# 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

### 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**.

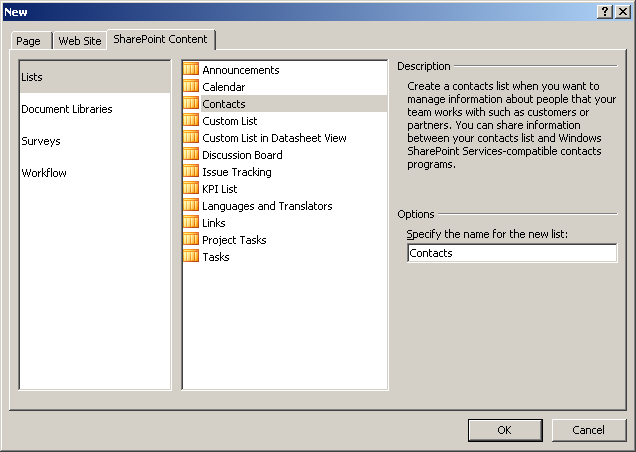
#### Open the **Demos** site collection by clicking **File -> Open Site** and entering a site name of **http://litwareinc.com/sites/Demo**.

### 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**.

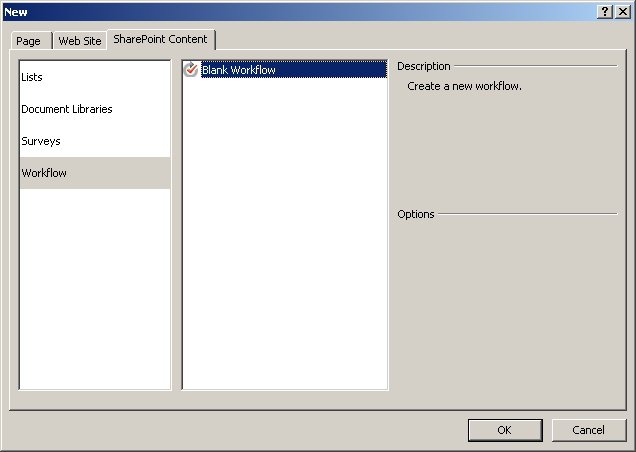


### Add a new workflow named **Lookup Notes** to the **Contacts** list.

#### 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.

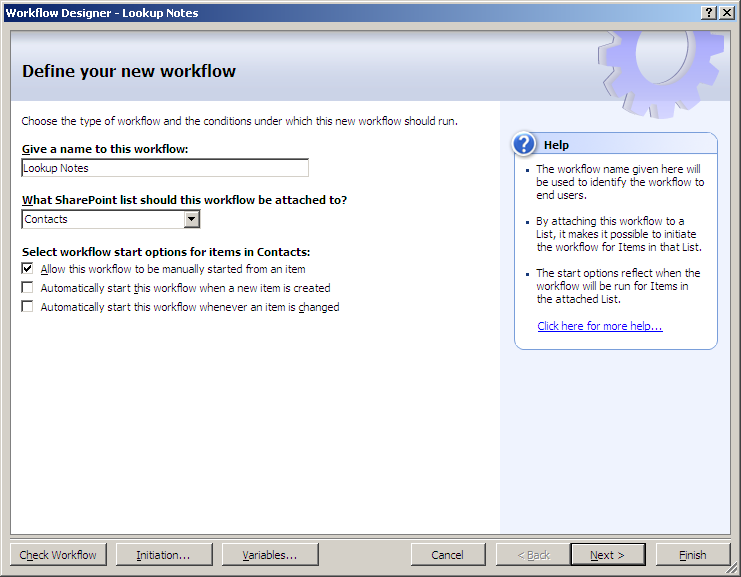


### Name the new workflow **Lookup Notes** and attach the new workflow to the Con**t**acts lists.

#### 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.



### 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**.

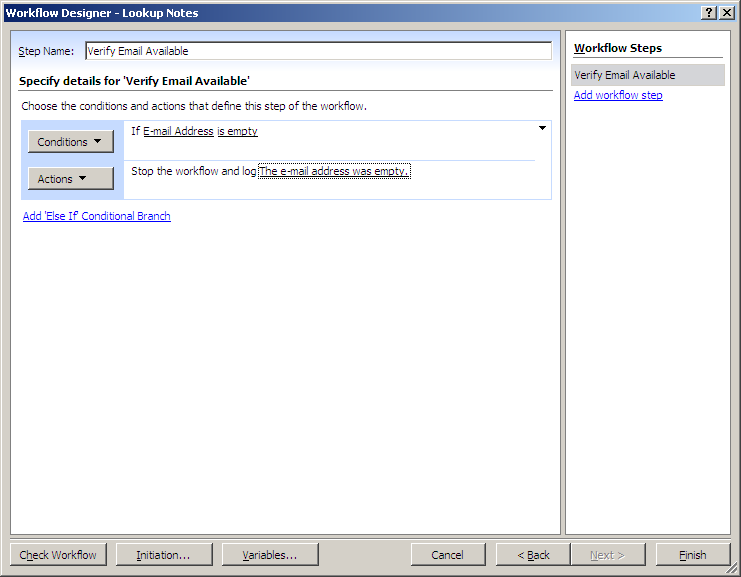
#### Add an action that of **Stop Workflow** to terminate the workflow.

##### 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.**”



### 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**.

### 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.

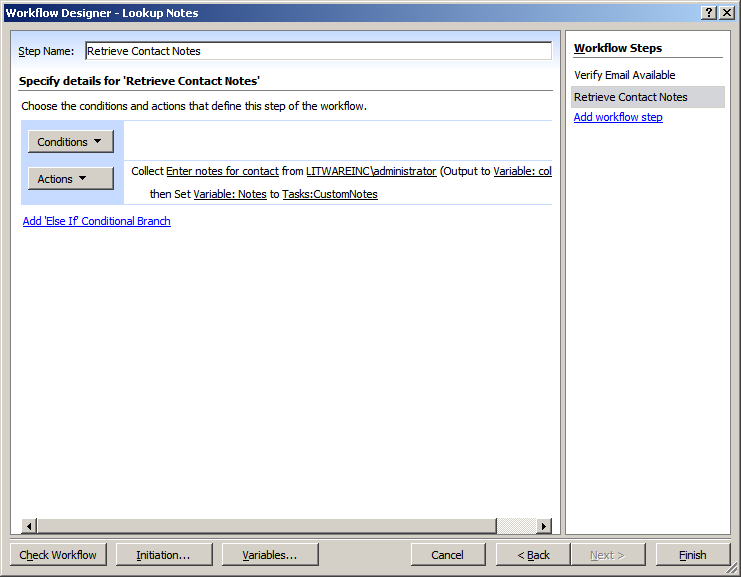
##### In the **Lookup Details** section’s **Field** drop down list, select **CustomNotes**.

##### 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.



### 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**.

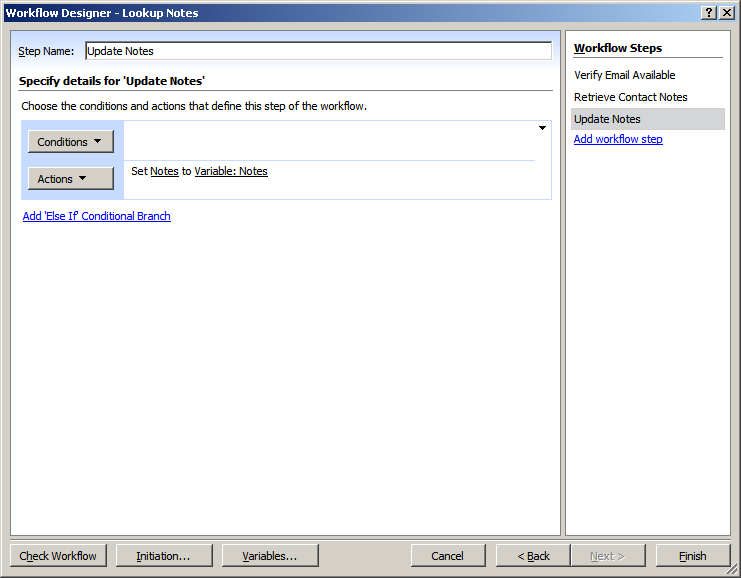
##### Click the **field** link in the action and select the **Notes** field.

##### 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.



### Save the workflow by clicking the **Finish** button in the Workflow Designer.

### 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.

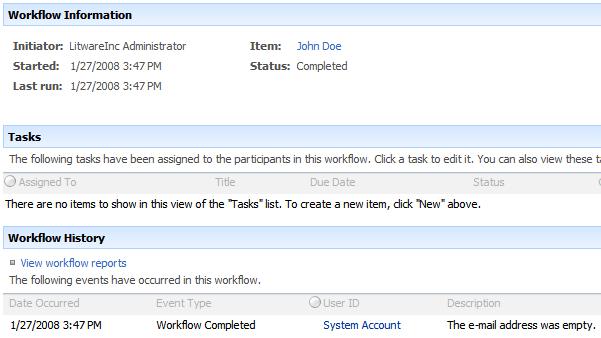
#### Start the workflow on the new contact.

##### 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.



### 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**.

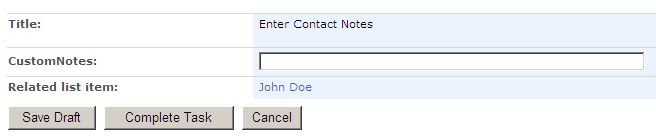
#### On the **Contacts** list page, click the **In Progress** link under **Lookup Notes** to see the workflow status.

#### Approve the notes task

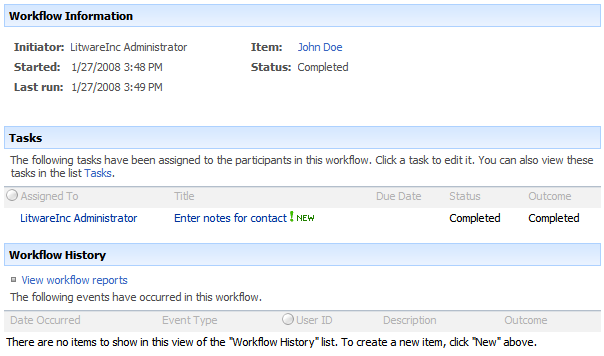
##### Click the **Edit Item** menu item on the task’s title.

##### Enter notes for the current contact.

##### Click **Complete Task** to approve the task.



#### Verify the workflow is completed and all of the tasks are completed as well.



## Exercise 2: Creating a custom SharePoint Designer condition

### 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**.

### 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);

### 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 **Functioname**, **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, mscorlib" Direction="In" />

<Parameter Name="\_2\_" Type="System.String, mscorlib" Direction="In" />

<Parameter Name="\_3\_" Type="System.Object, mscorlib" Direction="In" />

<Parameter Name="\_4\_" Type="System.String, mscorlib" Direction="In" />

</Parameters>

</Condition>

</Conditions>

</WorkflowInfo>

### 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" />

### 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\iisapp.vbs /a "SharePointDefaultAppPool" /r

## Exercise 3: Creating a custom SharePoint Designer activity

### 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**.

### 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); }

}

### Add a new **SqlConnection** activity to the **SqlGetValueActivity** designer.

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

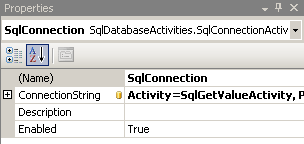
#### Drag a new **SqlConnection** activity onto the canvas and name it **SqlConnection**.

##### 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.



### Define the query embedded in the new **SqlConnection** activity.

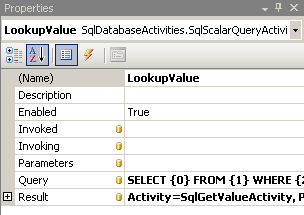
#### Select the **SqlScalarQueryActivity** inside the **SqlConnection** activity and change its name to **LookupValue**.

##### 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)

};

}

### 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, mscorlib"

Direction="In" />

<Parameter Name="ResultField" Type="System.String, mscorlib" Direction="In" />

<Parameter Name="Table" Type="System.String, mscorlib" Direction="In" />

<Parameter Name="QueryField" Type="System.String, mscorlib" Direction="In" />

<Parameter Name="QueryValue" Type="System.String, mscorlib" Direction="In" />

<Parameter Name="Result" Type="System.String, mscorlib" Direction="Out" />

</Parameters>

### 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

### 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**.

### Open the existing **Lookup Notes** workflow.

#### Expand the **Workflow** node in the **Folder List**.

#### Expand the Lookup Notes in the Folder List

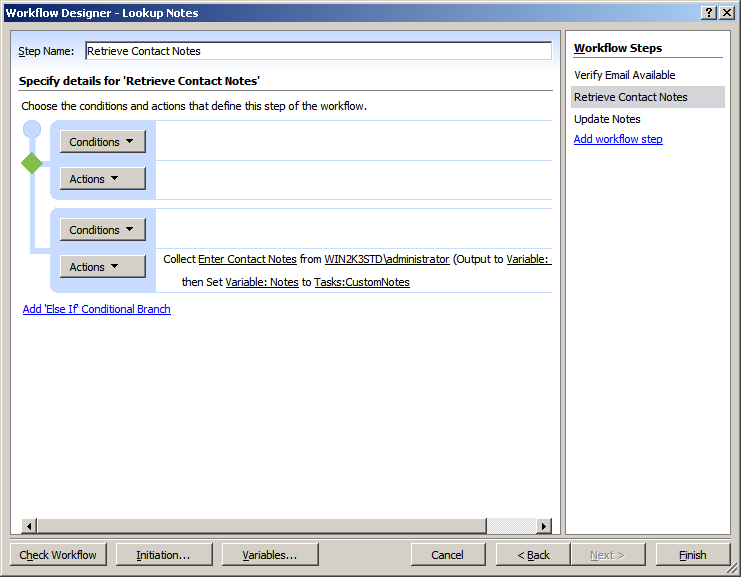
#### Double click **Lookup Notes.xoml** to open the workflow designer.

### 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**.



### 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**.

### 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**.

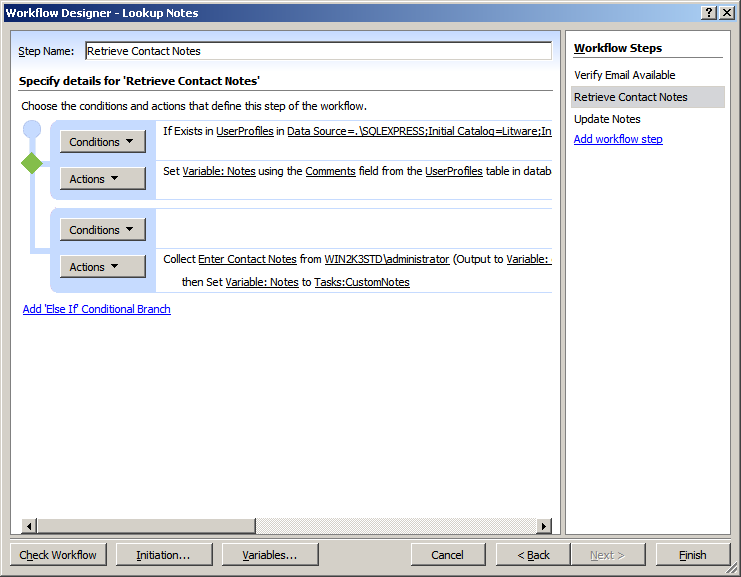
##### Click the **fourth** link in the action and set the value to **Data. Source=.\SQLEXPRESS;AttachDbFilename=C:\Labs\Files\Litware.mdf;Integrated Security=True;Connect Timeout=30;User Instance=True**.

##### 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**.



### Save the workflow by clicking the **Finish** button in the **Workflow Designer**.

### 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.

