoStart** tells SharePoint to try to restart the workflow if a problem occurs. This is important, as it will always fail on the first attempt to start since it is running in our process. Failure schedules a retry, which will work since it will run in SharePoint’s process.

// start the workflow

m\_web.Site.WorkflowManager.StartWorkflow(

listItem, association, txtInitiationData.Text, true);

#### Display a message box indicating the workflow is starting and clear the **initiation data** text box.

// update the UI to indicate the workflow has started

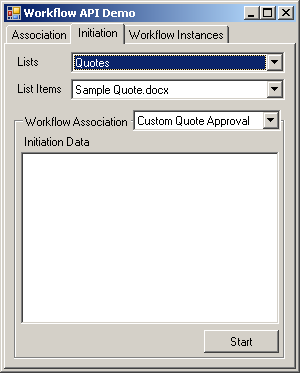
MessageBox.Show(this, "Workflow started");

txtInitiationData.Text = string.Empty;

### Test the application by starting a new instance of the **Quote Approval** workflow created in the previous exercise.

#### Start the application using the debugger by selecting **Debug -> Start Debugging** or pressing **F5**.

#### Switch to the **Initiation** tab, select the Quote list in the Lists dropdown, and select **Sample Quote.docx** in the list items drop down list.



#### In the **Workflow Association** drop down list, select **Quote Approval**.

#### Enter the following xml into the initiation data page. This is the initiation data to start an **Approval** workflow.

<my:myFields xml:lang="en-us" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:my="http://schemas.microsoft.com/office/infopath/2003/myXSD">

<my:Reviewers>

<my:Person>

<my:DisplayName> LITWAREINC \administrator</my:DisplayName>

<my:AccountId>LITWAREINC\administrator</my:AccountId>

<my:AccountType>User</my:AccountType>

</my:Person>

</my:Reviewers>

<my:CC></my:CC>

<my:DueDate xsi:nil="true"></my:DueDate>

<my:Description></my:Description>

<my:Title></my:Title>

<my:DefaultTaskType>1</my:DefaultTaskType>

<my:CreateTasksInSerial>true</my:CreateTasksInSerial>

<my:AllowDelegation>true</my:AllowDelegation>

<my:AllowChangeRequests>true</my:AllowChangeRequests>

<my:StopOnAnyReject xsi:nil="true"></my:StopOnAnyReject>

<my:WantedTasks xsi:nil="true"></my:WantedTasks>

<my:SetMetadataOnSuccess>false</my:SetMetadataOnSuccess>

<my:MetadataSuccessField></my:MetadataSuccessField>

<my:MetadataSuccessValue></my:MetadataSuccessValue>

<my:ApproveWhenComplete>false</my:ApproveWhenComplete>

<my:TimePerTaskVal xsi:nil="true"></my:TimePerTaskVal>

<my:TimePerTaskType xsi:nil="true"></my:TimePerTaskType>

<my:Voting>false</my:Voting>

<my:MetadataTriggerField></my:MetadataTriggerField>

<my:MetadataTriggerValue></my:MetadataTriggerValue>

<my:InitLock>false</my:InitLock>

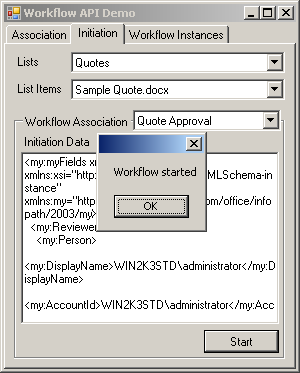
<my:MetadataStop>false</my:MetadataStop>

<my:ItemChangeStop>false</my:ItemChangeStop>

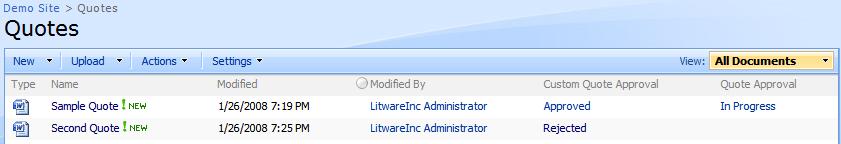
<my:GroupTasks>false</my:GroupTasks>

</my:myFields>

#### Click the **Start** button to start the workflow. The starting of the workflow will silently fail and SharePoint will automatically retry. This process may take up to 5 minutes.



#### In **Internet Explorer**, browse to the **Quotes** list and check the status of the workflow on the **Sample Quote** document. Most likely, the value will be **Starting**, when it changes to **In Progress**, the workflow is running.



## Exercise 3: Viewing Current Workflow Instances

### Connect the **Workflow** user control to the **MainForm** object.

#### Open the code for the **MainForm.cs** class by right clicking it in the **Solution Explorer** and choosing **View Code**.

#### In the **MainForm\_Load** method, add the code that will provide the **ctlWorkflow** object with the **SPWeb** object just created.

ctlWorkflow.Initialize(m\_web);

### Use the **SPWeb** object to initialize the UI of the workflow user control.

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

#### In the **Workflow** class, add a private field to store the **SPWeb** object needed to store the state of the user interface.

private SPWeb m\_web;

#### Locate the **Initialize** method and store the **SPWeb** parameter in the **m\_web** private field.

m\_web = web;

#### Populate the **Lists** combo box using the **SPWeb.Lists** collection and the **PopulateComboBox** helper method.

PopulateComboBox(m\_web.Lists, lstWebLists);

### Implement the **PopulateListItems** method that will populate the list items drop down list based on the currently selected list.

#### Locate the **PopulateListItems** method and populate the **lstListItems** drop down list using the **PopulateComboBox** method and the Items collection of the current list.

PopulateComboBox(list.Items, lstListItems);

### Implement the **PopulateWorkflows** method that will populate the workflows drop down list based on the currently selected list item.

#### Locate the **PopulateWorkflows** method and populate the **lstAssociations** drop down list using the **PopulateComboBox** method and the **Workflows** collection of the current list item.

PopulateComboBox(listItem.Workflows, lstAssociations);

### Implement the **DisplayWorkflow** method that will use the currently selected workflow to populate the workflow details UI.

#### Locate the **DisplayWorkflow** method and use current **SPWorkflow** object to initialize the initiator, started, and workflow state UI controls.

// setup the workflow state UI

txtInitiator.Text = workflow.AuthorUser.Name;

txtStarted.Text = workflow.Created.ToString();

txtWorkflowState.Text = workflow.InternalState.ToString();

#### Add the code to populate the **lstModifications** drop down list using the **PopulateListBox** and the **Modifications** collection of the current workflow.

// populate the list of modifications

PopulateListBox(workflow.Modifications, lstModifications);

#### Use the **SPWorkflowManager.GetItemTasks** method to retrieve the tasks for the current workflow. You will need to provide the **SPListItem** object the workflow attached to and the **SPWorkflowFilter** object associated with the current **SPWorkflow**.

// populate the list of tasks

SPWorkflowTaskCollection tasks =

m\_web.Site.WorkflowManager.GetItemTasks(

workflow.ParentItem, workflow.TaskFilter);

#### Using the retrieved **SPWorkflowTaskCollection**, populate the **lstTasks** drop down list using the **PopulateListBox** method.

PopulateListBox(tasks, lstTasks);

### Implement the **LookupModificationName** method that will convert a **SPWorkflowModification** object into a user-friendly string using the **SPWorkflowTemplate’s** metadata collection.

#### Locate the **LookupModificationname** method and add the code that formats the **SPWorkflowModification’s** **Id** property into the metadata string used to lookup the modification’s friendly name.

string metadataName = string.Format("Modification\_{0}\_Name", modification.Id);

#### Using the metadata string, retrieve the modification name from the **SPWorkflowTemplate** object the current **SPWorkflow** is based on.

return workflow.ParentAssociation.BaseTemplate[metadataName] as string;

### Implement the **LookupModificationContextData** method, which uses the **SPWorkflowModification** class’s **ContextData** property to return the context data associated with the modification.

#### Locate the **LookupModificationContextData** method and add the code to return the current **SPWorkflowModification’s** context data.

return modification.ContextData;

### Implement the **LookupAssociationData** method to return the association data stored in the workflow’s association to the list. As the workflow attached itself to the list, this data is set.

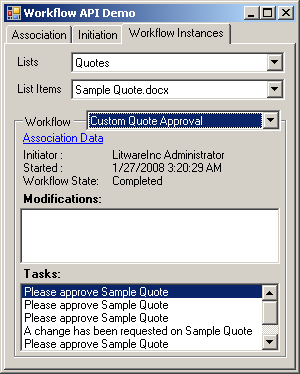
### Locate the **LookupAssociationData** method and add the code to return the current **SPWorkflow’s** association data by using its parent **SPWorkflowAssociation** object.

return workflow.ParentAssociation.AssociationData;

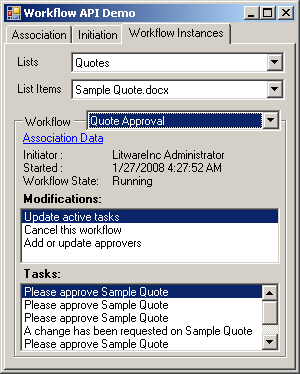
### Test the application by looking at the workflow instance started in the previous exercise.

#### Start the application using the debugger by selecting **Debug -> Start Debugging** or pressing **F5**.

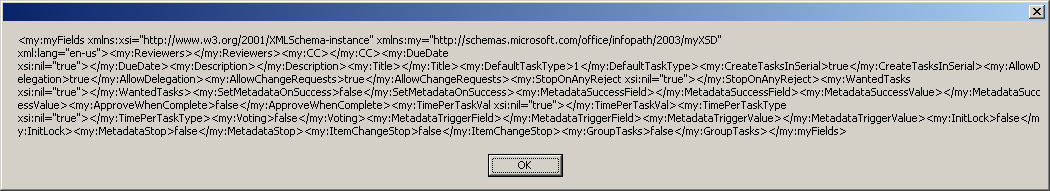
#### Switch to the **Workflow Instances** tab, select the Quote list in the Lists dropdown, and select **Sample Quote.docx** in the list items drop down list.



#### Work your way through the **Workflow** drop down to look at all workflows that are running or have run for the selected list/list item pair. Locate the workflow that has a workflow state or **Running**.



#### Click the **Association Data** link to display the workflow’s association data.



#### Double click one of the items in the **Modifications** list. The data displayed is the context data associated with the modification.

