

SPW401: Developing SharePoint Workflow Templates with Visual Studio

Student Lab Exercises

- Lab 02: Windows Workflow Foundation Primer
- Lab 03: Understanding the Windows Workflow Runtime
- Lab 04: Windows Workflow Foundation Integration with SharePoint 2007
- Lab 05: Programming with SharePoint Workflow API
- Lab 06: Developing SharePoint Workflow Templates with Visual Studio 2008
- Lab 07: Creating and Waiting on SharePoint Tasks
- Lab 08: Creating custom Workflow Association Forms
- Lab 09: Creating custom Instantiation and Modification Forms
- Lab 10: Integrating InfoPath Forms into a SharePoint Workflow Template
- Lab 11: Developing Custom Activities for a SharePoint Workflow Template
- Lab 12: Extending SharePoint Designer with Custom Activities

Revision: v3.0

Lab 02: Windows Workflow Foundation Primer

Lab Overview: Litware Inc. needs a system that will allow orders to be taken over the phone and entered using a simple console interface. Once the order is entered, a summary must be read back to the customer before it is approved.

If the order is approved the customer's account must be created using the existing customer web service. Next the order must be placed using the existing order web service. If anything fails in the creation of the customer or the order, they should be removed using the same web services that were used to create them.

Exercise 0: Setting up the project

- 1) Open the starter **Visual Studio 2008** solution at **\Labs\Lab02\StarterFiles\SequentialWorkflow.sln**.
- 2) While working on the lab, if you have an unresolved object, right click it and select **Resolve** to add the **using** statement referencing to its namespace.

Exercise 1: Entering the order using a While activity

- 1) Create a new **EnterOrder** sequential workflow.
 - Create a workflow named **EnterOrder.cs**.
*Right click **OrderSubmission** in the solution explorer and click **Add -> New Item**.
Select **Workflow** in the tree view on the left and select **Sequential Workflow (code)** in the window on the right.
Enter **EnterOrder.cs** as the name and click **Add**.*
 - Add four properties to the new **EnterOrder** class to store the state of the workflow.
*Right click **EnterOrder.cs** and select **View Code**.
Add the properties to the **EnterOrder** class.*

```
public Customer NewCustomer { get; private set; }
public Order NewOrder { get; private set; }
public bool OrderEntryDone { get; set; }
public bool OrderApproved { get; set; }
```

- Initialize the new properties in the **EnterOrder** class's constructor.

```
public EnterOrder()
{
    InitializeComponent();
    NewCustomer = new Customer();
    NewOrder = new Order();
    NewOrder.LineItems = new List<LineItem>();
}
```

- 2) Add the workflow activity that will prompt the user to enter a customer name and handle the data entry.
 - Right click **EnterOrder.cs** in the **Solution Explorer** and click **View Designer**.
 - Drag a **Code** activity from the toolbox onto canvas.

*In the properties pane, set the name property to **enterCustomer**.*

Double click the activity in the designer to generate the code handler

- In the **enterCustomer_ExecuteCode** method just generated enter the code to read a customer's name from the console

```
Console.WriteLine("Enter the customer's name: ");
NewCustomer.Name = Console.ReadLine();
```

- 3) Add a loop that will allow the user to enter one or more item ids representing the line items of the new order.

- Right click **EnterOrder.cs** in the **Solution Explorer** and click **View Designer**.
- Drag a **While** activity from the toolbox to the canvas immediately following the **enterCustomer** activity and name it **enterAllLineItems**.

*In the properties pane, set the **Name** property to **enterAllLineItems**.*

- Define the condition that will indicate whether the **While** activity continues to loop.

*Set the **Condition** property to **Declarative Rule Condition**.*

*Open the condition by clicking the plus icon to the left of the **Condition** property*

*Select the **ConditionName** sub property and click the ... button*

*In the **Select Condition** dialog, click the **New** button*

*Enter a condition of **OrderEntryDone == false** and click **OK** to close the dialog.*

*Use the **Rename** button to change the name of the new condition to **orderEntryDone**.*

*Click **OK** to complete the condition creation.*

- 4) Add another code activity into the while activity to handle the entry of a new line item.

- Drag a **Code** activity from the toolbox onto the canvas inside the **enterAllLineItems** activity.

*In the properties pane, set the **Name** property to **enterLineItem**.*

Double click the activity in the designer to generate the code handler.

- In the **enterLineItem_ExecuteCode** method just generated, enter the code to read a line item from the console.

```
Console.WriteLine("Enter a product Id (empty if no more): ");
string productId = Console.ReadLine();
```

- Add the code that will either create a new line item if a product id was entered and sets the **OrderEntryDone** to **true** if no product id was entered.

```
if (productId.Length != 0)
{
    LineItem lineItem = new LineItem();
    lineItem.ProductId = int.Parse(productId);
    NewOrder.LineItems.Add(lineItem);
}
else
{
    OrderEntryDone = true;
}
```

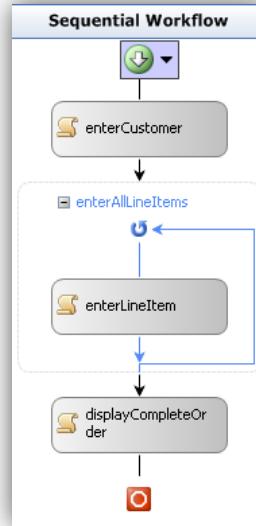
- 5) Create the final activity that will display the entire order in the console as preparation for approving the order.

- Right click **EnterOrder.cs** in the **Solution Explorer** and click **View Designer**.
- Drag one last **Code** activity from the toolbox onto the canvas immediately following the **enterAllLineItems** activity.

*In the properties pane, set the **Name** property to **displayCompleteOrder**
Double click the activity in the designer to generate the code handler.*

- In the **displayCompleteOrder_ExecuteCode** method, add the following code to display the order on the console.

```
Console.WriteLine("Review Order");
Console.WriteLine(" - Customer Name: {0}", NewCustomer.Name);
Console.WriteLine(" - Line Items");
foreach (LineItem lineItem in NewOrder.LineItems)
    Console.WriteLine("   - {0}", lineItem.ProductId);
```



- 6) Test order entry by running the workflow application and testing the order entry.
 - Click **Debug -> Start Without Debugging** in the menu bar.
 - Once the application has started, enter a customer name as **ABC Inc.**.
 - When asked for a product id, use an id of **1** for the first time and **2** for the second time.
 - When asked for the third product id, hit enter to indicate that no more line items are needed.
 - Review the output to verify the order is displayed correctly.

```
C:\WINDOWS\system32\cmd.exe
Enter the customer's name: ABC Inc.
Enter a product Id <empty if no more>: 1
Enter a product Id <empty if no more>: 2
Enter a product Id <empty if no more>:
Review Order
- Customer Name: ABC Inc.
- Line Items
  - 1
  - 2
Press any key to continue . . .
```

Exercise 2: Approving the order with an IfElse activity

- 1) Add an activity that will allow the user to verify that the previously displayed order was accurate.
 - Right click **EnterOrder.cs** in the **Solution Explorer** and click **View Designer**.
 - Drag a **Code** activity from the toolbox onto the canvas following the **displayCompleteOrder** activity.

*In the properties pane, set the **Name** property to **verifyOrder**.
Double click the activity in the designer to generate the code handler.*

 - In the **verifyOrder_ExecuteCode** method just generated, enter the code to read the approved flag from the console.

```
Console.WriteLine("Would you like to submit this order (yes/no)?: ");
OrderApproved = (Console.ReadLine() == "yes");
```
- 2) Add an if/else activity that will be used to perform different operations based on the results of the previous activity.
 - Drag an **IfElse** activity from the toolbox onto the canvas following the **verifyOrder** activity.

*In the properties pane, set the **Name** property to **submitOrCancel**.
Click the left branch and set the **Name** property to **ifSubmitOrder** in the properties pane.
Click the right branch and set the **Name** property to **ifCancelOrder** in the properties pane.*
- 3) Add a condition to the left branch that will only allow it to execute if the **OrderApproved** value is **true**.
 - Select the **ifSubmitOrder** branch of the **submitOrCancel** activity.
 - In the properties pane, add a new **Declarative Rule Condition** named **IsOrderApproved**.
 - Define the **IsOrderApproved** condition using the expression **OrderApproved == true**.
- 4) Add an activity to the **ifSubmitOrder** branch to display a message when the order is approved.
 - Drag a **Code** activity from the toolbox onto the canvas into the **ifSubmitOrder** branch.

*In the properties pane, set the **Name** property to **displayOrderApproved**.
Double click the activity in the designer to generate the code handler.*

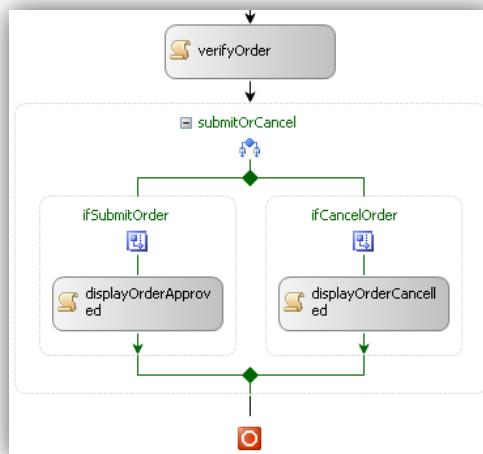
 - In the **displayOrderApproved_ExecuteCode** method just generated, enter the code to read the approved flag from the console.

```
Console.WriteLine("The order was approved.");
```

Add an activity to the **ifCancelOrder** branch to display a message when the order is not approved.

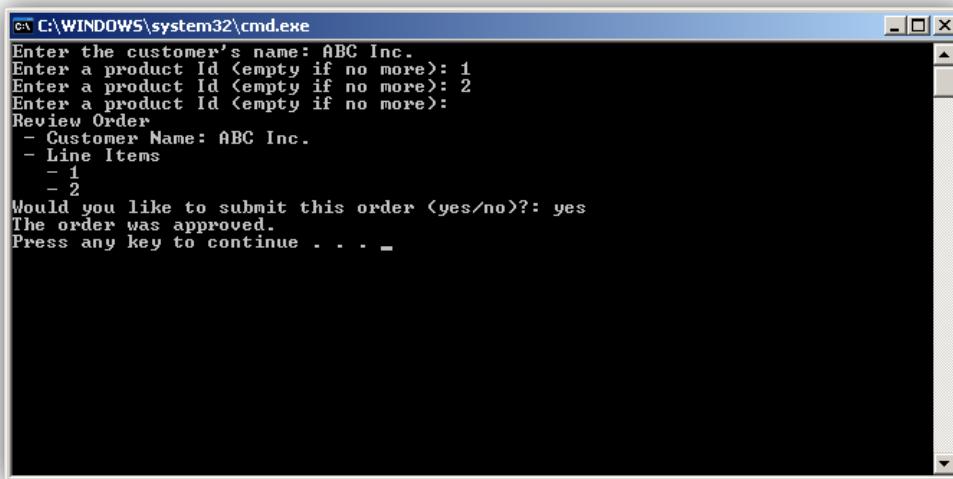
- Drag a **Code** activity from the toolbox onto the canvas into the **ifCancelOrder** branch.
*In the properties pane, set the **Name** property to **displayOrderCancelled**.
Double click the activity in the designer to generate the code handler.*
- In the **displayOrderCancelled_ExecuteCode** method just generated, enter the code to read the approved flag from the console.

```
Console.WriteLine("The order was cancelled");
```



5) Test order entry by running the workflow application and testing the order entry.

- Click **Debug -> Start Without Debugging** in the menu bar.
- Once the application has started, enter a customer name as **ABC Inc.**
- When asked for a product id, use an id of **1** for the first time and **2** for the second time.
- When asked for the third product id, hit enter to indicate that no more line items are needed.
- Review the output to verify the order is displayed correctly.
- Choose to approve the order by entering **yes**.
- Verify the result indicates the order was approved



Challenge: Submitting the order using web services

- 1) Create a new activity that will be embedded in the existing **EnterOrder** workflow

- Create a new activity named **SubmitOrder**

*Right click the **OrderSubmission** project in the **Solution Explorer** and click **Add -> New Item**.*

*Select **Workflow** in the list on the left and select **Activity** in the window on the right.*

*Enter **SubmitOrder.cs** as the name and click **Add**.*

- Drag a new **SendActivity** onto the **SubmitOrder** canvas.

*Name the new activity **createCustomer**.*

*Select the **ServiceOperationInfo** property and click its ... button.*

*Click the **Import** button in the **Choose Operation** dialog and choose the **ICustomerService** object.*

*Set the **ServiceOperationInfo** property using the **ICustomerService.CreateCustomer** method.*

*Set the **ChannelToken** property to **CustomerChannel**.*

*Expand the **ChannelToken** property, set the **EndpointName** to **WSHttpBinding_ICustomerService**, and set the **OwnerActivityName** to **SubmitOrder**.*

- Bind the return value of the service call to a property of the workflow.

*Select the **(ReturnValue)** property for the **createCustomer** activity and click the ... button.*

*Select the **Bind to a new member** tab.*

*Set the **New member name** to **NewCustomer** and verify **Create Property is selected**.*

*Click **OK** to create a new workflow property and bind the return value to that property.*

- Bind the name service parameter to the property previously created.

*Select the **Name** property for the **createCustomer** activity and click the ... button.*

*Find **NewCustomer** in the tree view, expand it, and select the **Name** property. This will bind the service parameter to the **SubmitOrder.NewCustomer.Name** property.*

*Click **OK** to create the binding.*

- Drag another new **SendActivity** onto the canvas following the **createCustomer** activity.

*Name the new activity **createOrder**.*

*Select the **ServiceOperationInfo** property and click its ... button.*

*Click the **Import** button in the **Choose Operation** dialog and choose the **OrderService** object.*

*Set the **ServiceOperationInfo** property using the **OrderService.CreateOrder** method.*

*Set the **ChannelToken** property to **OrderChannel**.*

*Expand the **ChannelToken** property and set the **EndpointName** to **WSHttpBinding_IOrderService** and set the **OwnerActivityName** to **SubmitOrder**.*

*Bind the **(ReturnValue)** property to a new **SubmitOrder.NewOrder** property.*

*Bind the **customerId** property to the **SubmitOrder.NewCustomer.Id** property.*

*Bind the **LineItems** property to the **SubmitOrder.NewOrder.LineItems** property.*

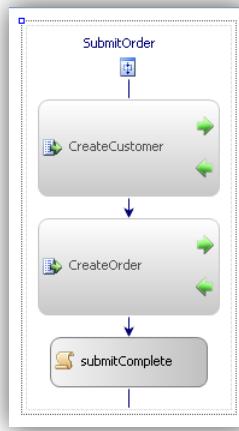
- Drag a **Code** activity onto the canvas following the **createOrder** activity.

*Name the new activity **submitComplete**.*

Double click the activity in the designer to generate the code handler.

- In the **submitComplete_ExecuteCode** method just generated, enter the code to read the approved flag from the console.

```
Console.WriteLine("Order submitted successfully.");
```



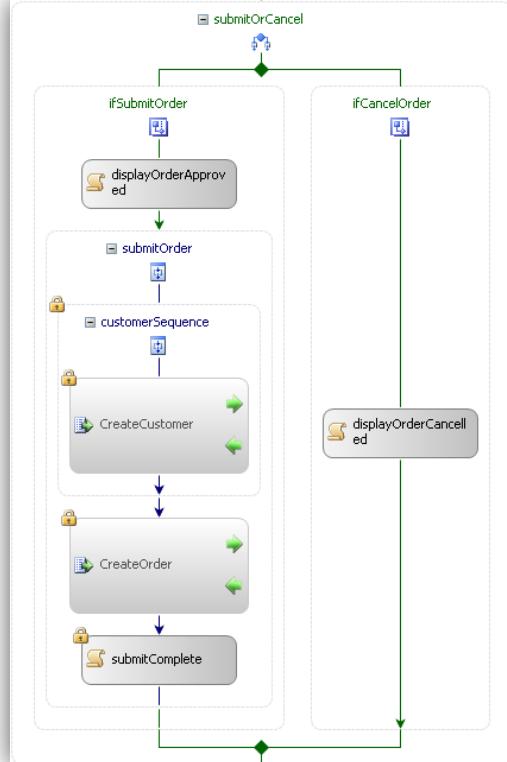
- Build the **OrderSubmission** project. This is done so it will show up in the **Toolbox**.

*Right click the project in the **Solution Explorer** and click **Build**.*

- 2) Embed a new **SubmitOrder** activity into the **ifSubmitOrder** if/else branch in the **EnterOrder** workflow.
 - Open the **EnterOrder** workflow in design mode
*Right click **EnterOrder.cs** in the solution explorer and select **View Designer**.*
 - Drag the new **SubmitOrder** activity onto the canvas following the **submitOrCancel** activity.
*Name the new activity **submitOrder**.*

*Bind the **NewCustomer** property to the **EnterOrder.NewCustomer** property.*

*Bind the **NewOrder** property to the **EnterOrder.NewOrder** property.*



3) Test order entry by adding a new order and submitting it to the services.

- Start both the service application and workflow application at the same time.

Right click the solution and click Set StartUp Projects...

*Verify the **Multiple startup projects** option is selected and both projects have an action of **Start**.*

*Start the project using the **Debug -> Start Without Debugging** menu option.*

- Once the application has started, enter a customer name as **ABC Inc**.
- When asked for a product id, use an id of **1** for the first time and **2** for the second time.
- When asked for the third product id, hit enter to indicate that no more line items are needed.
- Choose to approve the order by entering **yes**.
- Verify the result indicates the order was approved. Also, verify that the service console window shows that both a customer and order were created.

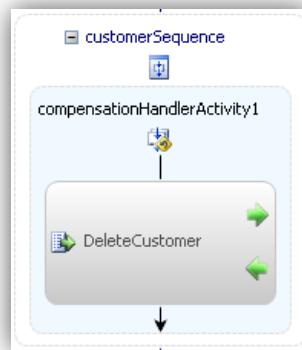
The image displays two separate command-line windows side-by-side. The left window, titled 'C:\WINDOWS\system32\cmd.exe', shows the interaction for creating a customer. It prompts for a customer name ('Enter the customer's name: ABC Inc.'), product IDs ('Enter a product Id (empty if no more): 1', '2'), and line items ('Would you like to submit this order <yes/no>?: yes'). It then displays a success message ('The order was approved.') and ends with 'Press any key to continue . . .' The right window, also titled 'C:\WINDOWS\system32\cmd.exe', shows the interaction for creating an order. It starts with a prompt ('Press any key to stop services...'), followed by 'Creating customer with name: ABC Inc.', and 'New Id: 4'. It then moves on to 'Creating order for customer: 4' and lists two line items: 'Lineitem: 1 - Blue Widget' and 'Lineitem: 2 - Orange Widget'. Finally, it shows 'New Id: 3'.

4) Test what happens if an invalid product id was entered.

- Run the same test again, but use a product id of 10 instead of 1.
- This should cause an error since the order creation service cannot find a product with id 10. We will explore ways of handling this in the next exercise.

Challenge: Handling failure with compensation

- 1) Add a compensation scope to the **SubmitOrder** activity that will allow the **createCustomer** activity to be “undone” if an error occurs
 - Open the **SubmitOrder** activity in design mode
*Right click **SubmitOrder.cs** in the solution explorer and click **View Designer**.*
 - Drag a new **CompensatableSequence** activity onto the canvas following the **createCustomer** activity.
*In the properties pane, set the **Name** property to **customerSequence**.*
 - Drag the existing **createCustomer** activity into the new **customerSequence** activity.
- 2) Implement the compensation process that will undo what the **createCustomer** activity by adding a **deleteCustomer** activity.
 - View the Compensation handler for the **customerSequence**
*Click the icon immediately below the **customerSequence** name
Select **View Compensation Handler** in the drop down menu.*
 - Drag a new **SendActivity** into the compensation handler
*In the properties pane, set the **Name** property to **deleteCustomer**.
Select the **ServiceOperationInfo** property and click its ... button.
Set the **ServiceOperationInfo** property using the **CustomerService.DeleteCustomer** method.
Set the **ChannelToken** property to **CustomerChannel**.
Bind the **customer** property to the **SubmitOrder.NewCustomer** property.*



- 3) Add a fault handler to the **SubmitOrder** activity that will execute when an error occurs and initiate the compensation handler you just created.
 - View the **Fault Handler** for the **SubmitOrder** activity.
*Click the icon immediately below the **SubmitOrder** name.
Select **View Fault Handlers** in the drop down menu.*
 - Drag a new **FaultHandler** onto the fault list in the fault handler.
*In the properties pane, set the **Name** property to **allFaultHandler**
Set the **FaultType** property to **System.Exception**.*
- 4) Log a message to the console indicating that an error occurred.
 - Drag a **Code** activity onto the fault handler canvas.
*In the properties pane, set the **Name** property to **submitError**.
Double click the activity in the designer to generate the code handler.*

- In the **submitError_ExecuteCode** method just generated, enter the code to read the approved flag from the console.

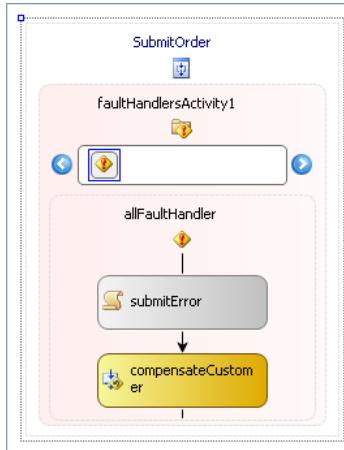
```
Console.WriteLine("Error: There was a problem submitting the order.  
Compensating by deleting the customer. ");
```

5) Cause the compensation handler for the **createCustomer** activity to fire and delete the customer

- Right click **SubmitOrder.cs** in the **Solution Explorer** and click **View Designer**.
- Drag a **Compensate** activity onto the fault handler canvas.

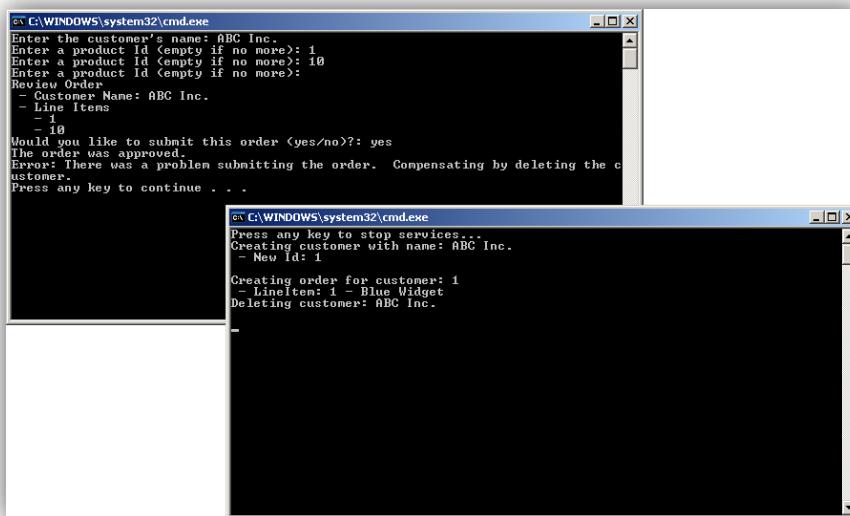
*In the properties pane, set the **Name** property to **compensateCustomer**.*

*Set the **TargetActivityName** property to **customerSequence**.*



6) Test order entry by adding a new order and submitting it to the services.

- Start both the service application and workflow application at the same time.
- Once the application has started, enter a customer name as **ABC Inc.**
- When asked for a product id, use an id of **1** for the first time and **10** for the second time.
- When asked for the third product id, hit enter to indicate that no more line items are needed.
- Choose to approve the order by entering **yes**.
- Verify the result shows order submission failing and the customer was deleted.



Challenge: Lookup the product names using the Replicator activity

- 1) Create a new activity that will be embedded in the existing **EnterOrder** workflow
 - Create a new activity named **LookupProduct**

*Right click the **OrderSubmission** project in the solution explorer and click **Add -> New Item**.*

*Select **Workflow** in the tree view on the left and select **Activity** in the window on the right.*

*Enter **LookupProduct.cs** as the name and click **Add**.*

- 2) Call the **GetProduct** method in the **OrderService** to lookup the name of the products entered.

- Add a property used to pass a **LineItem** to the **LookupProduct** activity.

*Right click **LookupProduct.cs** in the solution explorer and select **View Code**.*

*Add the following property to the **LookupProduct** class.*

```
public LineItem LineItem { get; set; }
```

- Drag a **SendActivity** onto the canvas and name it **getProduct**.

*In the properties pane, set the **Name** property to **getProduct**.*

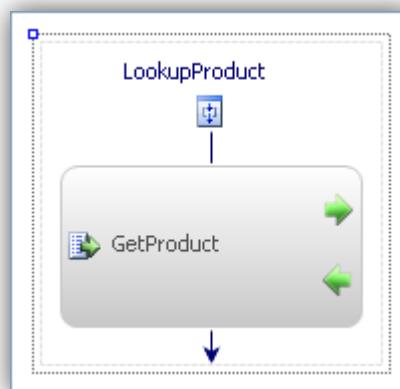
*Set the **ServiceOperationInfo** property using the **OrderService.GetProduct** method.*

*Set the **ChannelToken** property to **OrderChannel**.*

*Expand the **ChannelToken** property and set the **EndpointName** to **WSHttpBinding_IOrderService** and set the **OwnerActivityName** to **LookupProduct**.*

*Bind the **(ReturnValue)** property to the **LookupProduct.LineItem.Product** property.*

*Bind the **productId** property to the **LookupProduct.LineItem.ProductId** property.*



- 3) Add a fault handler to the **LookupProduct** activity to provide a default product name if an error occurs when calling the **GetProduct** method.

- View the **Fault Handler** for the **LookupProduct** activity.

*Click the icon immediately below the **LookupProduct** name.*

*Select **View Fault Handlers** in the drop down menu.*

- Drag a new **FaultHandler** onto the fault list in the fault handler and name it **allFaultHandler**.

*In the properties pane, set the **Name** property to **allFaultHandler***

*Set the **FaultType** property to **System.Exception**.*

- 4) Create a **Product** object that will simulate the results of the call to **GetProduct**.

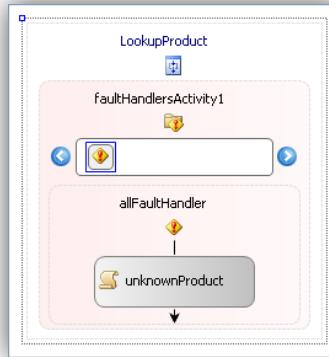
- Drag a **Code** activity onto the fault handler canvas and name it **unknownProduct**.

*In the properties pane, set the **Name** property to **unknownProduct**.*

Double click the activity in the designer to generate the code handler.

- In the **unknownProduct_ExecuteCode** method just generated, enter the code to create a new **Product** object.

```
LineItem.Product = new Product();
LineItem.Product.Id = LineItem.Product.Id;
LineItem.Product.Name = "Unknown";
```



- Integrate the **LookupProduct** activity into a **Replicator** activity so it is executed once for each line item in the order.

- Right click **OrderSubmission** in the **Solution Explorer** and click **Build**.
- Right click **EnterOrder.cs** in the **Solution Explorer** and click **View Designer**.
- Drag a **Replicator** activity onto the **EnterOrder** canvas following the **enterAllLineItems** activity and name it **lookupProducts**.

*In the properties pane, set the **Name** property to **lookupProducts**.*

*Bind the **InitialChildData** property to the **EnterOrder.NewOrder.LineItems** property.*

- Drag a **LookupProduct** activity into the **lookupProducts** activity and name it **lookupProduct**.

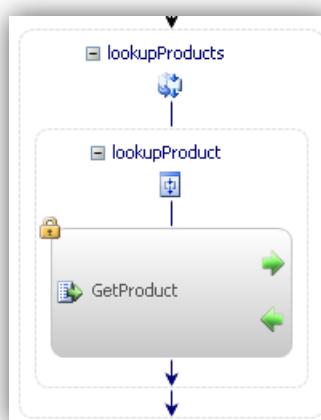
*In the properties pane, set the **Name** property to **lookupProduct**.*

- Implement the **lookupProducts ChildInitialized** handler.

*Set the **ChildInitialized** property to **lookupProducts_ChildInitialized** and hit enter*

*Add the following code to the **lookupProducts_ChildInitialized** method.*

```
LookupProduct send = e.Activity as LookupProduct;
send.LineItem = e.InstanceData as LineItem;
```



6) Update the code that generates the order summary to incorporate the product name.

- Find the **displayCompleteOrder_ExecuteCode** method and replace the last line

```
Console.WriteLine(" - {0}: {1}", lineItem.ProductId, lineItem.Product.Name);
```

7) Test order entry by adding a new order and submitting it to the services.

- Start both the service application and workflow application at the same time.
- Once the application has started, enter a customer name as **ABC Inc**.
- When asked for a product id, use an id of **1** for the first time and **10** for the second time.
- When asked for the third product id, hit enter to indicate that no more line items are needed.
- Review the output to verify the order is displayed correctly and includes the product names.

```
C:\WINDOWS\system32\cmd.exe
Enter the customer's name: ABC Inc.
Enter a product Id <empty if no more>: 1
Enter a product Id <empty if no more>: 10
Enter a product Id <empty if no more>:
Review Order
- Customer Name: ABC Inc.
- Line Items
- 1: Blue Widget
- 10: Unknown
Would you like to submit this order (yes/no)?: yes
The order was approved.
Error: There was a problem submitting the order. Compensating by deleting the customer.
Press any key to continue . . .
```

Lab 03: Understanding the Windows Workflow Runtime

Lab Overview: Now that your company has an automated system to submit orders, it is time to focus on the order fulfillment process. Currently this is a three-step process. As orders are received they are considered submitted. Once an account manager approves them, they move into the Approved step and are waiting to be shipped. Finally, when the order is shipped, the order moves into the closed state. At any point along the way, the order can be canceled.

At each stage of the process, the account managers and the shipping department will be notified the order is in the current state. Once the account manager or the shipping department completes their task, the order should advance through the process.

Exercise 0: Setting up the project

- 1) Open the starter Visual Studio 2008 solution at **\Labs\Lab03\StarterFiles\OrderWorkflow.sln**.

Exercise 1: Creating the ExternalDataExchange interfaces

- 1) Create a new workflow runtime service named **OrderManagerService**.
 - Create a new class named **OrderManagerService** in the **OrderWorkflow** project
Right click on OrderWorkflow in the Solution Explorer.
Select Add -> New Item.
Select Code in the list on the left and select Class in the window on the right.
Enter OrderManagerService.cs as the name and click Add.
 - Drive the **OrderManagerService** from the **WorkflowRuntimeService** class so it can be hosted in the Windows Workflow services list.

```
using System.Workflow.Runtime.Hosting;  
  
namespace OrderWorkflow  
{  
    public class OrderManagerService : WorkflowRuntimeService  
    {
```

- 2) Create an external communications interface named **IOrderManager**.
 - Apply the **ExternalDataExchange** attribute to the interface to identify it as an interface Windows Workflow can use to communicate with the hosting environment.
 - Add a method named **SetOrderStatus** to **IOrderManager** to be used when telling the hosting environment to update an order's status.

```
[ExternalDataExchange]  
public interface IOrderManager  
{  
    void SetOrderStatus(string status);  
}
```

- 3) Make **OrderManagerService** an implementation of **IOrderManager**
 - In the **OrderManagerService.cs** file, derive the **OrderManagerService** from the **IOrderManager** interface.

```
public class OrderManagerService : WorkflowRuntimeService, IOrderManager
{
}
```

- In the **OrderManagerService** class, implement the **IOrderManager.SetOrderStatus** method. This method will update the status of the order in the order database. If the order does not exist, it will be created.

```
public void SetOrderStatus(string status)
{
    // lookup the current workflow instance id
    Guid orderId = WorkflowEnvironment.WorkflowInstanceId;

    // find the order based on the current workflow instance id
    // FirstOrDefault return the order or null is none found
    DataClassesDataContext context = new DataClassesDataContext();
    Order order = context.Orders.FirstOrDefault(n => n.Id == orderId);

    // if the order wasn't found create a new order and insert it
    if (order == null)
    {
        order = new Order();
        order.Id = orderId;
        context.Orders.InsertOnSubmit(order);
    }

    // update the status and commit the changes
    order.Status = status;
    context.SubmitChanges();
}
```

- 4) Add an event to the **OrderManagerService** that will be used to communicate order updates from the service to the user interface.

- Add an **OrderChanged** event to the **OrderManagerService** class.

```
public event EventHandler OrderChanged;
```

- Modify **SetOrderStatus** so **OrderChanged** is called anytime an order is updated.

```
// fire the order changed event
if (OrderChanged != null)
    OrderChanged(this, EventArgs.Empty);
```

5) Create another workflow runtime service named **TaskManagerService**.

- Create a new class named **TaskManagerService** in the **OrderWorkflow** project
- Modify the definition of **TaskManagerService** to make it a public class.

```
public class TaskManagerService
{
}
```

6) Create an external communications interface named **ITaskManager**.

- Apply the **ExternalDataExchange** attribute to the interface to identify it as an interface Windows Workflow can use to communicate with the hosting environment.
- Add methods named **CreateTask** and **RemoveTask** to **ITaskManager** to be used to ask the hosting environment to create and delete user tasks.
- Add two events named **Complete** and **Cancel** to **ITaskManager** to be used when sending events from the hosting environment into the workflow engine.

```
[ExternalDataExchange]
public interface ITaskManager
{
    int CreateTask(string type);
    void RemoveTask(int taskId);

    event EventHandler<ExternalDataEventArgs> Complete;
    event EventHandler<ExternalDataEventArgs> Cancel;
}
```

7) Make **TaskManagerService** an implementation of **ITaskManager**

- In the **TaskManagerService.cs** file, derive the **TaskManagerService** from the **ITaskManager** interface.

```
public class TaskManagerService : ITaskManager
{
```

- Add a **TaskChanged** event to the **TaskManagerService** class.

```
public event EventHandler TaskChanged;
```

- In the **TaskManagerService** class, implement the **ITaskManager.CreateTask** method. This method will create a new task in the tasks database table.

```
public int CreateTask(string type)
{
    // lookup the current workflow instance id
    Guid orderId = WorkflowEnvironment.WorkflowInstanceId;

    // create the new task
    Task task = new Task();
    task.OrderId = orderId;
    task.Type = type;

    // add the task to the context
    DataClassesDataContext context = new DataClassesDataContext();
    context.Tasks.InsertOnSubmit(task);
    context.SubmitChanges();

    // fire the task changed event
    if (TaskChanged != null)
        TaskChanged(this, EventArgs.Empty);

    // return the new task's id
    return task.Id;
}
```

- In the **TaskManagerService** class, implement the **ITaskManager.RemoveTask** method. This method will remove a task when it is complete.

```
public void RemoveTask(int taskId)
{
    // find the task by id
    DataClassesDataContext context = new DataClassesDataContext();
    Task task = context.Tasks.FirstOrDefault(n => n.Id == taskId);

    // if the task was found, delete it
    if (task != null)
        context.Tasks.DeleteOnSubmit(task);
    context.SubmitChanges();

    // fire the task changed event
    if (TaskChanged != null)
        TaskChanged(this, EventArgs.Empty);
}
```

- Add two helper methods named **OnComplete** and **OnCancel**. These methods will wrap the **ITaskManager**'s events and make them easier to call.

```
public void OnComplete(Guid orderId) {
    if (Complete != null)
        Complete(null, new ExternalDataEventArgs(orderId));
}

public void OnCancel(Guid orderId) {
    if (Cancel != null)
        Cancel(null, new ExternalDataEventArgs(orderId));
}
```

Exercise 2: Setup Hosting Environment

- 1) Host the **Windows Workflow** runtime in the **ShoppingApplication** class.

- View the code for the **ShoppingApplication** class.

*Right click **ShoppingApplication.cs** in the **Solution Explorer** and click **View Code**.*

- Add a new member variable **_runtime** of type **WorkflowRuntime**.

```
WorkflowRuntime _runtime = new WorkflowRuntime();
```

- 2) Attach the **ExternalDataExchangeService** and the previously created services to the new **WorkflowRuntime** object.

- In the **ShoppingApplication_Load** method, add a new instance of the **ExternalDataExchangeService** class to the workflow runtime using the **AddService** method

```
// add the external data exchange service
_runtime.AddService(new ExternalDataExchangeService());
```

- Add new instances of the **OrderManagerService** and **TaskManagerService** to the newly created **ExternalDataExchangeService**. Use the **GetService** method of the **WorkflowRuntime** to get a reference to the **ExternalDataExchangeService**.

```
// add the order manager and task manager services to the runtime
_runtime.GetService<ExternalDataExchangeService>().AddService(
    new OrderManagerService());
_runtime.GetService<ExternalDataExchangeService>().AddService(
    new TaskManagerService());
```

- 3) Attach event handlers to the **OrderChanged** and **TaskChanged** events in the newly created **OrderManagerService** and **TaskManagerService**.

```
// register two event handlers that will update the UI
_runtime.GetService<OrderManagerService>().OrderChanged +=
    new EventHandler(ShoppingApplication_OrderChanged);
_runtime.GetService<TaskManagerService>().TaskChanged +=
    new EventHandler(ShoppingApplication_TaskChanged);
```

*Note: The **ShoppingApplication_OrderUpdated** and **ShoppingApplication_TaskUpdated** methods have already been implemented for you.*

- 4) Add code to start the workflow runtime when the initialization of the workflow runtime is completed.

```
// start the runtime
_runtime.StartRuntime();
```

- 5) Shutdown the workflow runtime and clean up when the form closes.

- Locate the **ShoppingApplication_FormClosing** method and add the shut down and dispose code.

```
// shutdown the runtime
_runtime.StopRuntime();
_runtime.Dispose();
```

Exercise 3: Create the primary workflow path

- 1) Create the three order states: **Submitted**, **Approved**, and **Closed**

- Create a new state machine workflow in the project named **OrderWorkflow**

*Right click **OrderWorkflow** in the **Solution Explorer** and click **Add -> New Item**.*

*Select **Workflow** in the tree view on the left and select **State Machine Workflow (code)** in the window on the right.*

*Enter **OrderPipeline.cs** as the name and click **Add**.*

- Define several constants used in the state machine.

*Right click **OrderPipeline.cs** in the **Solution Explorer** and click **View Code**.*

Add the following code to the class.

```
public const string SubmittedStatus = "Submitted";
public const string ApprovedStatus = "Approved";
public const string CompletedStatus = "Completed";
public const string CancelledStatus = "Cancelled";
public const string ApprovalType = "Approval";
public const string ShipType = "Ship";
```

- Drag three **State** activities onto the canvas named **SubmittedState**, **ApprovedState**, **ClosedState**.

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

*In the properties pane, set the **Name** property to **SubmittedState**.*

*Repeat previous two steps for **ApprovedState** and **ClosedState**.*

- Right click on the canvas and select **Properties**

*In the properties pane, set the **CompletedStateName** property to **ClosedState***

*Set the **InitialStateName** property to **SubmittedState***

- 2) Define the transition between the **SubmittedState** and the **ApprovedState**.

- Drag an **EventDriven** activity from the toolbox onto the **SubmittedState** activity and name it **orderApproved**.

*In the properties pane, set the **Name** property to **orderApproved**.*

*Double click the new **orderApproved** activity to open its design window.*

- Drag a new **HandleExternalEvent** activity onto the canvas and name it **handleOrderApproved**.

*In the properties pane, set the **Name** property to **handleOrderApproved**.*

*Set the **InterfaceType** to **OrderWorkflow.ITaskManager***

*Set **EventName** to **Complete***

- Place a new **SetState** activity following **handleOrderApproved** and name it **setStateApproved**.

*In the properties pane, set the **Name** to **setStateApproved**.*

*Set the **TargetStateName** property to **ApprovedState**.*

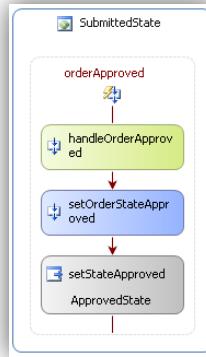
- Add a **CallExternalMethod** activity named **setOrderStateApproved** between the existing activities to initiate communication with the hosting environment.

*In the properties pane, set the **Name** property to **setOrderStateApproved**.*

*Set the **InterfaceType** property to **OrderWorkflow.IOrderManager**.*

*Set the **MethodName** property to **SetOrderStatus**.*

*Bind the **status** property to the **OrderPipeline.ApprovedStatus** property.*



- Switch back to the state view by clicking the **OrderPipeline** link in the upper left corner of the designer.

3) Define the transition between the **ApprovedState** and the **ClosedState**.

- Drag an **EventDriven** activity from the toolbox onto the **ApprovedState** and name it **orderCompleted**.

*In the properties pane, set the **Name** property to **orderCompleted**.*

- Drag a new **HandleExternalEvent** activity onto the canvas and name it **handleOrderCompleted**.

*In the properties pane, set the **Name** property to **handleOrderCompleted**.*

*Set the **InterfaceType** property to **OrderWorkflow.ITaskManager***

*Set the **EventName** property to **Complete***

- Place a new **SetState** activity following **handleOrderCompleted** and name it **setStateCompleted**.

*In the properties pane, set the **Name** property to **setStateCompleted**.*

*Set the **TargetStateName** property to **ClosedState**.*

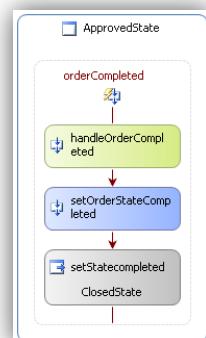
- Add a **CallExternalMethod** activity between the existing activities and name it **setOrderStateCompleted**.

*In the properties pane, set the **Name** property to **setOrderStateCompleted**.*

*Set the **InterfaceType** property to **OrderWorkflow.IOrderManager**.*

*Set the **MethodName** property to **SetOrderStatus**.*

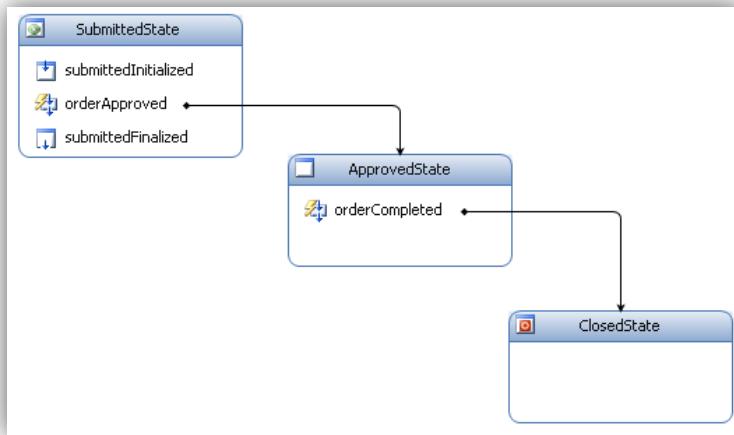
*Bind the **status** property to the **OrderPipeline.CompletedStatus** property.*



- Switch back to the state view by clicking the **OrderPipeline** link in the upper left corner of the designer.

- 4) Manage approval tasks whenever flow transitions into and out of the submitted state.

- Drag a **StateInitialization** activity onto the **SubmittedState** activity.
 - I. Name the new activity **submittedInitialized**.*
- Drag a **StateFinalized** activity onto the **SubmittedState** activity.
 - I. Name the new activity **submittedFinalized**.*

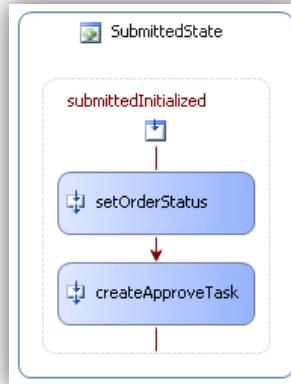


- Double click the **submittedInitialized** activity to open its design window.
- Set the initial state of the order by dragging a **CallExternalMethod** activity into **submittedInitialized** activity and name it **setOrderStatus**.

*In the properties pane, set the **Name** property to **setOrderStatus**.
 Set the **InterfaceType** property to **OrderWorkflow.IOrderManager**.
 Set the **MethodName** property to **SetOrderStatus**.
 Bind the **status** property to the **OrderPipeline.SubmittedStatus** property.*

- Create the approval task by dragging a **CallExternalMethod** activity onto the **submittedInitialized** activity and name it **createApproveTask**.

*In the properties pane, set the **Name** property to **createApproveTask**.
 Set the **InterfaceType** property to **OrderWorkflow.ITaskManager**.
 Set the **MethodName** property to **CreateTask**.
 Bind the **type** property to the **OrderPipeline.ApprovalType** property.
 Bind the **(ReturnValue)** property to a new **TaskId** property.*



- Switch back to the state view by clicking the **OrderPipeline** link in the upper left corner of the designer.

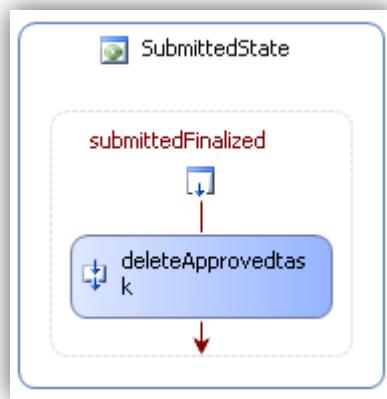
- Double click the **submittedFinalized** activity to open its design window.
- Delete the completed approval task by dragging a **CallExternalMethod** activity onto the **submittedFinalized** activity.

*Name the new activity **deleteApprovedTask**.*

*Set the **InterfaceType** property to **OrderWorkflow.ITaskManager**.*

*Set the **MethodName** property to **RemoveTask**.*

*Bind the **taskId** property to the **OrderPipeline.TaskId** property.*



- Switch back to the state view by clicking the **OrderPipeline** link in the upper left corner of the designer.

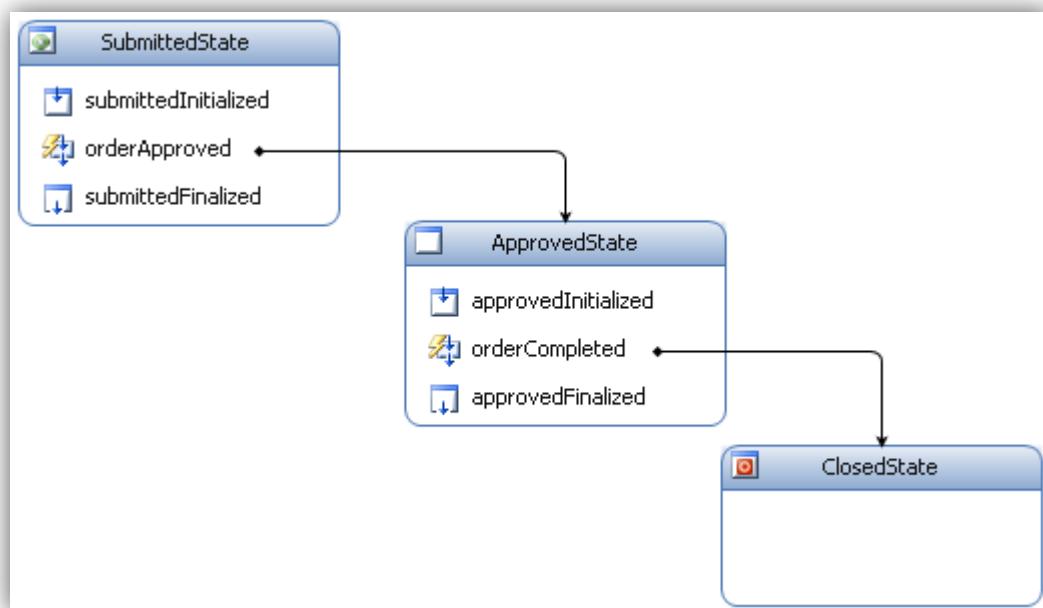
5) Manage ship tasks whenever flow transitions into and out of the approved state.

- Drag a **StateInitialization** activity onto the **ApprovedState** activity.

*In the properties pane, set the **Name** property to **approvedInitialized**.*

- Drag a **StateFinalized** activity onto the **ApprovedState** activity.

*In the properties pane, set the **Name** property to **approvedFinalized**.*



- Double click the **approvedInitialized** activity to open its design window.

- Create the approval task by dragging a **CallExternalMethod** activity onto the **approvedInitialized** activity and naming it **createShipTask**.

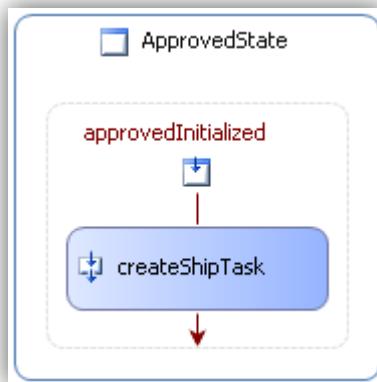
*In the properties pane, set the **Name** property to **createShipTask**.*

*Set the **InterfaceType** property to **OrderWorkflow.ITaskManager**.*

*Set the **MethodName** property to **CreateTask**.*

*Bind the **type** property to the **OrderPipeline.ShipType** property.*

*Bind the **(ReturnValue)** property to the **OrderPipeline.TaskId** property.*



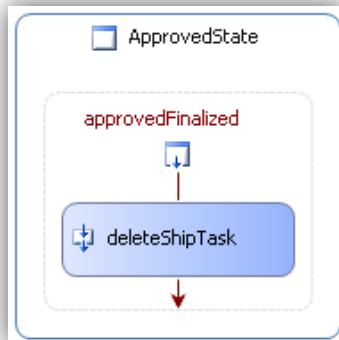
- Switch back to the state view by clicking the **OrderPipeline** link in the upper left corner of the designer.
- Double click the **approvedFinalized** activity to open its design window.
- Delete the completed approval task by dragging a **CallExternalMethod** activity onto the **approvedFinalized** activity and name it **deleteShipTask**.

*In the properties pane, set the **Name** property to **deleteShipTask**.*

*Set the **InterfaceType** property to **OrderWorkflow.ITaskManager**.*

*Set the **MethodName** property to **RemoveTask**.*

*Bind the **taskId** property to the **OrderPipeline.TaskId** property.*



Exercise 4: Connect the workflow to the user interface

- Implement the handler for the **Submit New Order** button to start a new **OrderPipeline** workflow.
 - Right click on the **ShoppingApplication.cs** file in the solution explorer and choose **View Code**.
 - Locate the **btnSubmitOrder_Click** method and use the **CreateWorkflow** method to create a new workflow instance of type **OrderPipeline**.

```
WorkflowInstance instance = _runtime.CreateWorkflow(typeof(OrderPipeline));
```

- Start the new instance using its **Start** method.

```
instance.Start();
```

- 2) When a task is displayed in the user interface, a button will be used to communicate with the workflow engine that the task is completed.

- Locate the **gridTasks_CellContentClick** and add code to call the **OnComplete** method of the **TaskManagerService**.

```
Task task = gridTasks.Rows[e.RowIndex].DataBoundItem as Task;  
_runtime.GetService<TaskManagerService>().OnComplete(task.OrderId);
```

Note: Most of the code in this method has been implemented for you as it does not relate to workflow. The first line in the above code segment is already implemented and is included for orientation purposes only.

- 3) When the workflow is completed, the runtime fires a **WorkflowCompleted** event. Use this to display a nice message to the user.

- Locate the **ShoppingApplication_Load** method and register a handler for the runtime's **WorkflowCompletedEvent**.

Register the method at the top of the function

```
// register the event handlers  
_runtime.WorkflowCompleted += new  
EventHandler<WorkflowCompletedEventArgs>(Runtime_WorkflowCompleted);
```

- Create a new method named **Runtime_WorkflowCompleted**.

```
void Runtime_WorkflowCompleted(object sender, WorkflowCompletedEventArgs e)  
{  
}
```

- Add code to access the database and loads the details of the order that was just completed.

```
// find the order that just completed  
DataClassesDataContext context = new DataClassesDataContext();  
Order order = context.Orders.FirstOrDefault(n => n.Id ==  
e.WorkflowInstance.InstanceId);
```

- If the order was found, display a message box with an order summary, then deletes the order.

```
// if the order exists, display a message and remove the order  
if (order != null) {  
    // display the message  
    MessageBox.Show(  
        string.Format("Order '{0}' was '{1}'.",  
        e.WorkflowInstance.InstanceId, order.Status));  
  
    // delete the order  
    context.Orders.DeleteOnSubmit(order);  
    context.SubmitChanges();  
}
```

- Finally update the order data source.

```
// update the order display
this.Invoke(new ThreadStart(delegate() {
    orderBindingSource.DataSource = context.Orders; }));
```

- 4) Test the application by passing an order through the system.
- When the application has loaded, click the **Submit New Order** to place an order in the pipeline.
 - A task of type approve should appear in the tasks list. Click the **Complete** button related to the task to indicate the order was approved.
 - This will change the status of the order and add a new task of type **Ship**. Click the task's **complete** button to indicate the order was shipped.
 - A message box will confirm the final status of the order and the lists should be cleared. The order has been successfully processed.

Challenge: Add the ability to cancel orders

- Add the global **orderCancelled** EventDriven activity to the **OrderPipeline**.
 - View the design for the **OrderPipeline** workflow
 - Right click **OrderPipeline.cs** in the solution explorer and select **View Designer**.*
 - Drag an **EventDriven** activity onto the canvas outside any existing **State** activities.

*Name the new activity **orderCancelled***

Note: *By placing this activity outside a state, it is active in all states.*
 - Drag a new **HandleExternalEvent** activity onto the canvas and name it **handleOrderCancelled**.

*In the properties pane, set the **Name** property to **handleOrderCancelled**.*

*Set the **InterfaceType** property to **OrderWorkflow.ITaskManager***

*Set the **EventName** property to **Cancel***
 - Place a new **SetState** activity following **handleOrderCancelled** and name it **setStateCancelled**.

*In the properties pane, set the **Name** property to **setStateCancelled**.*

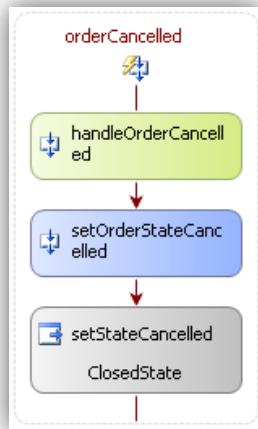
*Set the **TargetStateName** property to **ClosedState**.*
 - Add a **CallExternalMethod** activity between the existing activities and name it **setOrderStateCancelled**.

*In the properties pane, set the **Name** property to **setOrderStateCancelled**.*

*Set the **InterfaceType** property to **OrderWorkflow.IOrderManager**.*

*Set the **MethodName** property to **SetOrderStatus**.*

*Bind the **status** property to the **OrderPipeline.CancelledStatus** property.*



- 2) When an order is displayed in the user interface, a cancel button is associated with it. Connect the cancel button with the **ITaskManager.OnCancel** method.

- Right click **ShoppingApplication.cs** in the **Solution Explorer** and click **View Code**.
- Locate the **gridOrders_CellContentClick** and add code to call the **OnCancel** method of the **TaskManagerService**.

```
Order order = gridOrders.Rows[e.RowIndex].DataBoundItem as Order;
_runtime.GetService<TaskManagerService>().OnCancel(order.Id);
```

Note: Most of the code in this method has been implemented for you as it does not relate to workflow. The first line in the above code segment is already implemented and is included for orientation purposes only.

- 3) Test the application by passing an order through the system and cancelling it.
- When the application has loaded, click the **Submit New Order** to place an order in the pipeline.
 - An order should appear in the order list. Click the **cancel** button to cancel the order.
 - A message box will confirm the final status of the order and the lists should be cleared. The order has been successfully canceled.

Challenge: Use persistence for long running processes

- 1) Build a configuration file entry that will cause the persistence service to be loaded into the workflow runtime. The persistence service will store running workflows in a pre initialized database.
- Open the **app.config** file and add the following line to the **configuration/configSections** node.

```
<section name="workflowRuntime"
type="System.Workflow.Runtime.Configuration.WorkflowRuntimeSection,
System.Workflow.Runtime, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
```

- Add the following section to the configuration node.

```
<workflowRuntime Name="SampleApplication">
  <Services>
    <add type="System.Workflow.Runtime.Hosting.SqlWorkflowPersistenceService,
      System.Workflow.Runtime, Version=3.0.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35"
      ConnectionString="Data Source=.\SQLEXPRESS;
        AttachDbFilename=|DataDirectory|Database.mdf;
        Integrated Security=True;Connect Timeout=30;
        User Instance=True"
      UnloadOnIdle="true"/>
  </Services>
</workflowRuntime>
```

- 2) Update the creation of the **WorkflowRuntime** object to use the config file when loading.

- Open the code for **ShoppingApplication** form.
*Right click **ShoppingApplication.cs** in the solution explorer and click **View Code**.*
- Change the code creating the **_runtime** to the following code.

```
workflowRuntime _runtime = new WorkflowRuntime("workflowRuntime");
```

- 3) Add a handler to the workflow runtime what will log whenever the workflow instance is persisted to the database.

- Add another line of code following the registering of the **WorkflowCompleted** handler that handles the **WorkflowPersistence** event.

```
// register the event handlers
_runtime.WorkflowCompleted += new
EventHandler<WorkflowCompletedEventArgs>(Runtime_WorkflowCompleted);
_runtime.WorkflowPersisted += new
EventHandler<WorkflowEventArgs>(Runtime_WorkflowPersisted);
```

- 4) Test the application by passing an order through the system. Notice that after each step, the workflow is persisted.

Lab 04: Windows Workflow Foundation Integration with SharePoint 2007

Lab Overview: Litware Inc. has been using SharePoint 2003 to store documents for several years and has recently upgraded to SharePoint 2007. They have read about the integrated workflow support in SharePoint 2007. Based on what they have read, they would like to evaluate it using a simple Approval process.

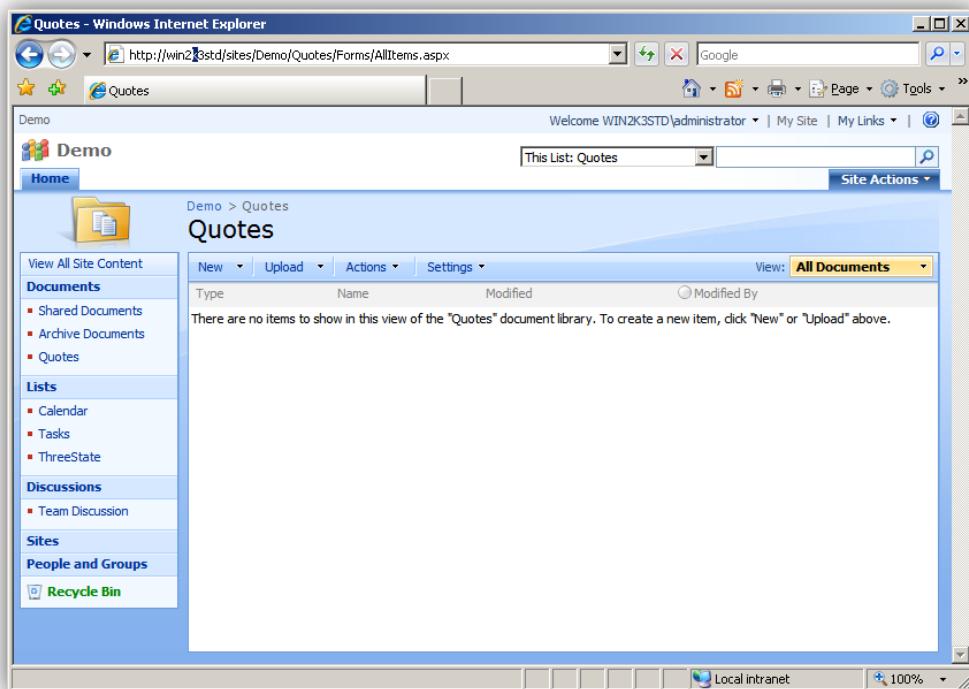
First, you will need to create a document library containing quotes submitted by others. Next, attach an Approval workflow to the Quotes document library. Configure the SharePoint to start a new Approval process for each new quote added to the list.

Exercise 0: Setting up the project

- 1) 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.

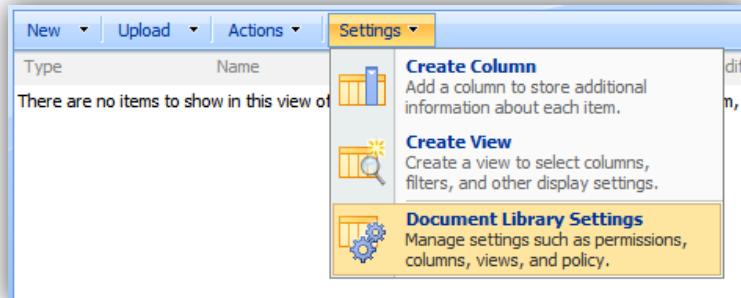
Exercise 1: Attaching an Approval Workflow

- 1) Create a new **Quotes** document library.
 - 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.

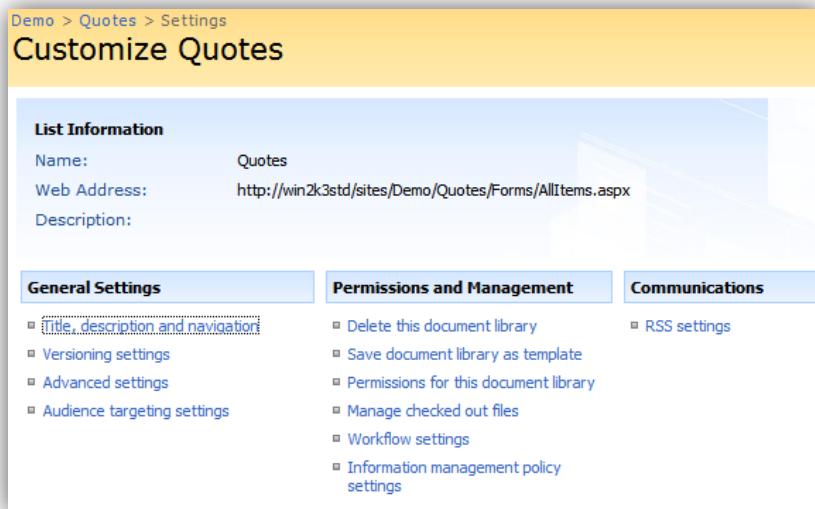


2) Associate an **Approval** workflow named **Quote approval** to the new **Quotes** document library.

- Starting from the **Quotes** document library, click the **Settings -> Document Library Settings** link.



- On the **Customize** page, click the **Workflow settings** link in the **Permissions and Management** section header.



- Choose the **Approval** workflow template and enter a name of **Quote Approval** in the name text box.

A screenshot of the 'Add a Workflow: Quotes' page. The title bar shows 'Demo > Quotes > Settings > Workflow settings > Add or Change a Workflow'. The page title is 'Add a Workflow: Quotes'. A message at the top says 'Use this page to set up a workflow for this document library.' On the left, there is a 'Workflow' section with instructions to select a workflow template. In the center, there is a 'Select a workflow template:' dropdown menu with four options: 'Approval' (selected), 'Collect Feedback', 'Collect Signatures', and 'Disposition Approval'. To the right of the dropdown is a 'Description' field containing the text: 'Routes a document for approval. Approvers can approve or reject the document, reassign the approval task, or request changes to the document.' At the bottom, there is a 'Name' section where the user has typed 'Quote Approval' into a text input field.

- Choose the existing **Tasks** list for the task list and **Workflow History** list for the history list.

Task List Select a task list to use with this workflow. You can select an existing task list or request that a new task list be created.	Select a task list: <input type="text" value="Tasks"/> Description: Use the Tasks list to keep track of work that you or your team needs to complete.
History List Select a history list to use with this workflow. You can select an existing history list or request that a new history list be created.	Select a history list: <input type="text" value="Workflow History"/> Description: History list for workflow.

- Choose the **Allow this workflow to be manually started** checkbox to allow manual workflow starting. Also choose the **Start this workflow when a new item is created** option to automatically start the workflow.

Start Options Specify how this workflow can be started.	<input checked="" type="checkbox"/> Allow this workflow to be manually started by an authenticated user with Edit Items Permissions. <input type="checkbox"/> Require Manage Lists Permissions to start the workflow. <input type="checkbox"/> Start this workflow to approve publishing a major version of an item. <input checked="" type="checkbox"/> Start this workflow when a new item is created. <input type="checkbox"/> Start this workflow when an item is changed.
---	--

- Click **Next** to move to the next step.

3) Define the **Approval** workflow specific parameters needed to finish the association of the **Quote Approval** workflow.

- In the **Workflow Tasks** section, leave the options as they are allowing only one approver to approve at a time and to give the approver the option of reassigning approval or resubmitting to the original submitter.

Workflow Tasks Specify how tasks are routed to participants and whether to allow tasks to be delegated or if participants can request changes be made to the document prior to finishing their tasks.	Assign tasks to: <input type="radio"/> All participants simultaneously (parallel) <input checked="" type="radio"/> One participant at a time (serial) Allow workflow participants to: <input checked="" type="checkbox"/> Reassign the task to another person <input checked="" type="checkbox"/> Request a change before completing the task
---	--

- In the **Default Workflow Start Values** section, add the **Administrator** to the approver list. If you'd like, enter a default message as well.

Default Workflow Start Values

Specify the default values that this workflow will use when it is started. You can opt to allow the person who starts the workflow to change or add participants.

Type the names of people you want to participate when this workflow is started. Add names in the order in which you want the tasks assigned (for serial workflows).

Assign a single task to each group entered (Do not expand groups).
 Allow changes to the participant list when this workflow is started

Type a message to include with your request:
Please approve this quote

- At the bottom of the **Default Workflow Start Values** section give each approver 1 day to complete their task and don't notify anyone else of their task.

Due Date

If a due date is specified and e-mail is enabled on the server, participants will receive a reminder on that date if their task is not finished.

Tasks are due by (parallel):

Give each person the following amount of time to finish their task (serial):

Notify Others

To notify other people when this workflow starts without assigning tasks, type names on the CC line.

- In the **Complete the Workflow** section, check the **Document is rejected** option to cancel the workflow if any approver rejects the quote.

Complete the Workflow

Specify when you want the workflow to be completed. If you do not select any options, the workflow will be completed when all tasks are finished.

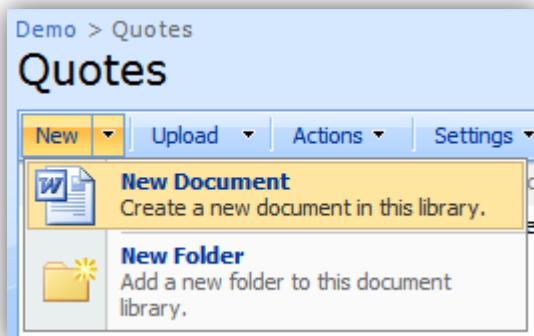
Complete this workflow when:
 Following number of tasks are finished:

Cancel this workflow when the:
 Document is rejected
 Document is changed

- Click **OK** to complete the workflow association.

4) Add a document to the **Quotes** document library.

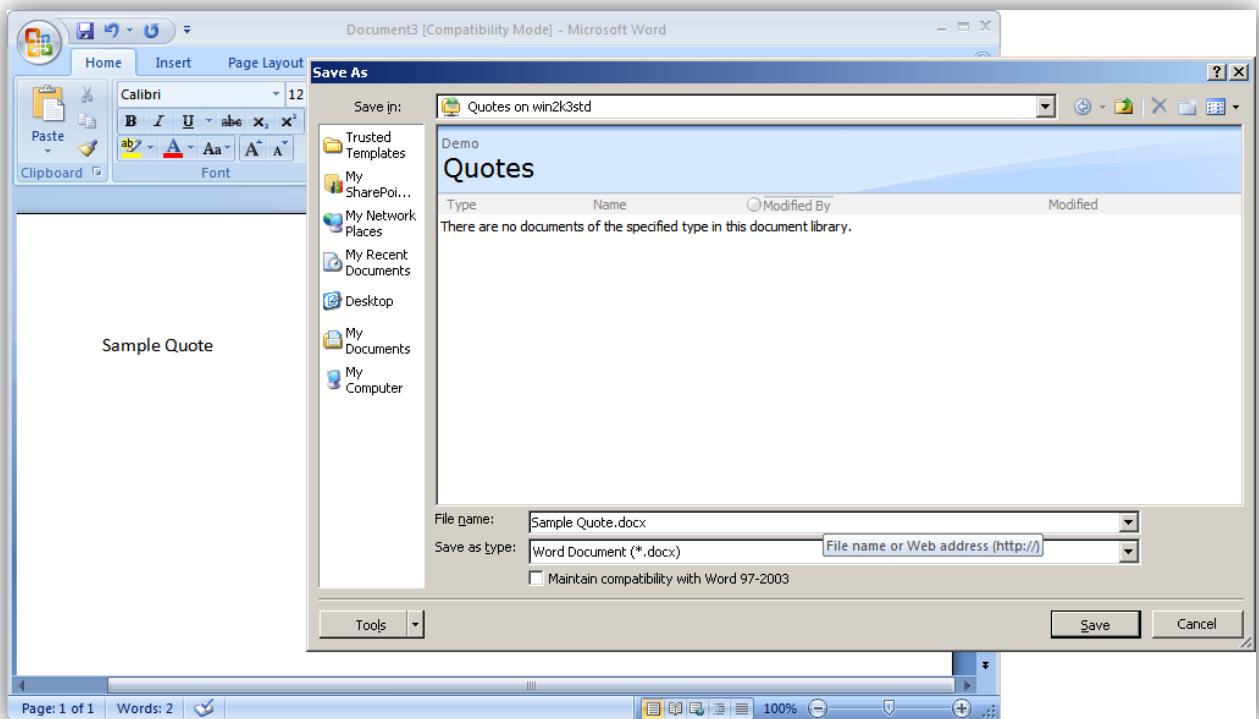
- Navigate to the **Quote** document library using the navigation links at the top of the page.
- Add a new **Quote** to the document library by clicking the **New -> New Document** button.



- If the warning dialog appears, 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.



5) Approve the new quote.

- Switch back to the **Quotes** document library in **Internet Explorer** and refresh the page. You should see the new document with the **Quote Approval** column set to **In Progress**.

Demo > Quotes				
Quotes				
New	Upload	Actions	Settings	View: All Documents
Type	Name	Modified	Modified By	Quote Approval
	Sample Quote ! NEW	1/16/2008 12:41 PM	WIN2K3STD\administrator	In Progress

- Click the **In Progress** link to view the workflow instance's statistics.
- In the **Tasks** section, locate the task item in the list and click its **Title**.

Demo > Quotes > Workflow Status
Workflow Status: Quote Approval

Workflow Information

Initiator:	WIN2K3STD\administrator	Document:	Sample Quote
Started:	1/16/2008 12:41 PM	Status:	In Progress
Last run:	1/16/2008 12:41 PM		

Update active tasks
 Add or update approvers
 Cancel this workflow

If an error occurs or this workflow stops responding, it can be terminated. Terminating the workflow will set its status to Canceled and will delete all tasks created by the workflow.
 Terminate this workflow now.

Tasks

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list Tasks.

Assigned To	Title	Due Date	Status	Outcome
WIN2K3STD\administrator	Please approve Sample Quote ! NEW	1/17/2008	Not Started	

- In the task form, enter some comments in the text box and click the **Approve** button to approve the workflow.

Demo > Tasks > Please approve Sample Quote
Tasks: Please approve Sample Quote

Delete Item
 This workflow task applies to Sample Quote.

Approval Requested

From: WIN2K3STD\administrator
Due by: 1/17/2008 12:41:54 PM

Please approve this quote

Type comments to include with your response:
This looks good to me.]

Other options
[Reassign task](#) [Request a change](#)

6) View the quote's approval status.

- In the **Workflow Status** form, verify the **Status** in the **Workflow Information** section shows **Approved**.

Workflow Information

Initiator: LitwareInc Administrator	Document: Sample Quote
Started: 1/26/2008 7:20 PM	Status: Approved
Last run: 1/26/2008 7:24 PM	

- In the **Workflow History** section, you should see a list of four history list items detailing the steps taken by the **Approval** workflow.

Workflow History

The following events have occurred in this workflow.

Date Occurred	Event Type	User ID	Description	Outcome
1/26/2008 7:20 PM	Workflow Initiated	LitwareInc Administrator	Quote Approval was started. Participants: LitwareInc Administrator	
1/26/2008 7:20 PM	Task Created	LitwareInc Administrator	Task created for LitwareInc Administrator. Due by: 1/27/2008 7:20:31 PM	
1/26/2008 7:24 PM	Task Completed	LitwareInc Administrator	Task assigned to LitwareInc Administrator was approved by LitwareInc Administrator. Comments: This looks good to me.	Approved by LitwareInc Administrator
1/26/2008 7:24 PM	Workflow Completed	LitwareInc Administrator	Quote Approval was completed.	Quote Approval on Sample Quote has successfully completed. All participants have completed their tasks.

7) Create another document and repeat the process with the exception of **Rejecting** the task instead of approving it.

- Name the document **Second Quote.docx**.

Demo Site > Quotes

Quotes

Type	Name	Modified	Modified By	Quote Approval
Word Document	Sample Quote ! NEW	1/26/2008 7:19 PM	LitwareInc Administrator	Approved
Word Document	Second Quote ! NEW	1/26/2008 7:25 PM	LitwareInc Administrator	Rejected

8) View statistics on all workflow instances

- Enable the **Reporting** Site collection feature.

*Navigate to **Site Actions -> Site Settings**.*

*Click the **Site collection features** link in the **Site Collection Administration** section.*

*Verify that the **Reporting** feature is active.*

 **Reporting**
Creates reports about information in Windows SharePoint Services.

Deactivate **Active**

- Navigate to the **Quotes** document library and click the status link on one of the quotes.

Workflow History					
Date Occurred	Event Type	User ID	Description	Outcome	
1/16/2008 12:55 PM	Workflow Initiated	WIN2K3STD\administrator	Quote Approval was started. Participants: WIN2K3STD\administrator		
1/16/2008 12:55 PM	Task Created	WIN2K3STD\administrator	Task created for WIN2K3STD\administrator. Due by: 1/17/2008 12:55:31 PM		
1/16/2008 12:56 PM	Task Completed	WIN2K3STD\administrator	Task assigned to WIN2K3STD\administrator was rejected by WIN2K3STD\administrator. Comments:	Rejected by WIN2K3STD\administrator	
1/16/2008 12:56 PM	Workflow Completed	WIN2K3STD\administrator	Quote Approval was completed.	Quote Approval on Second Quote has ended because WIN2K3STD\administrator has rejected.	

- Click the **View workflow reports** in the **Workflow History** section.

Quotes - View Workflow Reports

Use these reports to monitor how your business processes are running based on the history information of those workflows.

[Go Back to Quotes](#)

Document: Approval

- [Activity Duration Report](#)
Use this report to see how long it is taking for each activity within this workflow to complete, as well as how long it takes each instance to complete.
- [Cancellation & Error Report](#)
Use this report to see which workflows are being canceled or encounter errors before completion.

Document: Collect Feedback

- [Activity Duration Report](#)
Use this report to see how long it is taking for each activity within this workflow to complete, as well as how long it takes each instance to complete.
- [Cancellation & Error Report](#)
Use this report to see which workflows are being canceled or encounter errors before completion.

Document: Collect Signatures

- [Activity Duration Report](#)
Use this report to see how long it is taking for each activity within this workflow to complete, as well as how long it takes each instance to complete.
- [Cancellation & Error Report](#)
Use this report to see which workflows are being canceled or encounter errors before completion.

Quote Approval

- [Activity Duration Report](#)
Use this report to see how long it is taking for each activity within this workflow to complete, as well as how long it takes each instance to complete.
- [Cancellation & Error Report](#)
Use this report to see which workflows are being canceled or encounter errors before completion.

- In the **Quote Approval** section, locate and click the **Activity Duration Report** link to generate an Excel report. The first page contains a pivot chart summarizing the raw data. Switch to the second page to see the raw data the pivot table is based on.

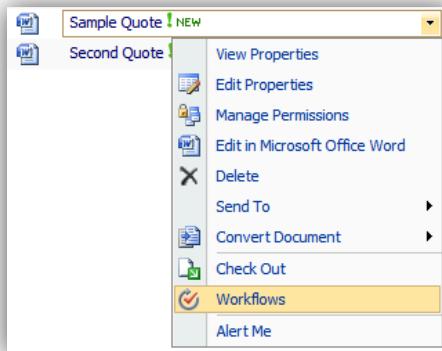
Workflow Completed						
A	B	C	D	E	F	G
1 Workflow ID	(All)					
2						
3 Average of Duration (Hours)	Event					
4 Group	Task Completed	Task Created	Workflow Completed	Workflow Initiated	Grand Total	
5 0	0.03	0.00	0.03	0.00	0.01	
6 Grand Total	0.03	0.00	0.03	0.00	0.01	
7						

A	B	C	D	E	F	G	H	I	J	K	L	M
1 Workflow I Workflow Workflow TList	Item	User	Date Occurred	Event	Group	Outcome	Duration (h)	Description Data				
2 {505d6e53 Quote App (c6964bff-b Quotes	Data.docx	WIN2K3ST	2008-01-16T12:09:31	Workflow I	0		0	Quote App <my:myField>				
3 {505d6e53 Quote App (c6964bff-b Quotes	Data.docx	WIN2K3ST	2008-01-16T12:09:31	Task Creat	0		0	Task creat <Data><All>				
4 {505d6e53 Quote App (c6964bff-b Quotes	Data.docx	WIN2K3ST	2008-01-16T12:12:20	Task Com	0 Approved b	0.047179	0.047179	Task assig <Data><New>				
5 {505d6e53 Quote App (c6964bff-b Quotes	Data.docx	WIN2K3ST	2008-01-16T12:12:21	Workflow C	0 Quote App	0.04724	0.04724	Quote App <my:myField>				
6 {8876a5b5 Quote App (c6964bff-b Quotes	Data2.doc	WIN2K3ST	2008-01-16T12:17:33	Workflow I	0		0	Quote App <my:myField>				
7 {8876a5b5 Quote App (c6964bff-b Quotes	Data2.doc	WIN2K3ST	2008-01-16T12:17:33	Task Creat	0		0	Task creat <Data><All>				
8 {8876a5b5 Quote App (c6964bff-b Quotes	Data2.doc	WIN2K3ST	2008-01-16T12:17:44	Task Com	0 Rejected b	0.003204	0.003204	Task assig <Data><New>				
9 {8876a5b5 Quote App (c6964bff-b Quotes	Data2.doc	WIN2K3ST	2008-01-16T12:17:44	Workflow C	0 Quote App	0.00316	0.00316	Quote App <my:myField>				

Exercise 2: Starting the Workflow Manually

1) Start the **Quote Approval** workflow manually.

- Navigate to the **Quotes** document library.
- Hover over the **Sample Quote** document name and click the **Workflows** option in the drop down.



- On the **Workflows** page click the **Quote Approval** workflow to start a new instance of it.

A screenshot of the "Workflows: Sample Quote" page. The title bar says "Demo > Quotes > Sample Quote > Workflows" and the main heading is "Workflows: Sample Quote". Below it says "Use this page to start a new workflow on the current item or to view the status of a running or completed workflow." There is a section titled "Start a New Workflow" with three options:

- Approval**: Routes a document for approval. Approvers can approve or reject the document, reassign the approval task, or request changes to the document.
- Collect Feedback**: Routes a document for review. Reviewers can provide feedback, which is compiled and sent to the document owner when the workflow has completed.
- Quote Approval**: Routes a document for approval. Approvers can approve or reject the document, reassign the approval task, or request changes to the document.

- On the **Start Workflow** page, leave the approvers list the same, but change the message to something other than the default.

A screenshot of the "Start \"Quote Approval\"": Sample Quote page. The title bar says "Demo > Quotes > Sample Quote > Workflows > Start Workflow" and the main heading is "Start \"Quote Approval\"": Sample Quote".

Request Approval

To request approval for this document, type the names of the people who need to approve it on the **Approvers** line. Each person will be assigned a task to approve your document. You will receive an e-mail when the request is sent and once everyone has finished their tasks.

Add approver names in the order you want the tasks assigned:

WIN2K3STD\administrator

Assign a single task to each group entered (Do not expand groups).

Type a message to include with your request:
Please approve this quote as soon as possible. The customer is waiting.

- Click the **Start** button to start the workflow.

2) Modify the running workflow instance.

- Click the **In Progress** link on the **Sample Quote** document to view the workflow instance's status.
- In the **Workflow Information** section, click the **Update active tasks** list to update all current tasks for the workflow instance.

Workflow Information

Initiator:	LitwareInc Administrator	Document:	Sample Quote
Started:	1/26/2008 7:45 PM	Status:	In Progress
Last run:	1/26/2008 7:45 PM	Update active tasks Add or update approvers Cancel this workflow	

- Change the due date of the tasks to today's date and update the message to indicate the change.

Demo > Quotes > Sample Quote > Workflow Status > Modify Workflow

Modify Workflow: Quote Approval

Update active tasks
Use this page to update active tasks. Completed tasks will be unaffected. Participants will be notified automatically when their task is updated.

Update the date tasks are due by:

Update the message included with your request:

Please approve this quote as soon as possible. The customer is waiting. If we don't get this approved immediately we won't get the project.

- Click **OK** to commit the changes.
- Verify the due date of the task has changed to today's date.

Tasks

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

Assigned To	Title	Due Date
LitwareInc Administrator	Please approve Sample Quote ! NEW	1/26/2008

3) Request a change to the workflow.

- On the **Workflow Status** page In the **Tasks** section, click the title of the approval task.
- On the tasks form, click the **Request** a change link at the bottom of the form.
- On the new **Tasks** page that is loaded, leave the user to request the change from as the workflow's owner.

Demo Site > Tasks > Please approve Sample Quote

Tasks: Please approve Sample Quote

X Delete Item

This workflow task applies to Sample Quote.

Request a Change

If this document needs to be changed before you can finish your task, use this form to request the change. After the change is made, you will again be asked to perform your task.

Request a change from:

The workflow owner: LitwareInc Administrator
 Another person:
Assign To: []

Type your request:
Please approve this quote as soon as possible

- Set the due date to today's date.

Due Date
If a due date is specified and email is enabled on the server, the task owner will receive a reminder on that date if their task is not finished.

Task is due by: 1/26/2008 []

- Click **Send** to submit the new task.
- A new task will exist in the **Tasks** list requesting the change.

Tasks					
The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list Tasks.					
Assigned To	Title	Due Date	Status	Outcome	
LitwareInc Administrator	Please approve Sample Quote [NEW]	1/26/2008	Completed	Change Requested of LitwareInc Administrator	
LitwareInc Administrator	A change has been requested on Sample Quote [NEW]	1/26/2008	Not Started		

4) Complete the request change task.

- In the **Tasks** section, click the **Title** of new task.
- Enter the response as if you'd made the changes and click **Send Response**.

- In the **Tasks** section, verify the request change task is completed and a new approval task exists.

Tasks					
The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list Tasks .					
Assigned To	Title	Due Date	Status	Outcome	
LitwareInc Administrator	Please approve Sample Quote ! NEW	1/26/2008	Completed	Change Requested of LitwareInc Administrator by LitwareInc Administrator	
LitwareInc Administrator	A change has been requested on Sample Quote ! NEW	1/26/2008	Completed	Finished by LitwareInc Administrator	
LitwareInc Administrator	Please approve Sample Quote ! NEW	1/27/2008	Not Started		

5) Complete the approval task.

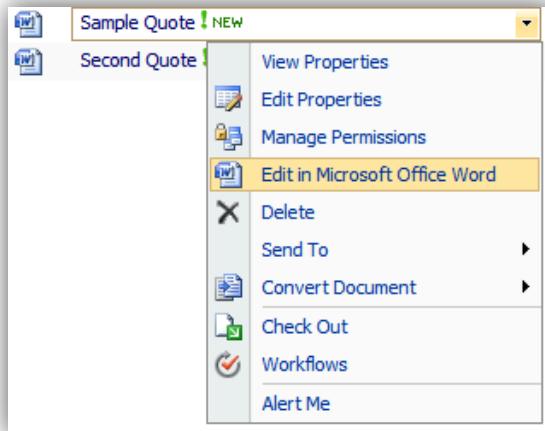
- Complete the new approval task by approving the tasks, just like in the previous exercise.
- The workflow's status in the **Workflow Information** section should be **Approved** and the **Workflow History** section should show the entire process.

Workflow History					
Date Occurred	Event Type	User ID	Description	Outcome	
1/26/2008 7:45 PM	Workflow Initiated	LitwareInc Administrator	Quote Approval was started. Participants: LitwareInc Administrator		
1/26/2008 7:45 PM	Task Created	LitwareInc Administrator	Task created for LitwareInc Administrator. Due by: 1/27/2008 7:45:50 PM		
1/26/2008 7:48 PM	Comment	LitwareInc Administrator	Tasks for Quote Approval on Sample Quote were updated by LitwareInc Administrator. Due by: None Task instructions: Please approve this quote		
1/26/2008 7:49 PM	Comment	LitwareInc Administrator	Tasks for Quote Approval on Sample Quote were updated by LitwareInc Administrator. Due by: 1/26/2008 12:00:00 AM Task instructions: Please approve this quote		
1/26/2008 7:52 PM	Task Completed	LitwareInc Administrator	LitwareInc Administrator has requested a change to the task assigned to LitwareInc Administrator. Comments: Please approve this quote as soon as possible	Change Requested of LitwareInc Administrator by LitwareInc Administrator	
1/26/2008 7:52 PM	Task Created	LitwareInc Administrator	Task created for LitwareInc Administrator. Due by: 1/26/2008 12:00:00 AM		
1/26/2008 7:54 PM	Task Completed	LitwareInc Administrator	Task assigned to LitwareInc Administrator for a requested change was completed by LitwareInc Administrator. Comments: The changes requested are integrated into the quote.	Finished by LitwareInc Administrator	
1/26/2008 7:54 PM	Task Created	LitwareInc Administrator	Task created for LitwareInc Administrator. Due by: 1/27/2008 7:54:22 PM		
1/26/2008 7:54 PM	Task Completed	LitwareInc Administrator	Task assigned to LitwareInc Administrator was approved by LitwareInc Administrator. Comments: Please approve this quote as soon as possible. The customer is waiting. Quote Approval was completed.	Approved by LitwareInc Administrator	
1/26/2008 7:54 PM	Workflow Completed	LitwareInc Administrator	Quote Approval on Sample Quote has successfully completed. All participants have completed their tasks.		

Exercise 3: Executing Workflow using Office 2007

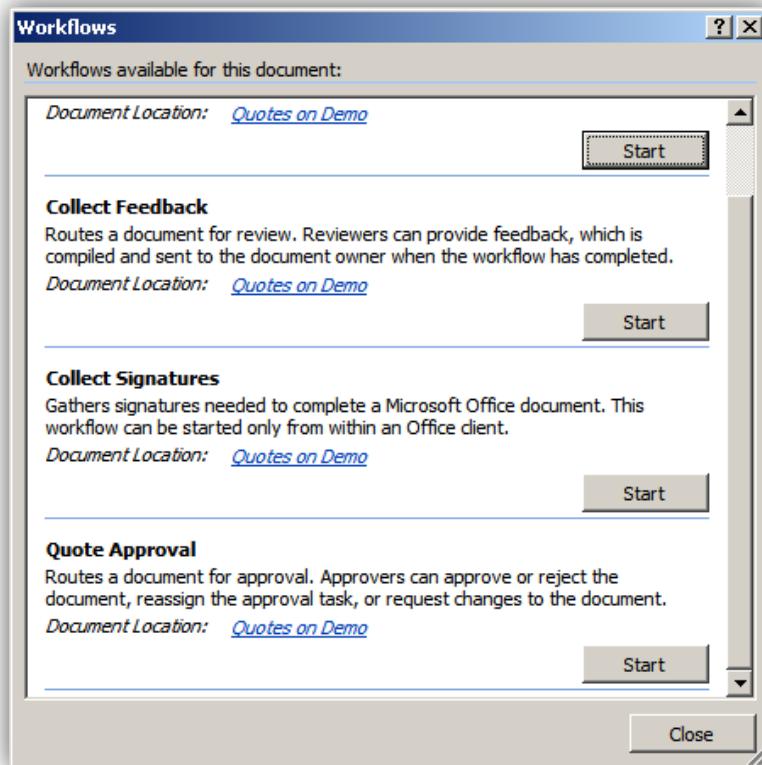
1) Open an existing quote document

- Navigate to the **Quotes** document library.
- Hover over the **Sample Quote** document name and click the **Edit in Microsoft Office Word** option in the drop down.



2) Start the **Quote Approval** in Word 2007

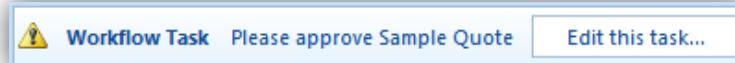
- Click the **Office Button** and select **Workflows** option from the menu.
- In the **Workflows** window, click the **Start** button under the **Quote Approval** workflow.



- Update the comments to something other than the default and click **Start** to start the workflow.

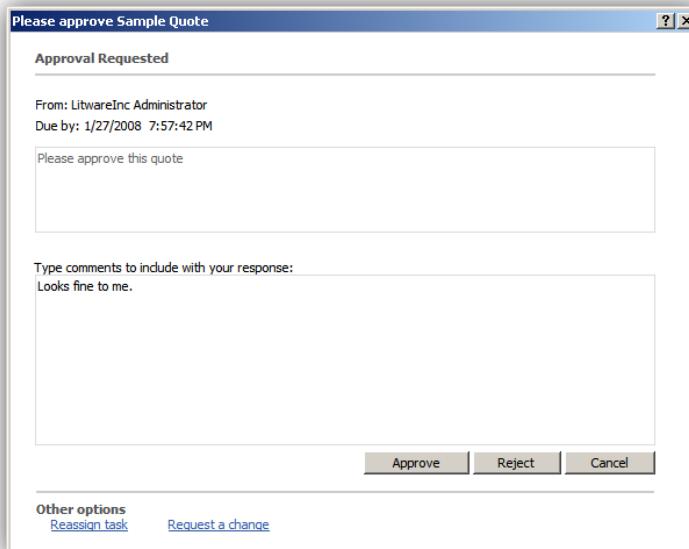


- Notice the **Workflow Task** bar at the top of the document. This notifies the editor of the document that they have a task that relates to this document.



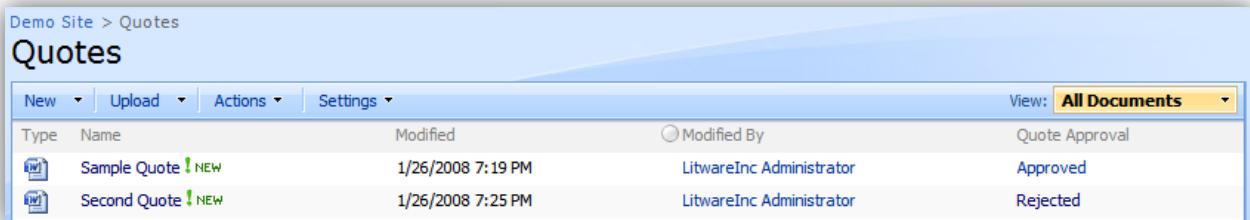
3) Approving the task in Word 2007

- Click the **Edit this task** button at the top of the document in Word 2007.
- In the **Task** dialog, enter some comments and click the **Approve** button to approve the document.



4) Verify the results in **Internet Explorer**.

- Navigate to the **Quotes** document library
- Verify that the **Sample Quote's Quote Approval** column contains **Approved**.



Type	Name	Modified	Modified By	Quote Approval
Word Document	Sample Quote <small>! NEW</small>	1/26/2008 7:19 PM	LitwareInc Administrator	Approved
Word Document	Second Quote <small>! NEW</small>	1/26/2008 7:25 PM	LitwareInc Administrator	Rejected

Challenge: Quote Content Type

Often specific document types are grouped by content type and not by document library. Therefore, it is more realistic to create a **Quotes** content type and attach the workflow to the content type.

Attach the same quote approval workflow to a new **Quote** content type and add the content type to any document library. Now as long as the document is a quote content type the workflow will be available no matter which document library it is in.

Do not forget to remove the workflow association from the **Quotes** document library or you may see that workflow association instead of the content type association.

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

- 1) 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

- 1) 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();
```

2) 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);
```

3) 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 **SPWorkflowAssociations 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;
```

4) 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.

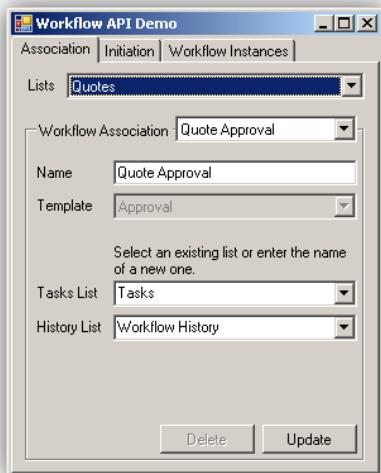
```
// select the first task or history list available
lstTaskList.SelectedItem = 0;
lstHistoryList.SelectedItem = 0;
```

- 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;
```

5) 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.



6) 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 **IstHistoryList.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 **SPWorkflowTemplate**, 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;
```

7) 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 **UpdateWorkflowAssocaiton** method.

```
// update the association
association.ParentList.UpdateWorkflowAssociation(association);
```

- 8) 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.

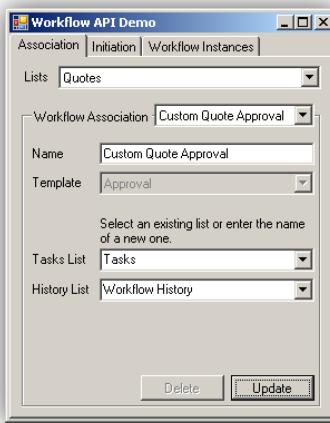
```
SPLIST parentList = association.ParentList;
```

- Remove the workflow association from the parent list using the **SPLIST.RemoveWorkflowAssociation** method. Do not forget to call the list's **Update** method.

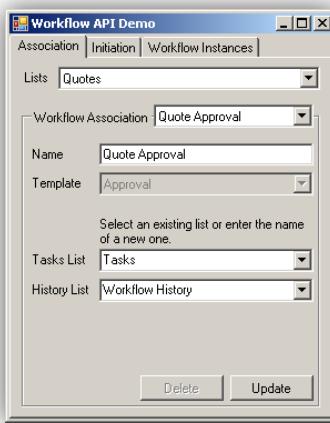
```
// remove the workflow association
parentList.RemoveWorkflowAssociation(association);
parentList.Update();
```

- 9) 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**.

Demo > Quotes > Settings > Workflow settings

Change Workflow Settings: Quotes

Use this page to view or change the workflow settings for this document library. You can also add or remove existing workflows. New workflows will not be applied to workflows already in progress.

Workflows	Workflow Name (click to change settings)	Workflows in Progress
	Custom Quote Approval	1
	Quote Approval	0

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 **Initiation** 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);
```

- 4) 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);
```

- 5) 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 =
    1stAssociations.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);
```

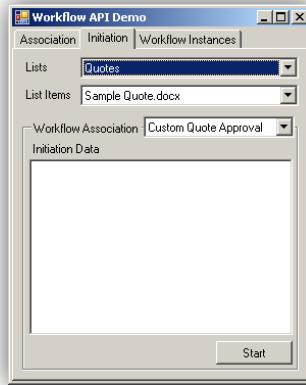
- Display a message box indicating the workflow is starting and clear the **initiation data** text box.

```
// update the UI to indicate the workflow has started
MessageBox.Show(this, "Workflow started");
txtInitiationData.Text = string.Empty;
```

- 6) 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.



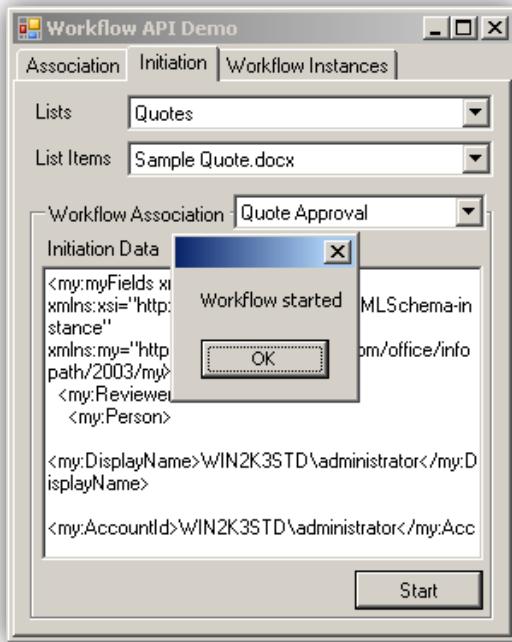
- In the **Workflow Association** drop down list, select **Quote Approval**.
- Enter the following xml into the initiation data page. This is the initiation data to start an **Approval** workflow.

```

<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.

Demo Site > Quotes					
Quotes					
Type	Name	Modified	Modified By	Custom Quote Approval	Quote Approval
	Sample Quote ! NEW	1/26/2008 7:19 PM	LitwareInc Administrator	Approved	In Progress
	Second Quote ! NEW	1/26/2008 7:25 PM	LitwareInc Administrator	Rejected	

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);
```

- 3) 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 **IstListItems** drop down list using the **PopulateComboBox** method and the **Items** collection of the current list.

```
PopulateComboBox(list.Items, lstListItems);
```

- 4) 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 **IstAssociations** drop down list using the **PopulateComboBox** method and the **Workflows** collection of the current list item.

```
PopulateComboBox(listItem.Workflows, lstAssociations);
```

- 5) 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 **IstModifications** 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 **IstTasks** drop down list using the **PopulateListBox** method.

```
PopulateListBox(tasks, lstTasks);
```

- 6) 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;
```

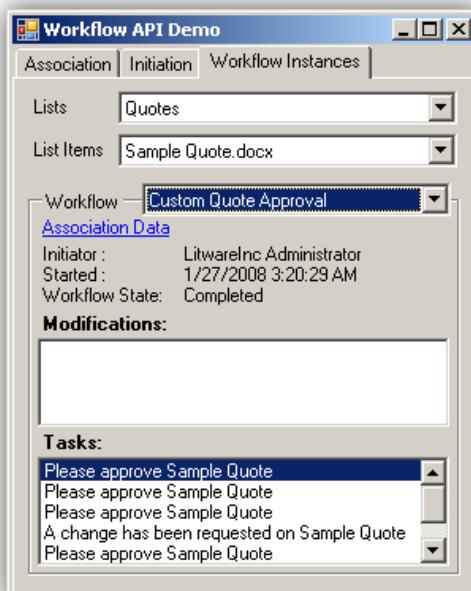
- 7) Implement the **LookupModificationContextData** method, which uses the **SPWorkflowModification** class's **ContextData** property to return the context data associated with the modification.
- Locate the **LookupModificationContextData** method and add the code to return the current **SPWorkflowModification**'s context data.

```
return modification.ContextData;
```

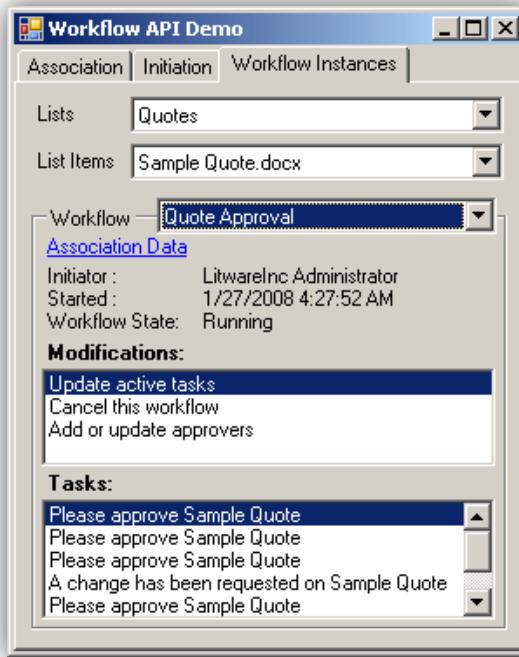
- 8) Implement the **LookupAssociationData** method to return the association data stored in the workflow's association to the list. As the workflow attached itself to the list, this data is set.
- 9) Locate the **LookupAssociationData** method and add the code to return the current **SPWorkflow**'s association data by using its parent **SPWorkflowAssociation** object.

```
return workflow.ParentAssociation.AssociationData;
```

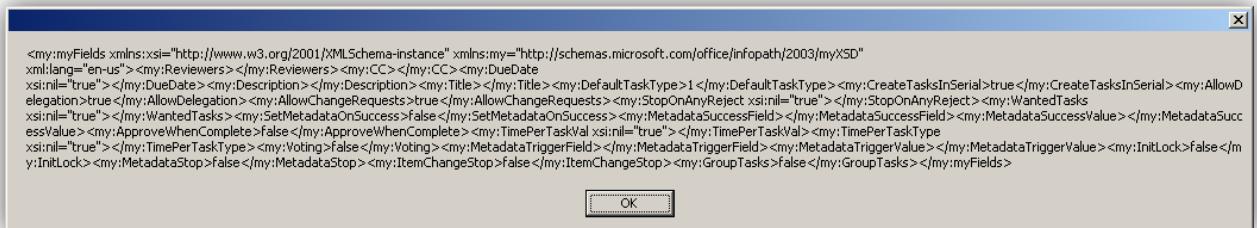
- 10) Test the application by looking at the workflow instance started in the previous exercise.
- Start the application using the debugger by selecting **Debug -> Start Debugging** or pressing **F5**.
 - Switch to the **Workflow Instances** tab, select the Quote list in the Lists dropdown, and select **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.



Lab 6: Developing SharePoint Workflow Templates with Visual Studio 2008

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.

To build this workflow, you will need to deal with many different aspects of SharePoint workflow, so you will start simple. In the first part or implementing the **DocumentArchive** project, you will focus on creating the workflow that will simply log a message to the Workflow History list.

Exercise 0: Setup

- 1) If you did not complete lab 4, you will need to create the **Demo** site collection.

Open SharePoint and browse to the Demo site collection

The url is <http://litwareinc.com/sites/Demo>.

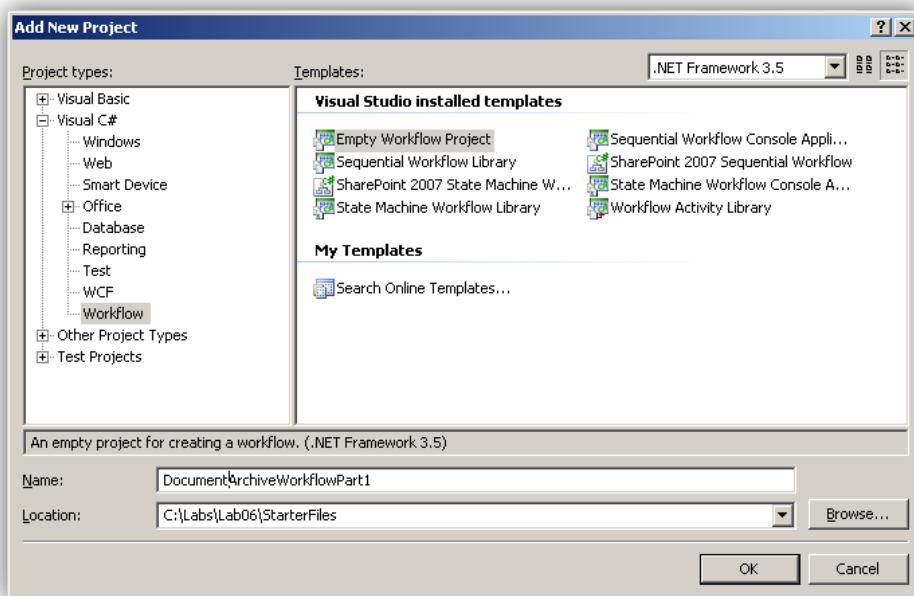
If the site collection does not exist, create it using the **CreateDemo.bat** file in the **C:\Labs\Files** folder.

- 2) Open the starter VS 2008 solution at **\Labs\Lab06\StarterFiles\DocumentArchive Part 1.sln**.

Exercise 1: Build Hello World Workflow

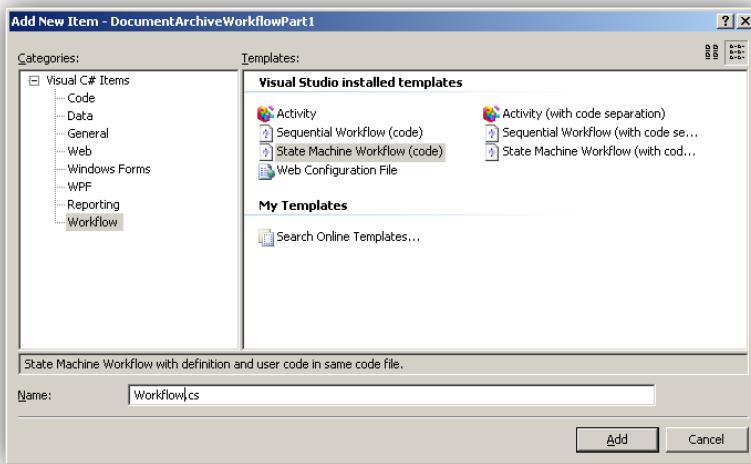
- 1) Add a new **Empty Workflow Project** named **DocumentArchiveWorkflowPart1**.

- Right click on the solution in the **Solution Explorer** and select **Add -> New Project**.
- In the **Add New Project** dialog, select the project type of **Workflow** on the left and a template of **Empty Workflow Project** on the right.



- Name the project **DocumentArchiveWorkflowPart1** and click **OK**.

- 2) Add a new state machine workflow named **Workflow** to the project.
- Right click on the **DocumentArchiveWorkflowPart1** project in the solution explorer and select **Add -> New Item...**
 - In the **Add New Item** dialog, select the **Workflow** category on the left and a template of **State Machine Workflow (code)** on the right.
 - Name the new item **Workflow.cs** and click **Add**.



- 3) Add a new state to the workflow named **WorkflowCompletedState**.

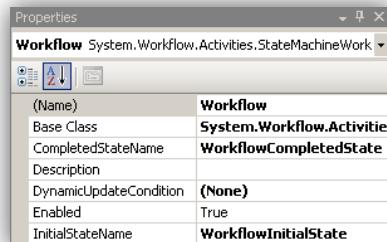
- Right click on the **State Machine** designer canvas and click **Add State**.
- Set the name of the new state to **WorkflowCompletedState**.

*In the properties pane, set the **Name** property to **WorkflowCompletedState**.*

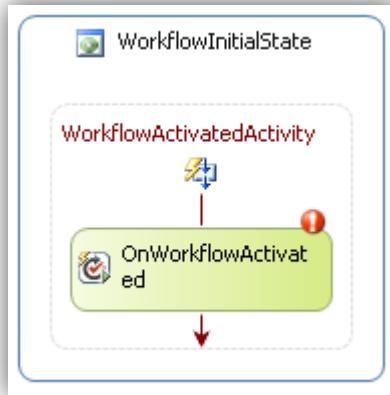


- 4) Set the initial and completed states of the workflow

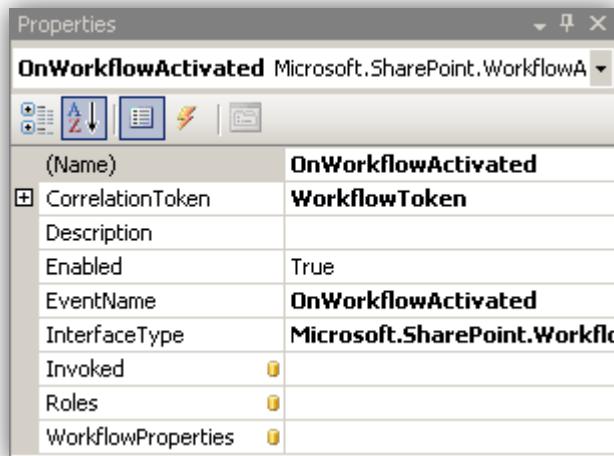
- Right click on the **State Machine** designer canvas and click **Properties**.
- Verify the **InitializedName** property in the Properties pane is set to **WorkflowInitialState**.
- Set the **CompletedStateName** property to **WorkflowCompletedState**.



- 5) Add an **OnWorkflowActivated** event driven activity to signal the start of the workflow.
- Right click on the **WorkflowInitialState** state and select **Add EventDriven**.
 - Rename the new event driven activity **WorkflowActivatedActivity**.
*Right click the new event driven activity and click **Properties**.
In the properties pane, set the **Name** property to **WorkflowActivatedActivity**.*
 - Drag an **OnWorkflowActivated** activity from the toolbox into the **WorkflowActivatedActivity** activity and name it **OnWorkflowActivated**.
*In the properties pane, set the **Name** property to **OnWorkflowActivated**.*

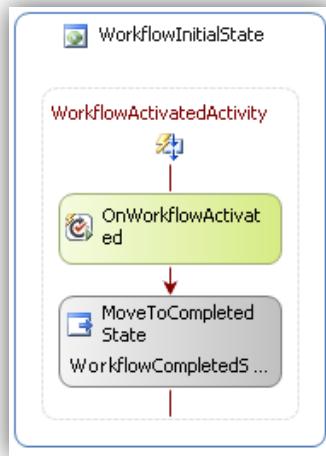


- Create a new correlation token for the **OnWorkflowActivated** activity. Scope it to the workflow.
*In the properties pane, set the **CorrelationToken** property to **WorkflowToken**.
Expand the **CorrelationToken** property in the property pane.
Set the **OwnerActivityName** property to **Workflow**.*



- 6) Transition to the **WorkflowCompletedState** state after the **OnWorkflowActivated** activity executes.
- Drag a **SetState** activity from the toolbox into the **WorkflowActivatedActivity** immediately following the **OnWorkflowActivated** activity.
*Make sure you use the **SetState** activity in the **Windows Workflow v3.0** group in the toolbox.*
 - Change the new activity's name to **MoveToCompletedState** and set the target state to **WorkflowCompletedState**.

Set the **TargetStateName** property to **WorkflowCompletedState**.

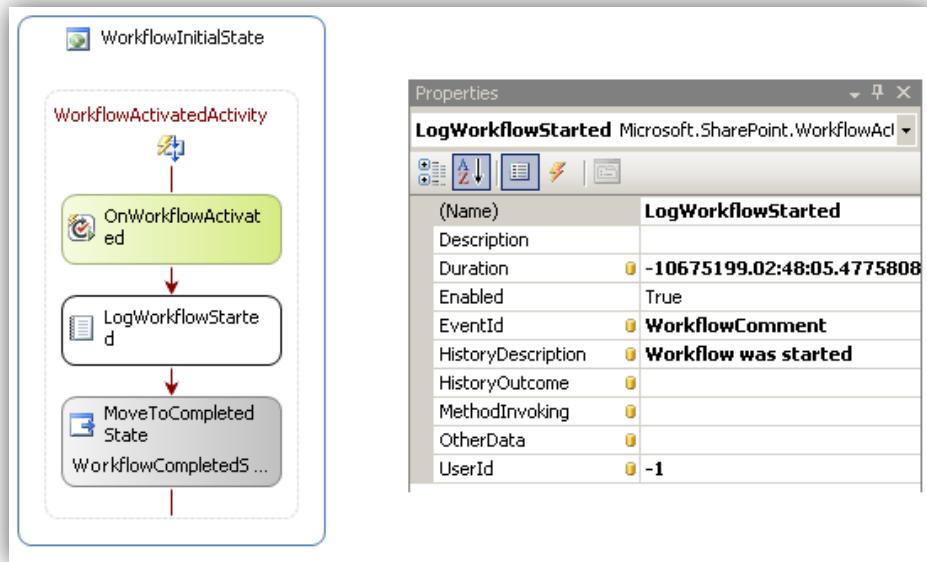


7) Log a started message to the workflow's history to indicate that the workflow ran.

- Drag a **LogToListActivity** from the toolbox into the **WorkflowActivatedActivity** between the **OnWorkflowActivated** and **MoveToCompletedState** activities.
- Change the new activity's name to **LogWorkflowStarted** and set its **HistoryDescription** to '**Workflow was started**'.

*In the properties pane, set the **Name** property to **LogWorkflowStarted**.*

*Set the **HistoryDescription** property to **Workflow was started**.*

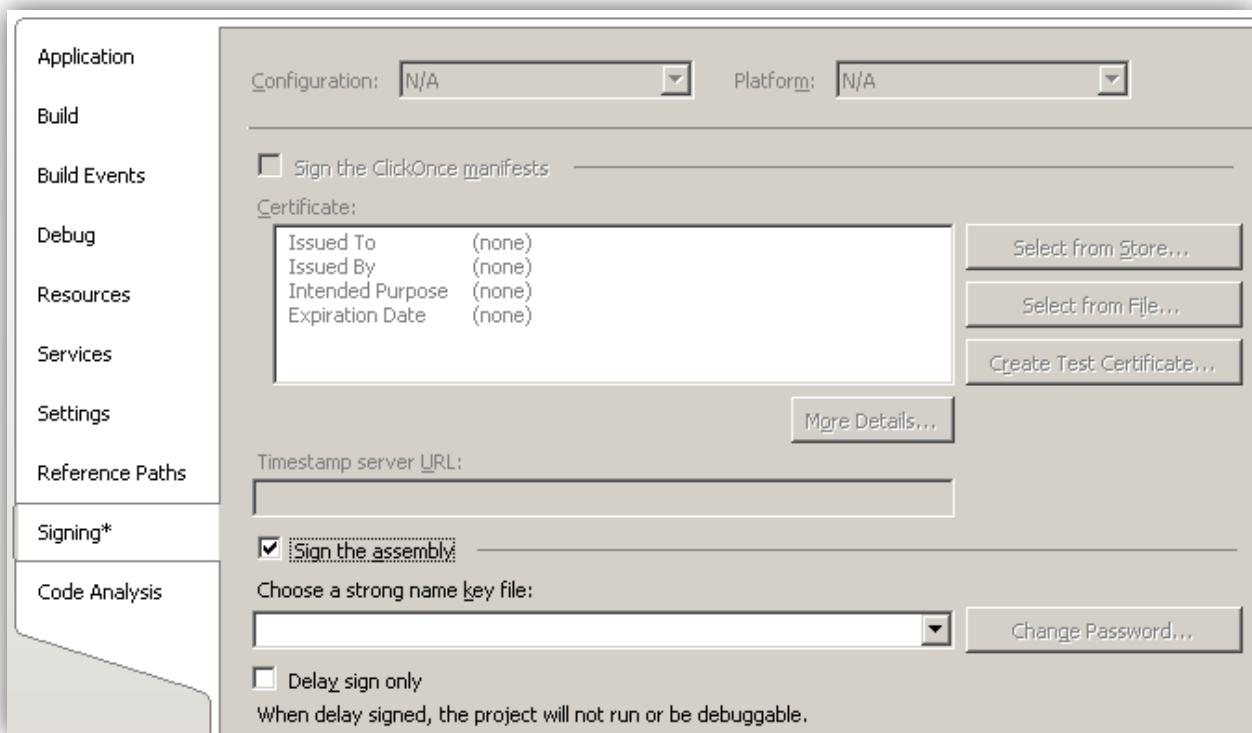


8) Build the project.

- Right click the project in the **Solution Explorer** and click **Build**.

Exercise 2: Install the workflow as a Feature

- 1) Sign the assembly to allow installation in the GAC.
 - Right click the project in the solution explorer and click **Properties**.
 - Click the **Signing** tab on the left hand side of the project properties window.
 - Check the **Sign the assembly** checkbox.



- Choose <New ...> new from the drop down list box.
- In the **Create Strong Name Key** dialog box, enter a **Key file name of Key**.
- Uncheck the **Protect my key file with a password** and click **OK**.



- Close the project properties window.

2) Create the folder that will contain the feature files.

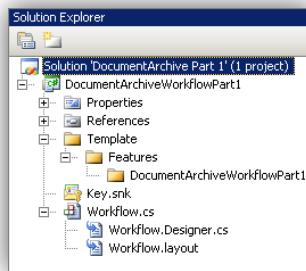
- Create a folder under the project named **Template**.

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

*Right click the folder and click **Rename** to rename the folder to **Template**.*

- Create another folder named under **Template** named **Features**.

- Create a third folder under **Features** named **DocumentArchiveWorkflowPart1**.



3) Create the **feature.xml** in the **DocumentArchiveWorkflowPart1** folder.

- Create a new file named **Feature.xml** in the **DocumentArchiveWorkflowPart1** folder.

*Right click the **DocumentArchiveWorkflowPart1** folder and click **Add -> New Item....***

*Select the **Data** category on the left and select a template of **XML file**.*

*Name the new file **Feature.xml** and click **Add**.*

- Insert the **Windows SharePoint Service Workflow Feature.xml** snippet.

*Right click in the editor and select **Insert Snippet....***

*In the popup window, select the **Windows SharePoint Service Workflow** option then select **Feature.xml Code**.*

- Replace **GUID** with a new Guid generated using **GuidGen**.

*Click **Tools -> Create Guid** then copy and paste the registry formatted guid into the snippet's GUID placeholder.*

- Replace **Title** with **Document Archive – Part 1**.

- Replace **Description** with “Contains workflows and custom forms designed to aid in document archiving”.

```
<Feature Id="29ac22d1-7d8c-4513-9566-11109ab25983"
    Title="Document Archive - Part 1"
    Description="Request approval to archive this document in another
document library."
    Version="12.0.0.0"
    Scope="Site"
    xmlns="http://schemas.microsoft.com/sharepoint/">
<ElementManifests>
    <ElementManifest Location="workflow.xml" />
</ElementManifests>
<Properties>
    <Property Key="GloballyAvailable" value="true" />
</Properties>
</Feature>
```

- Save and close the file.

- 4) Create the **workflow.xml** file needed to define the details of the workflow.
- Create a new file named Workflow.xml in the **DocumentArchiveWorkflowPart1** folder.
 - Insert the **Windows SharePoint Service Workflow Workflow.xml** snippet.

*Right click in the editor and select **Insert Snippet...***

*In the popup window, select the **Windows SharePoint Service Workflow** option.*

*In the second popup window, select **Workflow.xml Code**.*
 - Replace **GUID** with a new Guid generated using GuidGen.

*Click **Tools -> Create Guid***

Copy and paste the registry formatted guid into the snippet's GUID placeholder.
 - Replace **Title** with **Document Archive – Part 1**.
 - Replace **Description** with “**Requests approval to archive this document in another document library**”.
 - Replace **CodeBesideClass** with **DocumentArchiveWorkflowPart1.Workflow**.
 - Replace the **PublicKeyToken** of the **CodeBesideAssembly** with the public key token of the assembly.

*Open a new instance of the **Visual Studio 2008 Command Prompt** located in **Start -> All Programs -> Microsoft Visual Studio 2008 -> Visual Studio Tools**.*

*Navigate to **C:\Labs\Lab06\StarterFiles\DocumentArchiveWorkflowPart1\bin\Debug***

*Run **sn.exe -T DocumentArchiveWorkflowPart1.dll** to display the public key token.*
 - Remove the **TaskListContentTypeId**, **AssociationUrl**, **InstantiationUrl**, **ModificationUrl** attributes.

These will be used later.
 - Remove the content of the **MetaData** element.

These will also be used later.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow
    Name="Document Archive - Part 1"
    Description="Requests approval to archive this document in another
document library."
    Id="78c8fd78-0b7f-4893-9361-ff005c32b12c"
    CodeBesideClass="DocumentArchiveWorkflowPart1.Workflow"
    CodeBesideAssembly="DocumentArchiveWorkflowPart1, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=15812f954569663f"
    StatusUrl="_layouts/wrkStat.aspx">
    <Categories/>
    <MetaData>
    </MetaData>
  </Workflow>
</Elements>
```

- 5) Add post build steps to aid in developing and deploying the workflow.
- Right click the project in the **Solution Explorer** and click **Properties**.
 - Click the **Build Events** tab in the project properties window.
 - Enter the following commands into the **Post-build event command line** window.

*Copy the contents of the **Template** folder to SharePoint's **Template** folder.*

*Install the workflow assembly into the GAC using **gacutil.exe**.*

Reset the app domain that may have loaded the workflow assembly.

*Forcibly install the SharePoint feature using **stsadm.exe**.*

```
xcopy "$(ProjectDir)\TEMPLATE" "C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE" /E /Y
"$(DevEnvDir)..\..\SDK\v2.0\bin\gacutil.exe" /i "$(TargetPath)" /f
%windir%\system32\cscript.exe c:\windows\system32\isapp.vbs /a
"SharePointDefaultAppPool" /r
"C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\bin\stsadm.exe" -o installfeature -name
DocumentArchiveWorkflowPart1 -force
```

- Save and close the project properties window.

- 6) Rebuild and deploy the new workflow feature.

- Right click the project in the solution explorer and click **Rebuild**.

*If no code changes are made, **Visual Studio 2008** does not re-execute the post build events. If you need to force a redeploy do a rebuild.*

- Look at the **Output** window to determine the post build steps executed successfully.

*If the **Output** window is not visible, display it using **View -> Output**.*

- 7) 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 1** feature.

Name	Status
Collect Signatures Workflow	Active
Disposition Approval Workflow	Active
Document Archive - Part 1	Active

8) Create an association between the **Shared Documents** document library and the new **Document Archive – Part 1** 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 1** workflow template and a name of **Document Archive - Part 1**.

Use the default values for both list and startup options.

Demo > Shared Documents > Settings > Workflow settings > Add or Change a Workflow

Add a Workflow: Shared Documents

Use this page to set up a workflow for this document library.

Workflow

Select a workflow to add to this document library. If the workflow template you want does not appear, contact your administrator to get it added to your site collection or workspace.

Select a workflow template:

- Disposition Approval
- Document Archive - Part 1
- DocumentArchiveWorkflow
- Hello World Workflow

Description:

Requests approval to archive this document in another document library.

Name

Type a name for this workflow. The name will be used to identify this workflow to users of this document library.

Type a unique name for this workflow:

Document Archive - Part 1

9) Add a new document to the **Shared Documents** folder.

- Navigate to the **Shared Documents** document library in the **Demo** site.
- Click **New -> New document** to create a new document.
- Click **OK** to allow Word 2007 to open the file.
- Enter some data and save the document as **Document To Archive.docx**.

Demo Site > Shared Documents

Shared Documents

Share a document with the team by adding it to this document library.

New | Upload | Actions | Settings

View: All Documents

Type



Document to Archive ! NEW

Modified

1/26/2008 9:10 PM

Modified By

Litware Admin Guy

10) Run the **Document Archive – Part 1** workflow on the newly created document.

- Hover over the new document and select **Workflows** from the drop down menu.
- In the workflows page, click the **Document Archive – Part 1** to start the workflow.
- In the **Shared Documents** document library, verify the workflow has completed.
- Click the **Completed** link to view the workflow status and verify the started message was logged to the workflow's history.

Demo Site > Shared Documents > Workflow Status

Workflow Status: Document Archive - Part 1

Workflow Information

Initiator: Litware Admin Guy	Document: Document to Archive
Started: 1/26/2008 9:14 PM	Status: Completed
Last run: 1/26/2008 9:14 PM	

Tasks

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

Assigned To	Title	Due Date	Status	Outcome
There are no items to show in this view of the "Tasks" list. To create a new item, click "New" above.				

Workflow History

[View workflow reports](#)

The following events have occurred in this workflow.

Date Occurred	Event Type	User ID	Description	Outcome
1/26/2008 9:14 PM	Comment	System Account	Workflow has started.	

Challenge: Deploy the workflow as a Solution Package

- 1) Create the folder to store the **Solution Package** definition files.
 - Create a folder under the project named **Solution**.

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

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

*Rename the folder **Solution**.*
- 2) Create the **Manifest.xml** in the **Solution** folder.
 - Right click the **Solution** folder and click **Add -> New Item....**
 - Select the **Data** category on the left and select a template of **XML file**.
 - Name the new file **Manifest.xml** and click **Add**.

- 3) Paste the following XML into your manifest file.
- The **FeatureManifest** element defines the features that are part of this solution.
 - The **Assembly** elements define the target location for any assemblies that are part of the solution. The GAC is the target location in this case.

```
<?xml version="1.0" encoding="utf-8" ?>
<solution SolutionId="78c8fd78-0b7f-4893-9361-ff005c32b12c"
xmlns="http://schemas.microsoft.com/sharepoint/">
  <FeatureManifests>
    <FeatureManifest Location="DocumentArchiveworkflowPart1\Feature.xml" />
  </FeatureManifests>
  <Assemblies>
    <Assembly Location="DocumentArchiveworkflowPart1.dll"
DeploymentTarget="GlobalAssemblyCache"/>
  </Assemblies>
</solution>
```

- 4) Create the **Package.ddf** in the **Solution** folder. Its purpose is to define the generated package.
- Right click the **Solution** folder and click **Add -> New Item....**
 - Select the **General** category on the left and select a template of **Text file**.
 - Name the new file **Package.ddf** and click **Add**.
- 5) Paste the following text into your DDF file to define the files to be packaged.

```
.OPTION EXPLICIT      ; Generate errors
.Set CabinetNameTemplate=DocumentArchiveworkflowPart1.wsp
.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory
.Set CompressionType=MSZIP;** All files are compressed in cabinet files
.Set UniqueFiles="ON"
.Set Cabinet=on
.Set DiskDirectory1=Package

..\..\Solution\Manifest.xml manifest.xml

..\..\Template\Features\DocumentArchiveworkflowPart1\Feature.xml
DocumentArchiveworkflowPart1\Feature.xml
..\..\Template\Features\DocumentArchiveworkflowPart1\Workflow.xml
DocumentArchiveworkflowPart1\Workflow.xml

DocumentArchiveworkflowPart1.dll DocumentArchiveworkflowPart1.dll
```

- 6) Add another step to the post build process that creates the solution package.
- Right click the project in the **Solution Explorer** and click **Properties**.
 - Click the **Build Events** tab in the project properties window.
 - Enter the following commands into the **Post-build event command line** window at the top of the list.

```
makecab /f "$(ProjectDir)\Solution\Package.ddf"
```

- 7) Rebuild the project.
 - Right click on the project in the **Solution Explorer** and click **Rebuild**.
 - Verify the rebuild was successful.
- 8) 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 **DocumentArchiveWorkflowPart1** feature.

```
stsadm -o uninstallfeature -name DocumentArchiveWorkflowPart1 -force
```
 - Delete the folder at **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Features\DocumentarchiveWorkflowPart1**.
- 9) 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\DocumentArchiveworkflowPart1\bin\Debug\Package\DocumentArchiveworkflowPart1.wsp"
```
 - Deploy the solution to the farm using the **deploysolution** command in **stsadm**.

```
stsadm -o deploysolution -name DocumentArchiveworkflowPart1.wsp -allowgacdeployment -local
```
- 10) 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 1** feature is active.

Challenge: Using the workflow activation properties.

The **OnWorkflowActivated** activity provides a **WorkflowProperties** property. This property contains information about the **SPListItem** that the workflow is attached to. Use the **DisplayName** property in this object to populate the **HistoryOutcome** of the **LogToHistoryList** activity.

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

- 1) 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

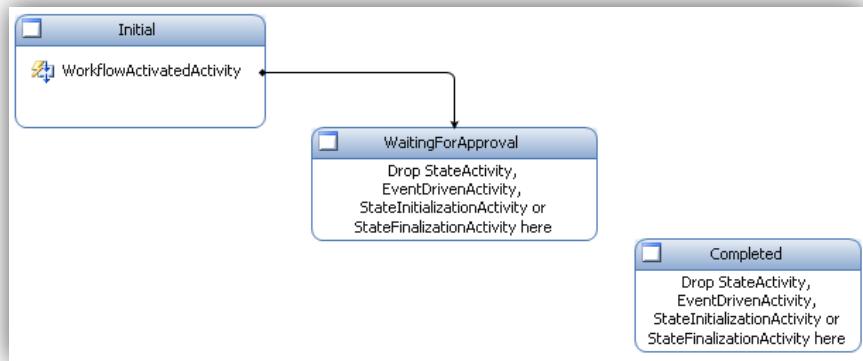
The url is <http://litwareinc.com/sites/Demo>.

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

- 2) Open the starter **VS 2008** solution at `\Labs\Lab07\StarterFiles\DocumentArchive Part 2.sln`.

Exercise 1: Creating SharePoint Tasks in a Workflow

- 1) 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 **MovingToWaitingForApproval**.*
 - 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.



- 2) Add a new **StateInitializationActivity** to the **WaitingForApproval** state and name it **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**.*



- 3) Create a new task in the **WaitingForApprovalInit** activity.

- Drag a new **CreateTask** activity from the Toolbox into the **WaitingForApprovalInit** activity and name it **CreateApprovalTask**

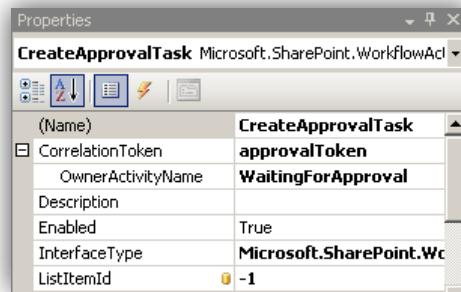
*In the properties pane, set the **Name** property to **CreateApprovalTask**.*

- In the **CreateApprovalTask**'s properties pane, create a new correlation token scoped to the **WaitingForApproval** 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.

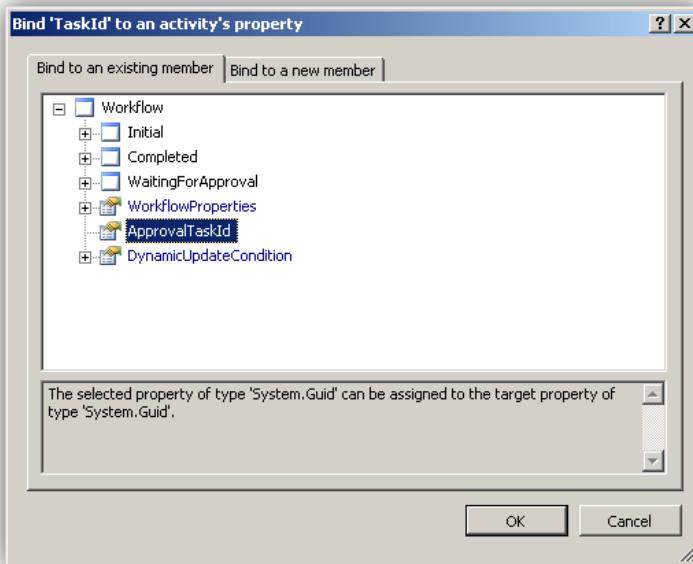
*View the workflow designer by clicking **Workflow.cs** in the **Solution Explorer** and selecting **View Designer**.*

*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;
```

- 4) 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**.*



- 5) 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; }
```

- 6) 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.

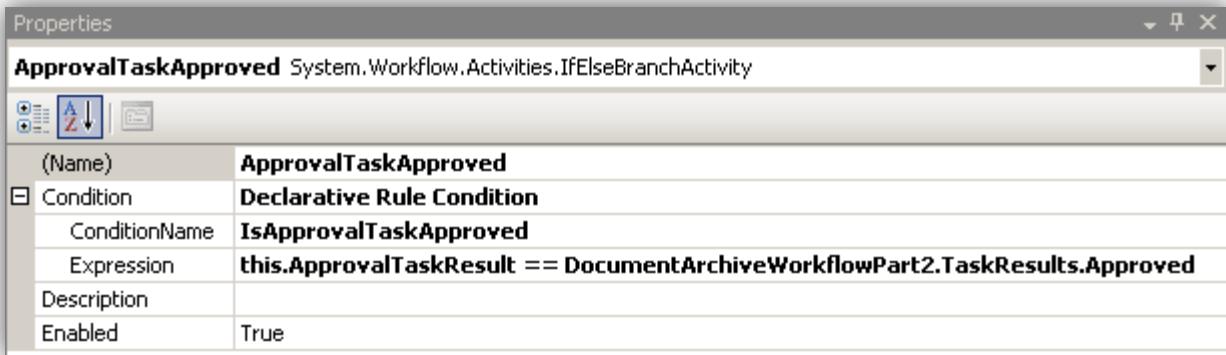
*Right click the **OnApprovalTaskChanged** activity and click **Properties**.
In the properties pane, set the **CorrelationToken** property to **ApprovalToken**.*
 - In the **OnApprovalTaskChanged** properties pane, bind **TaskId** to **ApprovalTaskId**.

*Open the **Binding** dialog by selecting the **TaskId** property and clicking the ... button.
In the **Binding** dialog select the **ApprovalTaskId** property and click **OK**.*
- 7) 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;
```
- 8) 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 **ApprovalTaskInProgress**.*
 - 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**.*
- 9) 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**.*



- 10) 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**.*

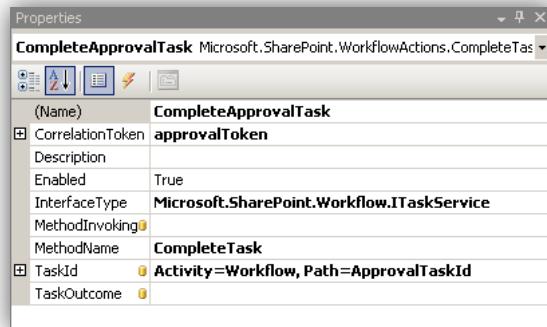
- Mark the task as complete using the **CompleteTask** activity if the task was determined to be complete.

*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.*



- 11) 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.

- 12) 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.

Name	Status
 Collect Signatures Workflow Gathers signatures needed to complete a Microsoft Office document.	<button>Deactivate</button> Active
 Disposition Approval Workflow Manages document expiration and retention by allowing participants to decide whether to retain or delete expired documents.	<button>Deactivate</button> Active
 Document Archive - Part 1 Request approval to archive this document in another document library.	<button>Deactivate</button> Active
 Document Archive - Part 2 Request approval to archive this document in another document library.	<button>Deactivate</button> Active

13) 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.

Demo > Shared Documents > Settings > Workflow settings > Add or Change a Workflow

Add a Workflow: Shared Documents

Use this page to set up a workflow for this document library.

Workflow Select a workflow to add to this document library. If the workflow template you want does not appear, contact your administrator to get it added to your site collection or workspace.	Select a workflow template: <input type="button" value="Disposition Approval"/> <input type="button" value="Document Archive - Part 1"/> <input checked="" type="button" value="Document Archive - Part 2"/> <input type="button" value="DocumentArchiveWorkflow"/>	Description: Requests approval to archive this document in another document library.
Name Type a name for this workflow. The name will be used to identify this workflow to users of this document library.	Type a unique name for this workflow: <input type="text" value="Document Archive - Part 2"/>	

14) 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**.*

Demo > Tasks > Archival Approval > Edit Item

Tasks: Archival Approval

The content of this item will be sent as an e-mail message to the person or group assigned to the item.

OK Cancel

Attach File Delete Item Spelling... * indicates a required field

Content Type	Workflow Task
A work item created by a workflow that you or your team needs to complete.	
Title *	Archival Approval
Priority	(2) Normal
Status	Completed
% Complete	100 %
Assigned To	WIN2K3STD\administrator

- 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.

Demo > Shared Documents > Workflow Status

Workflow Status: Document Archive - Part 2

Workflow Information

Initiator: WIN2K3STD\administrator	Document: Document To Archive
Started: 1/18/2008 12:17 PM	Status: Completed
Last run: 1/18/2008 12:19 PM	

Tasks

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

Assigned To	Title	Due Date	Status	Outcome
WIN2K3STD\administrator	Archival Approval ! NEW		Completed	

Exercise 2: Creating custom Task Forms

- 1) Add a new base class named **DocArchivePart2TaskForm** that will be used 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;
```

- 2) 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 **InkItem** of type **Hyperlink** to the class.
- Add a protected field named **IstArchiveList** 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; }  
}
```

- 3) 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 **SPWorkflow** 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);
```

4) 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);
}

```

- 5) 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);
}

```

- 6) 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 **IstArchiveList.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", IstArchiveList.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);
}

```

- 7) 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>
```

- 8) 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-propertiesheet">
        </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

- 1) 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="DocumentArchiveWorkflowPart2 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>
```

- 2) 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>
```

- 3) 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.

- 4) 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.*

Tasks : Archival Approval

Item Requiring Approval Please review this item/document.	Click on item hyperlink below to review item/document List: Shared Documents Item: Document To Archive
Archive Document Library Specify the document library used to store archived documents.	Archive Document Library: <input style="width: 150px; height: 25px;" type="button" value="Archive Documents"/>
<input style="width: 100px; height: 30px; margin-right: 10px;" type="button" value="Approve"/> <input style="width: 100px; height: 30px; margin-right: 10px;" type="button" value="Reject"/> <input style="width: 100px; height: 30px;" type="button" value="Cancel"/>	

- 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.

Demo > Shared Documents > Workflow Status

Workflow Status: Document Archive - Part 2

Workflow Information				
Initiator: WIN2K3STD\administrator Started: 1/18/2008 3:12 PM Last run: 1/18/2008 3:12 PM	Document: Document To Archive Status: Completed			
Tasks				
The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list Tasks .				
<input type="radio"/> Assigned To	Title	Due Date	Status	Outcome
WIN2K3STD\administrator	Archival Approval NEW			Completed
Workflow History				
<input type="checkbox"/> View workflow reports The following events have occurred in this workflow.				
Date Occurred	Event Type	<input type="radio"/> User ID	Description	Outcome
1/18/2008 3:12 PM	Comment	System Account	Workflow was started	Document To Archive
1/18/2008 3:12 PM	Comment	System Account	Approval task rejected.	

Challenge: Integrating Task Forms into Solution Packages

- 1) 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>
```

- 2) 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
```

- 3) 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 **DocumentArchiveWorkflowPart2** 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.

- 4) 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
```

- 5) 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.

Lab 08: Creating custom Workflow Association Forms

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

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

Exercise 0: Setup

- 1) 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.*

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

Exercise 1: Creating an Association Form base class

- 1) Create the **AssociationForm** base class in the UI folder.

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

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

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

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

```
public abstract class AssociationForm : LayoutsPageBase
{
}
```

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

- Create a **protected virtual** method named **PopulateControls** to populate the controls not directly tied to the association data.
- Create a **protected abstract** method named **BindAssociationData** that takes a single string parameter. This method populates the UI with the association data from a previously created association.
- Create a **protected abstract** method named **UpdateAssociationData** that returns a string. This method extracts the association data from the form on the submit event.

```
protected virtual void PopulateControls() { }
protected abstract void BindAssociationData(string value);
protected abstract string UpdateAssociationData();
```

- 3) Since a workflow can be associated to a list, content type, or a list's content type, create an enumeration to track the type of association being managed.
- Create a private **AssociationType** enumeration inside the **AssociationForm** class with three values: **List**, **ContentType**, or **ListContentType**.

```
private enum AssociationType
{
    List,
    ContentType,
    ListContentType
}
```

- Add a private field to the class named **_associtionType** of type **AssociationType**.

```
private AssociationType _associationType;
```

- 4) When the page is loaded, determine the association type of the association to manage. Using the **Url** and **Form** parameters, determine if the workflow is related to a list, content type, or both and whether it's already been created.
- In the **AssociationForm** class, override the **OnLoad** method.

```
protected override void OnLoad(EventArgs e)
{
}
```

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

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

```
// read the form level parameters
string listId = Request.Params["List"];
string ctypeId = Request.Params["ctype"];
string guidAssocId = Request.Params["GuidAssoc"];
```

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

```
// determine the type of association
if (!string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))
    _associationType = AssociationType.ListContentType;
else if (!string.IsNullOrEmpty(listId) && string.IsNullOrEmpty(ctypeId))
    _associationType = AssociationType.List;
else if (string.IsNullOrEmpty(listId) && !string.IsNullOrEmpty(ctypeId))
    _associationType = AssociationType.ContentType;
```

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

```
// validate the workflow association id
Guid? workflowAssociationId = new Guid?();
if (!string.IsNullOrEmpty(guidAssocId))
    workflowAssociationId = new Guid(guidAssocId);
```

- 5) Based on the association type and the existence of the workflow association, lookup the **SPList**, **SPContentType**, and **SPWorkflowAssociation** objects.
- Add **private** fields to the class to store the content type, list and workflow association SharePoint APIs.

```

private SPContentType _contentType;
private SPList _list;
private SPWorkflowAssociation _workflowAssociation;

```

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

```

// lookup the list, content type, and existing workflow associations
switch (_associationType)
{
    case AssociationType.ContentType:
        break;
    case AssociationType.List:
        break;
    case AssociationType.ListContentType:
        break;
}

```

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

```

case AssociationType.ContentType:
    _contentType = Web.AvailableContentTypes[new SPContentTypeId(ctypeId)];
    if (workflowAssociationId.HasValue)
        _workflowAssociation =
            _contentType.WorkflowAssociations[workflowAssociationId.Value];

```

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

```

case AssociationType.List:
    _list = Web.Lists[new Guid(listId)];
    if (workflowAssociationId.HasValue)
        _workflowAssociation =
            _list.WorkflowAssociations[workflowAssociationId.Value];

```

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

```

case AssociationType.ListContentType:
    _list = Web.Lists[new Guid(listId)];
    _contentType = _list.ContentTypes[new SPContentTypeId(ctypeId)];
    if (workflowAssociationId.HasValue)
        _workflowAssociation =
            _contentType.WorkflowAssociations[workflowAssociationId.Value];

```

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

```

// call the base implementation
base.OnLoad(e);

```

- Since many of the page's parameters are in the posted form, write them back into the page so they will be available in subsequent postbacks.
 - In the **AssociationForm** class, override the **OnPreRender** method.

```
protected override void OnPreRender(EventArgs e)
{ }
```

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

```
// register parameters in the hidden field
ClientScript.RegisterHiddenField("WorkflowName",
    Request.Params["WorkflowName"]);
ClientScript.RegisterHiddenField("WorkflowDefinition",
    Request.Params["WorkflowDefinition"]);
ClientScript.RegisterHiddenField("AddToStatusMenu",
    Request.Params["AddToStatusMenu"]);
ClientScript.RegisterHiddenField("AllowManual", Request.Params["AllowManual"]);
ClientScript.RegisterHiddenField("RoleSelect", Request.Params["RoleSelect"]);
ClientScript.RegisterHiddenField("GuidAssoc", Request.Params["GuidAssoc"]);
ClientScript.RegisterHiddenField("SetDefault", Request.Params["SetDefault"]);
ClientScript.RegisterHiddenField("HistoryList", Request.Params["HistoryList"]);
ClientScript.RegisterHiddenField("TaskList", Request.Params["TaskList"]);
ClientScript.RegisterHiddenField("UpdateLists", Request.Params["UpdateLists"]);
ClientScript.RegisterHiddenField("AutoStartCreate",
    Request.Params["AutoStartCreate"]);
ClientScript.RegisterHiddenField("AutoStartChange",
    Request.Params["AutoStartChange"]);
```

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

- Call the virtual **PopulateControls** method.

```
// populate controls unrelated to the association data
this.PopulateControls();
```

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

```
// bind the association data (if any)
if (_workflowAssociation != null)
    this.BindAssociationData(_workflowAssociation.AssociationData);
```

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

```
// call the base implementation
base.OnPreRender(e);
```

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

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

```
private static string BuildRedirectString(AssociationType associationType,
    string paramList, string paramCtype)
{ }
```

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

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

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

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

```
switch (associationType)
{
    case AssociationType.ContentType:
        return "WrkSetng.aspx?ctype=" + paramCtype;
    case AssociationType.List:
        return "WrkSetng.aspx?List=" + paramList;
    case AssociationType.ListContentType:
        return "WrkSetng.aspx?List=" + paramList + "&ctype=" + paramCtype;
    default:
        throw new InvalidOperationException();
}
```

- Create a **protected** event handler named **Cancel_Click**.

```
protected void Cancel_Click(object sender, EventArgs e)
{
}
```

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

```
// redirect back to the workflow settings page
SPUtility.Redirect(
    BuildRedirectString(_associationType, Request.Params["List"],
    Request.Params["ctype"]),
    SPRedirectFlags.RelativeToLayoutsPage, this.Context);
```

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

- Create a **protected** event handler named **Submit_Click**.

```
protected void Submit_Click(object sender, EventArgs e)
{
}
```

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

```
// get the association data
string associationData = UpdateAssociationData();
```

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

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

```
private SPList LookupOrCreateList(string paramList, SPListTemplateType listType)
{
}
```

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

```
// lookup the history list
if (paramList.StartsWith("z"))
{
}
else
{ }
```

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

```
Guid newListId =
    Web.Lists.Add(paramList.Substring(1), "Workflow Tasks", listType);
return Web.Lists[newListId];
```

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

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

- In the **Submit_Click** method, add two lines of code to get the **SPList** objects for the **Tasks** and **History** lists. Use the **LookupOrCreateList** method to help.

```
// create or find the lists
SPList taskList = LookupOrCreateList(
    Request.Params["TaskList"], SPListTemplateType.Tasks);
SPList historyList = LookupOrCreateList(
    Request.Params["HistoryList"], SPListTemplateType.WorkflowHistory);
```

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

- Back in the **Submit_Click** method; check if derived content types should be updated by checking the **UpdateLists** form parameter.

```
// check whether updates to content types cascade to their lists
bool updateContentTypeLists = (Request.Params["UpdateLists"] == "TRUE");
```

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

```
// create or update the workflow association
if (_workflowAssociation != null)
    UpdateWorkflowAssociation(
        Request.Params["TaskList"], Request.Params["HistoryList"],
        associationData, updateContentTypeLists);
else
    CreateWorkflowAssociation(
        Request.Params["TaskList"], Request.Params["HistoryList"],
        associationData, updateContentTypeLists);
```

12) If the workflow association already exists, update it.

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

```
private void PopulateWorkflowAssociation(string taskListName,
    string historyListName, string associationData)
{ }
```

- Use the form parameters to initialize several of the startup and association data parameters in the **_workflowAssociation** field.

```
// assign the form data to the new workflow association
_workflowAssociation.Name = Request.Params["WorkflowName"];
_workflowAssociation.AutoStartCreate =
    (Request.Params["AutoStartCreate"] == "ON");
_workflowAssociation.AutoStartChange =
    (Request.Params["AutoStartChange"] == "ON");
_workflowAssociation.AllowManual = (Request.Params["AllowManual"] == "ON");
_workflowAssociation.AssociationData = associationData;
```

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

```
// assign/update the tasks and history lists
if (this._associationType != AssociationType.ContentType) {
    Splist taskList = LookupOrCreateList(
        taskListName, SplistTemplateType.Tasks);
    Splist historyList = LookupOrCreateList(
        historyListName, SplistTemplateType.WorkflowHistory);

    if (_workflowAssociation.TaskListId != taskList.ID)
        _workflowAssociation.SetTaskList(taskList);

    if (_workflowAssociation.HistoryListId != historyList.ID)
        _workflowAssociation.SetHistoryList(historyList);
}
else {
    if (_workflowAssociation.TaskListTitle != taskListName)
        _workflowAssociation.TaskListTitle = taskListName;
    if (_workflowAssociation.HistoryListTitle != historyListName)
        _workflowAssociation.HistoryListTitle = historyListName;
}
```

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

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

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

```
PopulateWorkflowAssociation(taskListName, historyListName, associationData);
```

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

```
switch (_associationType)
{
    case AssociationType.ContentType:
        break;
    case AssociationType.List:
        break;
    case AssociationType.ListContentType:
        break;
}
```

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

```
case AssociationType.ContentType:
    _contentType.UpdateWorkflowAssociation(_workflowAssociation);
    if (updateContentTypeLists)
        _contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);
```

- If the association type is **List**, call the list's **UpdateWorkflowAssociation** method.

```
case AssociationType.List:
    _list.UpdateWorkflowAssociation(_workflowAssociation);
```

- If the association type is **ListContentType**, call the content type's **UpdateWorkflowAssociation** method.

```
case AssociationType.ListContentType:
    _contentType.UpdateWorkflowAssociation(_workflowAssociation);
```

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

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

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

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

```
// lookup the workflow template and the workflow name
string workflowName = Request.Params["WorkflowName"];
Guid workflowTemplateId = new Guid(Request.Params["WorkflowDefinition"]);
SPWorkflowTemplate workflowTemplate = web.WorkflowTemplates[workflowTemplateId];
```

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

```
switch (_associationType)
{
    case AssociationType.ContentType:
        break;
```

```

    case AssociationType.List:
        break;
    case AssociationType.ListContentType:
        break;
}

```

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

Call SPWorkflowAssociation.CreateSiteContentTypeAssociation to create the workflow association.

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

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

Optionally call SPContentType.UpdateWorkflowAssociationsOnChildren to update child content types.

```

case AssociationType.ContentType:
    _workflowAssociation =
        SPWorkflowAssociation.CreateSiteContentTypeAssociation(
            workflowTemplate, workflowName, taskListName, historyListName);
    PopulateWorkflowAssociation(taskListName, historyListName, associationData);
    _contentType.AddWorkflowAssociation(_workflowAssociation);
    if (updateContentTypeLists)
        _contentType.UpdateWorkflowAssociationsOnChildren(true, true, true);
}

```

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

Call SPWorkflowAssociation.CreateListAssociation to create the workflow association.

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

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

```

case AssociationType.List:
    _workflowAssociation =
        SPWorkflowAssociation.CreateListAssociation(
            workflowTemplate, workflowName,
            LookupOrCreateList(taskListName, SplistTemplateType.Tasks),
            LookupOrCreateList(historyListName,
                SplistTemplateType.WorkflowHistory));
    PopulateWorkflowAssociation(taskListName, historyListName, associationData);
    _list.AddWorkflowAssociation(_workflowAssociation);
}

```

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

Call SPWorkflowAssociation.CreateListContentTypeAssociation to create the workflow association.

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

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

```

case AssociationType.ListContentType:
    _workflowAssociation =
        SPWorkflowAssociation.CreateListContentTypeAssociation(
            workflowTemplate, workflowName,
            LookupOrCreateList(taskListName, SplistTemplateType.Tasks),
            LookupOrCreateList(historyListName,
                SplistTemplateType.WorkflowHistory));
    PopulateWorkflowAssociation(taskListName, historyListName, associationData);
    _contentType.AddWorkflowAssociation(_workflowAssociation);
}

```

- 14) Finish the **Submit_Click** button event handler by redirecting to the settings page for the current list or content type.

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

```
// redirect back to the workflow settings page
SPUtility.Redirect(
    BuildRedirectString(
        _associationType, Request.Params["List"], Request.Params["ctype"]),
    SPRedirectFlags.RelativeToLayoutsPage, this.Context);
```

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

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

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

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

```
protected string WorkflowAssociationName
{
    get { return Request.Params["workflowName"]; }
}
```

Exercise 2: Creating a custom Association Form

- 1) Add a new base class named **DocArchivePart3AssocForm** that will be used as the code behind for an ASPX page.

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

Right click the UI folder and click Add -> Class....

Name the new file DocArchivePart3AssocForm.cs and click Add.

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

```
public class DocArchivePart3AssocForm : AssociationForm
{ }
```

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

```
protected PeopleEditor ppkApprover;
protected DropDownList lstArchiveList;
```

- 2) Populate the form by overriding the **PopulateControls** and **BindAssociationData** methods.

- Override the **PopulateControls** method.

```
protected override void PopulateControls()
{ }
```

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

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

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

// add each visible document library to the list
foreach (SPList documentLibrary in documentLibraries)
    1stArchiveList.Items.Add(
        new ListItem(documentLibrary.Title, documentLibrary.ID.ToString()));
```

- Override the **BindAssociationData** method.

```
protected override void BindAssociationData(string value)
{ }
```

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

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

```
// deserialize the association data
AssocInitData assocInitData = AssocInitData.Deserialize(value);
```

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

```
// bind the controls to the association data
ppkApprover.CommaSeparatedAccounts = assocInitData.ApproverUserName;
lstArchiveList.SelectedValue = assocInitData.ArchiveListId.ToString();
```

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

- Override the **UpdateAssociationData** class.

```
protected override string UpdateAssociationData()
{ }
```

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

```
// populate the association data using the UI
AssocInitData assocInitData = new AssocInitData();
assocInitData.ApproverUserName = (ppkApprover.Entities[0] as PickerEntity).Key;
assocInitData.ArchiveListId = lstArchiveList.SelectedValue;
```

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

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

```
// serialize the return the association data
return assocInitData.Serialize();
```

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

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

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

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

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

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

```
<%@ Assembly Name="DocumentArchiveworkflowPart3, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=15812f954569663f" %>

<%@ Page Language="C#" MasterPageFile="~/_layouts/application.master"
EnableSessionState="true" ValidateRequest="False"
Inherits="DocumentArchiveworkflowPart3.UI.DocArchivePart3AssocForm" %>
```

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

```
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
   Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
   PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
   Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
   PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormSection"
   Src="/_controltemplates/InputFormSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormControl"
   Src="/_controltemplates/InputFormControl.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ButtonSection"
   Src="/_controltemplates/ButtonSection.ascx" %>
```

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

```
<asp:Content ID="PageTitle" ContentPlaceHolderID="PlaceHolderPageTitle"
runat="server">
  <%= this.IsNewAssociation?"Customize Workflow":"Change a workflow"
%></asp:Content>
<asp:Content ID="PageTitleInTitleArea"
ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea"
runat="server">
  <%= this.IsNewAssociation?"Customize Workflow: " +
this.WorkflowAssociationName:"Change a Workflow: " +
this.WorkflowAssociationName %></asp:Content>
<asp:Content ID="PageDescription"
ContentPlaceHolderID="PlaceHolderPageDescription"
runat="server">
  <%= this.IsNewAssociation?"Use this page to define the settings of a new
workflow. ":"Use this page to change the settings of an existing workflow."
%></asp:Content>
```

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

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

```
<asp:Content ID="Main" ContentPlaceHolderID="PlaceHolderMain" runat="server">
  <table cellspacing="0" cellpadding="0" style="border: none; width: 100%"
class="ms-propertiesheet">
    </table>
</asp:Content>
```

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

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

```
<%-- Default Approver Input --%>
<wssuc:InputFormSection Title="Default Approver" Description="Specify the user
who will be the default approver."
runat="server">
<template_inputformcontrols>
    <wssuc:InputFormControl runat="server" LabelText="Approver:>
        <Template_Control>
            <SharePoint:PeopleEditor id="ppkApprover"
                AllowEmpty="false"
                ValidatorEnabled="true"
                MultiSelect="false"
                SelectionSet="User"
                width='300px'
                runat="server" />
        </Template_Control>
    </wssuc:InputFormControl>
</template_inputformcontrols>
</wssuc:InputFormSection>
```

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

```
<%-- Archive List Input --%>
<wssuc:InputFormSection Title="Archive Document Library"
    Description="Specify the document library used to store archived documents."
runat="server">
<template_inputformcontrols>
    <wssuc:InputFormControl LabelText="Archive Document Library:"
runat="server">
        <Template_Control>
            <asp:DropDownList ID="lstArchiveList" runat="server" />
        </Template_Control>
    </wssuc:InputFormControl>
</template_inputformcontrols>
</wssuc:InputFormSection>
```

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

```
<%-- Submit and Cancel Buttons --%>
<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">
<template_buttons>
    <asp:PlaceHolder runat="server">
        <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
        ButtonHeightWidth" OnClick="Submit_Click" Text="OK" id="BtnSubmit" /> &nbsp;
        <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
        ButtonHeightWidth" OnClick="Cancel_Click" Text="Cancel" id="cmdCancel"
        causesvalidation=false />
    </asp:PlaceHolder>
</template_buttons>
</wssuc:ButtonSection>
```

Exercise 3: Integrating custom Association Forms into the workflow

- 1) Handle the **Invoked** event of the **OnWorkflowActivated** and deserialize the association data.
 - Open the Workflow code by right clicking on the Workflow class in the **Solution Explorer** and clicking **View Code**.
 - Add a private field named **AssocInitData** of type **AssocInitData** to the **Workflow** class.

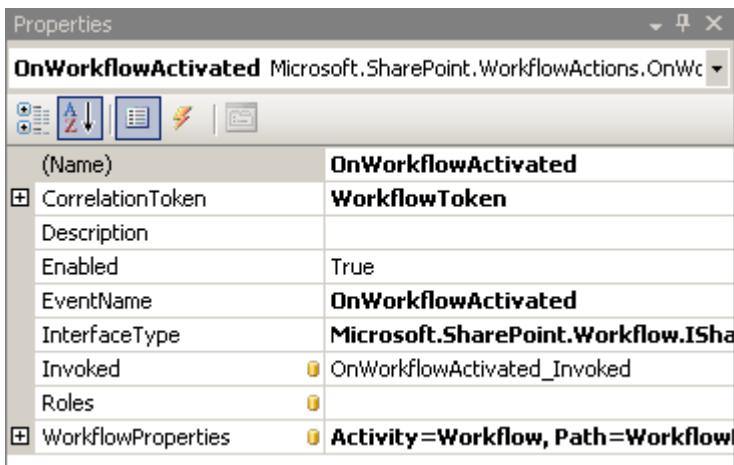
```
private AssocInitData AssocInitData { get; set; }
```

- Open the workflow in the designer by right clicking on the Workflow class in the **Solution Explorer** and clicking **View Designer**.
- Add an Invoked handler named **OnWorkflowActivated_Invoked** to the **OnWorkflowActivated** activity.

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

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

*In the properties pane, set **Invoked** to **OnWorkflowActivated_Invoked** and press **Enter**.*



- In the **OnWorkflowActivated_Invoked** method, deserialize the association data.

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

```
// deserialize init data, if none use defaults  
this.AssocInitData = AssocInitData.Deserialize(  
    WorkflowProperties.AssociationData);
```

- 2) Update the task creation to use the association data when initializing the task.
 - Locate the **CreateApprovalTask_Invoking** method.
 - Change the line that sets the **taskProperties.AssignedTo** property and get its value from **AssocInitData.ApproverUserName**.

```
//taskProperties.AssignedTo = "LITWAREINC\Administrator";  
taskProperties.AssignedTo = AssocInitData.ApproverUserName;
```

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

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

- 3) Update the task form to load the default archive list in the **OnLoad** method.
- Open the **DocArchivePart3TaskForm.cs** file in the UI folder.
 - Locate the **OnLoad** Method.
 - In the section where controls are initialized, set the **SelectedValue** of the **IstArchiveList** control using the extended properties of the task.

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

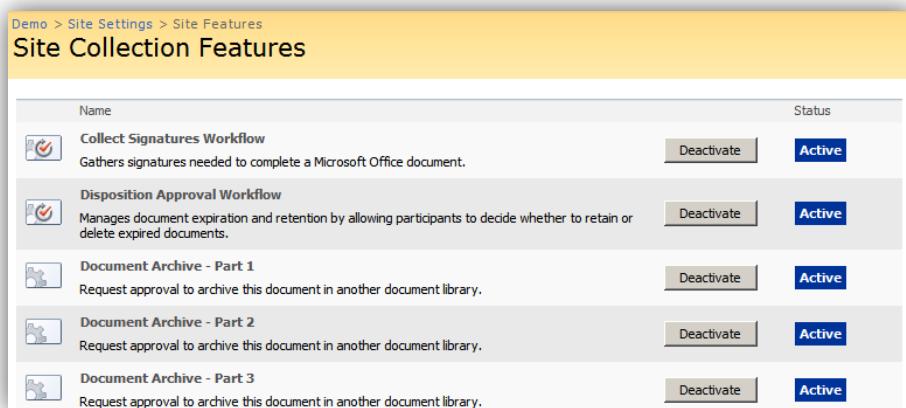
```
// initialize the archive list selection
Hashtable extendedProperties =
    SPWorkflowTask.GetExtendedPropertiesAsHashtable(_taskItem);
IstArchiveList.SelectedValue = extendedProperties["ArchiveListId"] as string;
```

Exercise 4: Deploying custom Association Forms

- 1) Register the new custom Association form by assigning the **AssociationUrl** attribute in the feature's **Workflow** element.

```
<workflow ...
    AssociationUrl ="_layouts/DocArchivePart3AssocForm.aspx"
    StatusUrl=" _layouts/wrkstat.aspx">
    <Categories/>
    <MetaData>
    </MetaData>
</Workflow>
```

- 2) 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.
- 3) Activate the new **DocumentArchiveWorkflowPart3** feature.
- Using **Internet Explorer** navigate to the **Demo** site collection at <http://litwareinc.com/sites/Demo>.
 - Open the features list by clicking **Site Actions -> Site Settings**.
 - On the **Site Settings** page, click **Site collection features** in the **Site Collection Administration** section.
 - Click the **Activate** button next to the **Document Archive – Part 3** feature.



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

- Navigate to the **Shared Documents** document library in the **Demos** site.
- Click **Settings -> Document Library Settings** to load the settings page.
- Click the **Workflow settings** link in the **Permissions and Management section**.
- Create a new workflow using the **Document Archive – Part 3** workflow template and a name of **Document Archive - Part 3**.

Use the default values for both list and startup options.

Demo > Shared Documents > Settings > Workflow settings > Add or Change a Workflow

Add a Workflow: Shared Documents

Use this page to set up a workflow for this document library.

Workflow

Select a workflow to add to this document library. If the workflow template you want does not appear, contact your administrator to get it added to your site collection or workspace.

Select a workflow template:

- Disposition Approval
- Document Archive - Part 1
- Document Archive - Part 2
- Document Archive - Part 3**

Description:

Requests approval to archive this document in another document library.

Name

Type a name for this workflow. The name will be used to identify this workflow to users of this document library.

Type a unique name for this workflow:

Document Archive - Part 3

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

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

Customize Workflow: Document Archive - Part 3

Use this page to define the settings of a new workflow.

Default Approver

Specify the user who will be the default approver.

Approver:

LitwareInc Administrator



Archive Document Library

Specify the document library used to store archived documents.

Archive Document Library:

Quotes

OK

Cancel

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

- Navigate to the Shared Documents document library, hover over the new document, and select **Workflows** from the drop down menu.
- In the workflows page, click the **Document Archive – Part 3** to start the workflow.
- In the **Shared Documents** document library, verify the workflow is running.

- Click the **In Progress** link to view the workflow status and verify the started message was logged to the workflow's history.
- Edit the task and set its **Status** to **Completed**.

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

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

Tasks : Archival Approval

Item Requiring Approval Please review this item/document.	Click on item hyperlink below to review item/document List: Shared Documents Item: Document to Archive
Archive Document Library Specify the document library used to store archived documents.	Archive Document Library: Quotes
<input type="button" value="Approve"/> <input type="button" value="Reject"/> <input type="button" value="Cancel"/>	

- 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.

Demo Site > Shared Documents > Workflow Status

Workflow Status: Document Archive - Part 3

Workflow Information				
Initiator: LitwareInc Administrator	Document: Document to Archive			
Started: 1/27/2008 11:45 AM	Status: Completed			
Last run: 1/27/2008 11:47 AM				
Tasks				
The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list Tasks.				
Assigned To	Title	Due Date	Status	Outcome
LitwareInc Administrator	Archival Approval ! NEW		Completed	

Challenge: Integrating Association Forms into Solution Packages

- Add the task form's .ASPX file to the **Manifest.xml** file.
 - Open the **Manifest.xml** file in the **Solution** folder.
 - Add a **TemplateFile** element telling the solution installer to install the association form when the solution is deployed.

```
<TemplateFiles>
  <TemplateFile Location="LAYOUTS\DocArchivePart3TaskForm.aspx"/>
  <TemplateFile Location="LAYOUTS\DocArchivePart3AssocForm.aspx"/>
</TemplateFiles>
```

- 2) Add the custom task form to the **Package.ddf** file.
- Open the **Package.ddf** file in the **Solution** folder.
 - Add a line that packages the **DocArchivePart3AssocForm.aspx** and puts it in the **LAYOUTS** folder in the **.cab** file.

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

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

- 3) 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 **DocumentArchiveWorkflowPart3** feature.

```
stsadm -o uninstallfeature -name DocumentArchiveworkflowPart3 -force
```

- Delete the folder at **C:\\Program Files\\Common Files\\Microsoft Shared\\web server extensions\\12\\Template\\Features\\DocumentarchiveWorkflowPart3**.
- Delete the **DocArchivePart3AssocForm.aspx** and **DocArchivePart3TaskForm.aspx** files from the **C:\\Program Files\\Common Files\\Microsoft Shared\\web server extensions\\12\\Template\\Layouts** folder.

- 4) Install and deploy the new solution package.

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

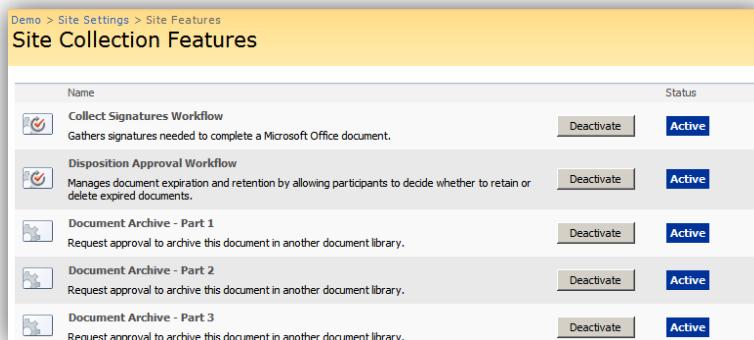
```
stsadm -o addsolution -filename
"C:\\Labs\\Lab08\\StarterFiles\\DocumentArchiveworkflowPart3\\bin\\Debug\\Package\\DocumentArchiveworkflowPart3.wsp"
```

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

```
stsadm -o deploysolution -name DocumentArchiveworkflowPart3.wsp -
allowgacdeployment -local
```

- 5) Verify the feature is still available and activated.

- Using **Internet Explorer** navigate to the **Demo** site collection at <http://litwareinc.com/sites/Demo>.
- Open the features list by clicking **Site Actions -> Site Settings**.
- On the **Site Settings** page, click **Site collection features** in the **Site Collection Administration** section.
- Verify the **Document Archive – Part 3** feature is active.



Lab 09: Creating custom Instantiation and Modification Forms

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

In this lab, you will be adding an Instantiation form and a Modification form to the workflow. This will allow the user to override the association settings when starting the workflow and even change them while the workflow is running. With these two forms added, the archive workflow will be complete.

Exercise 0: Setup

- 1) 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.
- 2) Open the starter **VS 2008** solution at `\Labs\Lab09\StarterFiles\DocumentArchive Part 4.sln`.

Exercise 1: Creating a Custom Instantiation Form

- 1) Add a new base class named **DocArchivePart4InitForm** that will be used as the code behind for an ASPX page.
 - Create a new class named **DocArchivePart4InitForm** in the **UI** folder.
*Right click the **UI** folder and click **Add -> Class....**
Name the new file `DocArchivePart4InitForm.cs` and click **Add**.*
 - Change the **DocArchivePart4InitForm** class into a public class that derives from `Microsoft.SharePoint.WebControls.LayoutsPageBase`.

```
public abstract class DocArchivePart4InitForm : LayoutsPageBase
{
}
```

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

```
private SPWorkflowAssociation _workflowAssociation;
private SPList _list;
private SPListItem _listItem;
```

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

- Add a **protected** field named **ppkApprover** of type **PeopleEditor** to the class.
- Add a **protected** field named **IstArchiveList** of type **DropDownList** to the class.
- Add a **protected** property named **WorkflowName** of type string to the class. Have it return the **Name** property of the **_workflowAssociation** field.
- Add a **protected** property named **ItemName** of type string to the class. Have it return the **DisplayName** property of the **_listItem** field.

```
protected PeopleEditor ppkApprover;
protected DropDownList IstArchiveList;

protected string WorkflowName
{
    get { return _workflowAssociation.Name; }
}

protected string ItemName
{
    get { return _listItem.DisplayName; }
}
```

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

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

```
protected override void OnLoad(EventArgs e)
{ }
```

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

```
// read the form level parameters
string listId = Request.Params["List"];
string listItemID = Request.Params["ID"];
string ctypeID = Request.Params["ctype"];
string templateID = Request.Params["TemplateID"];
```

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

```
// find the list and list item
_list = web.Lists[new Guid(listId)];
_listItem = _list.GetItemById(int.Parse(listItemId));
```

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

```
// check for the workflow association in the list
_workflowAssociation = _list.WorkflowAssociations[new Guid(templateID)];
```

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

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

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

```
// if the association wasn't found, check the content type
if (_workflowAssociation == null)
{
    SPContentType _contentType =
        _list.ContentTypes[new SPContentTypeId(ctypeId)];
    _workflowAssociation =
        _contentType.WorkflowAssociations[new Guid(templateId)];
}
```

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

```
// make sure the workflow association could be found
if (_workflowAssociation == null)
    throw new SPException("The requested workflow could not be found.");
```

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

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

```
// populate the UI
if (!this.IsPostBack)
{}
```

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

```
// filter the list of lists for all document libraries that aren't hidden
IEnumerable<SPList> documentLibraries =
    Web.Lists.Cast<SPList>().Where(n => n is SPDocumentLibrary && !n.Hidden);

// add each visible document library to the list
foreach (SPList documentLibrary in documentLibraries)
    1stArchiveList.Items.Add(new ListItem(documentLibrary.Title,
    documentLibrary.ID.ToString()));
```

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

```
// populate the form using the association data
AssocInitData assocInitData =
AssocInitData.Deserialize(_workflowAssociation.AssociationData);
ppkApprover.CommaSeparatedAccounts = assocInitData.ApproverUserName;
1stArchiveList.SelectedValue = assocInitData.ArchiveListId.ToString();
```

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

```
// call the base implementation
base.OnLoad(e);
```

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

- Create a **protected** event handler named **Cancel_Click**.
- Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list's default view url.

```
protected void Cancel_Click(object sender, EventArgs e)
{
    // redirect to the page in the source url parameter or the default page
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.UseSource, this.Context);
}
```

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

- Create a **protected** event handler named **Start_Click**.

```
protected void Start_Click(object sender, EventArgs e)
{
}
```

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

```
// populate the association data using the UI
AssocInitData assocInitData = new AssocInitData();
assocInitData.ApproverUserName = (ppkApprover.Entities[0] as PickerEntity).Key;
assocInitData.ArchiveListId = 1stArchiveList.SelectedValue;

// get the initiation data
string initiationData = assocInitData.Serialize();
```

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

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

```
// start the new workflow instance
Web.Site.WorkflowManager.StartWorkflow(_listItem,
    _workflowAssociation, initiationData);
```

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

```
// redirect to the list default view
SPUtility.Redirect(_list.DefaultViewUrl,
    SPRedirectFlags.UseSource, this.Context);
```

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

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

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

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

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

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

```
<%@ Assembly Name="DocumentArchiveworkflowPart4, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=15812f954569663f" %>

<%@ Page Language="C#" MasterPageFile="~/_layouts/application.master"
EnableSessionState="true" ValidateRequest="False"
Inherits="DocumentArchiveworkflowPart4.UI.DocArchivePart4InitForm" %>
```

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

```
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormSection"
Src="/_controltemplates/InputFormSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormControl"
Src="/_controltemplates/InputFormControl.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ButtonSection"
Src="/_controltemplates/ButtonSection.ascx" %>
```

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

```
<asp:Content ID="PageTitle" ContentPlaceHolderID="PlaceHolderPageTitle"
runat="server">
    Start Workflow</asp:Content>
<asp:Content ID="PageTitleInTitleArea"
ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea"
runat="server">
    <%= "Start \"\" + this.WorkflowName + "\": " + this.ItemName %></asp:Content>
```

- 8) 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%">
        </table>
    </asp:Content>
```

- Implement the first **InputFormSection** that allow entry of the primary approver.

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

```
<%-- Default Approver Input --%>
<wssuc:InputFormSection Title="Default Approver" Description="Specify the user
who will be the default approver." runat="server">
    <template_inputformcontrols>
        <wssuc:InputFormControl runat="server" LabelText="Approver:>
            <Template_Control>
                <SharePoint:PeopleEditor id="ppkApprover"
                    AllowEmpty="false"
                    ValidatorEnabled="true"
                    Multiselect="false"
                    runat="server"
                    SelectionSet="User"
                    width='300px' />
            </Template_Control>
        </wssuc:InputFormControl>
    </template_inputformcontrols>
</wssuc:InputFormSection>
```

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

```
<%-- Archive List Input --%>
<wssuc:InputFormSection Title="Archive Document Library"
    Description="Specify the document library used to store archived documents."
    runat="server">
    <template_inputformcontrols>
        <wssuc:InputFormControl LabelText="Archive Document Library:"
            runat="server">
            <Template_Control>
                <asp:DropDownList ID="lstArchiveList" runat="server" />
            </Template_Control>
        </wssuc:InputFormControl>
    </template_inputformcontrols>
</wssuc:InputFormSection>
```

- Implement the last section containing the **Start** and **Cancel** buttons.

```
<%-- Start and Cancel Buttons --%>
<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">
    <template_buttons>
        <asp:PlaceHolder runat="server">
            <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
ButtonHeightWidth" OnClick="Start_Click" Text="Start" id="BtnSubmit" /> &nbsp;
            <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
ButtonHeightWidth" OnClick="Cancel_Click" Text="Cancel" id="cmdCancel"
            causesvalidation=false />
        </asp:PlaceHolder>
    </template_buttons>
</wssuc:ButtonSection>
```

Exercise 2: Integration Instantiation Forms into the Workflow

- 1) Update the workflow to use the initiation data instead of the association data.
 - Open **Workflow.cs** as code by right clicking it in the **Solution Explorer** and clicking **View Code**.
 - Locate the **OnWorkflowActivated_Invoked** method.
 - This method uses **WorkflowProperties.AssociationData** to initialize the workflow. Change the code to use the **InitiationData** property instead.

```
private void OnWorkflowActivated_Invoked(object sender, ExternalDataEventArgs e)
{
    // deserialize init data, if none use defaults
    this.AssocInitData = AssocInitData.Deserialize(
        workflowProperties.InitiationData);
}
```

Exercise 3: Deploying custom Instantiation Forms

- 1) Register the new custom Instantiation form by assigning the **InstantiationUrl** attribute in the feature's **Workflow** element.

```
<workflow ...
    InstantiationUrl="_layouts/DocArchivePart4InitForm.aspx"
    StatusUrl="_layouts/wrkStat.aspx">
    <Categories/>
    <MetaData>
    </MetaData>
</Workflow>
```

- 2) 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.
- 3) Activate the new **DocumentArchiveWorkflowPart4** feature.
 - Using **Internet Explorer** navigate to the **Demo** site collection at <http://litwareinc.com/sites/Demo>.
 - Open the features list by clicking **Site Actions -> Site Settings**.
 - On the **Site Settings** page, click **Site collection features** in the **Site Collection Administration** section.
 - Click the **Activate** button next to the **Document Archive – Part 4** feature.

Demo > Site Settings > Site Features

Site Collection Features

Name	Status
Collect Signatures Workflow Gathers signatures needed to complete a Microsoft Office document.	Deactivate Active
Disposition Approval Workflow Manages document expiration and retention by allowing participants to decide whether to retain or delete expired documents.	Deactivate Active
Document Archive - Part 1 Request approval to archive this document in another document library.	Deactivate Active
Document Archive - Part 2 Request approval to archive this document in another document library.	Deactivate Active
Document Archive - Part 3 Request approval to archive this document in another document library.	Deactivate Active
Document Archive - Part 4 Request approval to archive this document in another document library.	Deactivate Active

- 4) Create an association between the **Shared Documents** document library and the new **Document Archive – Part 4** workflow.
- Navigate to the **Shared Documents** document library in the **Demos** site.
 - Click **Settings** -> **Document Library Settings** to load the settings page.
 - Click the **Workflow settings** link in the **Permissions and Management section**.
 - Create a new workflow using the **Document Archive – Part 4** workflow template and a name of **Document Archive - Part 4**.

Use the default values for both list and startup options.

Demo > Shared Documents > Settings > Workflow settings > Add or Change a Workflow

Add a Workflow: Shared Documents

Use this page to set up a workflow for this document library.

Workflow Select a workflow to add to this document library. If the workflow template you want does not appear, contact your administrator to get it added to your site collection or workspace.	Select a workflow template: <input type="button" value="Document Archive - Part 3"/> <input checked="" type="button" value="Document Archive - Part 4"/> <input type="button" value="DocumentArchiveWorkflow"/> <input type="button" value="Hello World Workflow"/>	Description: Requests approval to archive this document in another document library.
Name Type a name for this workflow. The name will be used to identify this workflow to users of this document library.	Type a unique name for this workflow: <input type="text" value="Document Archive - Part 4"/>	

- In the custom association form, enter the **Administrator** as both the default approver and the manager. Select the **Quotes** document library as the archive location.
*If you'd like, create an **Archive** document library to use instead.*

Customize Workflow: Document Archive - Part 4

Use this page to define the settings of a new workflow.

Default Approver Specify the user who will be the default approver.	Approver: <input type="text" value="LitwareInc Administrator"/>
Archive Document Library Specify the document library used to store archived documents.	Archive Document Library: <input type="text" value="Quotes"/>
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

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

- Navigate to the Shared Documents document library, hover over the new document, and select **Workflows** from the drop down menu.
- In the workflows page, click the **Document Archive – Part 4** to start the workflow.
- In the custom initiation page, verify the form initialized using the data provided in the custom association form.
- Change any values you would like and click **Start** to start a new instance of the workflow.

Start "Document Archive - Part 4": Document to Archive

Default Approver Specify the user who will be the default approver.	Approver: <input type="text" value="LitwareInc Administrator"/>
Archive Document Library Specify the document library used to store archived documents.	Archive Document Library: <input type="text" value="Quotes"/>
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

6) Complete the workflow by approving the task in the workflow status page.

Exercise 4: Creating a Custom Modification Form

- 1) Add a new base class named **DocArchivePart4ModForm** that will be used as the code behind for an ASPX page.
- Create a new class named **DocArchivePart4ModForm** in the **UI** folder.

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

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

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

```
public abstract class DocArchivePart4ModForm : LayoutsPageBase
{ }
```

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

```
private SPList _list;
private SPListItem _listItem;
private SPWorkflow _workflow;
private SPWorkflowModification _modification;
```

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

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

```
protected PeopleEditor pplApprover;
protected PeopleEditor pplManager;
```

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

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

```
protected override void OnLoad(EventArgs e)
{ }
```

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

```
// read the form level parameters
string listId = Request.Params["List"];
string listItemID = Request.Params["ID"];
string workflowId = Request.Params["WorkflowInstanceId"];
string modificationId = Request.Params["ModificationID"];
```

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

```
// find the list and workflow modification objects
_list = Web.Lists[new Guid(listId)];
_listItem = _list.GetItemById(Convert.ToInt32(listItemID));
_workflow = _listItem.Workflows[new Guid(workflowId)];
_modification = _workflow.Modifications[new Guid(modificationId)];
```

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

```
// populate the UI
if (!this.IsPostBack)
{
}
```

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

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

```

// deserialize the association data
ModificationData modData =
ModificationData.Deserialize(_modification.ContextData);

// bind the controls to the association data
pplApprover.CommaSeparatedAccounts = modData.ApproverUserName;

```

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

```

// call the base implementation
base.OnLoad(e);

```

- 4) Implement the event handler called after a click of the **Cancel** button.

- Create a **protected** event handler named **Cancel_Click**.
- Use the **SPUtility.Redirect** method to handle the automatic redirection to either the source url parameter or the current list's default view url.

```

protected void Cancel_Click(object sender, EventArgs e)
{
    // redirect to the page in the source url parameter or the default page
    SPUtility.Redirect(_list.DefaultViewUrl,
        SPRedirectFlags.UseSource, this.Context);
}

```

- 5) Implement the event handler that will be called when the **Start** button is clicked.

- Create a protected event handler named **Start_Click**.

```

protected void Submit_Click(object sender, EventArgs e)
{
}

```

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

```

// populate the modification data using the UI
ModificationData modData = new ModificationData();
modData.ApproverUserName = (pplApprover.Entities[0] as PickerEntity).Key;

// serialize the modification data
string modificationData = modData.Serialize();

```

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

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

```

// apply the modification to the workflow
web.Site.WorkflowManager.ModifyWorkflow(_workflow,
    _modification,
    modificationData);

```

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

```
// redirect to the page defined in the source url parameter or the default page
SPUtility.Redirect(_list.DefaultviewUrl,
    SPRedirectFlags.UseSource, this.Context);
```

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

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

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

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

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

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

```
<%@ Assembly Name="DocumentArchiveWorkflowPart4, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=15812f954569663f" %>

<%@ Page Language="C#" MasterPageFile="~/_layouts/application.master"
EnableSessionState="true" ValidateRequest="False"
Inherits="DocumentArchiveworkflowPart4.UI.DocArchivePart4ModForm" %>
```

- Register the tag prefix and locations of the SharePoint web and user controls used to format the page.

```
<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormSection"
Src="/_controltemplates/InputFormSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormControl"
Src="/_controltemplates/InputFormControl.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ButtonSection"
Src="/_controltemplates/ButtonSection.ascx" %>
```

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

```
<asp:Content ID="PageTitle" ContentPlaceHolderID="PlaceHolderPageTitle"
runat="server">
    Modify Workflow</asp:Content>
<asp:Content ID="PageTitleInTitleArea"
ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
    <%= "Modify" %></asp:Content>
```

- 7) 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-propertiesheet">
        </table>
</asp:Content>
```

- Implement the first **InputFormSection** that allows modification of the approver.

```
<%-- Default Approver Input --%>
<wssuc:InputFormSection Title="Default Approver" Description="Specify the user
who will be the default approver." runat="server">
    <template_inputformcontrols>
        <wssuc:InputFormControl runat="server" LabelText="Approver:>
            <Template_Control>
                <SharePoint:PeopleEditor id="pplApprover"
                    AllowEmpty="false"
                    ValidatorEnabled="true"
                    Multiselect="false"
                    runat="server"
                    SelectionSet="User"
                    width='300px' />
            </Template_Control>
        </wssuc:InputFormControl>
    </template_inputformcontrols>
</wssuc:InputFormSection>
```

- Implement the last section containing the **OK** and **Cancel** buttons.

```
<%-- Submit and Cancel Buttons --%>
<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">
    <template_buttons>
        <asp:PlaceHolder runat="server">
            <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
ButtonHeightWidth" OnClick="Submit_Click" Text="OK" id="BtnSubmit" /> &nbsp;
            <asp:Button UseSubmitBehavior="false" runat="server" class="ms-
ButtonHeightWidth" OnClick="Cancel_Click" Text="Cancel" id="cmdCancel"
causesvalidation=false />
        </asp:PlaceHolder>
    </template_buttons>
</wssuc:ButtonSection>
```

Exercise 5: Enabling Modifications in the Workflow

- Communicate with the SharePoint Workflow hosting environment to allow modification of this workflow.
 - Open **Workflow.cs** in the designer by right clicking it in the **Solution Explorer** and selecting **View Designer**.
 - Double click the **WorkflowActivatedActivity** activity to open its designer.
 - Drag an **EnableWorkflowModification** activity from the toolbox into the **WorkflowActivatedActivity** immediately following the **OnWorkflowActivated** activity and name it **EnableModification**..

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

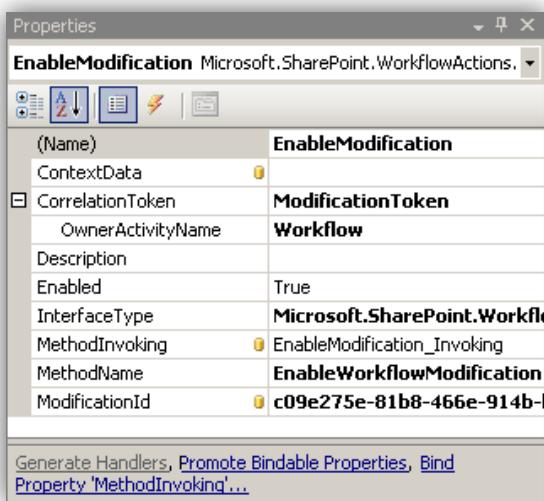
- Assign a correlation token to the modification and assign the id of the modification to enable.

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

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

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

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



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

*In the properties pane, set the **Invoking** property to **EnableModification_Invoking** and press **Enter**.*

```
private void EnableModification_Invoking(object sender, EventArgs e)
{
    // populate the modification data object
    ModificationData modData = new ModificationData();
    modData.ApproverUserName = AssocInitData.ApproverUserName;

    // serialize and assign the context data to the modification
    EnableWorkflowModification activity = sender as EnableWorkflowModification;
    activity.ContextData = modData.Serialize();
}
```

- Add an **EventDrivenActivity** to the workflow that will receive the notification the workflow has been modified. When that happens, it will update the **AssocInitData** object and reset any tasks already created.
 - On the **State Machine** designer canvas, right click and select **Add EventDriven**.
 - Rename the new event driven activity to **WorkflowModifiedActivity**.

*In the properties pane, set the **Name** property to **WorkflowModifiedActivity**.*
 - Drag an **OnWorkflowModified** activity from the toolbox into the **WorkflowModifiedActivity** activity and name it **OnWorkflowModified**.

*In the properties pane, set the **Name** property to **OnWorkflowModified**.*
 - Update the new activity to respond to the modification previously enabled.

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

*Set the **ModificationId** property to **c09e275e-81b8-466e-914b-bae95eb27dfa**.*
 - Implement the **Invoked** event and retrieve the updated modification data for integration back into the running workflow.

*Set the **Invoked** property to **OnWorkflowModified_Invoked** and press **Enter**.*

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

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

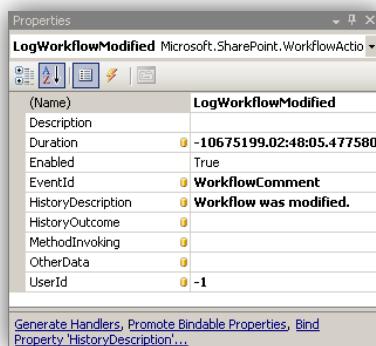
```

private void OnWorkflowModified_Invoked(object sender, ExternalDataEventArgs e)
{
    SPMModificationEventArgs args = e as SPMModificationEventArgs;

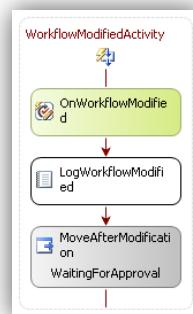
    // deserialize the modification data and apply the changes
    ModificationData modData = ModificationData.Deserialize(args.data);
    AssocInitData.ApproverUserName = modData.ApproverUserName;
}

```

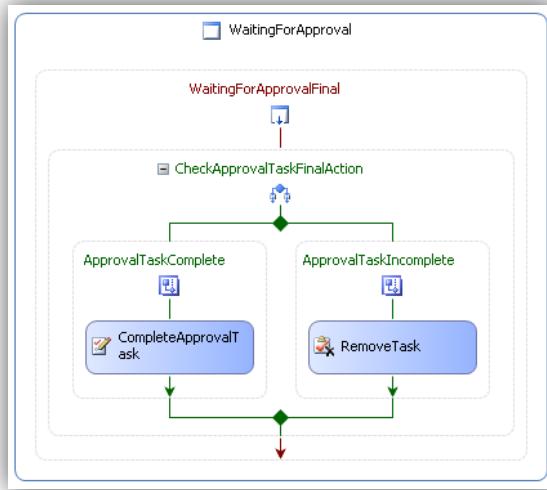
- 3) Log the fact that the modification has taken place and transition back to the **WaitingForApproval** state.
- Drag a **LogToListActivity** from the toolbox into the **WorkflowModifiedActivity** activity and name it **LogWorkflowModified**.
*In the properties pane, set the **Name** property to **LogWorkflowModified**.
Set the **HistoryDescription** property to “Workflow was modified.”*



- Drag a **SetState** activity from the toolbox into the **WorkflowModifiedActivity** at the end of the sequence and name it **MoveAfterModification**.
*In the properties pane, set the **Name** property to **MoveAfterModification**.
Set the **TargetStateName** property to **WaitingForApproval**.*



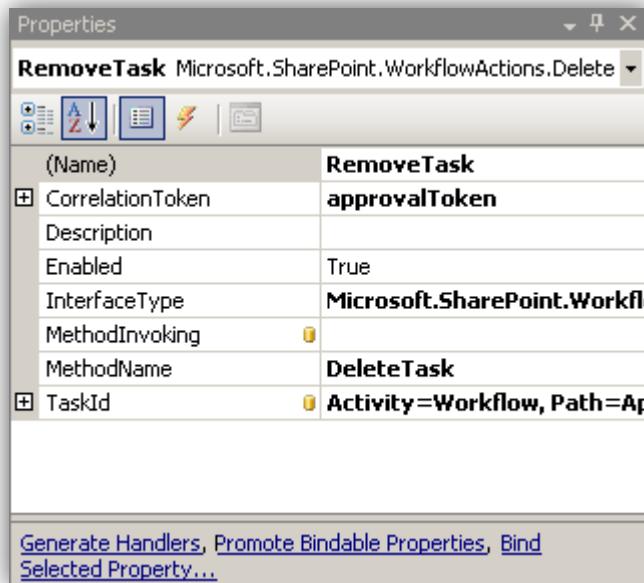
- 4) Since modifications will happen when tasks already exist, they will need to be recreated after the modification. To do this, the finalizer and initializer for the state come into play.
- Open the **WaitingForApprovalFinal** activity in the State Machine designer canvas.
 - Drag a **DeleteTask** activity from the toolbox into the **ApprovalTaskIncomplete** branch and name it **RemoveTask**.



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

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

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



Exercise 6: Deploying custom Modification Forms

- Register the new custom modification form by assigning the **ModificationUrl** attribute in the feature's **Workflow** element and registering the modification metadata.
 - Open the **workflow.xml** file in the **Template/Features/DocumentArchiveWorkflowPart4** folder.
 - Add the **ModificationUrl** attribute to the **Workflow** element. Point it toward **_layouts/DocArchivePart4ModForm.aspx**.

```

<workflow ...
    ModificationUrl="_layouts/DocArchivePart4ModForm.aspx"
    StatusUrl="_layouts/wrkstat.aspx">
    <Categories/>
    <MetaData>
    </MetaData>
</workflow>

```

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

*Use the format **Modification_{0}_Name** where the {0} token is replaced by the modification's id.*

```

<MetaData>
    <Modification_c09e275e-81b8-466e-914b-bae95eb27dfa_Name>Change
    approvers</Modification_c09e275e-81b8-466e-914b-bae95eb27dfa_Name>
</MetaData>

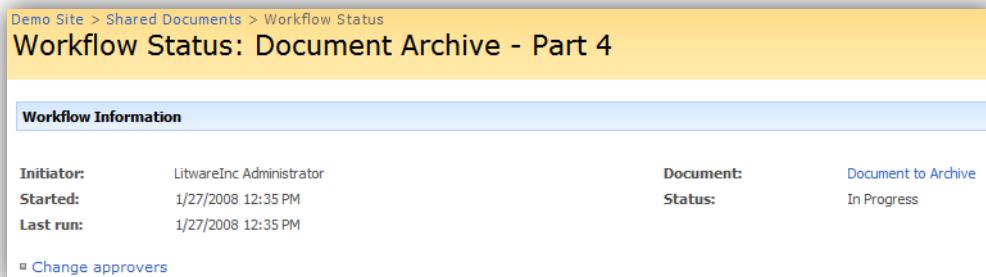
```

2) Rebuild the workflow.

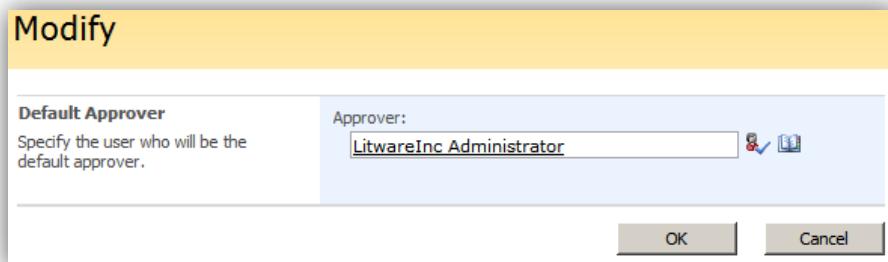
- Right click the project in the **Solution Explorer** and click **Rebuild**.
- In the **Output** window, verify the post build actions completed successfully.

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

- Navigate to the **Shared Documents** document library, hover over the new document, and select **Workflows** from the drop down menu.
- In the workflows page, click the **Document Archive – Part 4** to start the workflow.
- When the custom instantiation page opens, click **Start** to create the workflow instance with the default values.
- In the Shared Documents UI, click the **In Progress** link to view the status of the workflow. Click the **Change approvers** link in the Workflow Information section to modify the workflow.



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



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

Workflow History			
Date Occurred	Event Type	User ID	Description
1/27/2008 12:35 PM	Comment	System Account	Workflow was started
1/27/2008 12:36 PM	Comment	System Account	Workflow was modified.

- Complete the workflow by approving the task.

Challenge: Integrate Instantiation and Modification Forms into Solution Packages

- Add the task form's .ASPX file to the **Manifest.xml** file.

- Open the **Manifest.xml** file in the **Solution** folder.
- Add a **TemplateFile** element telling the solution installer to install the association form when the solution is deployed.

```
<TemplateFiles>
  <TemplateFile Location="LAYOUTS\DocArchivePart4TaskForm.aspx"/>
  <TemplateFile Location="LAYOUTS\DocArchivePart4AssocForm.aspx"/>
  <TemplateFile Location="LAYOUTS\DocArchivePart4InitForm.aspx"/>
  <TemplateFile Location="LAYOUTS\DocArchivePart4ModForm.aspx"/>
</TemplateFiles>
```

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

- Open the **Package.ddf** file in the **Solution** folder.
- Add lines that package the **DocArchivePart4InitForm.aspx** and **DocArchivePart4ModForm.aspx** and puts it in the **LAYOUTS** folder in the .cab file.

```
..\..\Template\Layouts\DocArchivePart4InitForm.aspx
LAYOUTS\DocArchivePart4InitForm.aspx
..\..\Template\Layouts\DocArchivePart4ModForm.aspx
LAYOUTS\DocArchivePart4ModForm.aspx
```

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

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

Click Start -> Run and enter cmd.

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

- Execute **stsadm.exe** to uninstall the **DocumentArchiveWorkflowPart4** feature.

```
stsadm -o uninstallfeature -name DocumentArchiveWorkflowPart4 -force
```

- Delete the folder at **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Features\DocumentarchiveWorkflowPart4**.
- Delete the **DocArchivePart4AssocForm.aspx**, **DocArchivePart4InitForm.aspx**, **DocArchivePart4TaskForm.aspx**, and **DocArchivePart4ModForm.aspx** files from the **C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\Template\Layouts** folder.

4) Install and deploy the new solution package.

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

```
stsadm -o addsolution -filename  
"C:\Labs\Lab09\StarterFiles\DocumentArchiveWorkflowPart4\bin\Debug\Package\DocumentArchiveWorkflowPart4.wsp"
```

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

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

5) Verify the feature is still available and activated.

- Using **Internet Explorer** navigate to the **Demo** site collection at <http://litwareinc.com/sites/Demo>.
- Open the features list by clicking **Site Actions -> Site Settings**.
- On the **Site Settings** page, click **Site collection features** in the **Site Collection Administration** section.
- Verify the **Document Archive – Part 4** feature is active.

Site Collection Features		
Name	Status	
Collect Signatures Workflow Gathers signatures needed to complete a Microsoft Office document.	Deactivate	Active
Disposition Approval Workflow Manages document expiration and retention by allowing participants to decide whether to retain or delete expired documents.	Deactivate	Active
Document Archive - Part 1 Request approval to archive this document in another document library.	Deactivate	Active
Document Archive - Part 2 Request approval to archive this document in another document library.	Deactivate	Active
Document Archive - Part 3 Request approval to archive this document in another document library.	Deactivate	Active
Document Archive - Part 4 Request approval to archive this document in another document library.	Deactivate	Active

Challenge:

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

Lab 10: Integrating InfoPath Forms into a SharePoint Workflow Template

Lab Overview: Designing workflow forms in ASP.NET is not a task easily delegated to web designers. There is no simple way to provide web designers with a visual design environment; therefore, workflow form design is currently the responsibility of developers. This is not a good scenario for forms that may need minor layout and branding adjustments that don't change the way they work.

The solution to this problem is to use InfoPath. InfoPath provides a functional user interface that interacts with XML data. This is perfect for interacting with SharePoint Workflow Association, Initiation, Task, and Modification data. In this lab, you'll be building a simple approval workflow using InfoPath as the user interface.

Exercise 0: Setup

- 1) If you did not complete lab 4, you will need to create the **Demo** site collection.

Open SharePoint and browse to the Demo site collection

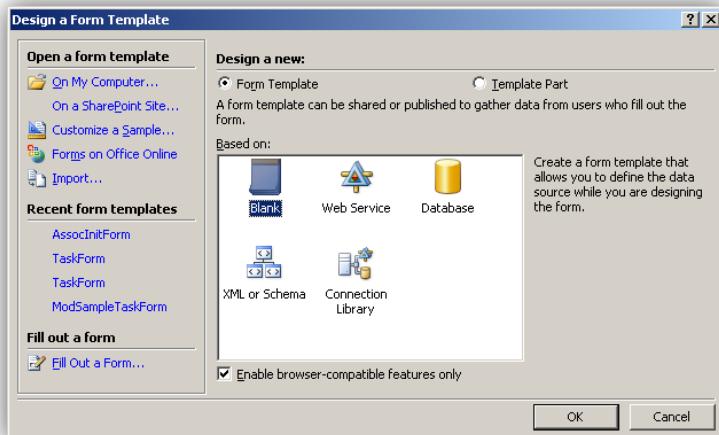
The url is <http://litwareinc.com/sites/Demo>.

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

- 2) Open the starter **VS 2008** solution at `\Labs\Lab10\StarterFiles\InfoPathApprovalWorkflow.sln`.

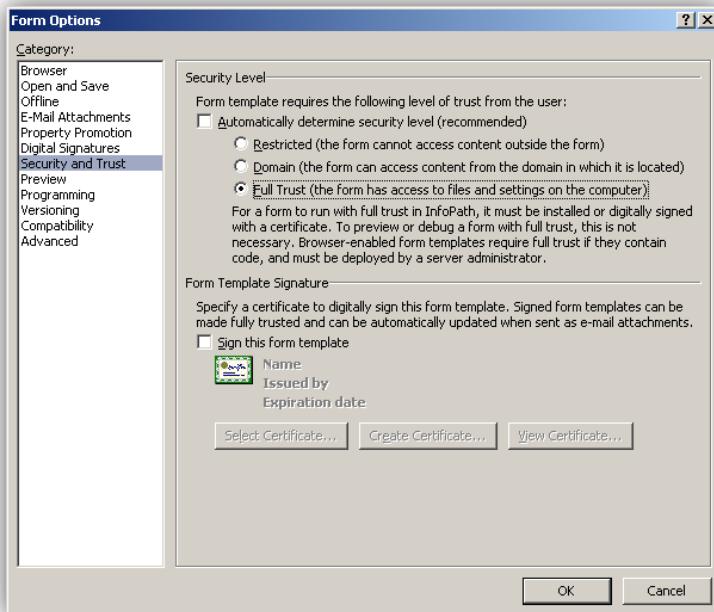
Exercise 1: InfoPath Association Forms

- 1) Create a new InfoPath form named **AssocInitForm.xsn** in the project's **Forms** folder.
 - Open InfoPath by clicking the **Start Button** and selecting **All Programs -> Microsoft Office -> Microsoft Office InfoPath 2007**.
 - In the **Getting Started** window, click the **Design a Form Template** link in the **Design a form** section.
 - In the **Design a Form Template** window, select a **Blank** form template and make sure the **Enable browser-compatible features only** check box is checked and click **OK**.



2) Update the form's security options to grant the form full trust when hosted in Forms Services.

- Click the **Tools -> Form Options** item in the menu bar.
- In the **Form Options** dialog, select **Security and Trust** in the **Category** list box.
- Uncheck **Automatically determine security level** and select **Full Trust**.
- Click **OK** to close the dialog.



3) Add a secondary data source that will allow the hosting environment to provide extra data to the form. This is required to use the **Contact Selector** in the next step.

- In **Visual Studio 2008**, add a new file named **Context.xml** to the **Forms** folder.

*Right click on the **Forms** folder in the **Solution Explorer** and select **Add -> New Item**.*

*Select the **Data** category in the left side of the screen and then select **Xml File** in the **Template** list.*

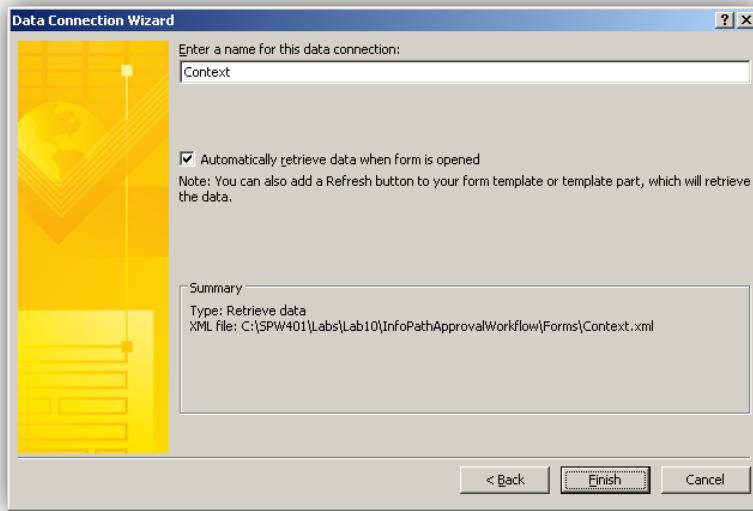
*Enter a name of **Context.xml** and click **Add**.*

- Enter the following XML into the new **Context.xml** file.

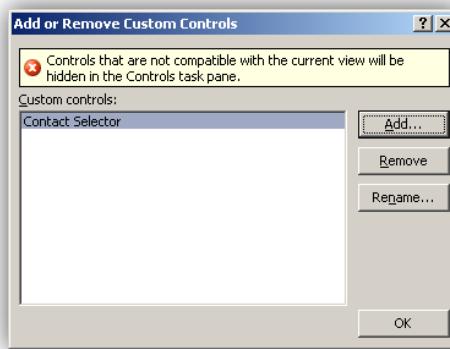
```
<Context
  isStartWorkflow="true"
  isRunAtServer="false"
  provideAllFields="true"
  siteUrl="" />
```

- In InfoPath, click **Tools -> Data Connections** in the menu bar.
- In the **Data Connections** dialog, click the **Add** button.
- Select the **Create a new connection to** option and then select **Receive data** and click **Next**.
- On the next page, choose to receive data from an **XML document**.
- Browse to the **Context.xml** file just created and click **Next**.
- On the next page, select to include the file as a resource and click **Next**.

- On the final page, click **Finish** to create the data source.
- In the **Data Connections** dialog, click **Close** to close the window.



- 4) Add a **Contact Selector** ActiveX control to the controls list for adding to the form later.
- Locate the **Controls** task pane on the right hand side of the page.
*If it is not visible, click the drop down menu at the top of the task pane and select **Controls**.*
 - Click **Add or Remove Custom Controls** to add the **Contact Selector** control.
 - In the **Add or Remove Custom Controls** dialog, click the **Add** button.
 - In the first page of the wizard, choose **ActiveX Control** and click **Next**.
 - Select **Contact Selector** in the control list and click **Next**.
 - Choose to not include a cab file and click **Next**.
 - Select a binding property of **Value** and click **Next**.
 - In the **Field or group type** drop down list, choose **Field or Group (any data type)** and click **Finish**.
 - On the summary page, click **Close**.
 - Now the **Contact Selector** should be visible in the **Add or Remove Custom Controls** dialog. Close the dialog when you are done.



- 5) Add a **Contact Selector** to the form and bind it to the data source.

- Drag a **Contact Selector** from the **Controls** task pane onto the design surface.
- Switch to the **Data Source** view using the drop down menu at the top of the task pane.

*Notice the **Contact Selector** control has already created a new group named **group1**.*



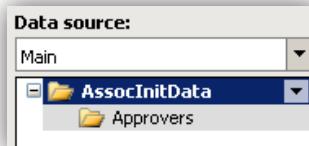
- Rename the **myFields** and **group1** groups to **AssocInitData** and **Approvers**.

*Right click **myFields** and select **Properties**.*

*In the **Field or Group Properties** dialog, change the name to **AssocInitData**.*

*Click **OK** to close the dialog.*

*Repeat to change **group1** to **Approvers**.*



- Add a **Person** repeating group as a child of the **Approvers** group.

*Right click the **Approvers** group and select **Add**.*

*In the **Add Field or Group** dialog, enter a name of **Person**.*

*Change the **Type** to **Group**.*

*Check the **Repeating** checkbox.*

*Click **OK** to create the new group.*



- Add three more string fields named **DisplayName**, **AccountId**, **AccountType** as children of the **Person** group.

*Right click the **Person** group and click **Add**.*

*In the **Add Field or Group** dialog, enter a name of **DisplayName**.*

*Verify the **Data type** is **string**.*

*Click **OK** to create the new field.*

*Repeat the above steps for **AccountId** and **AccountType**.*



6) Add a **Text Box** control to the form to be used to enter any instructions to the approver.

- Add an **Instructions** field to the **AssocInitData** group.

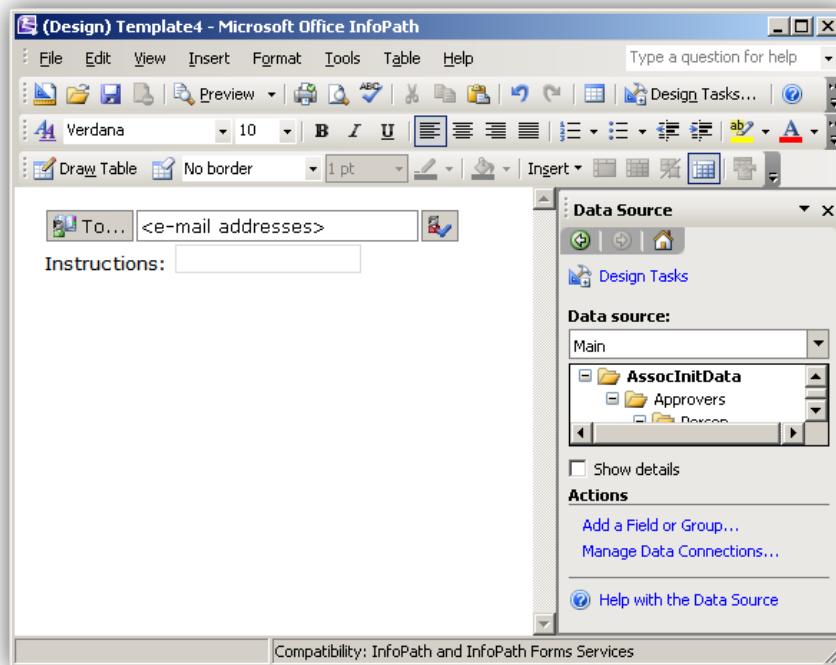
*In the **Data Source** task pane, right click the **AssocInitData** group and select **Add**.*

*In the **Add Field or Group** dialog, enter a name of **Instructions**.*

*Verify the **Data type** is **string**.*

*Click **OK** to create the new field.*

- Drag the **Instructions** field onto the design canvas to create a new text box.



7) Add two buttons to the page for the **OK** and **Cancel** operations.

- Switch to the **Controls** task pane and drag two button controls onto the design canvas.
- Change the name of one button to **OK**.

*Right click the button and select **Button Properties**.*

*In the **Button Properties** dialog, change the **Label** value to **OK**.*

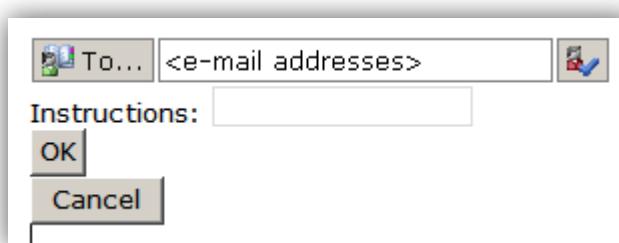
*Click **OK** to close the dialog.*

- Change the name of the other button to **Cancel**.

*Right click the button and select **Button Properties**.*

*In the **Button Properties** dialog, change the **Label** value to **Cancel**.*

*Click **OK** to close the dialog.*



8) Define rules for the **Cancel** button that will cause it to close the form.

- Open the **Rules** dialog to manage the rules that will execute when the button is clicked.

*Right click the cancel button and select **Button Properties**.*

*Click the **Rules** button to open the **Rules** dialog.*

- Add a new rule named **Close Form** that will be executed when the button is clicked.

*Click the **Add** button to add a new rule.*

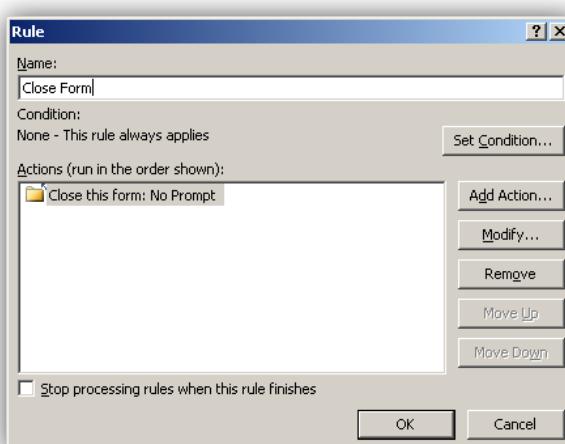
*Change the **Name** property in the **Rule** dialog to **Close Form**.*

- Add a close form action that will close the current form.

*Click **Add Action** to open the **Action** dialog.*

*In the **Action** dialog, select an action of **Close the form***

*Click **OK** to close the dialog.*



- Close all of the open dialogs.

9) Define rules for the **OK** button that will cause it to submit and then close the form.

- Open the **Rules** dialog to manage the rules that will execute when the button is clicked.

*Right click the cancel button and select **Button Properties**.*

*Click the **Rules** button to open the Rules dialog.*

- Add a new rule named **Submit and Close Form** that will be executed when the button is clicked.

*Click the **Add** button to add a new rule.*

*Change the **Name** property in the Rule dialog to **Submit and Close Form**.*

- Add a submit form action that will submit the data to Forms Services.

*Click **Add Action** to open the **Action** dialog.*

*In the **Action** dialog, select an action of **Submit using a data connection**.*

*Click the **Add** button to add a new submission data connection.*

*In the **Data Connection Wizard** create a new **Submit** data connection and click **Next**.*

*Select **To the hosting environment** and click **Next**.*

*Keep the default name of **Submit** and click **Finish**.*

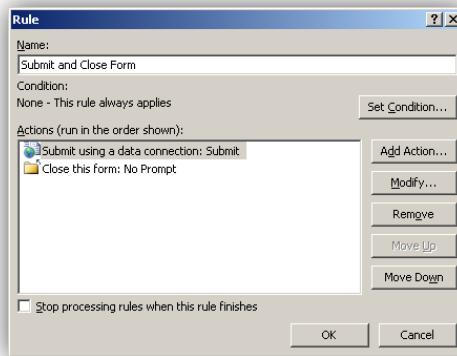
*Back in the **Action** dialog, verify the data connection **Submit** is selected and click **OK** to close the dialog.*

- Add a close form action that will close the current form.

*Click **Add Action** to open the **Action** dialog.*

*In the **Action** dialog, select an action of **Close the form***

*Click **OK** to close the dialog.*



- Close all of the open dialogs.

10) Update the layout of the form to look nicer. Add a title of **Association Form** that is size 14pt and bold and place the controls in a layout table.

- Using InfoPath like a word processor, add the text **Association Form** to the top of the page. Use a font size of 14pt and bold.
- Create a layout grid of 2x3 and place the approver and instructions controls in the top two rows. Adjust the height of the Instructions text box so it can contain multiple lines.
- Place the buttons in the lower right cell of the layout table and right justify them.

- 11) Save the new form template into the **Forms** folder of the Visual Studio Project.
 - Click the **Save** toolbar button.
 - Save the form in **C:\Labs\Lab10\StarterFiles\InfoPathApprovalWorkflow\Forms** with a name of **AssocInitForm.xsn** and close InfoPath.

- 12) Extract the schema file from the new InfoPath form.
 - Locate the **AssocInitForm.xsn** file in the **Windows Explorer** and rename it **AssocInitForm.xsn.cab**.
 - Open the new cab file and copy the **myschema.xsd** file contained within it into the root folder of the **InfoPathApprovalWorkflow** project.
 - Change the name of the **AssocInitForm.xsn.cab** back to **AssocInitForm.xsn**.

- 13) Update the **AssocInitData** class by using the **xsd.exe** tool to regenerate the **AssocInitData** class.
 - In the **InfoPathApprovalWorkflow** folder, rename the **myschema.xsd** to **AssocInitData.xsd**.
 - Open a **Visual Studio 2008 Command Prompt** in the **InfoPathApprovalWorkflow** folder.

*Click the **Start Button** and click **All Programs** -> **Microsoft Visual Studio 2008** -> **Visual Studio Tools** -> **Visual Studio 2008 Command Prompt**.*

*In the command prompt, navigate to **C:\Labs\Lab10\StarterFiles\InfoPathApprovalWorkflow**.*
 - Execute **xsd.exe** to regenerate the class that will serialize xml to the format defined in the InfoPath schema.

xsd.exe AssocInitData.xsd /c /n:InfoPathApprovalWorkflow
 - Open the generated **AssocInitData.cs** in **Visual Studio 2008**.

*Comment out the **anyAttrField** field and property if it exists*

Exercise 2: InfoPath Instantiation Form using Views

- 1) Add a new view named **Instantiation View** to the **InfoPath** form
 - Design **AssocInitForm.xsn** in InfoPath.
 - Select the **Views** task pane using the drop down list at the top of the task pages window.
 - Rename the default view to **Association View**.

*Select the default view and click the **View Properties** button.*

*Change the **View name** to **Association View** and click **OK**.*
 - Add a new view named **Instantiation View** to the form.

*Click the **Add a New View** link on the **Views** task pane.*

*In the **Add View** dialog, enter the name of **Instantiation View** and click **OK**.*
 - Switch back to the **Association View** by clicking it in the list of **Views**.



- 2) Copy all of the controls from the **Association View** to the **Instantiation View** and update them to display an instantiation view

- Copy and paste the contents of the **Association View** into the **Instantiation View**.

*Select the entire content of the **Association View** in the designer and click **Edit -> Copy**.*

*Switch to the **Instantiation View** using the **Views** task pane.*

*Click **Edit -> Paste** to paste the contents into the new view.*

- Change the header from **Association Form** to **Instantiation Form**.

- Change the text of the **OK** button to **Start**.

*Right click the **OK** button and click **Button Properties**.*

*In the **Button Properties** dialog, change the name from **OK** to **Start**.*

*Click **OK** to close the dialog.*

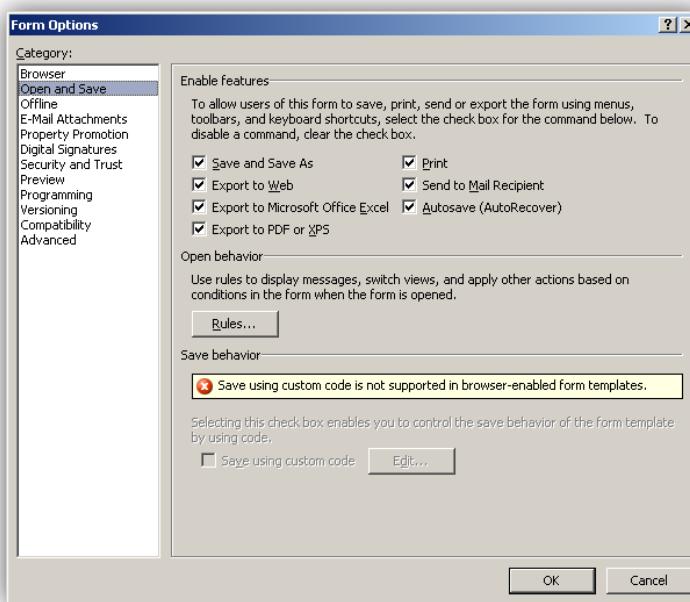


- 3) Add form load rules that will choose the appropriate view based on the **isStartWorkflow** field in the Context data source.

- Using the **Form Options** dialog, open the **Open** behavior rules for the form.

*Click **Tools -> Form Options** in the menu bar.*

*Switch to the **Open and Save** category and click the **Rules** button.*



- Add rule to switch to association form if **isStartWorkflow == "false"**.

*Click the **Add** button to add a new rule.*

*Set the **Name** of the new rule to **Is Not Starting Workflow**.*

*Click the **Set Condition** button to set the conditions under which the actions will execute.*

*On the left hand side drop down list, click **Select a Field or Group**.*

*In the **Select a Field or Group** dialog, choose the **Context** data source and the **isStartWorkflow** field and click **OK**.*

*In the center drop down list, verify **is equal to** is selected.*

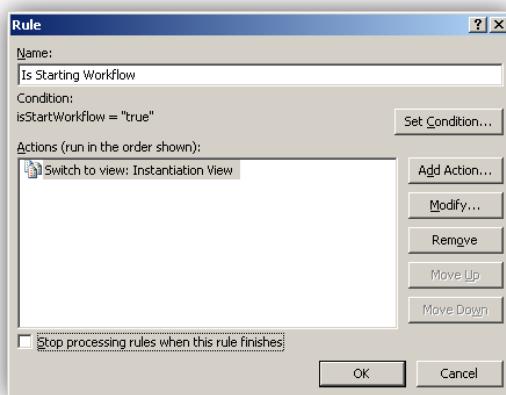
*In the right drop down list, click **type text**, enter the text **false**, and click **OK**.*

*Click **OK** again to close the condition.*

*Click **Add Action** to add a new switch view action.*

*Select an action of **Switch Views** and a view of **Association View**.*

*Click **OK** to close the dialogs back to the **Rules** dialog.*



- Add rule to switch to instantiation form if **isStartWorkflow == "true"**.

*Click the **Add** button to add a new rule.*

*Set the **Name** of the new rule to **Is Starting Workflow**.*

*Click the **Set Condition** button to set the conditions under which the actions will execute.*

*On the left hand side drop down list, click **Select a Field or Group**.*

*In the **Select a Field or Group** dialog, choose the **Context** data source and the **isStartWorkflow** field and click **OK**.*

*In the center drop down list, verify **is equal to** is selected.*

*In the right drop down list, click **type text** and enter the text **true** and click **OK**.*

*Click **OK** again to close the condition.*

*Click **Add Action** to add a new switch view action.*

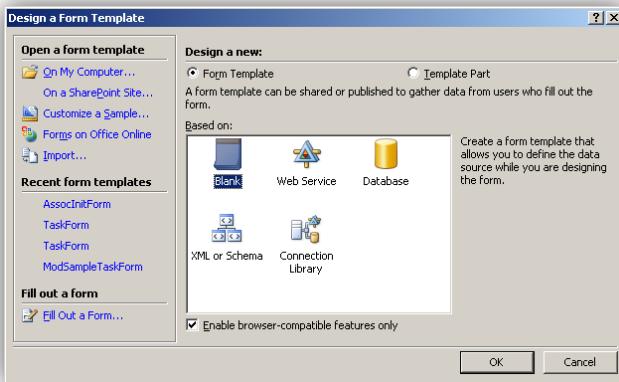
*Select an action of **Switch Views** and a view of **Instantiation View**.*

*Click **OK** to close the dialogs back to the designer.*

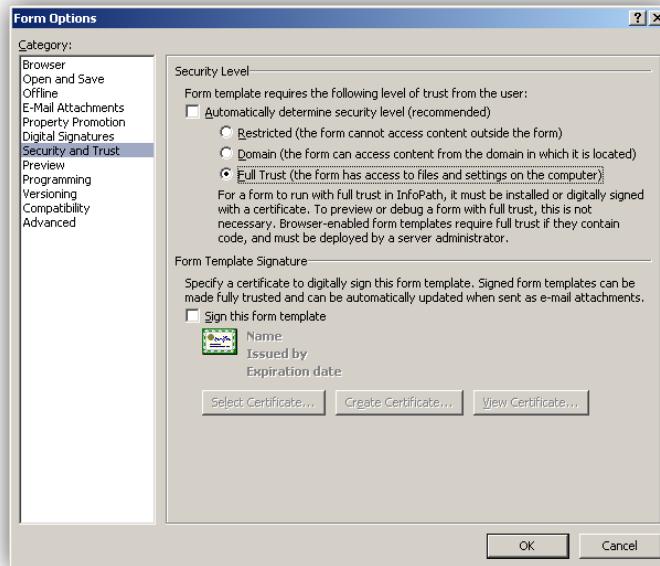
Exercise 3: InfoPath Task Forms

1) Create a new InfoPath form named **TaskForm.xsn**.

- Open **InfoPath** by clicking the **Start Button** and selecting **All Programs -> Microsoft Office -> Microsoft Office InfoPath 2007**.
- In the **Getting Started** window, click the **Design a Form Template** link in the **Design a form** section.
- In the **Design a Form Template** window, select a **Blank form** template and make sure the **Enable browser-compatible features only** check box is checked and click **OK**.



- 2) Update the form's security options so the form will be granted full trust when hosted in Forms Services.
 - Click the **Tools -> Form Options** item in the menu bar.
 - In the **Form Options** dialog, select **Security and Trust** in the **Category** list box.
 - Uncheck **Automatically determine security level** and select **Full Trust**.
 - Click **OK** to close the dialog.



- 3) Add a secondary data source that will allow the form to access fields in the current task.
 - In **Visual Studio 2008**, add a new file named **ItemMetadata.xml** to the **Forms** folder.

*Right click on the **Forms** folder in the **Solution Explorer** and select **Add -> New Item**. Select the **Data** category in the left side of the screen and then select **Xml File** in the **Template** list. Enter a name of **ItemMetadata.xml** and click **Add**.*
 - Enter the following XML into the new **ItemMetadata.xml** file.

```
<z:row xmlns:z="#RowsetSchema"
       ows_Instructions="" />
```

- Add a new data connection using the **ItemMetadata.xml** file.

*In InfoPath, click **Tools -> Data Connections** in the menu bar.*

*In the **Data Connections** dialog, click the **Add** button.*

*Select the **Create a new connection to** option and then select **Receive data** and click **Next**.*

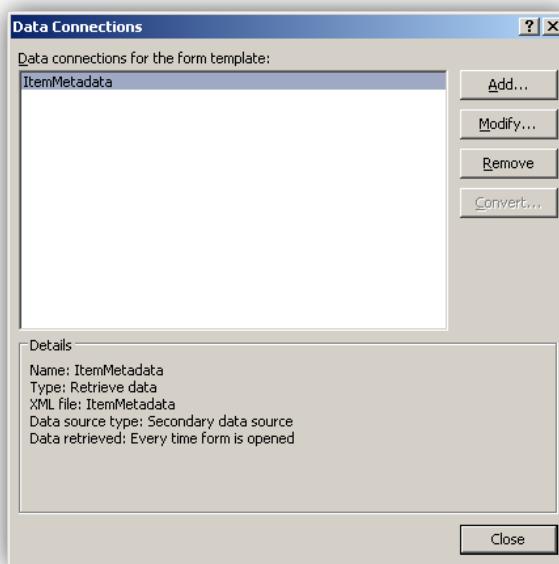
*On the next page, choose to receive data from an **XML document**.*

*Browse to the **ItemMetadata.xml** file just created and click **Next**.*

*On the next page select to include the file as a resource and click **Next**.*

*On the final page, click **Finish** to create the data source.*

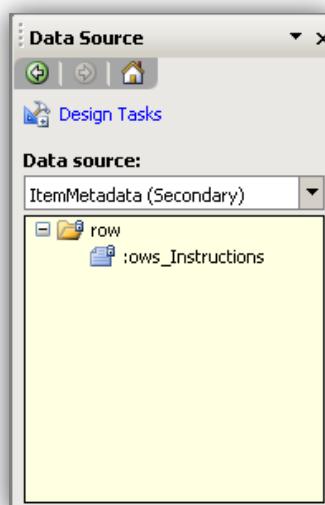
*In the **Data Connections** dialog, click **Close** to close the window.*



- Using the new data source, add a new control using the **ows_Instructions** field.

*In the **Data Source** task pane, use the **Data source** drop down to change to the **ItemMetadata** data source.*

*Drag the **ows_Instructions** field onto the designer to create an **Instructions** control.*



- 4) Define the output data that will be transmitted back to the workflow when the task is submitted.

- Add a new **Comments** field of type string to the primary data source.

*In the **Data Source** task pane, right click the **myFields** group and select **Add**.*

*In the **Add Field or Group** dialog, enter a name of **Comments**.*

*Verify the **Data type** is **string**.*

*Click **OK** to create the new field.*

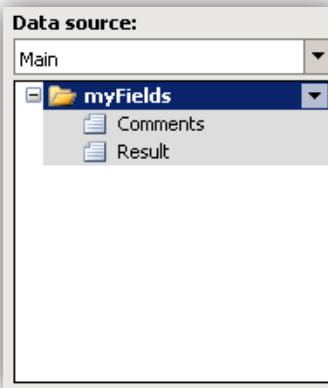
- Add a new **Result** field of type string to the primary data source.

*In the **Data Source** task pane, right click the **myFields** group and select **Add**.*

*In the **Add Field or Group** dialog, enter a name of **Result**.*

*Verify the **Data type** is **string**.*

*Click **OK** to create the new field.*



- Drag the **Comments** field onto the design canvas to create a new text box.

- 5) Add three buttons to the page for the **Approve**, **Reject**, and **Cancel** operations.

- Switch to the **Controls** task pane and drag two button controls onto the design canvas.

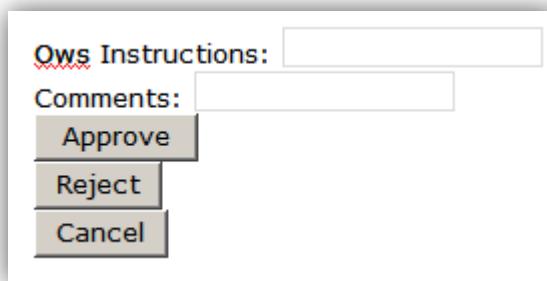
- Change the name of one button to **Approve**.

*Right click the button and select **Button Properties**.*

*In the **Button Properties** dialog, change the **Label** value to **Approve**.*

*Click **OK** to close the dialog.*

- Repeat the previous step for the **Reject** and **Cancel** buttons.



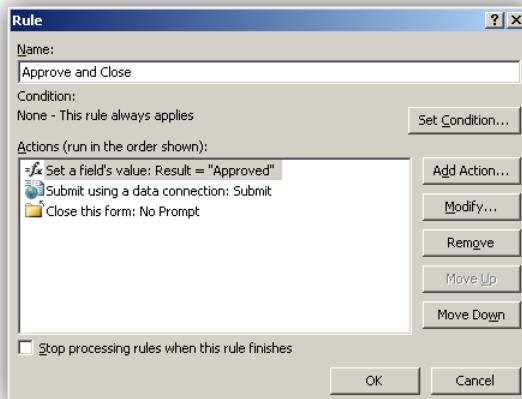
- 6) Define rules for the **Cancel** button that will cause it to close the form.

- Open the **Rules** dialog to manage the rules that will execute when the button is clicked.

*Right click the **cancel** button and select **Button Properties**.*

*Click the **Rules** button to open the **Rules** dialog.*

- Add a new rule named **Close Form** that will be executed when the button is clicked.
*Click the **Add** button to add a new rule.*
*Change the **Name** property in the **Rule** dialog to **Close Form**.*
 - Add a close form action that will close the current form.
*Click **Add Action** to open the **Action** dialog.*
*In the **Action** dialog, select an action of **Close the form***
*Click **OK** to close the dialog.*
Close all of the open dialogs.
- 7) Define rules for the **Approve** button that will cause it to submit and then close the form.
- Open the **Rules** dialog to manage the rules that will execute when the button is clicked.
*Right click the cancel button and select **Button Properties**.*
*Click the **Rules** button to open the **Rules** dialog.*
 - Add a new rule named **Approve and Close Form** that will be executed when the button is clicked.
*Click the **Add** button to add a new rule.*
*Change the **Name** property in the **Rule** dialog to **Approve and Close Form**.*
 - Add a **Set a field's value** action that will set the **Result** field to **Approved**.
*Click **Add Action** to open the **Action** dialog.*
*In the **Action** dialog, select an action of **Set a field's value**.*
*Click the button following the **Field** text box and choose the **Result** field.*
*In the **Value** text box, enter **Approved** and click **OK** to create the action.*
*Back in the **Action** dialog, verify the data connection **Submit** is selected and click **OK** to close the dialog.*
 - Add a submit form action that will submit the data to Forms Services.
*Click **Add Action** to open the **Action** dialog.*
*In the **Action** dialog, select an action of **Submit using a data connection**.*
*Click the **Add** button to add a new submission data connection.*
*In the **Data Connection Wizard** create a new **Submit** data connection and click **Next**.*
*Select **To the hosting environment** and click **Next**.*
*Keep the default name of **Submit** and click **Finish**.*
*Back in the **Action** dialog, verify the data connection **Submit** is selected and click **OK** to close the dialog.*
 - Add a close form action that will close the current form.
*Click **Add Action** to open the **Action** dialog.*
*In the **Action** dialog, select an action of **Close the form***
*Click **OK** to close the dialog.*

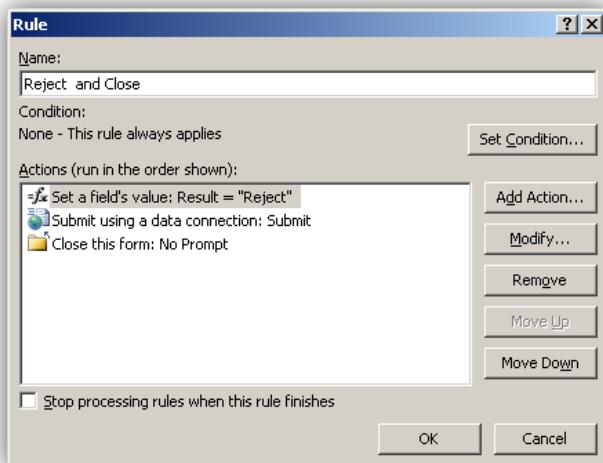


8) Repeat the previous step to define the rules for the **Rejected** button.

- Perform the previous steps with the following changes:

*Use the rule name of **Reject and Close Form**.*

*Set the **Result** field value to **Rejected**.*



9) Update the layout of the form to look nicer. Add a title of **Task Form** that is size 14pt, bold, and place the controls in a layout table.

- Using InfoPath like a word processor, add the text **Association Form** to the top of the page. Use a font size of 14pt and bold.
- Create a layout grid of 2x3 and place the instructions and comments controls in the top two rows. Adjust the height of the text boxes so they can contain multiple lines.
- Place the buttons in the lower right cell of the layout table and right justify them.

10) Save the new form template into the **Forms** folder of the Visual Studio Project.

- Click the **Save** toolbar button.
- Save the form in **C:\Labs\Lab10\StarterFiles\InfoPathApprovalWorkflow\Forms** with a name of **TaskForm.xsn**.

11) Update the workflow to support providing values to the task form and receiving results back.

- In **CreateApprovalTask_Invoking**, add an extended property named Instructions getting its value from the **AssocInitData** property.

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

*Locate the **CreateApprovalTask_Invoking** and add an **Instructions** property to the task properties.*

*Get the value of the extended property from the **AssocInitData** property of the workflow.*

```
createApprovalTask.TaskProperties.ExtendedProperties.Add(  
    "Instructions", InitData.Instructions);
```

- In **ApprovalTaskChanged_Invoked** retrieve the extended property with the Guid id of **{52578fc3-1f01-4f4d-b016-94ccbcf428cf}** representing the **Comments** column.

*Locate the **ApprovalTaskChanged_Invoked** method.*

Add the line to retrieve the comments extended property.

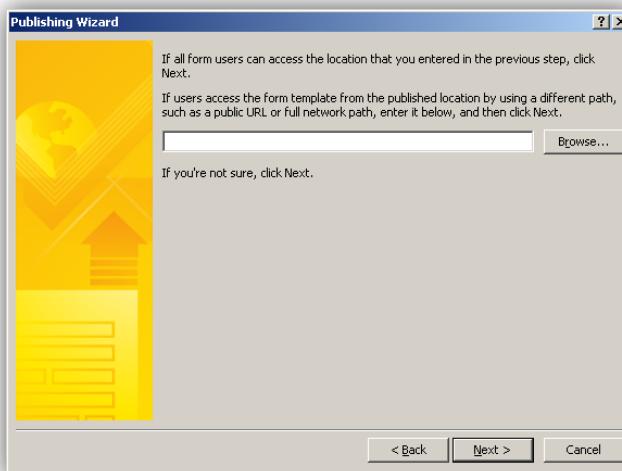
```
this.ApprovalTaskComments = args.afterProperties.ExtendedProperties[  
    new Guid("52578fc3-1f01-4f4d-b016-94ccbcf428cf")] as string;
```

- In **ApprovalTaskChanged_Invoked** retrieve the **Result** expended property and convert it into a **TaskResult** enum value.

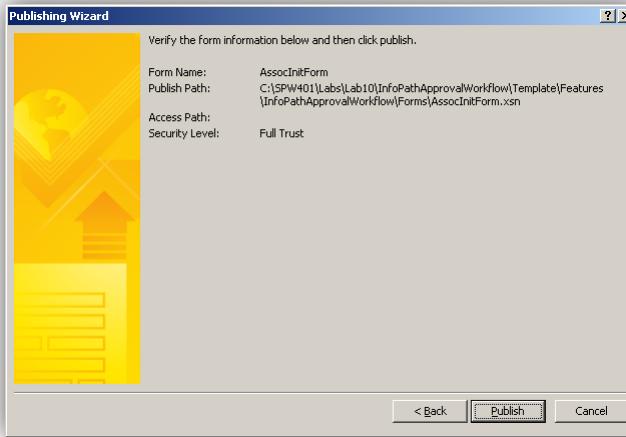
```
string result = args.afterProperties.ExtendedProperties["Result"] as string;  
this.ApprovalTaskResult = (TaskResult)Enum.Parse(typeof(TaskResult), result);
```

Exercise 4: Deploying InfoPath Workflow Forms

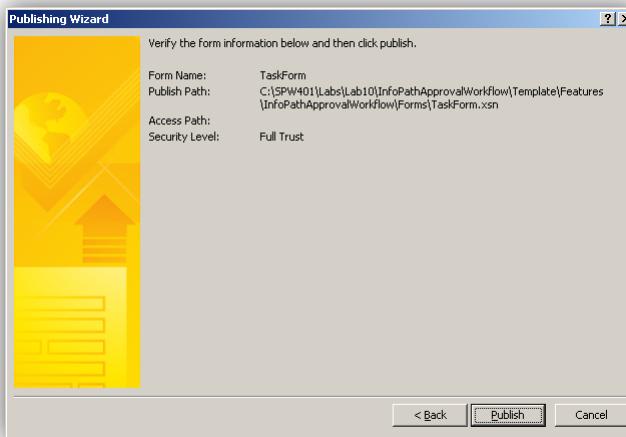
- 1) Publish the **AssocInitForm.xsn** form to the **Template\Features\InfoPathApprovalWorkflow\Forms** folder in the project.
 - Design the **Forms\AssocInitForm.xsn** form template in InfoPath.
 - Publish the form by clicking **File -> Publish** in the menu bar.
 - In the **Publishing Wizard**, choose to publish to a network location and click **Next**.
 - Publish the document to the **C:\Labs\Lab10\StarterFiles\InfoPathApprovalWorkflow\Template\Features\InfoPathApprovalWorkflow\Forms\AssocInitForm.xsn** location.
 - When asked for the public URL make sure the text box is empty and click **Next**.



- On the final page of the wizard, verify the results and click **Publish** and then **Close**.



- 2) Using the process in the previous step, publish the **Task.xsn** form to the **Template\Features\InfoPathApprovalWorkflow\Forms** folder in the project.



- 3) Update the **feature.xml** file to indicate the location of the InfoPath forms and include a receiver assembly that will install those forms to Forms Services.
- Open the **Feature.xml** file in the **Template\Features\InfoPathApprovalWorkflow** folder.
 - Add a **ReceiverAssembly** and **ReceiverClass** to the **Feature** element.

```
ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0,
Culture=neutral, PublicKeyToken=71e9bcce111e9429c"
ReceiverClass="Microsoft.Office.workflow.Feature.WorkflowFeatureReceiver"
```

- Add a **Property** element to the **Properties** element that will set a property of **RegisterForms** to the **Forms*.xsn** value.

```
<Properties>
<Property Key="GloballyAvailable" value="true" />
<Property Key="RegisterForms" value="Forms*.xsn" />
</Properties>
```

- 4) Update the **workflow.xml** file to define new form URLs and content types for tasks.
- Open the **Workflow.xml** file in the **Template\Features\InfoPathApprovalWorkflow** folder.
 - Add an **AssociationUrl** and **InstantiationUrl** to the **Workflow** element.

These values are special Forms Services hosting pages.

```
AssociationUrl="/_layouts/CstwrkfLIP.aspx"
InstantiationUrl="/_layouts/IniwrkfLIP.aspx">
```

- Add a **TaskListContentTypeld** attribute to the **Workflow** element.

*This is a special **Forms Services** content type with the appropriate view and edit forms already defined.*

```
TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"
```

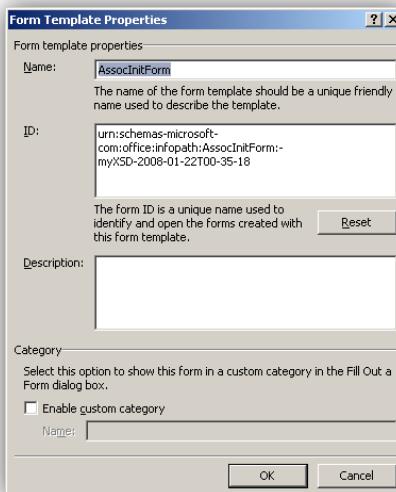
- 5) Add form registration URNs that are used by the special Forms Services pages to identify the InfoPath form to load.

- Lookup the URN of the **AssocInitForm.xsn** using the form's **Properties** window.

*Right click the form in **Windows Explorer** and select **Design**.*

*When the form is loaded, click **File > Properties**.*

*Copy the **ID** property from the dialog.*



- Use the URN for the form to populate the Association and Instantiation Form URNs in the workflow metadata.

*Open **Workflow.xml** and add the following metadata into the form.*

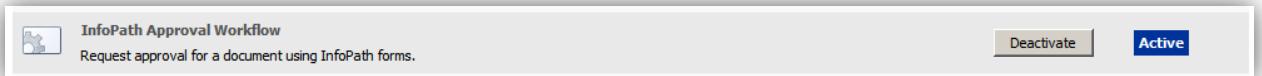
*Replace the values in **Association_FormURN** and **Instantiation_FormURN** with the URN you copied from InfoPath.*

```
<MetaData>
  <Association_FormURN>urn:schemas-microsoft-com:office:infopath:AssocInitForm:-myXSD-2008-01-22T00-35-18</Association_FormURN>
  <Instantiation_FormURN>urn:schemas-microsoft-com:office:infopath:AssocInitForm:-myXSD-2008-01-22T00-35-18</Instantiation_FormURN>
  <Task0_FormURN>urn:schemas-microsoft-com:office:infopath:TaskForm:-myXSD-2008-01-22T03-21-33</Task0_FormURN>
</MetaData>
```

- Repeat the previous steps to populate **Task0_FormURN** with the URN of **TaskForm.xsn**.

- 6) Rebuild the project to force redeployment of the forms.
 - Right click on the project in the **Solution Explorer** and click **Rebuild**.

- 7) Activate the new **InfoPathApprovalWorkflow** 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 **InfoPath Approval Workflow** feature.



- 8) Create an association between the **Shared Documents** document library and the new **InfoPath Approval Workflow** 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 **InfoPath Approval Workflow** workflow template and a name of **InfoPath Approval**.

Use the default values for both list and startup options.

The screenshot shows the "Add a Workflow" dialog for the "Shared Documents" library. The title bar says "Add a Workflow: Shared Documents". The main area has a sub-header "Use this page to set up a workflow for this document library." Below it, there's a "Workflow" section with a dropdown menu showing "Hello World Workflow", "InfoPath Approval Workflow" (which is selected), "Three-state", and "Wss Custom Activities Workflow". To the right, there's a "Description" field containing "A simple workflow example that demonstrates the use of InfoPath forms.". Below the dropdown, there's a "Name" section with a text input field containing "InfoPath Approval".

- In the custom association form, enter Administrator as the approver and add "Please approve this document." as the instructions.

The screenshot shows the "Association Form" dialog. It has fields for "Approver:" (with "LitwareInc Administrator" selected), "Instructions:" (containing "Please approve this document."), and buttons for "OK" and "Cancel".

9) Run the **InfoPath Approval** workflow on a document.

- Navigate to the Shared Documents document library and hover over the new document and select **Workflows** from the drop down menu.
- In the workflows page, click the **InfoPath Approval** 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.

Instantiation Form

Approver:

Instructions:

Start **Cancel**

10) Approve the document using the Custom Task Form.

- Click the task's title in the workflow status page.
- Notice that the **Instructions** text box defaults to the value set in the **Instantiation Form**.
- Click the **Approve** button to approve the task.

Task Forms

Instructions:

Comments:

Approve **Reject** **Cancel**

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

Demo Site > Shared Documents > Workflow Status

Workflow Status: InfoPath Approval

Workflow Information

Initiator: LitwareInc Administrator	Document: Document to Archive
Started: 1/27/2008 1:47 PM	Status: Completed
Last run: 1/27/2008 1:48 PM	

Tasks

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

Assigned To	Title	Due Date	Status	Outcome
LitwareInc Administrator	Approve document. ! NEW		Completed	

Lab 11: Developing Custom Activities for a SharePoint Workflow Template

Lab Overview: Often there are pieces of code used many times in a solution. In a traditional .NET application, you encapsulate them as a class or a function. In Windows Workflow, you encapsulate them as Activities.

In this lab, you will be creating two custom activities that will provide database access. To do this you will need to create activities as well as validators and designers to ensure the activities are used properly.

Exercise 1: Building a simple Activity

- 1) Create a new **Workflow Activity Library** project to the solution named **SqlDatabaseActivities**.
 - Right click the solution in the **Solution Explorer** and click **Add -> New Project**.
 - Select the **Workflow** option in the **Project types** list box and select **Workflow Activity Library** in the Templates list.
 - Enter a name of **SqlDatabaseActivities** and click **OK**.
- 2) Create a new **Activity** named **SqlScalarQueryActivity** in the new project.
 - Right click on the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Activity**.
 - Enter a name of **SqlScalarQueryActivity.cs** and click **OK**.
 - Modify the class definition to change the base class to **Activity** instead of **SequenceActivity**.

```
public partial class SqlScalarQueryActivity : Activity
{
}
```

- 3) Define the dependency properties named **Parameters** and **Result**.
 - Define the static **DependencyProperty** object for **Parameters**.

*Create a new **public static DependencyProperty** object named **ParametersProperty**. Set its value using the **DependencyProperty.Register** static method. Define its name as **Parameters**, its propertyType as **typeof(SqlParameter[])**, and its ownerType of **typeof(SqlScalarQueryActivity)**.*

```
public static DependencyProperty ParametersProperty =
    DependencyProperty.Register("Parameters",
        typeof(SqlParameter[]), typeof(SqlScalarQueryActivity));
```
 - Define a helper property that wraps access to the new dependency property.

*Define a string **Parameters** property with a get and set section. Implement get using the **base.GetValue** method and casting the result to a string. Implement set using the **base.SetValue** method. Add a **Category** attribute to the property with a value of **Parameters** (this places the property in the **Parameters** group in the designer).*

```

[Category("Parameters")]
public SqlParameter[] Parameters
{
    get { return base.GetValue(ParametersProperty) as SqlParameter[]; }
    set { base.SetValue(ParametersProperty, value); }
}

```

- Repeat the process to create a **Result** dependency property and wrapping property definition. The only difference is the type. Make **Result** an object.

```

public static DependencyProperty ResultProperty =
    DependencyProperty.Register("Result",
        typeof(System.Object), typeof(SqlScalarQueryActivity));

[Category("Parameters")]
public object Result
{
    get { return base.GetValue(ResultProperty); }
    set { base.SetValue(ResultProperty, value); }
}

```

- Define the **Query** dependency property. Make it a **MetaData** property so it can only be modified at design time.

- Define the static **DependencyProperty** object for **Query**.

Create the new public static DependencyProperty just like before.

Add a fourth parameter to the call to DependencyProperty.Register that is a PropertyMetadata object with an options parameter of DependencyPropertyOptions.Metadata.

```

public static DependencyProperty QueryProperty =
    DependencyProperty.Register("Query",
        typeof(System.String), typeof(SqlScalarQueryActivity),
        new PropertyMetadata(DependencyPropertyOptions.Metadata));

```

- Define the wrapper property for **Query** the same as any other dependency property wrapper property.

```

[Category("Parameters")]
public string Query
{
    get { return base.GetValue(QueryProperty) as string; }
    set { base.SetValue(QueryProperty, value); }
}

```

- Add two events to the activity, **Invoking** and **Invoked**, which can be bound using the same mechanism as the properties.

- Define the static **DependencyProperty** object for **Invoking**.

Create a new public static DependencyProperty object named InvokingEvent.

Set its value using the DependencyProperty.Register static method.

Define its name as Invoked, its propertyType as typeof(EventHandler), and its ownerType of typeof(SqlScalarQueryActivity).

```

public static DependencyProperty InvokingEvent =

```

```
DependencyProperty.Register("Invoking",
    typeof(EventHandler), typeof(SqlScalarQueryActivity));
```

- Define the wrapper event for **Invoking** that will provide a standard mechanism for attaching and removing event handlers.

*Define an event named **Invoked** of type **EventHandler** with an add and remove seciton.*

*Implement **add** using the **base.AddHandler** method.*

*Implement **remove** using the **base.RemoveHandler** method.*

*Add a **Category** attribute to the property with a value of **Handlers**.*

```
[Category("Handlers")]
public event EventHandler Invoking
{
    add { base.AddHandler(InvokingEvent, value); }
    remove { base.RemoveHandler(InvokingEvent, value); }
}
```

- Define a second dependency property and event for the **Invoked** event.

```
public static DependencyProperty InvokedEvent =
    DependencyProperty.Register("Invoked",
        typeof(EventHandler), typeof(SqlScalarQueryActivity));

[Category("Handlers")]
public event EventHandler Invoked
{
    add { base.AddHandler(InvokedEvent, value); }
    remove { base.RemoveHandler(InvokedEvent, value); }
}
```

- Implement the **Execute** method so the query is made and the result is stored in the **Result** property.

- Override the **Execute** method of the **Activity** class.

```
protected override ActivityExecutionStatus Execute(ActivityExecutionContext
executionContext)
{
}
```

- Raise the **Invoking** event using the **Activity.RaiseEvent** method.

```
// raise the invoking event
this.RaiseEvent(InvokingEvent, this, EventArgs.Empty);
```

- Open the **SqlConnection** using a hardcoded connection string (this will be changed later)

```
using (SqlConnection connection = new SqlConnection("Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\\Labs\\Files\\Litware.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True"))
{
    connection.Open();
}
```

- While the connection is open, create and execute the new **SqlCommand** based on the **Query** and **Parameters** dependency properties.

```
// create the sql command
SqlCommand command = new SqlCommand(this.Query, connection);
if (this.Parameters != null)
    foreach (SqlParameter parameter in this.Parameters)
        command.Parameters.Add((parameter as ICloneable).Clone() as
SqlParameter);
this.Result = command.ExecuteScalar();
```

- After the **SqlConnection**'s using scope, raise the **Invoked** event to indicate the activity is completed.

```
// raise the Invoked event
this.RaiseEvent(InvokedEvent, this, EventArgs.Empty);
```

- Return an **ActivityExecutionStatus** of **Closed** to let the executor know this activity is completed and it should move on to the next activity.

```
// return the closed activity status
return ActivityExecutionStatus.Closed;
```

7) Build the **SqlDatabaseActivities** project.

- Right click the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Build**.

8) Create a new **Sequential Workflow Console Application** project to the solution named **SqlDatabaseActivitiesConsole**.

- Click **File -> New Project** in the menu bar.
- Select the **Workflow** option in the **Project types** list box and select **Sequential Workflow Console Application** in the **Templates** list.
- Select **Add to Solution** in the **Solution** drop down list.
- Enter a name of **SqlDatabaseActivitiesConsole** and click **OK**.

9) Add a new **SqlScalarQueryActivity** to **Workflow1** and use it to make a database query.

- View **Workflow1.cs** in design mode by right clicking it in the **Solution Explorer** and selecting **View Designer**.
- Drag a new **SqlScalarQueryActivity** onto the canvas and name it **LookupComments**.

*Drag a **SqlScalarQueryActivity** onto the canvas.*

*In the properties pane, set the **Name** property to **LookupComments**.*

- Define the query as a lookup into the **UserProfiles** table.

*In the properties pane, set the **Query** to the following query.*

```
SELECT Comments FROM UserProfiles WHERE Email = @email
```

- Define an **Invoking** method that will be used to define the **@email** parameter.

*In the properties pane, set the **Invoking** property to **LookupComments_Invoking** and press **Enter**.*

*In the **LookupComments_Invoking** method, cast the sender parameter to a **SqlScalarActivityQuery**.*

*Set the activities **Parameters** property to an array containing a single **SqlParameter** named **@email** with a value of Administrator@litwareinc.com.*

```
private void LookupComments_Invoking(object sender, EventArgs e)
```

```

{
    SqlScalarQueryActivity activity = sender as SqlScalarQueryActivity;

    activity.Parameters = new SqlParameter[]
    {
        new SqlParameter("@email", "Administrator@litwareinc.com")
    };
}

```

- Define an **Invoked** event handler that will be used to display the activity's Result property to the console.

*Back in the designer's properties pane, set the **Invoked** property to **LookupComments_Invoked** and press **Enter**.*

*In the **LookupComments_Invoked** method, cast the **sender** parameter to a **SqlScalarActivityQuery**.*

*Use **Console.WriteLine** to display the activity's **Result** property to the console.*

```

private void LookupComments_Invoked(object sender, EventArgs e)
{
    SqlScalarQueryActivity activity = sender as SqlScalarQueryActivity;

    Console.WriteLine("Result - {0}", activity.Result);
}

```

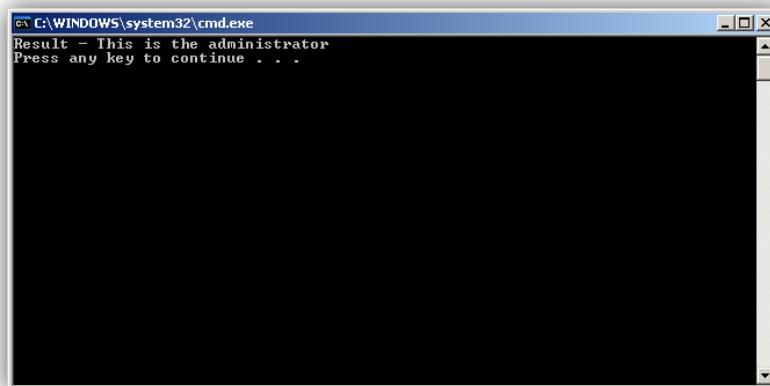
- Run the workflow and verify that the output displays the appropriate result.

- Verify the **SqlDatabaseActivitiesConsole** is bold in the **Solution Explorer**.

*If not, right click it and select **Set as Startup Project**.*

- Build and execute the workflow.

*Click **Debug -> Start Without Debugging**.*



Exercise 2: Creating a Composite Activity

- 1) Create a new **Activity** named **SqlConnectionActivity** in the **SqlDatabaseActivities** project.
 - Right click on the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Activity**.
 - Enter a name of **SqlConnectionActivity.cs** and click **OK**.
- 2) Define the **SqlConection** property that will be used to expose the connection to other components.
 - Define a property with a public getter and private setter.

```
public SqlConnection Connection { get; private set; }
```

- Add a **Browsable** attribute to the property so it does not show up in the designer.

```
[Browsable(false)]  
public SqlConnection Connection { get; private set; }
```

- 3) Define the dependency property named **ConnectionString** of type **string**.

```
public static DependencyProperty ConnectionStringProperty =  
    DependencyProperty.Register("ConnectionString",  
        typeof(System.String), typeof(SqlConnectionActivity));  
  
[Category("Parameters")]  
public string ConnectionString  
{  
    get { return base.GetValue(ConnectionStringProperty) as string; }  
    set { base.SetValue(ConnectionStringProperty, value); }  
}
```

- 4) Implement the execute method to start the processing of the child activities.

- Override the **Execute** method of the **Activity** class.

```
protected override ActivityExecutionStatus Execute(ActivityExecutionContext  
executionContext)  
{  
}
```

- If no child activities exist that are enabled, return with an **ActivityExecutionStatus** of **Closed**.

```
// check if there's any activities to execute  
if (base.EnabledActivities.Count == 0)  
    return ActivityExecutionStatus.Closed;
```

- Create and open a new **SqlConnection** using the **ConnectionString** dependency property.

```
// open the database connection  
if (this.Connection == null)  
{  
    this.Connection = new SqlConnection(this.ConnectionString);  
    this.Connection.Open();  
}
```

- Find the first activity in the list of enabled activities, start it and register for status change message from that activity.

*Access and store the first item in the **base.EnabledActivities** collection
Use the **Activity.RegisterForStatusChange** method to register for the **ClosedEvent**.
Use the **ActivityExecutionContext.ExecuteActivity** to start the first activity executing.*

```
// start the first child activity and register for it's state change event
Activity firstEnabledActivity = base.EnabledActivities[0];
firstEnabledActivity.RegisterForStatusChange(Activity.ClosedEvent, this);
executionContext.ExecuteActivity(firstEnabledActivity);
```

- Return an **ActivityExecutionStatus** of **Executing** to notify the runtime that this activity is still running.

```
// tell the runtime this activity is still executing
return ActivityExecutionStatus.Executing;
```

- 5) Implement the **IActivityEventListener<ActivityExecutionStatusChangedEventArgs>** interface so the activity can receive activity events from its child activities.

- Modify the class to derive from **IActivityEventListener<ActivityExecutionStatusChangedEventArgs>**.

```
public partial class SqlConnectionActivity: SequenceActivity,
    IActivityEventListener<ActivityExecutionStatusChangedEventArgs>
```

- Implement the **OnEvent** method to handle the receipt of a status change event from a child activity.

```
public void OnEvent(object sender, ActivityExecutionStatusChangedEventArgs e)
{ }
```

- Validate that the sender parameter of the **OnEvent** method is an **ActivityExecutionContext**.

```
// validate the sender
ActivityExecutionContext executionContext = sender as ActivityExecutionContext;
if (executionContext == null)
    throw new ArgumentException("sender");
```

- Using the **Activity.UnregisterForStatusChange** method, cancel the subscription to events from the activity that has closed.

```
// unregister the handler for status events
e.Activity.UnregisterForStatusChange(Activity.ClosedEvent, this);
```

- If the current activity's **ExecutionStatus** property indicates a **Canceling** or **Faulting** state, close the activity.

```
// if the activity is cancelling or faulting, cleanup
if ((this.ExecutionStatus == ActivityExecutionStatus.Canceling) ||
(this.ExecutionStatus == ActivityExecutionStatus.Faulting)) {
    executionContext.CloseActivity();
}
```

- If the current activity's **ExecutionStatus** property indicates that the activity is in the **Executing** state, find the next activity and execute it.

*Use the **FindNextChild** method to lookup the next child activity to be executed (this will be implemented in the next step).
If another activity was found, register to the closed event and tell the executor to execute the activity.
If no more activities were found, tell the execution context to close the **SqlConnectionActivity**.*

```

// if the activity is executing, schedule the next child
else if (this.ExecutionStatus == ActivityExecutionStatus.Executing) {
    // find the next activity in the sequence
    Activity nextActivity = FindNextChild();
    if (nextActivity != null) {
        // register and start the next activity
        nextActivity.RegisterForStatusChange(Activity.ClosedEvent, this);
        executionContext.ExecuteActivity(nextActivity);
    }
    else {
        // close this activity
        executionContext.CloseActivity();
    }
}

```

- 6) Create a **FindNextActivity** method that will find the next child activity to execute.

- Define a private method named **FindNextChild** that returns an **Activity** object.

```

private Activity FindNextChild()
{
}

```

- Check if there are any enabled activities in the list, if not return null.

```

// check if any activities are in the list
if (base.EnabledActivities.Count == 0)
    return null;

```

- Find the last activity in the list that has a status of closed.

```

// find the last activity in the list that is closed
Activity lastClosedActivity = base.EnabledActivities.LastOrDefault(
    n => n.ExecutionStatus == ActivityExecutionStatus.Closed);

```

- If the last closed activity is the last activity in the list, return null.

```

// if this is the last closed activity, return false
int indexOfLastClosedActivity =
    base.EnabledActivities.IndexOf(lastClosedActivity);
if (indexOfLastClosedActivity == base.EnabledActivities.Count - 1)
    return null;

```

- Return the activity following the last closed activity.

```

// return try to indicate a task was found
return base.EnabledActivities[indexOfLastClosedActivity + 1];

```

- 7) Handle the activity's **OnClosed** method to make sure the **SqlConnection** is closed.

- Override the **OnClosed** method of the **Activity** class.

```

protected override void OnClosed(IServiceProvider provider)

```

```
{  
}
```

- If the connection is not null, close it (if necessary) and set it to null.

```
// close the database connection  
if (this.Connection != null)  
{  
    if (this.Connection.State != System.Data.ConnectionState.Closed)  
        this.Connection.Close();  
    this.Connection = null;  
}
```

- Call the base class's implementation.

```
// call the base implementation  
base.OnClosed(provider);
```

8) Update the **SqlScalarQueryActivity** to use the parent **SqlConnectionActivity**'s **SqlConnection** instead of creating its own.

- Open the code for **SqlScalarQueryActivity** by right clicking it in the **Solution Explorer** and selecting **View Code**.
- Add a private method named **FindParentConnection** that accepts searches up the activity tree and finds the first **SqlConnectionActivity**.

*Use the **Activity.Parent** property to find the parent.*

*If the parent is a **SqlConnectionActivity**, return it.*

*Set the **activity** variable to its parent, and keep looking.*

```
private SqlConnectionActivity FindParentConnection(Activity current)  
{  
    while (current.Parent != null)  
    {  
        if (current.Parent is SqlConnectionActivity)  
            return current.Parent as SqlConnectionActivity;  
        current = current.Parent;  
    }  
    return null;  
}
```

- In the **Execute** method, remove all code that creates or opens the **SqlConnection** object.
- Immediately following the call where the **InvokingEvent** is raised, add the code to find the parent **SqlConnectionActivity** using the **FindParentConnection**.

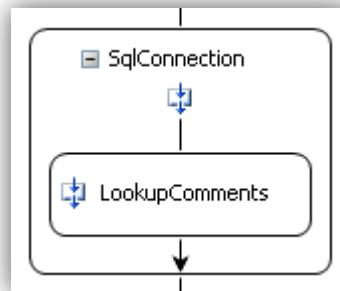
```
// find the parent SqlConnectionActivity  
SqlConnectionActivity connectionActivity = FindParentConnection(this);  
if (connectionActivity == null)  
    throw new InvalidOperationException("No parent connection was found.");
```

- Using the parent **SqlConnectionActivity**'s **Connection** property, store the **SqlConnection** in a variable named **connection**.

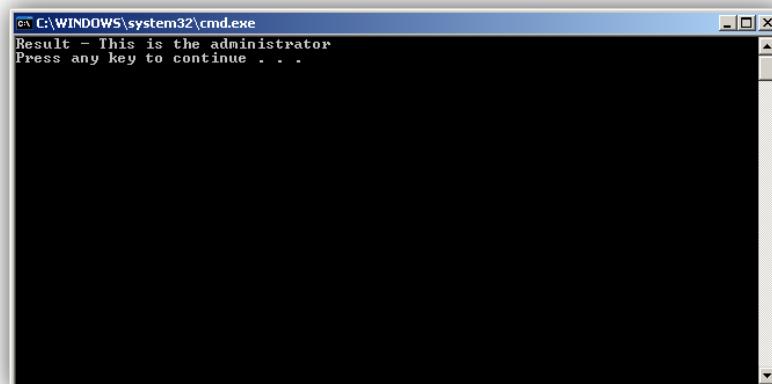
```
// store the connection for later
```

```
SqlConnection connection = connectionActivity.Connection;
```

- 9) Build the **SqlDatabaseActivities** project.
 - Right click the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Build**.
 - 10) Add a new **SqlConnectionActivity** to **Workflow1** and move the **SqlScalarQueryAction** into it.
 - View **Workflow1.cs** in design mode by right clicking it in the **Solution Explorer** and selecting **View Designer**.
 - Drag a new **SqlConnectionActivity** onto the canvas and name it **SqlConnection**.
*Drag a **SqlConnectionActivity** onto the canvas.*
*In the properties pane, set the **Name** property to **SqlConnection**.*
 - Define the connection string property as a lookup into the **UserProfiles** table.
*In the properties pane, set the **ConnectionString** property to the following value.*
- ```
Data Source=.\SQLEXPRESS;AttachDbFilename=C:\Labs\Files\Litware.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True
```
- Drag the existing **SqlScalarQueryActivity** into the new **SqlConnection** activity.



- 11) Run the workflow and verify that the output displays the appropriate result.
  - Verify the **SqlDatabaseActivitiesConsole** is bold in the **Solution Explorer**.  
*If not, right click it and select **Set as Startup Project**.*
  - Build and execute the workflow. The result should be the same as before, but now the connection is shared.  
*Click **Debug -> Start Without Debugging**.*



## Exercise 3: Adding Validation to the Activities

- 1) Create a new **class** named **SqlScalarQueryActivityValidator** in the new project.
  - Right click on the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Class**.
  - Enter a name of **SqlScalarQueryActivityValidator.cs** and click **OK**.
  - Modify the class definition so it is **public** and derives from **ActivityValidator**.

```
public class SqlScalarQueryActivityValidator : ActivityValidator
{
}
```

- Open the code for the **SqlScalarQueryActivity** and attach the **Validator** attribute defining the activity's validator.

*Open the code for **SqlScalarQueryActivity.cs** by right clicking it in the **Solution Explorer** and clicking **View Code**.*

*Attach the **Validator** attribute to the **SqlScalarQueryActivity** class using the type of the new **SqlScalarQueryActivityValidator** class as the parameter.*

```
[ActivityValidator(typeof(SqlScalarQueryActivityValidator))]
public partial class SqlScalarQueryActivity : Activity
```

- 2) Validate that the **Query** property is not **null** or an empty string.

- Override the **Validate** method from the **ActivityValidator** class.

```
public override ValidationErrorCollection Validate(ValidationManager manager,
object obj)
{ }
```

- Cast the **obj** parameter to a **SqlScalarQueryActivity** object and store it for later.

```
SqlScalarQueryActivity activity = obj as SqlScalarQueryActivity;
```

- Run the base validator and store any errors in a new errors variable of type **ValidationErrorCollection**.

```
// build the errors collection using the base validation method
ValidationErrorCollection errors = base.Validate(manager, obj);
```

- If the validator is running in a real environment (not as part of the activity's designer), validate that the **Query** property is not null or empty.

*Check if the **activity.Parent** attribute is not null. If it isn't, continue the validation.*

*Check if the **activity.Query** property is null or empty using **string.IsNullOrEmpty** method.*

*If the query property is empty, create a new validation error using the **ValidationError.GetNotSetValidation** method and add it to the **error** collection.*

```
// only perform validation if in design or runtime mode
if (activity.Parent != null) {
 // validate that a query was defined
 if (String.IsNullOrEmpty(activity.Query))
 errors.Add(ValidationError.GetNotSetValidation("Query"));
}
```

- Return the collection of errors.

```
// return the errors
return errors;
```

3) Verify the **SqlScalarQueryActivity** has a parent **SqlConnectionActivity**.

- Create a new method named **HasSqlConnectionParent** that looks through an activity's parents to find an object of a specific type.

*Use the **Activity.Parent** property to find the parent.*

*If the parent is a **SqlConnectionActivity**, return true.*

*Set the activity variable to its parent, and keep looking.*

```
private bool HasSqlConnectionParent(Activity current) {
 while (current.Parent != null) {
 if (current.Parent is SqlConnectionActivity)
 return true;
 current = current.Parent;
 }
 return false;
}
```

- In the **Validate** method, immediately after validating the **Query** parameter, check if the activity has a **SqlConnectionActivity** as a parent.

```
// validate the activity has an ancestor that is a SqlConnectionActivity
if (!HasSqlConnectionParent(activity))
 errors.Add(new ValidationError(
 "SqlScalarQueryActivities must be placed in a SqlConnectionActivity",
 100, false));
```

4) Create a new class named **SqlConnectionActivityValidator** in the **SqlDatabaseActivities** project.

- Right click on the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Class**.
- Enter a name of **SqlConnectionActivityValidator.cs** and click **OK**.
- Modify the class definition so it derives from **ActivityValidator**.

```
public class SqlConnectionActivityValidator : CompositeActivityValidator
{ }
```

- Open the code for the **SqlConnectionActivity** and attach the **Validator** attribute.

*Open the code for **SqlConnectionActivity.cs** by right clicking it in the **Solution Explorer** and clicking **View Code**.*

*Attach the **Validator** attribute to the **SqlConnectionActivity** class using the type of the new **SqlConnectionActivityValidator** class as the parameter.*

```
[ActivityValidator(typeof(SqlConnectionActivityValidator))]
public partial class SqlConnectionActivity : CompositeActivity
```

5) Validate that the **ConnectionString** property is not null or an empty string.

- Override the **Validate** method from the **ActivityValidator** class.

```
public override ValidationExceptionCollection Validate(ValidationManager manager,
object obj)
{ }
```

- Cast the **obj** parameter to a **SqlScalarQueryActivity** object and store it for later.

```
SqlConnectionActivity activity = obj as SqlConnectionActivity;
```

- Run the base validator and store any errors in a new errors variable of type **ValidationErrorCollection**.

```
// build the errors collection using the base validation method
ValidationErrorCollection errors = base.Validate(manager, obj);
```

- If the validator is running in a real environment (not as part of the activity's designer), validate that the **ConnectionString** property is not null or empty.

*Check if the **activity.Parent** attribute is not null. If it isn't, continue the validation.*

*Check if the **activity.ConnectionString** property is null or empty using **string.IsNullOrEmpty** method.*

*If the connection string property is empty, create a new validation error using the **ValidationError.GetNotSetValidationErrors** method and add it to the **error** collection.*

```
// only perform validation if in design or runtime mode
if (activity.Parent != null)
{
 // validate that a connection string was defined
 if (string.IsNullOrEmpty(activity.ConnectionString))
 errors.Add(ValidationError.GetNotSetValidationErrors("ConnectionString"));
}
```

- Return the collection of errors.

```
// return the errors
return errors;
```

## 6) Verify the **SqlConnectionActivity** does not have any child activities that derive from **IEventHandler**.

- Create a new method named **HasChildIEventActivity** that looks through a composite activity's child activities and checks if any derive from **IEventHandler**.

```
private bool HasChildIEventActivity(CompositeActivity activity)
{
}
```

- Loop through each enabled child activity and check if the activity derives from **IEventHandler**. If so, return **true**.
- Check each child activity to determine if it is a **CompositeActivity**. If so, call **HasChildIEventActivity** and return true if it returns **true**.

```
// loop through each child activity that is enabled
foreach (Activity childActivity in activity.EnabledActivities) {
 // if the child activity is an IEventActivity, log an error
 if (childActivity is IEventActivity)
 return true;

 // if the child activity is a composite activity, check its children
 CompositeActivity compositeActivity = childActivity as CompositeActivity;
 if (compositeActivity != null)
 if (HasChildIEventActivity(compositeActivity))
 return true;
}
```

- If no child activities are **IEventActivities**, return **false**.

```
// no child activities are IEventActivity
```

```
return false;
```

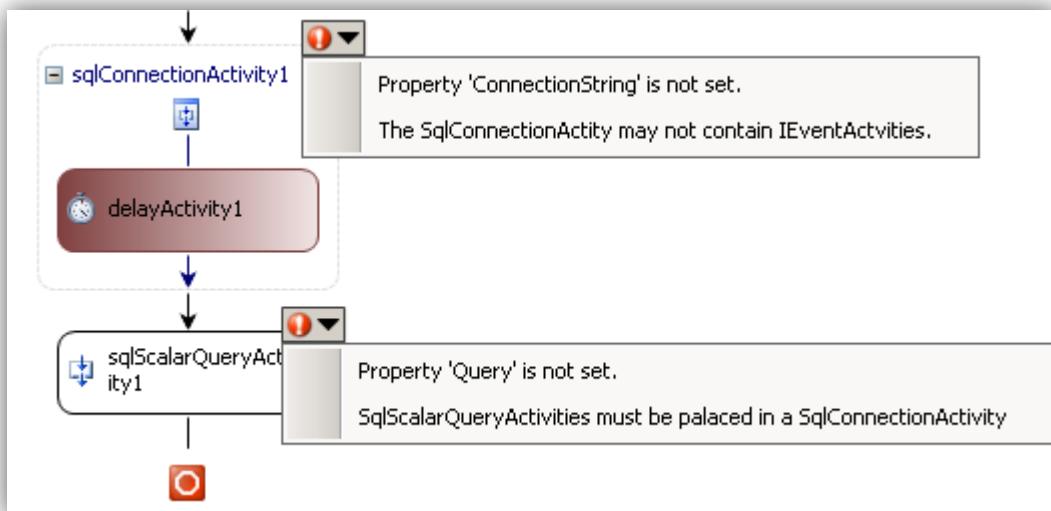
- In the **Validate** method, immediately after validating the **ConnectionString** parameter, check if the activity has a **SqlConnectionActivity** as a parent.

```
// validate the activity has no child IEventActivity activities
if (HasChildIEventActivity(activity))
 errors.Add(new ValidationError(
 "SqlConnectionActivities cannot contain IEventActivity activities. ",
 100, false));
```

- 7) Build the **SqlDatabaseActivities** project.
  - Right click the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Build**.
- 8) Test the validators by trying to create activities that are missing key properties.
  - Open **Workflow1.cs** in design mode by right clicking it and clicking **View Designer**.
  - Drag a new **SqlConnectionActivity** onto the canvas.
  - Drag a new **Delay** activity into the new **SqlConnectionActivity**.
  - Drag a new **SqlScalarQueryActivity** onto the canvas.
  - Verify each of the validators has fired.

*The **SqlConnectionActivity** should have an error indicating an empty **ConnectionString** and containing an **IEventActivity**.*

*The **SqlScalarQueryActivity** should have an error indicating an empty query and no parent **SqlConnectionActivity**.*



- Clean up by deleting the activities just added.

## Exercise 4: Adding custom designers to the Activities

- 1) Create a new class named **SqlScalarQueryActivityDesigner** in the **SqlDatabaseActivities** project.
  - Right click on the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Class**.
  - Enter a name of **SqlScalarQueryActivityDesigner.cs** and click **OK**.
  - Modify the class definition so it is **public** and derives from **ActivityDesigner**.

```
public class SqlScalarQueryActivityDesigner : ActivityDesigner
{
}
```

- Open the code for the **SqlScalarQueryActivity** and attach the **Designer** attribute defining the activity's designer.

*Open the code for **SqlScalarQueryActivity.cs** by right clicking it in the **Solution Explorer** and clicking **View Code**.*

*Attach the **Designer** attribute to the **SqlScalarQueryActivity** class using the type of the new **SqlScalarQueryActivityDesigner** class as the parameter.*

```
[Designer(typeof(SqlScalarQueryActivityDesigner))]
public partial class SqlScalarQueryActivity : Activity
```

- 2) Restrict **SqlScalarQueryActivities** from being placed on the canvas unless it is in a **SqlConnectionActivity**.

- Create a new method named **HasSqlConnectionParent** that looks through an activity's parents to find an object of a specific type.

*Use the **Activity.Parent** property to find the parent.*

*If the parent is a **SqlConnectionActivity**, return true.*

*Set the **activity** variable to its parent, and keep looking.*

```
private bool HasSqlConnectionParent(Activity current)
{
 while (current.Parent != null)
 {
 if (current.Parent is SqlConnectionActivity)
 return true;
 current = current.Parent;
 }
 return false;
}
```

- Override the **CanBeParentedTo** method to restrict the activities that this activity can be contained in.

*Define the **CanBeParentedTo** method.*

*If the **Activity** property in the parent designer object is of type **SqlConnectionActivity**, return true.*

*If not, check if any of the parent objects are of type **SqlConnectionActivity**.*

```
public override bool CanBeParentedTo(CompositeActivityDesigner
parentActivityDesigner)
{
 // if the parent activity is a SqlConnectionActivity
 if (parentActivityDesigner.Activity is SqlConnectionActivity)
 return true;

 // if not, check if a parent is a SqlConnectionActivity
 return HasSqlConnectionParent(parentActivityDesigner.Activity);
}
```

- 3) Create a new class named **SqlConnectionActivityDesigner** in the **SqlDatabaseActivities** project.

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

- Enter a name of **SqlConnectionActivityDesigner.cs** and click **OK**.
- Modify the class definition so it is **public** and derives from **SequentialActivityDesigner**.

```
public class SqlConnectionActivityDesigner : SequentialActivityDesigner
{
}
```

- Open the code for the **SqlDesignerActivity** and attach the **Designer** attribute defining the activity's designer.

*Open the code for **SqlConnectionActivity.cs** by right clicking it in the **Solution Explorer** and clicking **View Code**.*

*Attach the **Designer** attribute to the **SqlConnectionActivity** class using the type of the new **SqlConnectionActivityDesigner** class as the parameter.*

```
[Designer(typeof(SqlConnectionActivityDesigner))]
public partial class SqlConnectionActivity : SequenceActivity
```

4) Restrict the activities that can be placed in the **SqlConnectionActivity**.

- Override the **CanInsertActivities** method in the **SqlConnectionActivityDesigner**.
- If any of the activities in **activitiesToInsert** parameter derive from **IEventActivity**, return **false**.

```
public override bool CanInsertActivities(HitTestInfo insertLocation,
 IReadOnlyCollection<Activity> activitiesToInsert)
{
 // check if any activities are of type IEventActivity
 return !activitiesToInsert.Any(n => n is IEventActivity);
}
```

5) Add a new verb to the designer that will allow workflow developers to right click and add a new **SqlScalarQueryActivity** to the **SqlConnectionActivity**.

- Create a private event handler named **AddScalarQuery\_Click**.

```
private void AddScalarQuery_Click(object sender, EventArgs e) { }
```

- Cast the designer's activity to a **CompositeActivity** variable for later.

```
//Access the designer's activity using the base classes Activity property.
CompositeActivity activity = base.Activity as CompositeActivity;
```

- Create a list of new Activity objects and create a new **SqlScalarQueryActivity** in the list.

```
// create the list of new activites
List<Activity> activities = new List<Activity>();
activities.Add(new SqlScalarQueryActivity());
```

- Create a **ConnectorHitTestInfo** object identifying where to add the activity. In this case add it as the last activity in the **SqlConnectionActivity**.

```
// determine where to add the activities
ConnectorHitTestInfo location =
 new ConnectorHitTestInfo(this,
 HitTestLocations.Designer, activity.Activities.Count);
```

- Add the child activities using the base class's **InsertActivities** method.

```
// add the activities to the designer
base.InsertActivities(location, activities.AsReadOnly());
```

- Override the **Verbs** property.

```
protected override ActivityDesignerVerbCollection Verbs { get {} }
```

- Create a new **ActivityDesignerVerbCollection** and populate it by calling the base class's implementation.

```
// load the verbs from the base class
ActivityDesignerVerbCollection verbs = new ActivityDesignerVerbCollection();
verbs.AddRange(base.Verbs);
```

- Add the new verb named **Add Scalar Query** the calls the **AddScalarQuery\_Click**.

```
// add the new verb
verbs.Add(
 new ActivityDesignerVerb(this, DesignerVerbGroup.Actions, "Add Scalar Query",
 new EventHandler(AddscalarQuery_Click)));
```

- Return the new list of verbs.

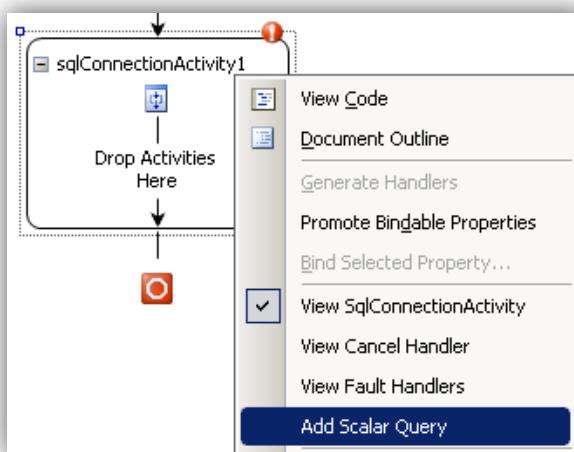
```
// return the list of verbs
return verbs;
```

6) Build the **SqlDatabaseActivities** project.

- Right click the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Build**.

7) Test the designers by trying to place activities in positions that are not valid

- Open **Workflow1.cs** in design mode by right clicking it and clicking **View Designer**.
- Drag a new **SqlConnectionActivity** onto the canvas.
- Try to drag a new **Delay** activity into the new **SqlConnectionActivity** and verify it will not work.
- Try to drag a new **SqlScalarQueryActivity** outside of the **SqlConnectionActivity** and verify it will not work.
- Right click on the **SqlConnectionActivity** and select **Add Scalar Query** to add a new **SqlScalarQueryActivity** to the **SqlConnectionActivity**.



### Challenge: Designer Themes and Toolbox Items

Add a designer theme to the **SqlConnectionActivityDesigner** that changes the background color of the **SqlConnectionActivity**. To do this you will need to create a class derived from

**CompositeDesignerTheme** and set the theme's properties in the constructor. Once the theme is complete, attach it to the designer using the **ActivityDesignerTheme** attribute.

Add a toolbox item to the **SqlConnectionActivity** that adds a **SqlConnectionActivity** and an embedded **SqlScalarQueryActivity** when a **SqlConnectionActivity** is added to the canvas. To do this you will need to create a class derived from **ActivityToolboxItem** and override **CreateComponentsCore**. Once the toolbox item is complete, attach it to the activity using the **ToolboxItem** attribute.

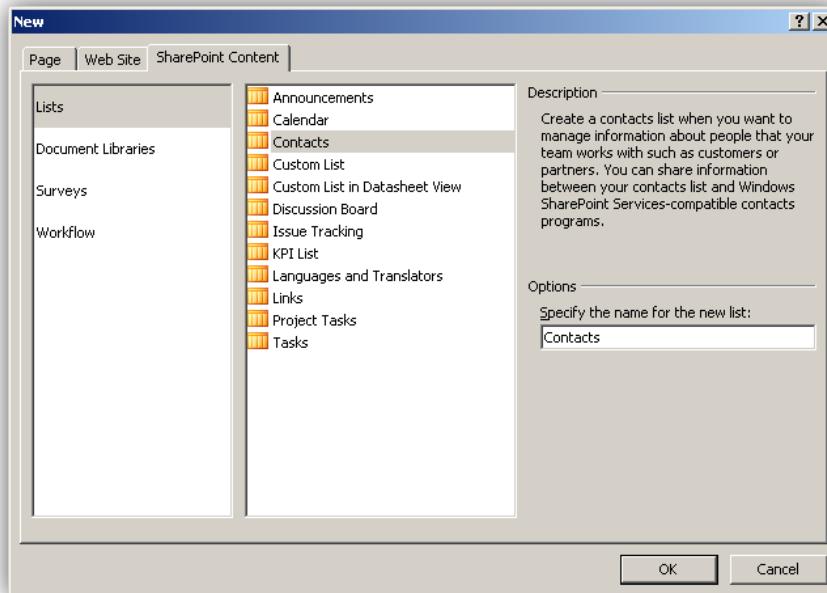
## Lab 12: Extending SharePoint Designer with Custom Activities

**Lab Overview:** Litware Inc has a need to integrate contact information that has currently been stored in a database into a **Contacts** list on a SharePoint site. A two-step process is designed. First, it checks if a new or updated contact has an email address and then looks up notes based on that email address. In the event no value exists for the email address, a user is assigned a task to provide them. The notes are then assigned to the Contact in its notes field.

This solution is fluid enough that the design team does not want a developer to manage it. Using the SharePoint designer is the best choice as it allows each department to manage their needs. To do this, a developer will need to create some custom conditions and actions that allow the IT departments to design the workflows themselves using the SharePoint Designer.

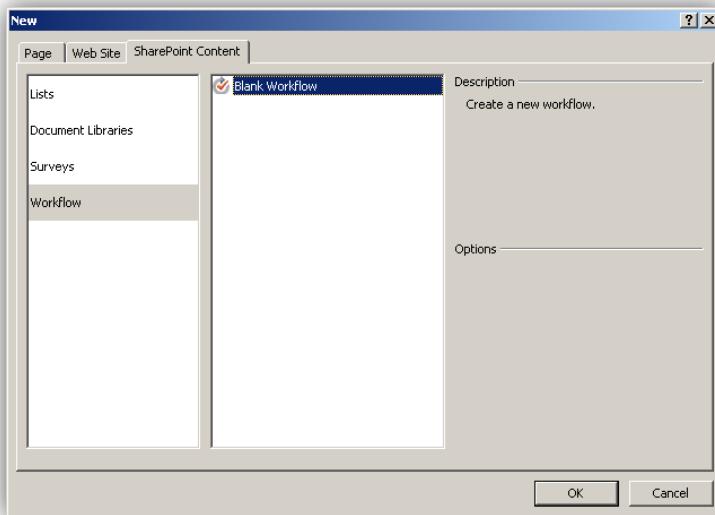
### Exercise 1: Creating a simple workflow in SharePoint Designer

- 1) Start the **SharePoint Designer** on open the site at <http://litwareinc.com/sites/Demo>.
  - Start the **SharePoint Designer** by clicking the **Start Button** and selecting **All Programs -> Microsoft Office -> Microsoft Office SharePoint Designer 2007**.
  - Open the **Demos** site collection by clicking **File -> Open Site** and entering a site name of <http://litwareinc.com/sites/Demo>.
- 2) Create a new list in the site named **Contacts** using list type **Contacts**.
  - In the **Folder List** on the left hand side, right click the site at the top of the tree and click **New -> SharePoint Content**.
  - In the **New** dialog, select **Lists** in the list box on the left and select **Contacts** on the center list box.
  - Enter a name of **Contacts** and click **OK**.



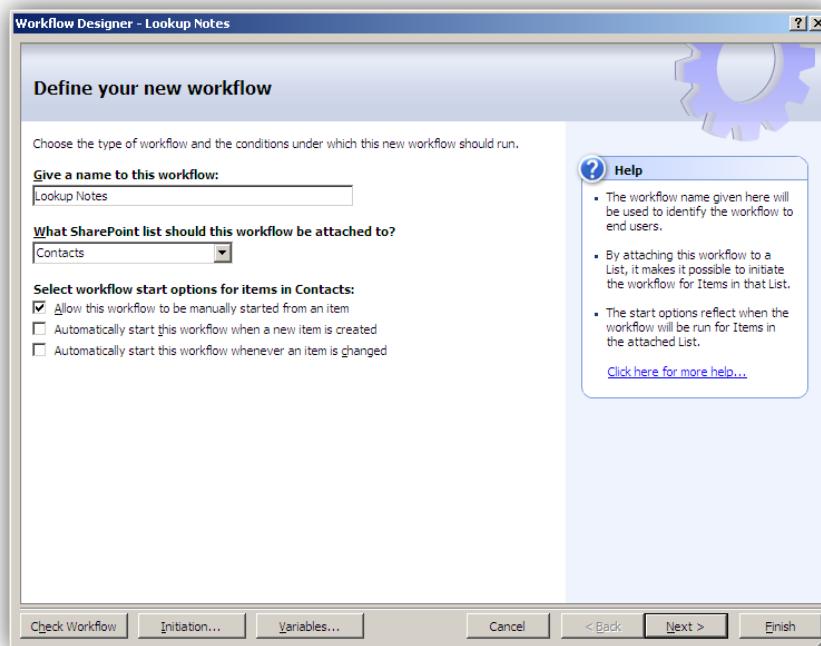
3) Add a new workflow named **Lookup Notes** to the **Contacts** list.

- In the **Folder List** on the left hand side, right click the site at the top of the tree and click **New -> SharePoint Content**.
- In the **New** dialog, select **Workflow** in the list box on the left and select **Blank Workflow** on the center list box.
- Click **OK** to create the new workflow.



4) Name the new workflow **Lookup Notes** and attach the new workflow to the **Contacts** lists.

- In the **Workflow Designer Wizard**, set the name of the workflow to **Lookup Notes**.
- Choose the **Contacts** list as the list the workflow should be attached to.
- Click **Next** to move on to defining the workflow steps.



5) Create a step that will terminate the workflow if the contact's email field is empty.

- Set the **Step Name** to **Verify Email Available**.
- Add a condition of type **Compare Contacts Field**.

*Click the **Condition** button and select **Compare Contacts Field**.*

*Click the **field** link in the condition and select the **E-mail Address** field.*

*Click the **equals** link and select **is empty**.*

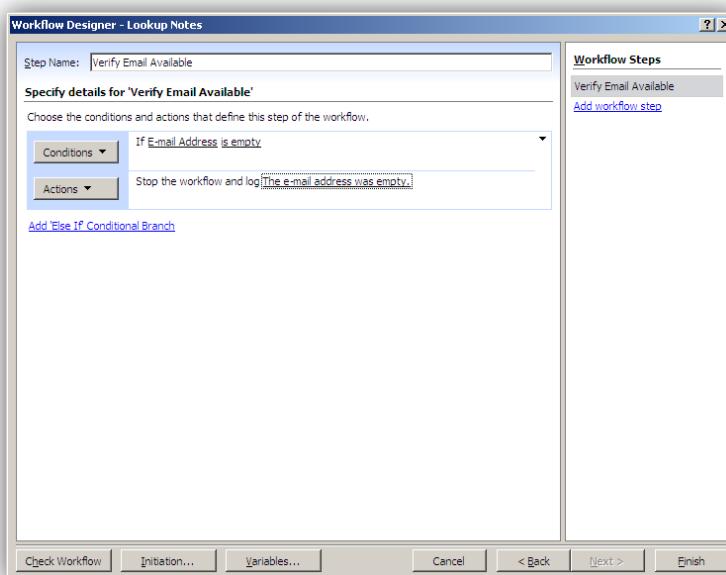
- Add an action that of **Stop Workflow** to terminate the workflow.

*Click the **Actions** button and select **More Actions**.*

*In the **Workflow Actions** dialog, select the **Core Actions** category.*

*Select the **Stop Workflow** action from the list and click **Add**.*

*Click the **this message** link and enter "The e-mail address was empty."*



6) Add a new step that will be used to retrieve the notes from the user.

- Click the **Add workflow step** link on the left hand side of the dialog.
- Set the **Step Name** to **Retrieve Contact Notes**.
- Add an action that will request notes from a user.

*Click the **Actions** button and select **More actions**.*

*In the **Workflow Actions** dialog, select the **Task Actions** category.*

*Select the **Collect Data from a User** action from the list and click **Add**.*

*Click the **data** link to start the **Custom Task Wizard**.*

*Click **Next** to move to the next step.*

*Enter a name of "**Enter notes for contact**" and click **Next**.*

*Click **Add** to add a single field named **CustomNotes** of type **Single line of text** and then click **Next**.*

*Click **Finish** on both dialogs to complete the wizard.*

*Click the **this user** link and select a user of **Administrator**.*

7) Store the results of the user activity into a workflow variable so it can be used later.

- Add an action that will store the result of the task in a workflow variable.

*Click the **Actions** button and select **More actions**.*

*In the **Workflow Actions** dialog, select the **Core Actions** category.*

*Select the **Set Workflow Variable** action from the list and click **Add**.*

*Click the **workflow variable** link in the action and select the **Create a new variable** link*

*Give the variable a name of **Notes** and a type of **String** and click **OK**.*

*Click the **value** link in the action and click the **Fx** button.*

- Select the field from the task that was just completed.

*In the **Lookup Details** section's **Source** drop down list, select the **Tasks** list.*

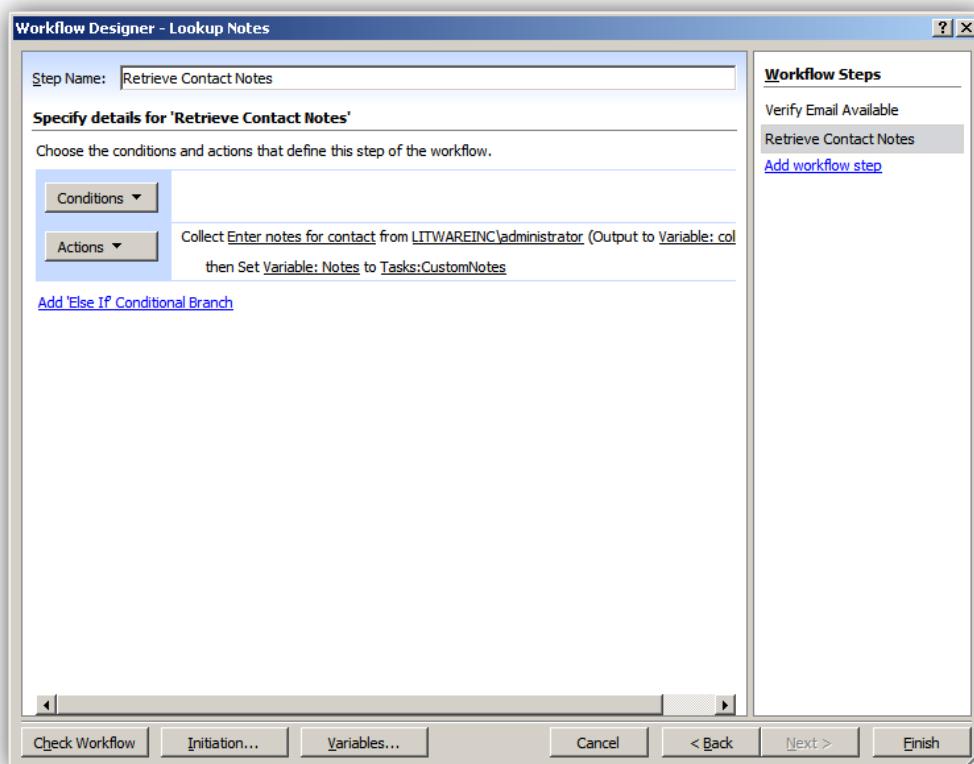
*In the **Lookup Details** section's **Field** drop down list, select **CustomNotes**.*

*In the **Find the List Item** section's **Field** drop down list, select **Tasks:ID**.*

*In the **Find the List Item** section's **Value** drop down list, click the **Fx** button.*

*Select a **Source of Workflow Data** and a field of **Variable: collect** and click **OK**.*

*Click **OK** again to complete the action.*



8) Using the **Notes** workflow variable, set the **Notes** field of the current item.

- Click the **Add workflow step** link on the left hand side of the dialog.
- Set the **Step Name** to **Update Notes**.
- Add an action that will update a field on the current item.

*Click the **Actions** button and select **More actions**.*

*In the **Workflow Actions** dialog, select the **Core Actions** category.*

*Select the **Set Field in Current Item** action from the list and click **Add**.*

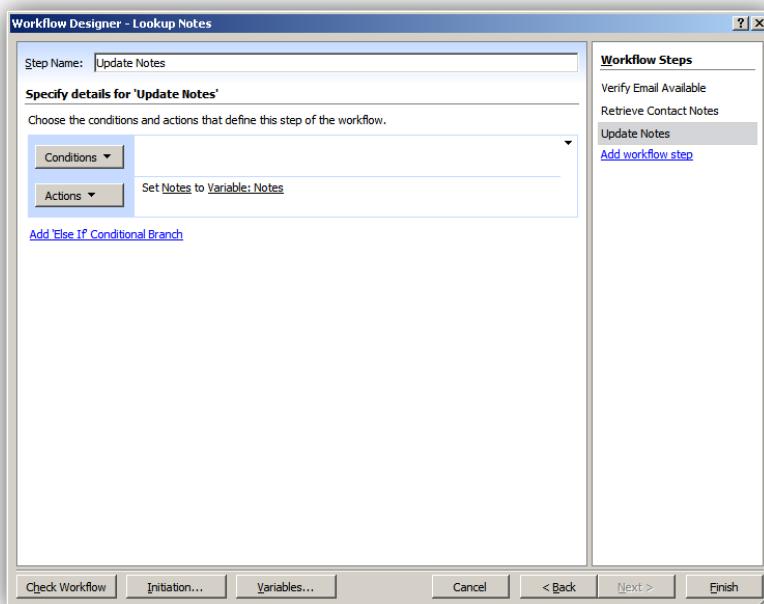
*Click the **field** link in the action and select the **Notes** field.*

*Click the **value** link in the action and click the **Fx** button.*

*In the **Lookup Details** section's **Source** drop down list, select the **Workflow Data** item.*

*In the **Lookup Details** section's **Field** drop down list, select **Variable: Notes** and click **OK**.*

*Click **OK** again to complete the action.*



9) Save the workflow by clicking the **Finish** button in the Workflow Designer.

10) Test the workflow in the browser by creating a new contact with no email and starting the **Lookup Notes** workflow.

- Navigate to the **Contacts** list using **Internet Explorer**.

*Navigate to <http://litwareinc.com/sites/Demos>.*

*Click the **Contacts** link on the left hand navigation bar.*

- Create a new contact with an empty email address.
- Start the workflow on the new contact.

*Click **Workflows** in the item's drop down menu and start a workflow of type **Lookup Notes**.*

*On the instantiation page, click **Start**.*

- On the **Contacts** list page, click the **Completed** link under **Lookup Notes** to see the workflow status.
- Verify the workflow is completed and there is a history event indicating the email address was empty.

**Workflow Information**

|            |                          |         |           |
|------------|--------------------------|---------|-----------|
| Initiator: | LitwareInc Administrator | Item:   | John Doe  |
| Started:   | 1/27/2008 3:47 PM        | Status: | Completed |
| Last run:  | 1/27/2008 3:47 PM        |         |           |

**Tasks**

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

| Assigned To                                                                                           | Title | Due Date | Status |
|-------------------------------------------------------------------------------------------------------|-------|----------|--------|
| There are no items to show in this view of the "Tasks" list. To create a new item, click "New" above. |       |          |        |

**Workflow History**

[View workflow reports](#)

The following events have occurred in this workflow.

| Date Occurred     | Event Type         | User ID        | Description                   |
|-------------------|--------------------|----------------|-------------------------------|
| 1/27/2008 3:47 PM | Workflow Completed | System Account | The e-mail address was empty. |

11) Test the workflow with a contact that has an email address.

- Edit the existing contact and provide an email address of **Administrator@litwareinc.com**.
- Start the workflow on the existing contact.

*Click **Workflows** in the item's drop down menu and start a workflow of type **Lookup Notes**.*

*On the instantiation page, click **Start**.*

- On the **Contacts** list page, click the **In Progress** link under **Lookup Notes** to see the workflow status.
- Approve the notes task

*Click the **Edit Item** menu item on the task's title.*

*Enter notes for the current contact.*

*Click **Complete Task** to approve the task.*

|                                                                                                                              |                      |  |  |
|------------------------------------------------------------------------------------------------------------------------------|----------------------|--|--|
| <b>Title:</b>                                                                                                                | Enter Contact Notes  |  |  |
| <b>CustomNotes:</b>                                                                                                          | <input type="text"/> |  |  |
| <b>Related list item:</b>                                                                                                    | John Doe             |  |  |
| <input type="button" value="Save Draft"/> <input type="button" value="Complete Task"/> <input type="button" value="Cancel"/> |                      |  |  |

- Verify the workflow is completed and all of the tasks are completed as well.

**Workflow Information**

|            |                          |         |           |
|------------|--------------------------|---------|-----------|
| Initiator: | LitwareInc Administrator | Item:   | John Doe  |
| Started:   | 1/27/2008 3:48 PM        | Status: | Completed |
| Last run:  | 1/27/2008 3:49 PM        |         |           |

**Tasks**

The following tasks have been assigned to the participants in this workflow. Click a task to edit it. You can also view these tasks in the list [Tasks](#).

| Assigned To              | Title                                         | Due Date | Status    | Outcome   |
|--------------------------|-----------------------------------------------|----------|-----------|-----------|
| LitwareInc Administrator | Enter notes for contact <a href="#">! NEW</a> |          | Completed | Completed |

**Workflow History**

[View workflow reports](#)

The following events have occurred in this workflow.

| Date Occurred                                                                                                    | Event Type | User ID | Description | Outcome |
|------------------------------------------------------------------------------------------------------------------|------------|---------|-------------|---------|
| There are no items to show in this view of the "Workflow History" list. To create a new item, click "New" above. |            |         |             |         |

## Exercise 2: Creating a custom SharePoint Designer condition

- 1) Add a new **SqlConditions** class to the **SqlDatabaseActivities** project.
  - In Visual Studio 2008, right click **SqlDatabaseActivities** in the **Solution Explorer** and click **Add -> Class**.
  - Name the class **SqlConditions.cs** and click **Add**.
  - Modify the class's definition to make it **public**.
- 2) Add a static method named **ItemExists** that will check for an item in the database.
  - Create a public static **ItemExists** method that returns a **bool** and accepts a **WorkflowContext**, list id, and item id along with the method specific parameters.

```
public static bool ItemExists(WorkflowContext context, string listId, int
 itemId, string table, string queryField, object queryValue, string
 connectionString)
{ }
```

- Add code that opens the **SqlConnection** using the **connectionString** property.

```
// create the SqlConnection object
using (SqlConnection connection = new SqlConnection(connectionString)) {
 connection.Open();
}
```

- Format the query and execute the query using the **table**, **queryField**, and **queryValue** parameters. Place the code in the previous **using** statement.

```
// build the query using the parameters
string query =
 string.Format("SELECT TOP 1 COUNT(*) FROM {0} WHERE {1} = @queryValue",
 table, queryField);

// create the SqlCommand and the parameters
SqlCommand command = new SqlCommand(query, connection);
command.Parameters.Add(new SqlParameter("@queryValue", queryValue));

// execute the query and return true if an item was found
return ((int)command.ExecuteScalar() == 1);
```

- 3) Create an **.ACTIONS** file in the **Template/1033/Workflow** folder.

- Create a **Template** folder in the project.

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

*Change the name of the new folder to **Template**.*

- Create a **1033** folder in the new **Template** folder.

*Right click the **Template** folder in the **Solution Explorer** and click **Add -> New Folder**.*

*Change the name of the new folder to **1033**.*

- Create a **Workflow** folder in the new **1033** folder.

*Right click the **1033** folder in the **Solution Explorer** and click **Add -> New Folder**.*

*Change the name of the new folder to **Workflow**.*

- Create a new **SqlDatabaseActivites.ACTIONS** file in the new **Workflow** folder.

*Right click the **Workflow** folder in the **Solution Explorer** and click **Add -> New Item**.*

In the **Add New Item** dialog, select **Data** in the **Categories** list and **Xml File** in the **Templates** list.

Set the name to **SqlDatabaseActivities.ACTIONS** and click **Add**.

- In the **Actions** file, add the following XML to define the new condition.

The condition defines the **Functionname**, **ClassName**, and **Assembly** of the condition method.

The **Sentence** and **FieldBind** defines the designer experience and references method parameters using the **\_1\_** syntax.

The **Parameters** define the type of each parameter in the condition method.

```
<workflowInfo>
 <Conditions And="and" Or="or" Not="not" When="If" Else="Else if">
 <Condition Name="Exists in database" FunctionName="ItemExists"
 ClassName="SqlDatabaseActivities.SqlConditions" Assembly="SqlDatabaseActivities,
 Version=1.0.0.0, Culture=neutral, PublicKeyToken=2848957be3dd88b2"
 AppliesTo="all" UsesCurrentItem="true">
 <RuleDesigner Sentence="Exists in %1 in %2 where %3 equals %4">
 <FieldBind Id="1" Field="_1_" Text="table" />
 <FieldBind Id="2" Field="_4_" Text="database" />
 <FieldBind Id="3" Field="_2_" Text="field" />
 <FieldBind Id="4" Field="_3_" Text="value" />
 </RuleDesigner>
 <Parameters>
 <Parameter Name="_1_" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="_2_" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="_3_" Type="System.Object, mscorelib" Direction="In" />
 <Parameter Name="_4_" Type="System.String, mscorelib" Direction="In" />
 </Parameters>
 </Condition>
 </Conditions>
</workflowInfo>
```

- 4) Update the **web.config** file to allow the custom activities to be loaded by SharePoint's workflow engine.

- Open the **web.config** file at **C:\inetpub\LitwarePublicSite** in **Visual Studio 2008**.
- Locate the **System.Workflow.ComponentModel.WorkflowCompiler/authorizedTypes**.
- Add another **authorizedType** referencing the **SqlDependencyActivities** assembly.

```
<authorizedType Assembly="SqlDatabaseActivities, Version=1.0.0.0,
 Culture=neutral, PublicKeyToken=2848957be3dd88b2"
 Namespace="SqlDatabaseActivities" TypeName="*" Authorized="True" />
```

- 5) Add post build steps to install the assembly in the gac, copy files to the **Template** folder, and reset the app pool.

- Right click the project in the **Solution Explorer** and click **Properties**.
- Select the **Build Events** tab and enter the following **Post Build** steps

```
xcopy "$(ProjectDir)\TEMPLATE" "C:\Program Files\Common Files\Microsoft
Shared\web server extensions\12\TEMPLATE" /E /Y
"$(DevEnvDir)..\..\SDK\v2.0\bin\gacutil.exe" /i "$(TargetPath)" /f
%windir%\system32\cscript.exe c:\windows\system32\isapp.vbs /a
"SharePointDefaultAppPool" /r
```

## Exercise 3: Creating a custom SharePoint Designer activity

- 1) Create a new **SqlGetValueActivity** in the **SqlDatabaseActivities** project.
  - Right click the **SqlDatabaseActivities** project in the **Solution Explorer** and click **Add -> Activity**.
  - Enter a name of **SqlGetValueActivity.cs** and click **Add**.
- 2) Define the dependency properties the **SharePoint Designer** will use to communicate with the **activity**.
  - The **ConnectionString** property will define the connection string to use.
  - The **Table** property defines the table to use in the query.
  - The **QueryField** and **QueryValue** properties will define the row to use.
  - The **ResultField** will define the column to return.
  - The **Result** property will hold the result.

```
public static DependencyProperty ConnectionStringProperty =
 DependencyProperty.Register("ConnectionString",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String ConnectionString
{
 get { return ((string)(base.GetValue(ConnectionStringProperty))); }
 set { base.SetValue(ConnectionStringProperty, value); }
}

public static DependencyProperty ResultFieldProperty =
 DependencyProperty.Register("ResultField",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String ResultField
{
 get { return ((string)(base.GetValue(ResultFieldProperty))); }
 set { base.SetValue(ResultFieldProperty, value); }
}

public static DependencyProperty TableProperty =
 DependencyProperty.Register("Table",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String Table
{
 get { return ((string)(base.GetValue(TableProperty))); }
 set { base.SetValue(TableProperty, value); }
}

public static DependencyProperty QueryFieldProperty =
```

```

DependencyProperty.Register("QueryField",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String QueryField
{
 get { return ((string)(base.GetValue(QueryFieldProperty))); }
 set { base.SetValue(QueryFieldProperty, value); }
}

public static DependencyProperty QueryValueProperty =
 DependencyProperty.Register("QueryValue",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String QueryValue
{
 get { return ((string)(base.GetValue(QueryValueProperty))); }
 set { base.SetValue(QueryValueProperty, value); }
}

public static DependencyProperty ResultProperty =
 DependencyProperty.Register("Result",
 typeof(System.String), typeof(SqlDatabaseActivities.SqlGetValueActivity));

[Category("Parameters")]
public String Result
{
 get { return ((string)(base.GetValue(ResultProperty))); }
 set { base.SetValue(ResultProperty, value); }
}

```

- 3) Add a new **SqlConnection** activity to the **SqlGetValueActivity** designer.

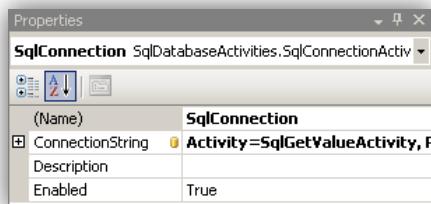
- Right click **SqlGetValueActivity.cs** in the **Solution Explorer** and click **View Designer**.
- Drag a new **SqlConnection** activity onto the canvas and name it **SqlConnection**.

*Drag a new **SqlConnection** activity onto the canvas.*

*In the properties pane, set the **Name** property to **SqlConnection**.*

*Select the **ConnectionString** property and click the ... button.*

*Select the **ConnectionString** property from the parent's property list.*



4) Define the query embedded in the new **SqlConnection** activity.

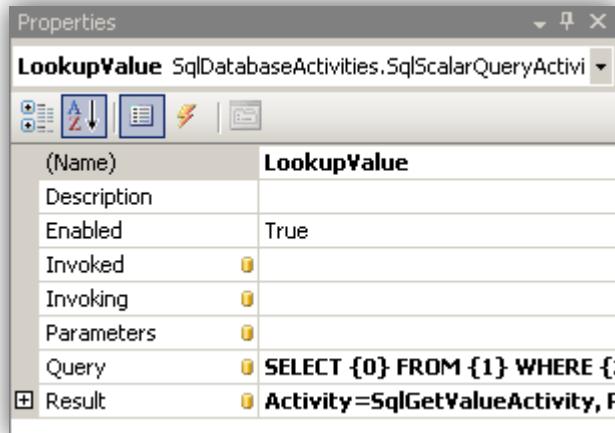
- Select the **SqlScalarQueryActivity** inside the **SqlConnection** activity and change its name to **LookupValue**.

*Right click the **SqlScalarQueryActivity** in the **SqlConnection** activity and click **Properties**.*

*In the properties pane, set the **Name** property to **LookupValue**.*

*Bind the **Result** property to the parent's **Result** property.*

*Set the **Query** property to **SELECT {0} FROM {1} WHERE {2} = @queryValue**.*



- Add an Invoking handler to the activity that will initialize the **Query** and **Parameters** property.

*In the properties pane, set the **Invoking** property of **LookupValue\_Invoking** and press **Enter**.*

*Cast the sender property to a **SqlScalarQueryActivity** object.*

*Use **string.Format** to replace the tokens in the query with the **ResultField**, **Table**, and **QueryField** properties.*

*Create a new **SqlParameter** object to map the **QueryValue** property to the **@queryValue** query parameter.*

*Put the new **SqlParameter** into an array of **SqlParameters** and assign it to the activity's **Parameters** property.*

```
private void LookupValue_Invoking(object sender, EventArgs e)
{
 SqlScalarQueryActivity activity = sender as SqlScalarQueryActivity;

 activity.Query = string.Format(
 activity.Query, this.ResultField, this.Table, this.QueryField);
 activity.Parameters = new SqlParameter[]
 {
 new SqlParameter("@queryValue", this.QueryValue)
 };
}
```

5) Update the **.ACTIONS** file to define the new activity's integration with the **SharePoint Designer**.

- Add the **Actions** and **Action** elements to the file.

*The **ClassName** and **Assembly** attributes define the location of the activity.*

```
<Actions Sequential="then" Parallel="and">
 <Action Name="Sql Value Lookup"
 ClassName="SqlDatabaseActivities.SqlGetValueActivity"
 Assembly="SqlDatabaseActivities, Version=1.0.0.0, Culture=neutral,
 PublicKeyToken=2848957be3dd88b2"
 AppliesTo="all" Category="Sql" >
 </Action>
</Actions>
```

- Add the interface definition for the action as a **RuleDesigner** element embedded in the **Action** element.

*The **sentence** defines the format of the action in the **SharePoint Designer**.*

*The **FieldBind** elements map named properties in the activity to indexed tokens in the sentence.*

*The **DesignerType** attribute defines the user interface provided for choosing the value.*

```
<RuleDesigner Sentence="Set %6 using the %1 field from the %2 table in database
%5 where the %3 field equals %4">
 <FieldBind Field="ResultField" Id="1" />
 <FieldBind Field="Table" Id="2" />
 <FieldBind Field="QueryField" Id="3" />
 <FieldBind Field="QueryValue" Id="4" />
 <FieldBind Field="ConnectionString" Id="5" />
 <FieldBind Field="Result" DesignerType="ParameterNames" Id="6" />
</RuleDesigner>
```

- Define the actual types for the named parameters using a **Parameters** element immediately following the **RuleDesigner** element.

*Each field referenced in the **FieldBind** elements has a corresponding parameter element.*

*The **Name** and **Type** match the properties of the activity.*

*The **Direction** determines if the action provides or receives the value.*

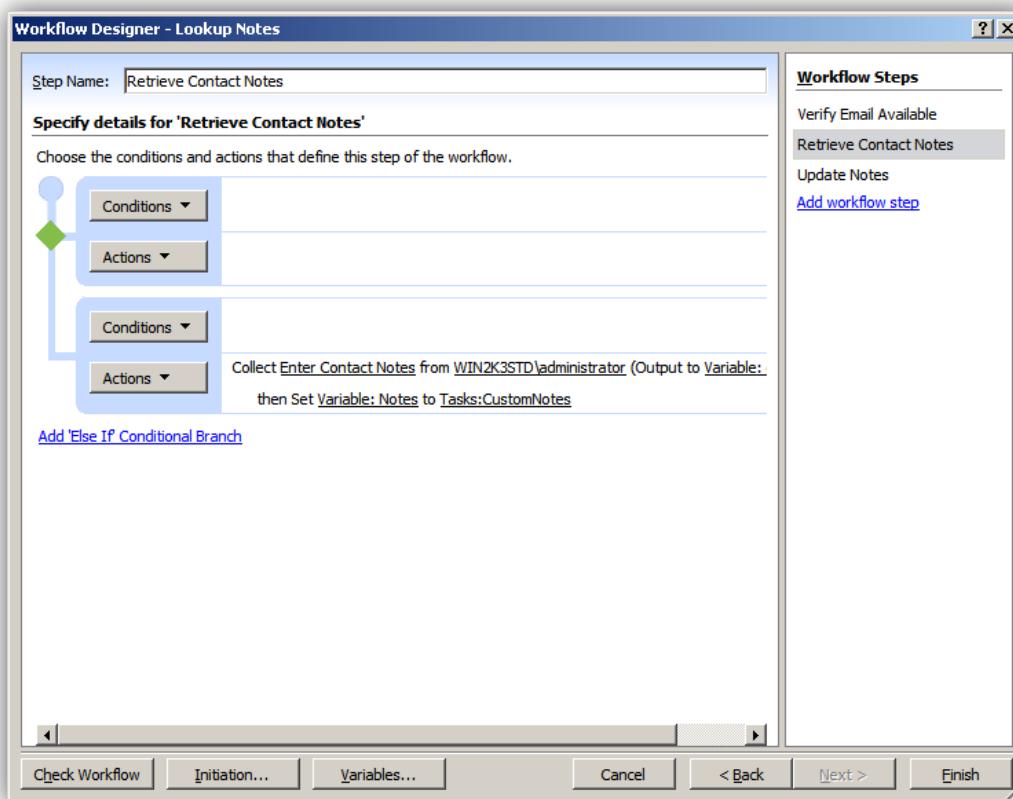
```
<Parameters>
 <Parameter Name="ConnectionString" Type="System.String, mscorelib"
 Direction="In" />
 <Parameter Name="ResultField" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="Table" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="QueryField" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="QueryValue" Type="System.String, mscorelib" Direction="In" />
 <Parameter Name="Result" Type="System.String, mscorelib" Direction="Out" />
</Parameters>
```

6) Rebuild the project in **Visual Studio 2008**.

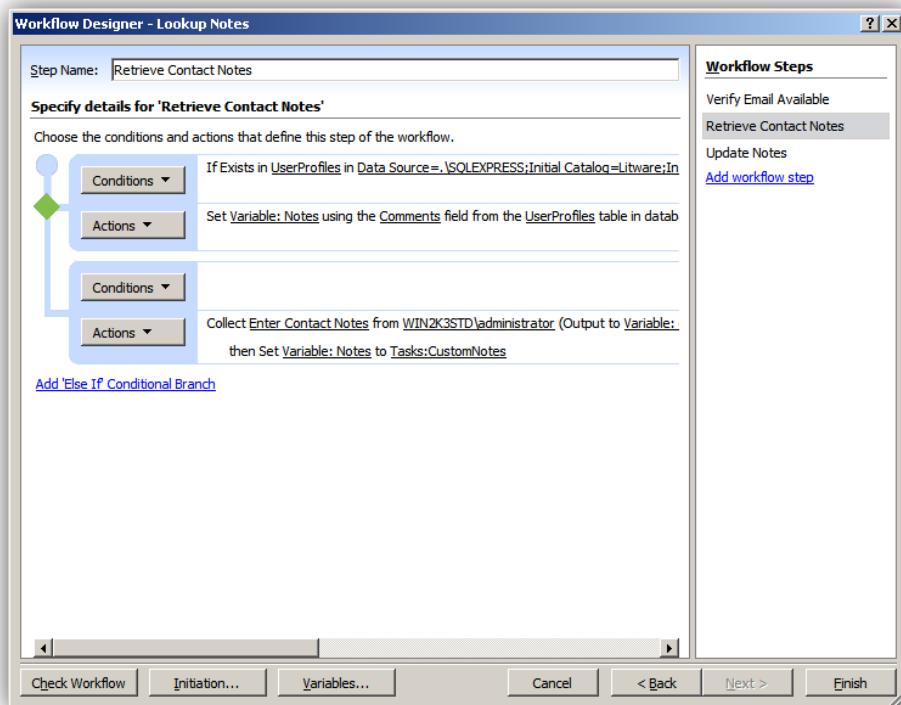
- Right click the project in the solution explorer and click **Rebuild**.

## Exercise 4: Updating SharePoint Designer Workflow using new Activities

- 1) Close and reopen the SharePoint Designer to force it to unload the **SqlDatabaseActivity** assembly.
  - Start the SharePoint Designer by clicking the **Start Button** and selecting **All Programs -> Microsoft Office -> Microsoft Office SharePoint Designer 2007**.
  - Open the **Demos** site collection by clicking **File -> Open Site** and entering a site name of <http://litwareinc.com/sites/Demos>.
- 2) Open the existing **Lookup Notes** workflow.
  - Expand the **Workflow** node in the **Folder List**.
  - Expand the **Lookup Notes** in the Folder List
  - Double click **Lookup Notes.xaml** to open the workflow designer.
- 3) Add an **Else If** conditional branch o the **Retrieve Contact Notes**
  - Select the **Retreive Contact Notes** step in the **Workflow Steps** section of the **Workflow Designer**.
  - Click the **Add 'Else If' Conditional Branch** in the **Workflow Designer**.
  - Move the new branch up by clicking its drop down menu in the upper left corner and selecting **Move Branch Up**.



- 4) Add a condition that performs a lookup into the database to determine if the email address is available.
- Add a condition to the top branch of type **Exists in database**.
- Click the **Condition** button and select **Exists in database**.  
 Click the **table** link in the condition and enter **UserProfiles**.  
 Click the **database** link in the condition and enter **Data**.  
**Source=.\SQLEXPRESS;AttachDbFilename=C:\ Labs\Files\Litware.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True.**  
 Click the **field** link in the condition and enter **Email**.  
 Click the **value** link in the action and click the **Fx** button.  
 In the **Lookup Details** section's **Source** drop down list, select the **Current Item** item.  
 In the **Lookup Details** section's **Field** drop down list, select **E-mail Address** and click **OK**.*
- 5) Add an action that will perform the lookup in the database and store the notes in the **Notes** variable.
- Add an action to the top branch of type **Sql Value Lookup**.
- Click the **Actions** button and select **More actions**.  
 In the **Workflow Actions** dialog, select the **Sql** category.  
 Select the **Sql Value Lookup** action from the list and click **Add**.  
 Click the **first** link in the action and select the **Variable: Notes** item.  
 Click the **second** link in the action and set the value to **Comments**.  
 Click the **third** link in the action and set the value to **UserProfiles**.  
 Click the **fourth** link in the action and set the value to **Data**.  
**Source=.\SQLEXPRESS;AttachDbFilename=C:\ Labs\Files\Litware.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True.**  
 Click the **fifth** link in the action and set the value to **Email**.  
 Click the **sixth** link in the action and click the **Fx** button.  
 In the **Lookup Details** section's **Source** drop down list, select the **Current Item** item.  
 In the **Lookup Details** section's **Field** drop down list, select **E-mail Address** and click **OK**.*



- 6) Save the workflow by clicking the **Finish** button in the **Workflow Designer**.
- 7) Test the workflow in the browser by creating a new contact with an email of **Administrator@litwareinc.com** and starting the **Lookup Notes** workflow.
  - Navigate to the **Contacts** list using **Internet Explorer**.
 

*Navigate to <http://litwareinc.com/sites/Demos>.*

*Click the **Contacts** link on the left hand navigation bar.*
  - Create a new contact with an email address of **Administrator@litwareinc.com**.
  - Start the workflow on the new contact.
 

*Click **Workflows** in the item's drop down menu and start a workflow of type **Lookup Notes**.*

*On the instantiation page, click **Start**.*
  - On the **Contacts** list page, click the **Completed** link under **Lookup Notes** to see the workflow status.
  - Verify that the workflow is completed and that the contact's notes are now populated using the value in the database.

Last Name *	John Doe
First Name	
Full Name	
E-mail Address	Administrator@litwareinc.com
Company	
Job Title	
Business Phone	
Home Phone	
Mobile Phone	
Fax Number	
Address	
City	
State/Province	
ZIP/Postal Code	
Country/Region	
Web Page	Type the Web address: ( <a href="#">Click here to test</a> ) <input type="text" value="http://"/> Type the description: <input type="text"/>
Notes	 This is the administrator