# 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

### Open the starter **Visual Studio 2008** solution at **\Labs\Lab02\StarterFiles\SequentialWorkflow.sln**.

### 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

### 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>();

}

### 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.Write ("Enter the customer’s name: ");

NewCustomer.Name = Console.ReadLine();

### 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.

### 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.Write("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;

}

### 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 to 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.

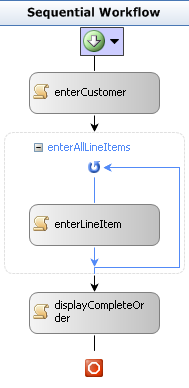
Console.WriteLine("Review Order");

Console.WriteLine(" – Customer Name: {0}", NewCustomer.Name);

Console.WriteLine(" – Line Items");

foreach (LineItem lineItem in NewOrder.LineItems)

Console.WriteLine(" - {0}", lineItem.ProductId);



### Test order entry by running the workflow application and testing the order entry.

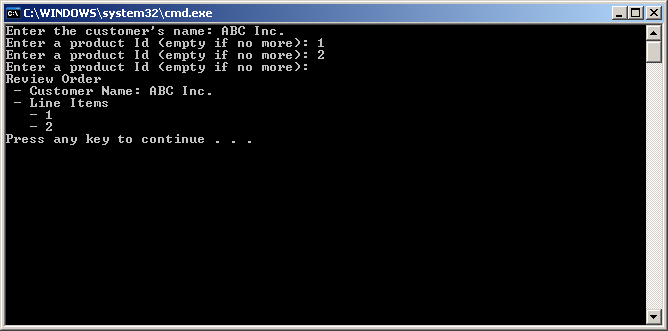
#### Click **Debug -> Start Without Debugging** in the menu bar.

#### 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.



## Exercise 2: Approving the order with an IfElse activity

### 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.Write("Would you like to submit this order (yes/no)?: ");

OrderApproved = (Console.ReadLine() == "yes");

### 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 theproperties pane.

##### Click the right branch and set the **Name** property to **ifCancelOrder** in the properties pane.

### 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**.

### 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.

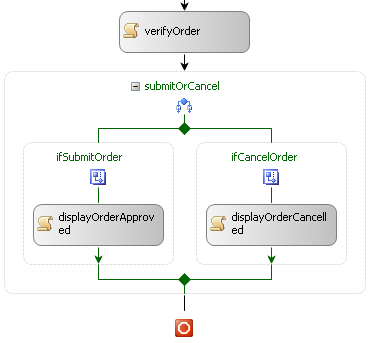
#### Drag a **Code** activity from the toolbox onto the canvas into the **ifCancelOrder** branch.

##### 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.Write("The order was cancelled");



### 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**.

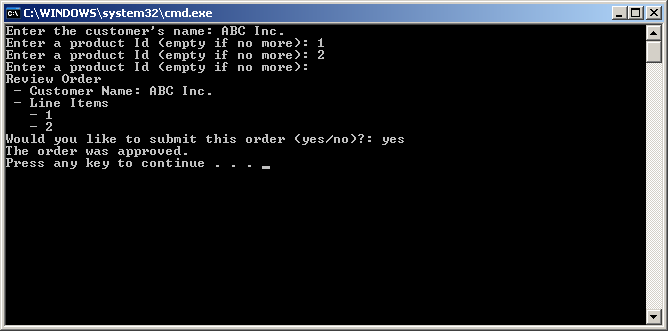
#### When asked for a product id, use an id of **1** for the first time and **2** for the second time.

#### 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

### 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 C**reate Property i**s 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.

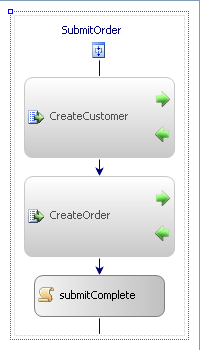
#### Drag a **Code** activity onto the canvas following the **createOrder** activity.

##### 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**.

### Embed a new **SubmitOrder** activity into the **ifSubmitOrder** if/else branch in the **EnterOrder** workflow.

#### Open the **EnterOrder** workflow in design mode

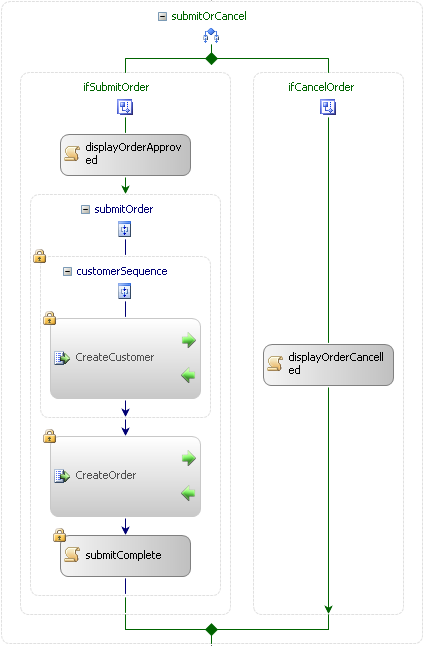
##### Right click **EnterOrder.cs** in the solution explorer and select **View Designer**.

#### 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.



### 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.

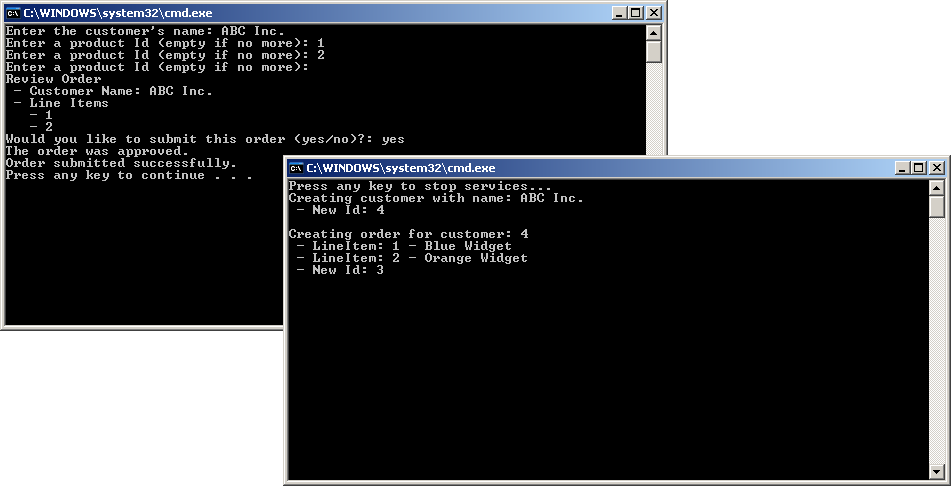
#### Once the application has started, enter a customer name as **ABC Inc**.

#### 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.



### 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

### 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.

### 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

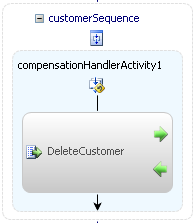
##### In the properties pane, set the **Name** property to **deleteCustomer**.

##### 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.



### 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**.

### 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. ");

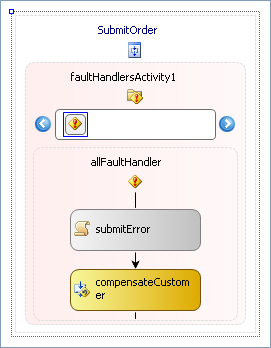
### Cause the compensation handler for the **createCustomer** activity to fire and delete the customer

#### 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**.



### 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.

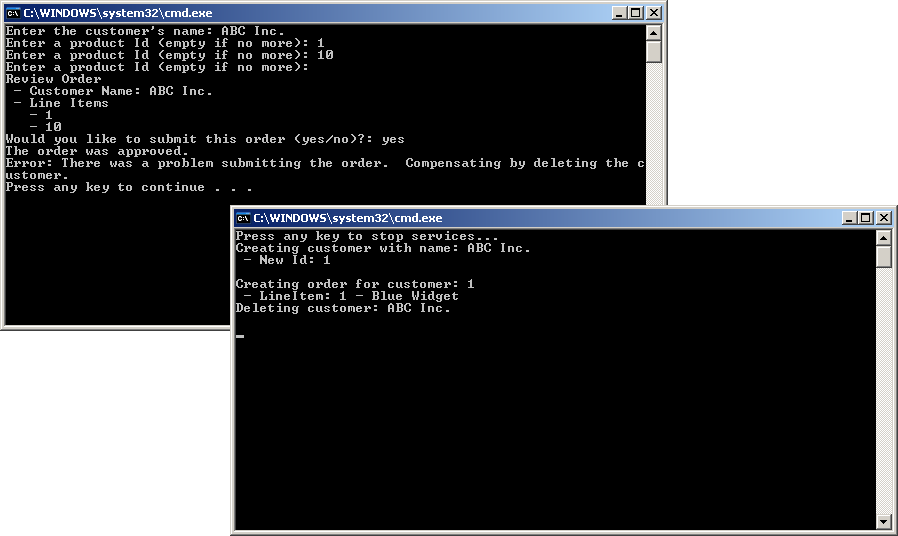
#### Once the application has started, enter a customer name as **ABC Inc**.

#### 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

### 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**.

### 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**.

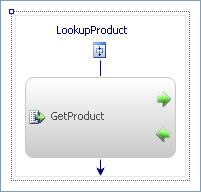
##### Set the **ServiceOperationInfo** property using the **OrderService.GetProduct** method.

##### 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.



### 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**.

### 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**.

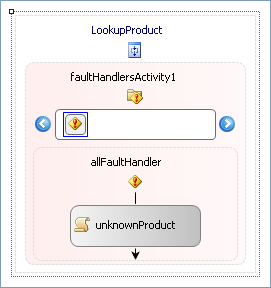
##### Double click the activity in the designer to generate the code handler.

#### 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**.

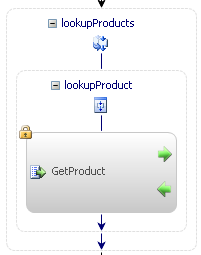
#### Implement the **lookupProducts ChildInitialized** handler.

##### 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;



### 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);

### 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.

