Chapter 10

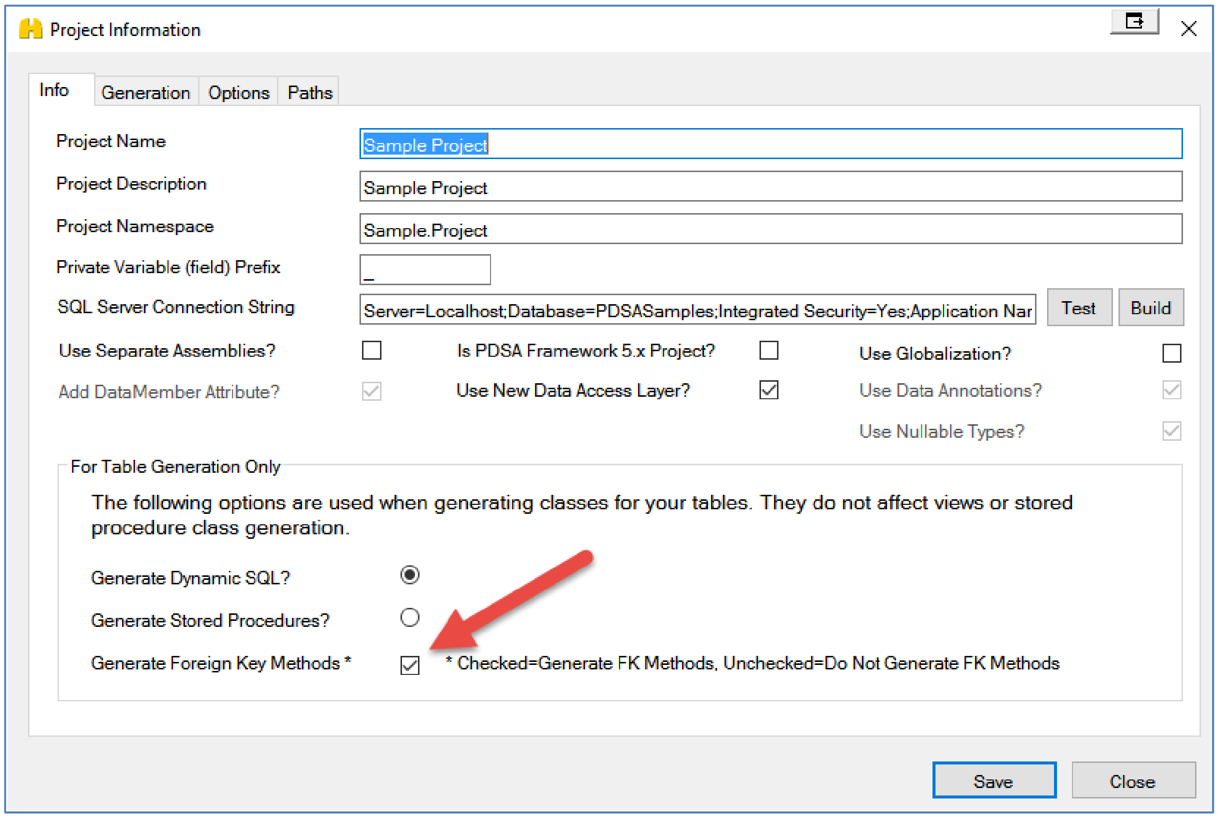
Relationships Using the Data Access Layer

Table of Contents

At some point you will need to work with relationships between tables. This chapter will help you with working with relationships within your .NET applications.

Create Child Table References

Haystack has the ability to generate references to child tables when you choose the "Generate Foreign Key Methods" check box on the Project Information setup screen as shown in Figure 1.



Project Name	Sample Project
Project Description	Sample Project
Project Namespace	Sample.Project
Private Variable (field) Prefix	_
SQL Server Connection String	Server=Localhost;Database=PDSASamples;Integrated Security=Yes;Application Nar

Use Separate Assemblies?	☐	Is PDSA Framework 5.x Project?	☐	Use Globalization?	☐
Add DataMember Attribute?	✓	Use New Data Access Layer?	✓	Use Data Annotations?	✓
				Use Nullable Types?	✓

For Table Generation Only

The following options are used when generating classes for your tables. They do not affect views or stored procedure class generation.

Generate Dynamic SQL?	●
Generate Stored Procedures?	○
Generate Foreign Key Methods *	✓

* Checked=Generate FK Methods, Unchecked=Do Not Generate FK Methods

Figure 1: Select the Generate Foreign Key Methods check box

Relationships in PDSASamples Database

In the PDSASamples database there are three related tables; Customer, OrderHeader and OrderLineItem as shown in Figure 2. A Customer can have one or more order header records. An Order Header can have one or more order line item records. This is a classic foreign key relationship paradigm in a relational database.

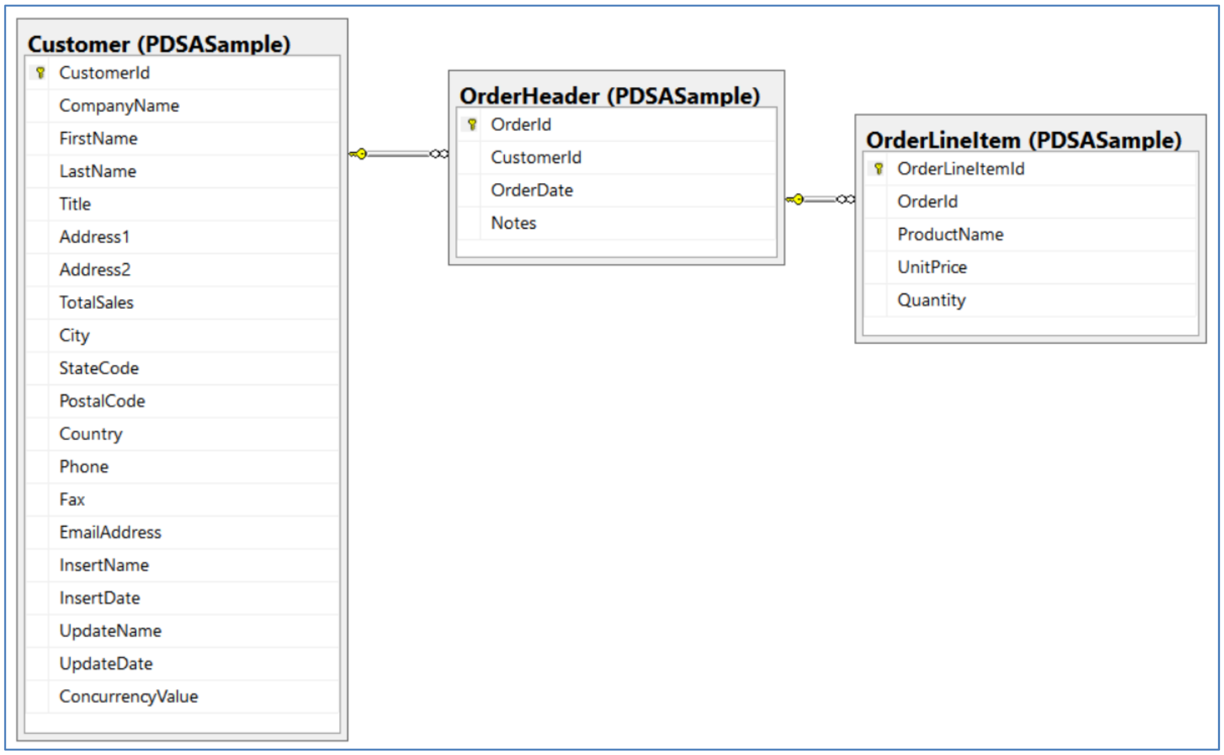


Figure 2: Customer, OrderHeader and OrderLineItem have FK relationships

Generated Relationship Code

When you generate the code in Haystack for the above relationship, the following items that are added in addition to your normal generated code.

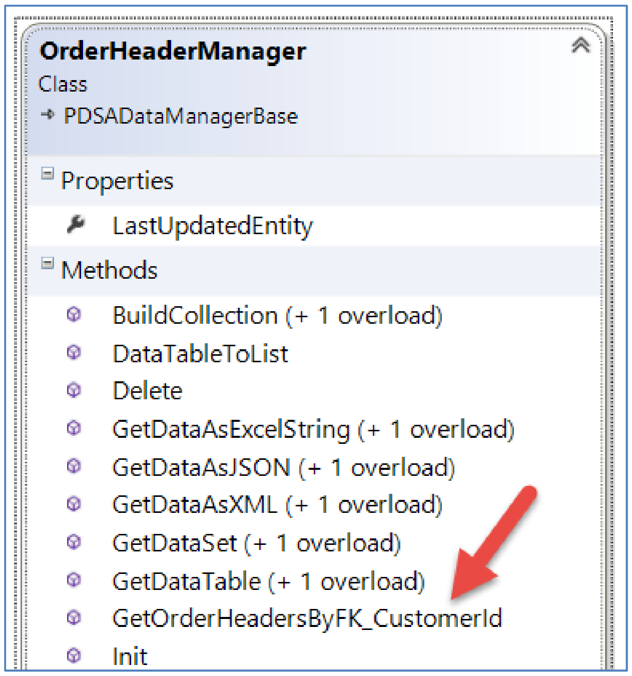
NOTE: The method and property names generated in Table 1 are for these particular tables. The names generated for your classes will be different. We infer the method names based on each foreign key index name. If you name your FK indexes with a good name, that helps create a more readable method name.

File	What is Added
OrderHeaderManager.cs	A method named **GetOrderHeadersByFK_CustomerId** is generated to load a collection of OrderHeader objects based on the customer id passed into this method. See Figure 3.
OrderLineItemManager.cs	A method named **GetOrderLineItemsByFK_Orders** is generated to load a collection of OrderLineItem objects based on the order id passed into this method.
CustomerViewModel.cs	A property named **OrderHeaderCollection** is a generic list used to store a collection of OrderHeader objects. See Figure 4.
	A method name **LoadOrderHeaderCollection** is created to load the **OrderHeaderCollection** property if no values exist in the collection. See Figure 4.
OrderHeaderViewModel.cs	A property name **OrderLineItemCollection** is a generic list used to

	store a collection of OrderLineItem objects.
	A method name **LoadOrderLineItemCollection** is created to load the **OrderLineItemCollection** property if no values exist in the collection.

Table 1: Methods generated by Haystack to support relationships

These generated methods and properties should give you a good start when you have to deal with relationships in your application.


OrderHeaderManager
Class
→ PDSADataManagerBase

Properties
- LastUpdatedEntity

Methods
- BuildCollection (+ 1 overload)
- DataTableToList
- Delete
- GetDataAsExcelString (+ 1 overload)
- GetDataAsJSON (+ 1 overload)
- GetDataAsXML (+ 1 overload)
- GetDataSet (+ 1 overload)
- GetDataTable (+ 1 overload)
- GetOrderHeadersByFK_CustomerId
- Init

Figure 3: Manager Class has a foreign key method generated

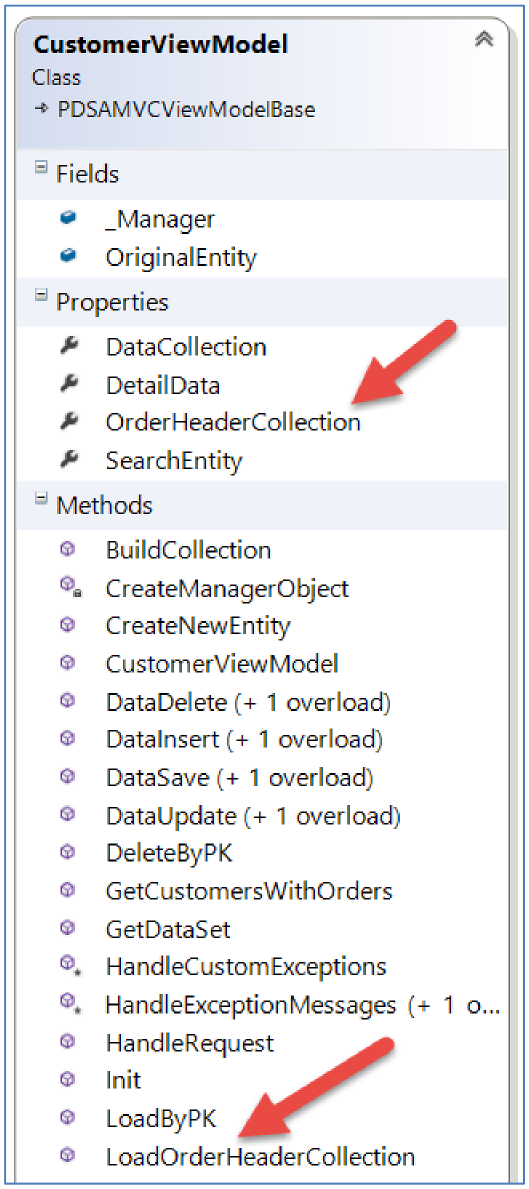


Figure 4: View Model class will have a property for child tables and a method to load those child objects.

One to Many Relationships in View Model

When you generate either using the WPF or MVC templates, any “Parent” view model classes will have a property and method that can hold a relationship to “Child” objects. We like putting relationships in View Model (Figure 3) classes instead of within Entity classes as this keeps your Entity classes clean and is better for serializing the Entity to any front-end application. Of course, you are free to add these relationships to your Entity classes if you wish as we explain in the next section.

The following two samples are pure generated code from Haystack. Take a look at the OrderHeaderViewModel.cs class to see the relationship properties and methods as outlined in Table 1.

WPF Sample

Sample: DataAccessLayer-General\Relationships-MVC\Relationships.sln

MVC Sample

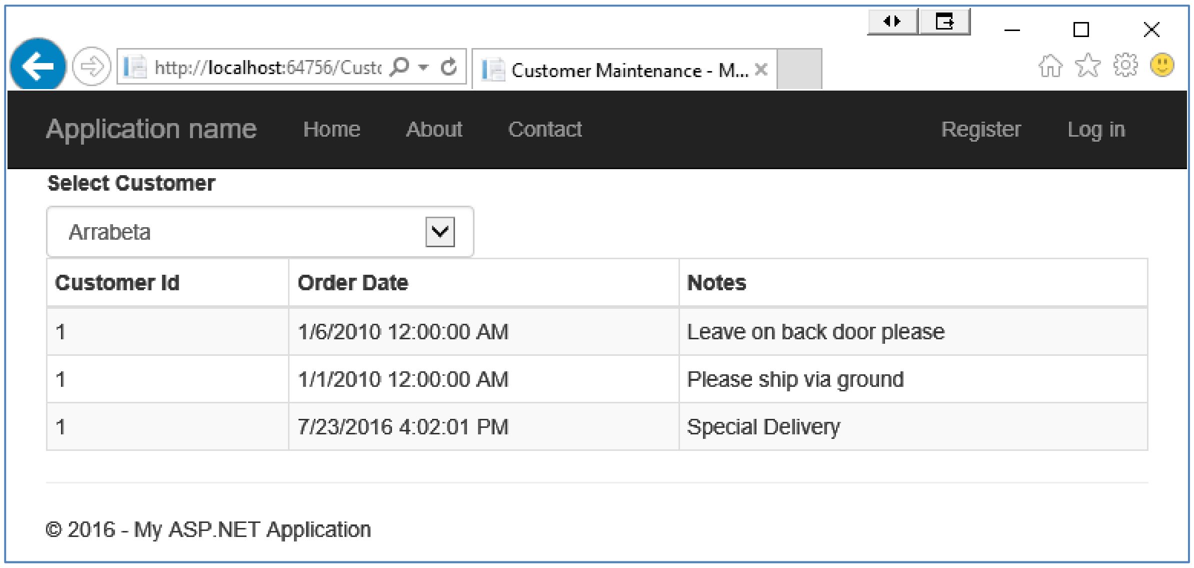
Sample: DataAccessLayer-General\Relationships-WPF\Relationships.sln

Load Child Tables using View Model

In this sample you will write code to load only those customers that have orders and display those customers in an MVC DropDown list. Once you select a customer, you will display the orders for those customers in a HTML table as shown in Figure 5.

MVC Sample

Create a new MVC web application and add the generated code and the appropriate DLLs.



Customer Id	Order Date	Notes
1	1/6/2010 12:00:00 AM	Leave on back door please
1	1/1/2010 12:00:00 AM	Please ship via ground
1	7/23/2016 4:02:01 PM	Special Delivery

© 2016 - My ASP.NET Application

Figure 5: Selecting orders for a specific customer

Let’s start with building a method in the CustomerManager.cs class to return a list of only those customers that have orders.

```
public List<Customer> GetCustomersWithOrders() {
  List<Customer> ret = new List<Customer>();

  SqlBuilder.Init();
  SqlBuilder.SelectWhereItems.Clear();
  SqlBuilder.SQLSelect = "SELECT * FROM PDSASample.Customer
             WHERE CustomerId IN (SELECT CustomerId
                FROM PDSASample.OrderHeader)";

  ret = BuildCollection();

  return ret;
}
```

Next, open the CustomerViewModel.cs class and add a method to load call the previous method you just wrote and populate the DataCollection property of the CustomerViewModel class with the list of customers. If you get customers back, then you will also set the DetailData property with the first Customer object in the collection.

```
public void GetCustomersWithOrders() {
  CustomerManager mgr = new CustomerManager();

  DataCollection = mgr.GetCustomersWithOrders();

  if (DataCollection.Count > 0) {
    if (DetailData.CustomerId == null) {
      DetailData = DataCollection[0];
    }

    LoadOrderHeaderCollection();
  }
}
```

Create a new folder under the \Views folder called Customer. Create a new view in that folder called Customers.cshtml. Add the code shown below to create the page that looks like Figure 5.

```
@model Relationships.CustomerViewModel

@{
  ViewBag.Title = "Customer Maintenance";
}

@using (Html.BeginForm()) {
  <div class="row">
    <div class="col-md-12">
      @Html.LabelFor(m => m.DetailData.CustomerId,
         "Select Customer")
      @Html.DropDownListFor(m => m.DetailData.CustomerId,
         new SelectList(Model.DataCollection, "CustomerId",
             "CompanyName"),
              new { @class = "form-control",
                    onchange = "loadOrders();" })
    </div>
  </div>

  <table class="table table-condensed table-bordered
                table-striped table-hover">
    <thead>
      <tr>
        <th>Customer Id</th>
        <th>Order Date</th>
        <th>Notes</th>
      </tr>
    </thead>
    <tbody>
      @foreach (var item in Model.OrderHeaderCollection) {
        <tr>
          <td>@item.CustomerId</td>
          <td>@item.OrderDate</td>
          <td>@item.Notes</td>
        </tr>
      }
    </tbody>
  </table>
}

@section scripts {
  <script type="text/javascript">
    function loadOrders() {
      $("form").submit();
    }
  </script>
}
```

Add a new controller under the \Controllers folder in your project named CustomerController.cs. Add the following two methods.

```
[HttpGet]
public ActionResult Customer() {
  CustomerViewModel vm = new CustomerViewModel();
  ActionResult ret = View(vm);

  vm.GetCustomersWithOrders();

  return ret;
}
```

```
[HttpPost]
public ActionResult Customer(CustomerViewModel vm) {
  ActionResult ret = View(vm);

  vm.GetCustomersWithOrders();
  vm.LoadOrderHeaderCollection();

  return ret;
}
```

If you now run the project you should see a list of customers in the drop down list and if you select a new customer from the list you should see their list of orders in an HTML table.

Load Child Tables Using Property in Entity Class

If you do not mind a strong-coupling between your Manager classes and your Entity classes you can add properties and methods to load and hold data from various tables in each different entity and manager classes. For instance, in your Customer class you can have an OrderHeaderCollection property (Figure 6) to hold a list of OrderHeader objects. In your CustomerManager class you create a method named LoadOrdersIntoCustomerObject() to which you pass in a Customer entity object that you wish to populate its OrderHeaderCollection with the OrderHeader objects for that customer.

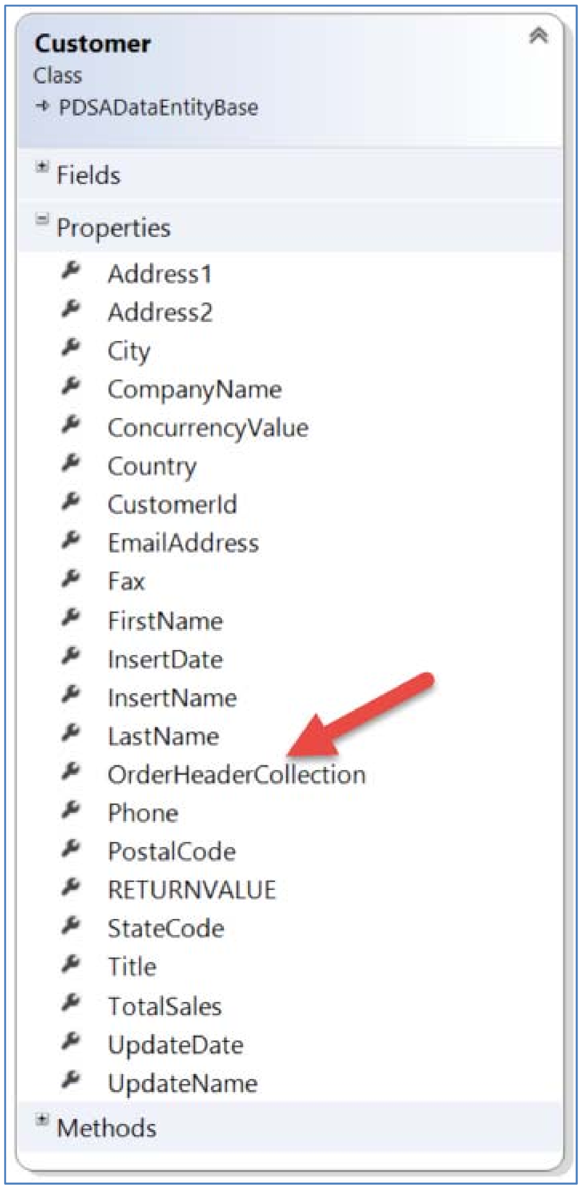
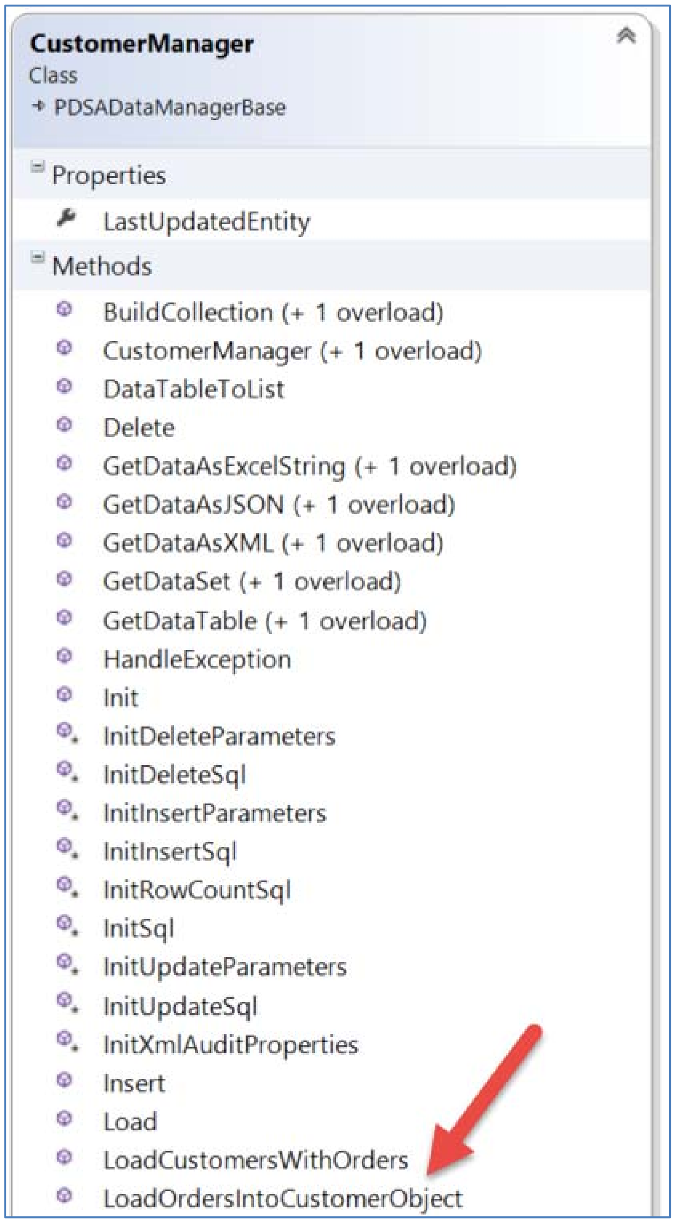


Figure 6: You can add child table properties directly to your Entity classes if you want.


CustomerManager
Class
→ PDSADataManagerBase

Properties
- LastUpdatedEntity

Methods
- BuildCollection (+ 1 overload)
- CustomerManager (+ 1 overload)
- DataTableToList
- Delete
- GetDataAsExcelString (+ 1 overload)
- GetDataAsJSON (+ 1 overload)
- GetDataAsXML (+ 1 overload)
- GetDataSet (+ 1 overload)
- GetDataTable (+ 1 overload)
- HandleException
- Init
- InitDeleteParameters
- InitDeleteSql
- InitInsertParameters
- InitInsertSql
- InitRowCountSql
- InitSql
- InitUpdateParameters
- InitUpdateSql
- InitXmlAuditProperties
- Insert
- Load
- LoadCustomersWithOrders
- LoadOrdersIntoCustomerObject

Figure 7: Add method to populate child objects into Entity class

Modify Generated Code

Open Customer.cs

Add the following property

```
public List<OrderHeader> OrderHeaderCollection { get; set; }
```

Open CustomerManager.cs

Add the following method to load just those customers that have orders.

```
public List<Customer> LoadCustomersWithOrders() {
  List<Customer> ret = new List<Customer>();

  // Clear WHERE clauses
  SqlBuilder.SelectWhereItems.Clear();
  // Build SQL Statement To Get Customers with Orders
  SqlBuilder.SQLSelect = "SELECT * FROM PDSASample.Customer ";
  SqlBuilder.SQLSelect += "WHERE CustomerId IN
        (SELECT CustomerId FROM PDSASample.OrderHeader)";
  // Build SQL Statement
  SqlBuilder.BuildSQLForSelect();

  // Build Collection of Customers
  ret = BuildCollection();

  return ret;
}
```

Add another method to populate OrderHeader objects into a Customer entity object.

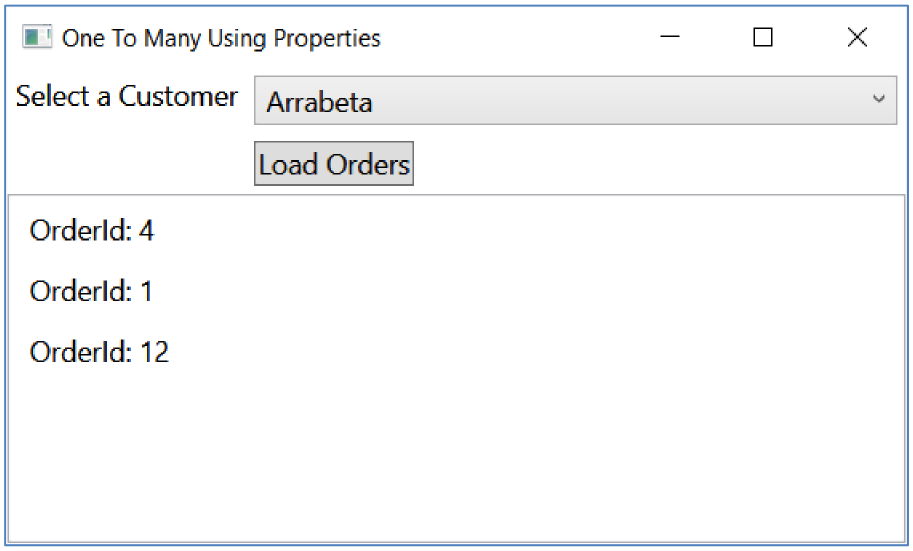
```
public void LoadOrdersIntoCustomerObject(Customer entity) {
  OrderHeaderManager mgr = new OrderHeaderManager();

  try {
    entity.OrderHeaderCollection =
       mgr.GetOrderHeadersByFK_CustomerId(entity);
  }
  catch (Exception ex) {
    LastException = ex;
    Message = ex.ToString();
  }
}
```

WPF Sample

Create a new WPF application, or just follow along with the following sample.

Sample: DataAccessLayer-General\ Relationships-OneToMany-Property\OneToMany-Property.sln



```
OrderId: 4
OrderId: 1
OrderId: 12
```

Figure 8: Testing out the relationships in the entity class

Open MainWindow.xaml

Add the XAML shown below to the window to make it look like that shown Figure 8.

```
<Window x:Class="OneToMany.MainWindow"
    // NORMAL NAMESPACE STUFF HERE
        Loaded="Window_Loaded"
        FontSize="14"
        Title="One To Many Using Properties">
  <Window.Resources>
    <Style TargetType="TextBlock">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="Button">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="ComboBox">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="ListBox">
      <Setter Property="Margin"
              Value="4" />
    </Style>
  </Window.Resources>
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition />
    </Grid.RowDefinitions>
    <TextBlock x:Name="textBlock"
               TextWrapping="Wrap"
               Text="Select a Customer" />
    <ComboBox x:Name="comboBox"
              Grid.Column="1"
              DisplayMemberPath="CompanyName"
              SelectedValuePath="CustomerId" />
    <Button x:Name="button"
            Content="Load Orders"
            HorizontalAlignment="Left"
            Grid.Column="1"
            Grid.Row="1"
            Click="button_Click" />
    <ListView x:Name="listBox"
              Grid.ColumnSpan="2"
              Grid.Row="2" />
  </Grid>
</Window>
```

Add a Window_Loaded event to the WPF window.

```
private void Window_Loaded(object sender, RoutedEventArgs e) {
  CustomerManager mgr = new CustomerManager();

  comboBox.ItemsSource = mgr.LoadCustomersWithOrders();
}
```

Add a Click event to the button.

```
private void button_Click(object sender, RoutedEventArgs e) {
  Customer entity = null;
  CustomerManager mgr = new CustomerManager();

  entity = (Customer)comboBox.SelectedItem;
  if (entity != null) {
    mgr.LoadOrdersIntoCustomerObject(entity);
    listBox.ItemsSource = entity.OrderHeaderCollection;
  }
}
```

Run the WPF application and see the orders appear.

Combo Box on Edit Page for Foreign Key Tables

When you generate a WPF user control and the table for which you are generating has a FK to another table, a combo box will be generated and loaded with the data from that table automatically. For an example of this, load and generate the Customer, OrderHeader and OrderLineItem tables from Haystack. Drag and drop the ucOrderHeaderGrid user control onto the Main Window. Run the sample and you should see a combo box appear, loaded with customer objects, when you go into either Add or Edit mode.

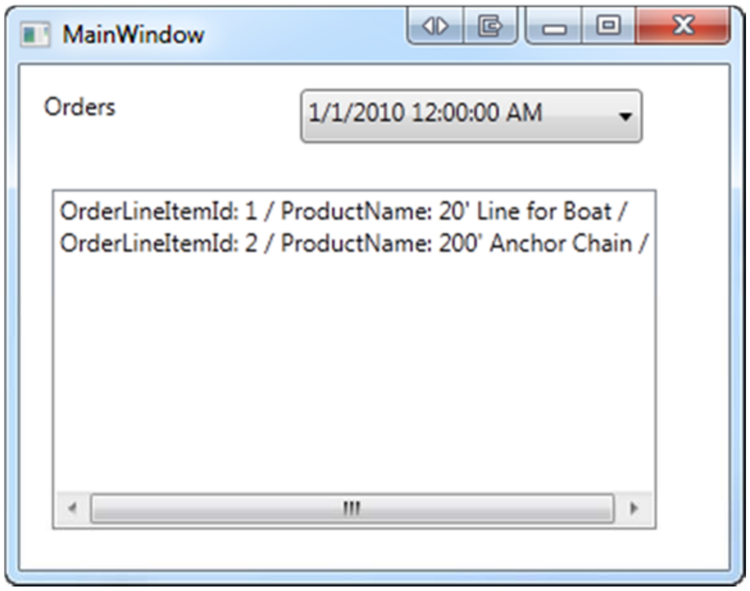
WPF Sample

Sample: DataAccessLayer-General\Relationships-WPF\Relationships.sln

Load Child Tables in WPF Using Manager Class

In this sample you will use the PDSASamples database that comes with Haystack to display all Order Line Items for a specific Order. You will place a Combo Box and a List Box control on a WPF Window and hook them together with an ObjectDataProvider. Your final version will look Figure 9.

Sample: DataAccessLayer-General\ Relationships-OneToMany-ComboList\OneToMany-ComboList.sln



```
OrderLineItemId: 1 / ProductName: 20' Line for Boat /
OrderLineItemId: 2 / ProductName: 200' Anchor Chain /
```

Figure 9: Selecting Line Items for an Order

Create a new WPF Application.

Add the appropriate Haystack DLLs.

Add all the generated code from the WPF folder.

On the MainWindow.xaml add a TextBlock

Set the Text property to “Orders”

Add a ComboBox control to the right of the TextBlock.

Add a ListView below the TextBlock and ComboBox

Go to the XAML view of MainWindow and in the <Window element add the following XAML namespaces.

NOTE: Use whatever namespaces you used when generating the classes with Haystack. The namespace I used was “Sample.Project”.

```
xmlns:mgr="clr-namespace:Sample.Project"
xmlns:entity="clr-namespace:Sample.Project"
```

Add a <Window.Resources> element just below the <Window…> and before the <Grid> element:

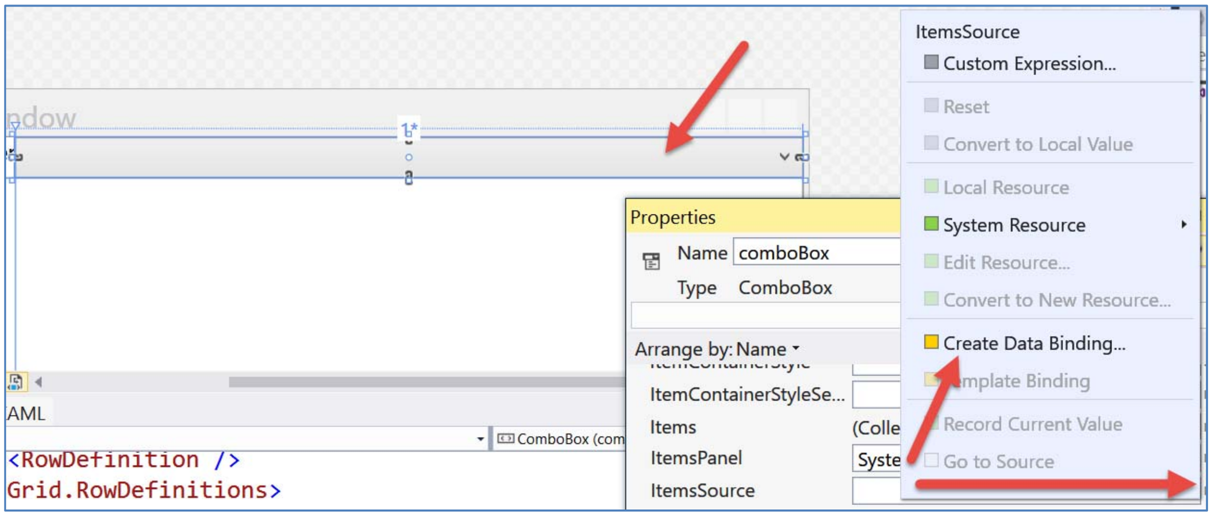
```
<Window.Resources>
  <ObjectDataProvider x:Key="odpOrders"
                      ObjectType="{x:Type mgr:OrderHeaderManager}"
                      MethodName="BuildCollection" />

  <ObjectDataProvider x:Key="odpLineItems"
                      ObjectType="{x:Type
                            mgr:OrderLineItemManager}"
                      MethodName="GetOrderLineItemsByFK_Orders">
    <ObjectDataProvider.MethodParameters>
      <entity:OrderHeader />
    </ObjectDataProvider.MethodParameters>
  </ObjectDataProvider>
</Window.Resources>
```

Build the Project.

Click once on the ComboBox control to give it focus and bring up the Properties window.

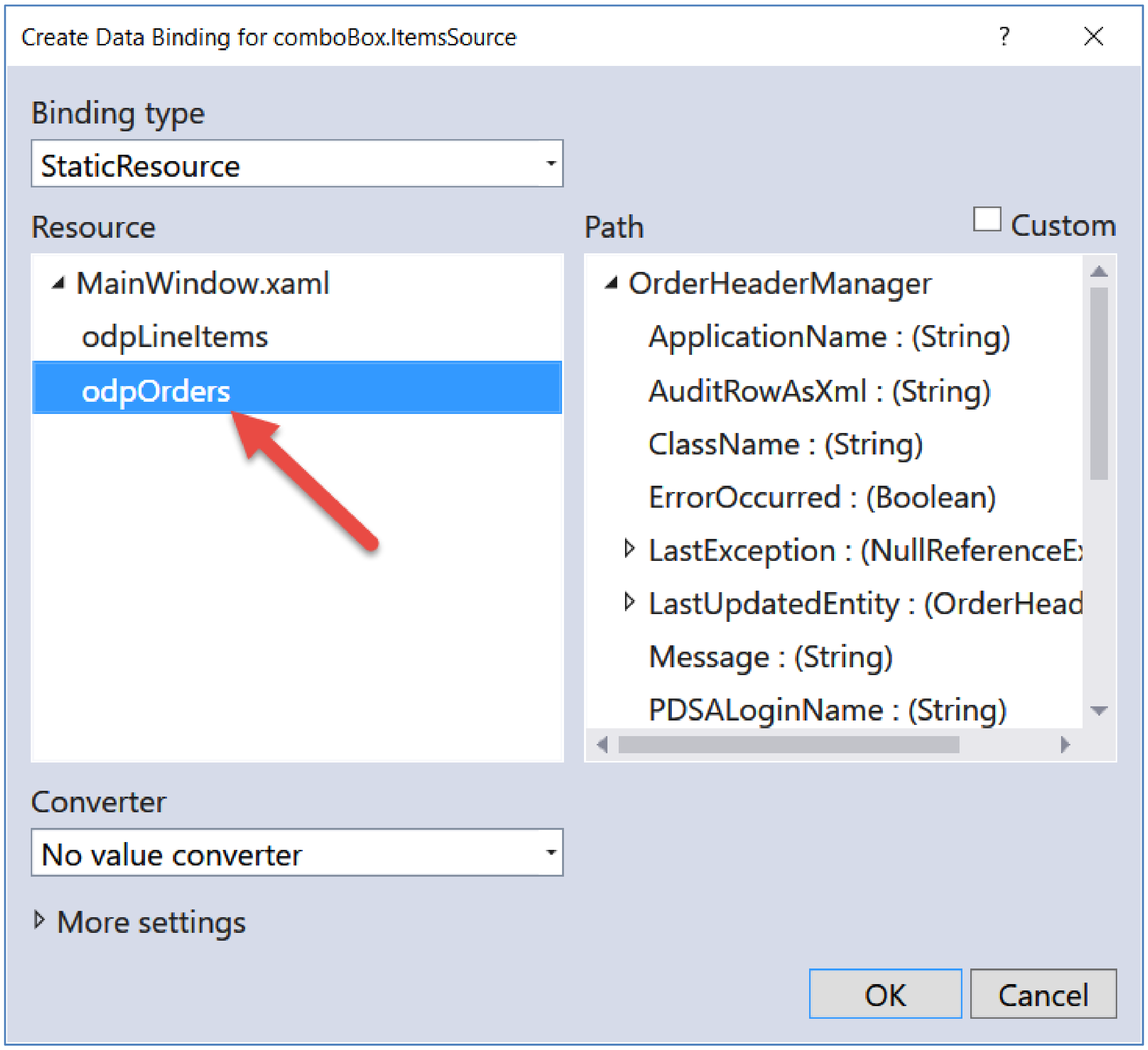
Click on the Advanced Properties (the little box) on the ItemsSource property and select Apply Data Binding… as shown in Figure 10.



```
<RowDefinition />
Grid.RowDefinitions>
```

Figure 10: Apply a data binding to Combo Box

Select StaticResource | Window.Resources | odpOrders from the list as shown in Figure 11.


Create Data Binding for comboBox.ItemsSource

Binding type: StaticResource

Resource
- MainWindow.xaml
  - odpLineItems
  - odpOrders

Path
- OrderHeaderManager
  - ApplicationName : (String)
  - AuditRowAsXml : (String)
  - ClassName : (String)
  - ErrorOccurred : (Boolean)
  - LastException : (NullReferenceE
  - LastUpdatedEntity : (OrderHead
  - Message : (String)
  - PDSALoginName : (String)

Custom

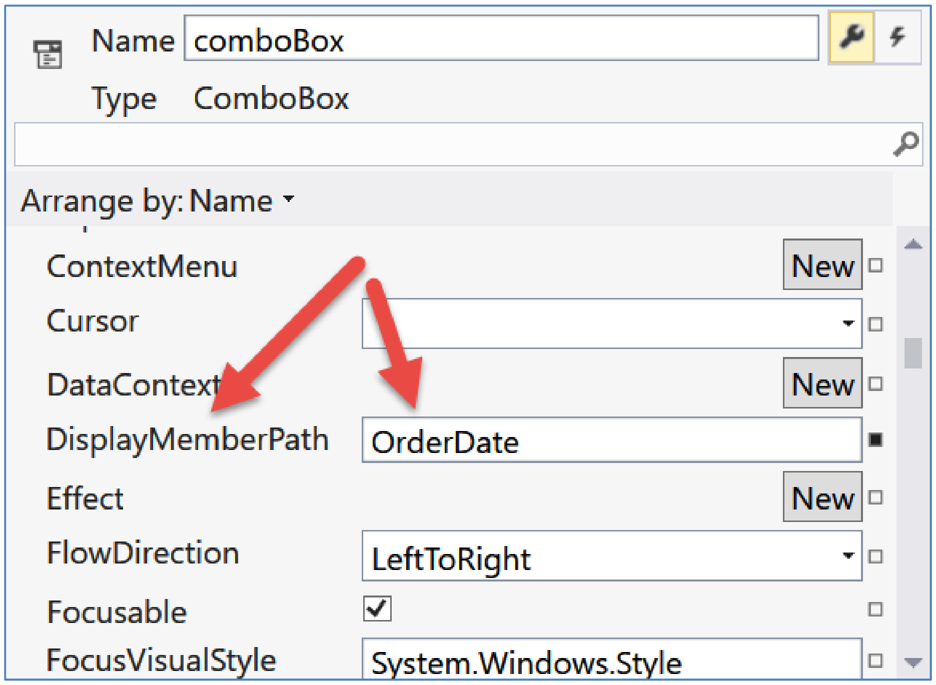
Converter: No value converter

More settings

Figure 11: Select the odpOrders Object Data Provider

Click off this window to set the binding.

While still in the Properties Window for the ComboBox select the **DisplayMemberPath** property and type in **OrderDate** as shown in Figure 12.



Name comboBox
Type ComboBox

ContextMenu	New
Cursor	
DataContext	New
DisplayMemberPath	OrderDate
Effect	New
FlowDirection	LeftToRight
Focusable	✓
FocusVisualStyle	System.Windows.Style

Figure 12: Set the DisplayMemberPath to the property you wish to display in the ComboBox

If you run the project right now you should see data in the combo box.

Now, let's hook up the list box to display the line items.

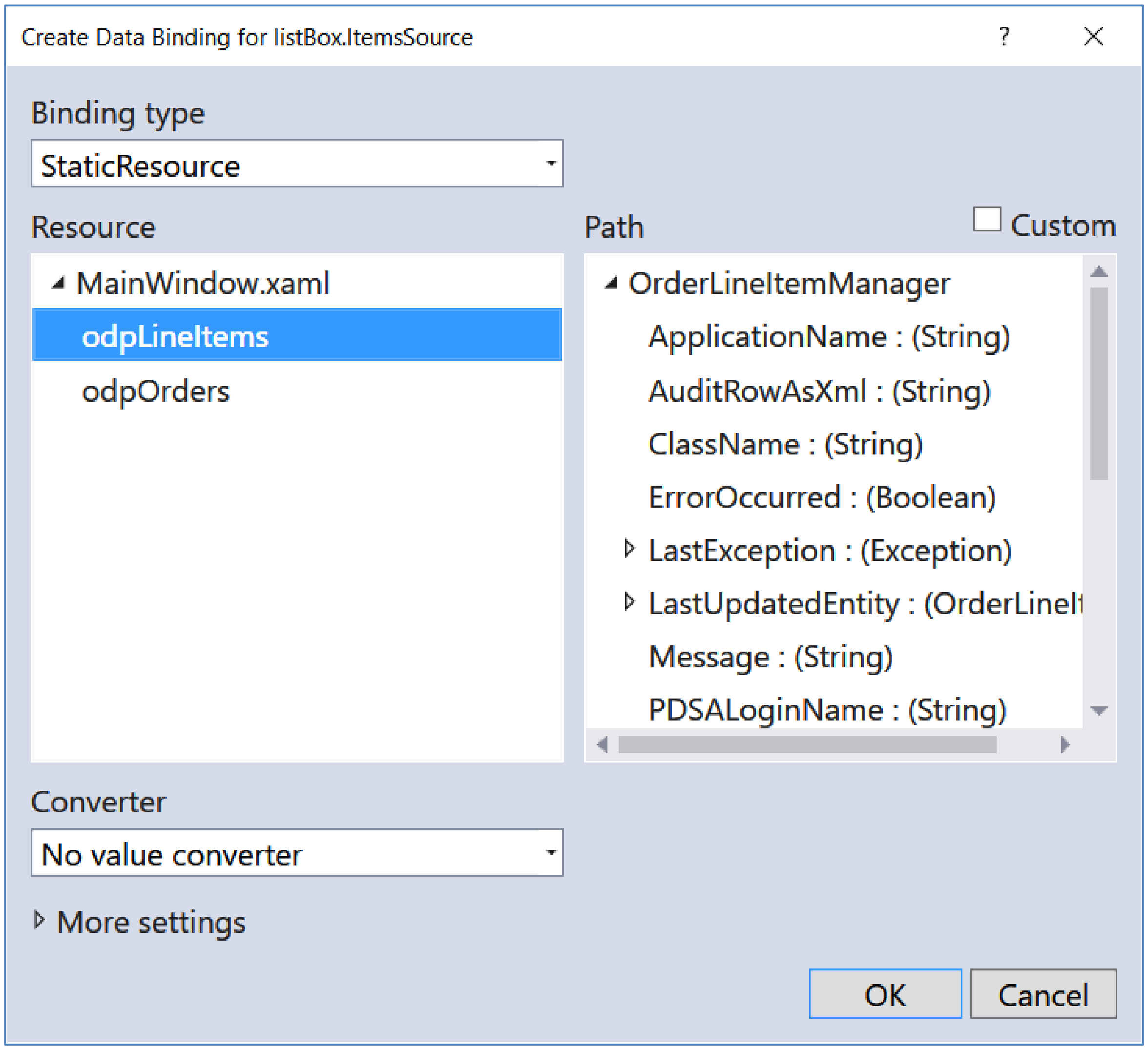
In the ComboBox you will need to write the following XAML between the <ComboBox…> and the </ComboBox> tags. If the </ComboBox> ending tag is not there, you will need to create it.

```
<ComboBox.SelectedValue>
  <Binding Source="{StaticResource odpLineItems}"
           Path="MethodParameters[0]"
           BindsDirectlyToSource="True"
           UpdateSourceTrigger="PropertyChanged" />
</ComboBox.SelectedValue>
```

Click on the List Box control to give it focus.

In the Properties Window select ItemsSource and click on the Advanced Properties box to Apply a Data Binding.

Choose the odpLineItems as shown in Figure 13.


Create Data Binding for listBox.ItemsSource

Binding type: StaticResource

Resource:

- MainWindow.xaml
  - odpLineItems
  - odpOrders

Path:

- OrderLineItemManager
  - ApplicationName : (String)
  - AuditRowAsXml : (String)
  - ClassName : (String)
  - ErrorOccurred : (Boolean)
  - LastException : (Exception)
  - LastUpdatedEntity : (OrderLineI
  - Message : (String)
  - PDSALoginName : (String)

Converter: No value converter

Figure 13: Select the odpLineItems Object Data Provider

Run the application and you should be able to select an order and see the ToString() representation of the line items for that order (Figure 9). Feel free to fix up the ListBox to look like however you want.

Summary

In this chapter you learned how to create relationships between tables in Haystack and add some additional code to create a MVC page that takes advantage of those relationships.

Chapter Index