Developer Session labs

The Developer Session labs will introduce you to the programming techniques that you need to develop add-ins for Microsoft Dynamics POS.

The exercises in these labs start out simple to give you a feel for the add-in architecture, and then develop in complexity. We hope that by the completion of these labs, you will be able to begin developing your own add-ins that fully utilize the rich features of Microsoft Dynamics POS.

There are seven labs in the Developer Session:

Note
These labs use Microsoft Dynamics POS Alpha. Customizations based on this release may need modifications to work with the final release of Microsoft Dynamics POS.

lab 1

Creating a basic add-in

There are six key parts to a basic Microsoft Dynamics POS add-in:

- The AddIn attribute that marks a class as an add-in
- A reference to the DLL that contains the Microsoft.Rms.AddInView.V300 namespace
- References to the Microsoft .NET 3.5 framework namespaces that provide the base functionality for add-ins
- The add-in manifest file
- The installation location for the compiled add-in
- The pipeline segments cache and pipeline store cache

In his lab, you will learn how to use the above six key parts to create a basic add-in. At the end of the lab, you will have a basic working add-in that displays the following dialog box when Microsoft Dynamics POS starts.

Note
You can find a finished example of each lab in the **Reference** subfolder in each of the lab folders in the SDKBootCamp folder on your desktop.

Throughout this document, you will be instructed to view these reference projects to study finished code or to copy and paste infrastructure code that is either lengthy or not directly related to the mechanics of writing a successful add-in.

Feel free to view these reference projects at any time. Note also that these projects compile and can be deployed on your computer.

Create the project

The following procedure walks you through creating a class library project for your add-in.

1. Run Microsoft Visual Studio 2008 as Administrator.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Project types:** pane in the **New Project** dialog box, expand **Visual C#**, click **Windows**, and then click **Class Library** in the **Templates:** pane.
4. In the **Name:** text area type, *HappyHourDiscount*.
5. Click the **Browse** button.
6. In the Project Location dialog box, navigate to *C:\Documents and Settings\yourusername\Desktop\SDKBootCamp\Session1_CreatingAnAddIn\Working*.
7. Uncheck the **Create directory for solution** check box.
8. Click **OK**.

Create a basic add-in

By doing the procedures in this section, you will perform all the steps necessary to write code that customizes Microsoft Dynamics POS. You will add the necessary references and using statements, create a public class marked with the AddIn attribute, create an application manifest file suitable for a basic add-in, implement an add-in view, and deploy your add-in by copying the appropriate files to the installation directory and updating the pipeline segments cache and add-in store.

Add the necessary references

Microsoft Dynamics POS add-ins require the use of types in the System.AddIn and Microsoft.Rms.AddInViews.V300 namespaces. The following steps walk you through adding these references and inserting useful using statements.

1. On the **Project** menu, click **Add Reference...**.
2. In the **Add Reference** dialog box, click the **.NET** tab.
3. Press and hold the **Ctrl** key and click **System.AddIn**, **System.AddIn.Contract**., and **System.Windows.Forms**.
4. Click **OK**.
5. On the **Project** menu, click **Add Reference...**.
6. In the **Add Reference** dialog box, click the **Browse** tab.
7. In the **Add Reference** dialog box, navigate to `C:\Program Files\Microsoft Dynamics - Point of Sale\AddInViews\`.
8. Select **Microsoft.Rms.AddInViews.V300.dll**. Click **OK**.
9. Right-click on **Class1.cs** in **Solution Explorer** and click **Rename**. Rename Class1.cs to `PosAddIn.cs`. If you are prompted to change all references, click **Yes**.
10. Open the `PosAddIn.cs` file and add the following using statements:

```
using System.AddIn;

using System.AddIn.Contract;

using System.Windows.Forms;

using Microsoft.Rms.AddInViews.V300;

using Microsoft.Rms.AddInViews.V300.ServiceModel;
```

Mark the add-in class with the AddIn attribute:

The AddIn attribute marks a class for discovery by Microsoft Dynamics POS. Classes marked with the AddIn attribute must be made public, or they will not be discovered by POS and will therefore not be loaded. In general, classes marked with the AddIn attribute will also be public and implement an interface from the Microsoft.Rms.AddInViews.V300 namespace or a sub-namespace belonging to it.

- In the `PosAddIn.cs` file, add the following line immediately above the definition of the `PosAddIn` class:

```
[AddIn ("Happy Hour Add-in", Version="1.0.0.0")]
```

Note
When you define your own add-in classes, remember that classes marked with the **AddIn** attribute must be public. **Visual Studio 2008** does not mark as public classes that you add by using the **Project** menu or **Solution Explorer**.

Create the manifest file

The manifest file is placed in the installation directory and is used by `AddInUtil.exe` to update the pipeline segments cache and the add-in store.

Note
In the alpha version of Microsoft Dynamics POS, an incorrect manifest file can cause your add-in not to load or create hard-to-find crashing bugs in your add-in.

1. In the Solution Explorer pane in the **Visual Studio 2008** application window, right-click the **HappyHourDiscount** node, click **Add**, then click **New Item**.
2. In the **Categories** pane in the **Add New Item** dialog box, click **General**.
3. In the **Templates** pane, select **Application Manifest File**.
4. In the **Name** text area, type *HappyHourDiscount.manifest*. Click **Add**.
5. Replace the contents of the `HappyHourDiscount.manifest` file with the following XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<AddinManifest>
<AddinAssembly
assembly="HappyHourDiscount.dll"
implementedViews="Microsoft.Rms.AddInViews.V300.ServiceModel.IPosAddIn
"
namedPermissionSet="FullTrust"
/>
</AddinManifest>
```

Modify PosAddIn to implement IPOSAddIn

Add-ins must implement the **IPosAddIn** interface. This interface has a single method, **SetPosApplication**, which is called by Microsoft Dynamics POS to give the add-in a handle to the running instance of POS. In general, classes marked with the AddIn attribute will also implement an interface from the Microsoft.Rms.AddInViews.V300 namespace or a sub-namespace belonging to it.

1. Declare that `PosAddIn` implements **IPosAddIn**.
2. Create an instance member in `PosAddIn` of the type **IPosApplication** to store the application handle to the running instance of Microsoft Dynamics POS. (Call it `PosApp`.)
3. Implement the **IPosAddIn.SetPosApplication** method. Assign its `posApplication` parameter to `PosApp` and then show a message box with the title "Happy Hour Add-in" that displays the message, "Happy Hour Add-in has a handle to POS."
4. Save your work. Then, on the **Build** menu, click **Build Solution**.

Deploy the add-in

To deploy an add-in, you must place the add-in DLL file and manifest file in an add-in specific folder in the AddIns folder in the installation directory. For the Happy Hour Discount add-in, HappyHourDiscount is a logical name for this directory and is used throughout the portion of this lab that deals with the Happy Hour add-in.

Note
Throughout this lab you will be instructed to copy over both a DLL and a PDB file. The PDB file contains information for the Visual Studio 2008 debugger and is necessary for setting breakpoints and stepping through code. You will almost always deploy retail builds to your customers' computers, and will therefore only deploy DLL files, not PDB files.

1. Create a folder called `HappyHourDiscount` in the `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\` folder.
2. Open the **SDKBootCamp** folder on the Desktop and navigate to the `Session1_CreatingAnAddIn\Working\HappyHourDiscount\bin\Debug` subfolder. Copy the `HappyHourDiscount.dll` file and `HappyHourDiscount.pdb` file from here to `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\`.

   Note
   Do not copy the Microsoft.Rms.AddInViews.dll file from your build directory to the installation directory. Microsoft Dynamics POS uses the version of this file that exists in the installation path.
3. Open the SDKBootCamp folder on the Desktop and navigate to the `Session1_CreatingAnAddIn\HappyHourDiscount` subfolder. Copy the `HappyHourDiscount.manifest` file from here to `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\`.
4. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
5. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

   Note
   For the purposes of this lab, you might consider leaving this command window open so that you can use the command history to execute this command. Simply move the

focus to the window, press the up arrow key until the command appears, and press Enter.

Because this is an alpha version of the product, several warning messages appear when you run AddInUtil.exe. Inspect them briefly to be sure that the HappyHourDiscount is not mentioned. Otherwise, ignore them.

Debug the add-in

In this section, you will set up Visual Studio so that you can debug your add-ins by starting Microsoft Dynamics POS from within Visual Studio. Note that you have to deploy your files before you can debug them.

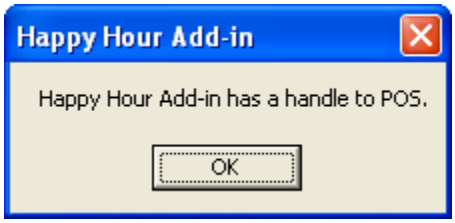
Note
Debugging must be set up once for every add-in project that you write. We have included a .user file that contains the debugging setup data in almost every project in this lab so that you do not have to perform these steps at the beginning of each lab.

1. On the **Project** menu in Visual Studio 2008, click **HappyHourDiscount Properties...**.
2. On the **HappyHourDiscount** tab, click the **Debug** tab.
3. In the **Start Action** radio button group, select **Start external program:**.
4. In the **Start external program:** text area, type:

   C:\Program Files\Microsoft Dynamics - Point of Sale\pos.exe
5. In the **Working directory:** text area in the **Start Options** group, type:

   C:\Program Files\Microsoft Dynamics - Point of Sale
6. Close the **HappyHourDiscount** tab.
7. On the **Debug** menu, click **Start Debugging**.
8. Login in to Microsoft Dynamics POS when prompted.

When Microsoft Dynamics POS starts up, the **Happy Hour Add-in** diaog box is displayed:



Conclusion: Creating a basic add-in

In this lab, you created an add-in that displayed a message box when Microsoft Dynamics POS started up. While this functionality is very rudimentary, your fully functioning add-in demonstrates how to code, deploy, and run any add-in.

lab 2

User interface extension 1

Now that we have created an add-in, we will add code to provide various customizations to the user interface of Microsoft Dynamics POS. In this lab, you will:

- Add a menu item to the Manager View by exposing IAddInMenuItemProvider
- Add a user control to the custom pane in POS View
- Add a status panel to the status bar in POS View
- Add a function menu in POS View
- Add a task item to the task pane in POS View

Note
You can find a finished example of each lab in the **Reference** subfolder in each of the lab folders in the SDKBootCamp folder on your desktop.

Throughout this document, you will be instructed to view these reference projects to study finished code or to copy and paste infrastructure code that is either lengthy or not directly related to the mechanics of writing a successful add-in.

Feel free to view these reference projects at any time. Note also that these projects compile and can be deployed on your computer.

Setup

At the beginning of most of the labs, you will start by navigating to the Working folder in the folder for the lab and double-click the .csproj file.

1. Run Visual Studio 2008 as Administrator.
2. From the **File** menu, click **Open**, and then click **Project/Solution**.
3. In the **Favorite Links** pane, click **Desktop**
4. Navigate to the Session2_UIExtension_1\Working folder.
5. Select the **HappyHourDiscount.csproj** file, and click **Open**.

Add a menu item to Manager View

In this section, you will add a menu item to the bottom of the **Inventory** menu that appears in Manager View.

Add the menu-related classes to the project:

1. On the **Project** menu, click **Add Class**.
2. In the **Name:** text area on the **Add New Item** dialog box, type **AddInMenuItem.cs**. Click **Add**.
3. On the **Project** menu, click **Add Class**.
4. In the **Name:** text area on the **Add New Item** dialog box, type **AddInMenuItemProvider.cs**. Click **Add**.
5. Add the following `using` statements to both **AddInMenuItem.cs** and **AddInMenuItemProvider.cs**:

```
using System.AddIn;

using Microsoft.Rms.AddInViews.V300;

using Microsoft.Rms.AddInViews.V300.UIExtension;
```

Code AddInMenuItem.cs and AddInMenuItemProvider.cs

To add a menu to the Manager View in POS, you need to create a class that implements `IAddInMenuProvider` and another class that implements `IAddInMenu`. You then use the `IAddInMenu` object in the `IAddInMenuProvider` constructor to hook the handler up to the menu.

1. In the AddInMenuItem.cs file, Derive `AddInMenuItem` from `IAddInMenuItem`.
2. On the Desktop, find and open the `SDKBootCamp` folder. Navigate to the `Session1_CreatingAnAddIn\Reference\` folder.
3. Open the `AddInMenuItem.cs` file and copy the body of the `AddInMenuItem` class paste it into the body of your `AddInMenuItem` class.
4. In the **AddInMenuItemProvider.cs** tab in the code window, mark the `AddInMenuProvider` class as public.
5. Derive `AddInMenuItemProvider` from `IAddInMenuItemProvider`.
6. Apply the `AddIn` attribute to the `AddInMenuItemProvider` class. The title of the add-in is "Happy Hour Add-in," and the version is "1.0.0.0".
7. Give `AddInMenuItemProvider` a private `AddInMenuItem` field an assign it to **null**. Call this member `settingsHappyHourDiscount`, since this will eventually launch a settings dialog box.
8. Give `AddInMenuItemProvider` a private method that takes an `IAddInMenuItem` parameter and returns void. Call it `DiscountHandler`. You can leave the body empty, or, if you like, display a message box so that you can see when this code is run.
9. Right-click `IAddInMenuItemProvider`, click **Implement Interface**, and then click **Implement Interface**.
10. In the `GetSubmenus` method create a new IAddInMenuItem array, initialize it to contain the private AddInMenuItem member you created above, and return the array.
11. Insert a public default constructor into the `AddInMenuItemProvider` class.
12. In the body of the constructor that you just created, assign `settingsHappyHourDiscount` to a new `AddInMenuItem` that is keyed to the Inventory menu, has a single element array of sub-items that contains a menu item with the title "Happy Hour Discount," and is bound to the `DiscountHandler` method that you created above.

Description of the AddInMenuItem class

Study the code you just pasted in to the body of `AddInMenuItem`. It provides a basic implementation of the `IAddInMenuItem` interface. It has a constructor that takes four parameters, and assigns the remaining members of the `IAddInMenuItem` to reasonable values. The four parameters that are used by the constructor are necessary for adding a menu item to Microsoft Dynamics POS. The following table explains these parameters.

Parameter	Description
string key	A string that uniquely identifies the menu. This is preferably a GUID.
IAddInMenuItem[] subItems	An array of submenus that belong to this menu.
string text	The text that appears on this menu.
System.Action<IAddInMenuItem> onExecute	The method that is invoked when this menu item is clicked.

The *key* argument, in addition to uniquely identifying a menu that you create, can also identify a menu in Microsoft Dynamics POS. For your convenience, the `ManagerMenuItem` class provides fields that evaluate to strings that identify the top-level menus. The following table shows these fields:

Field	Description
Edit	Identifies the **Edit** menu item.
File	Identifies the **File** menu item.
Help	Identifies the **Help** menu item.
Inventory	Identifies the **Inventory** menu item.
People	Identifies the **People** menu item.
Report	Identifies the **Report** menu item.
Settings	Identifies the **Settings** menu item.
Tools	Identifies the **Tools** menu item.
Transactions	Identifies the **Transactions** menu item.
View	Identifies the **View** menu item.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any views previously implemented in this add-in.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the HappyHourDiscount.manifest file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInView.V300.UIExtension.IAddInMenuItemProvider`. When you are done with this step, the value of the implementedViews attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Save the manifest file.

Build and deploy the add-in

These steps are provided throughout this document. Remember that when debugging, you must copy the new files to the installation directory before you will see your changes reflected in Microsoft Dynamics POS. Also remember that you must run AddInUtil.exe every time you change the application manifest file. It is easy to miss one or both of those steps in the heat of a debugging session!

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called `HappyHourDiscount` in the `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\` folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the `Session2_UIExtension1\Working\bin\Debug` subfolder. Copy the `HappyHourDiscount.dll` file and `HappyHourDiscount.pdb` file from here to `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\`.
4. Open the SDKBootCamp folder on the Desktop and navigate to the `Session2_UIExtension1\` subfolder. Copy the `HappyHourDiscount.manifest` file from here to `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\`.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

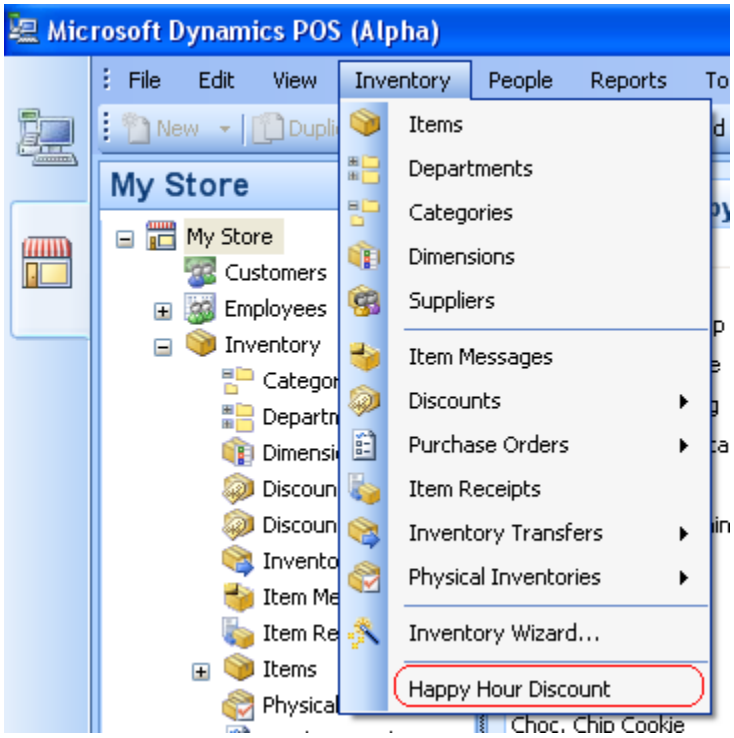
Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

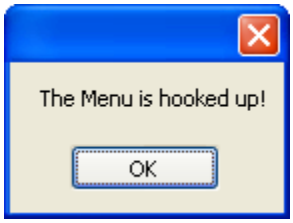
   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

2. Log in to Microsoft Dynamics POS.
3. Switch to Manager View.
4. Click the **Inventory** menu.

Notice the Happy Hour Discount menu item at the bottom of the Inventory menu:



5. If you decided to display a dialog box for this menu item, click the Happy Hour Discount menu item to display your dialog.



Summary for Adding a menu item to Manager View

In the preceding section, you implemented the IAddInMenuItemProvider view and updated the manifest file to reflect the new view you added to your project. In addition, you optionally implemented IAddInMenuItem to construct a menu item that Microsoft Dynamics POS could display.

Add a user control to the custom pane in POS View

In this section, you will add a browser control to the Custom Pane. To do this, you will create a class that implements `IHeaderPane` to add a custom control to the project, another class that implements `IHeaderPaneProvider` to enable you to display the control to the Microsoft Dynamics POS the Custom Pane, and write methods and events to update the control at regular intervals. Your Custom Pane will display the following graphic:

Deploy the Happy Hour HTML files.

The customization in this section enables you to display any html page in the Custom Pane. To achieve the desired effect for the Happy Hour add-in, use the files provide in the html folder in the folder for this lab.

1. On the Desktop, find and open the SDKBootCamp folder. Navigate to the `Session2_UIExtension1\html\` folder.
2. Copy the `4thcoffeeaddinhtml.swf` and `fourthcoffeeaddinhtml.html` files to `C:\Program Files\Microsoft Dynamic – Point of Sale\Content\HtmlStatusPane\`.

Add a custom control to the project

In this section, you will add a browser control to the project and configure its properties.

1. On the **Project** menu, click **Add User Control**.
2. Type *HeaderPaneControl.*cs in the **Name:** text area of the **Add New Item** dialog box. Click **Add**.
3. Expand the **Toolbox**, expand the **Common Controls** node, and drag a **WebBrowser** control onto the user control you added.
4. In the **Anchor** property in the **Properties** pane for **WebBrowser1**, type *Top, Left, Bottom, Right*.
5. In the **ScrollBarsEnabled** property, choose **False**.
6. Change the **TabIndex** property to 1.
7. View the code for **HeaderPaneControl.cs**.

Code the custom control

In this section, you will copy code that will allow the `WebBrowser1` control to be updated at regular intervals. (The `HeaderPaneProvider` class will start the timer that will regularly call the updates.) Your user control code needs a handler for the timer event, a method to perform the actual update, and a utility function to help establish the location of the displayed URL.

View the `HeaderPaneControl.cs` file in the `Session2_UIExtension_1\Reference` subfolder of the SDKBootcamp folder to see how the following parts are implemented, and then copy them into your code. Be sure to build your project and add any missing references or using statements.

Item	Description
HeaderPaneControl	A public default constructor. Simply calls `InitializeComponent` and `SetHtmlPane`.

SetHtmlPane	Finds the HTML document to display, navigates the control to it, and refreshes the control.
SetHtmlPane_Handler	Some plumbing to handle timer events. Simply throws away the parameters and updates the control.
GetCurrentProcessFolder	A utility function that finds the location of the currently running instance of POS. This is your installation directory.

Implemenent IHeaderPaneProvider

The following five steps create a minimally functional header pane provider.

1. Using the **Add Class** command on the **Project** menu, add a new class file called **HeaderPaneProvider.cs** to the **HappyHourDiscount** project.
2. Add the following `using` statements **HeaderPaneProvider.cs**:

   ```
   using System.AddIn;

   using Microsoft.Rms.AddInViews.V300;

   using Microsoft.Rms.AddInViews.V300.UIExtension;
   ```

3. Make the `HeaderPaneProvider` class public, mark it with the `AddIn` attribute, and make it implement `IHeaderPaneProvider`.
4. Implement the `IHeaderPaneProvider` interface by right-clicking `IHeaderPaneProvider` in the **HeaderPaneProvider.cs** file, and then clicking **Implement Interface**.
5. Replace the default implementation with code that creates a new `HeaderPaneControl` by using the default constructor, stores the header pane control in a variable (call it `control`), and returns the variable.

Previously, you created a simple implementation of the `IHeaderPaneProvider` method `GetControl`. Now, view the `HeaderPaneProvider.cs` file from the reference project for this lab and copy the implementation there into your `GetControl` method. Rebuild your solution and copy over any using statements that you need to get your code to compile.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any existing views. Furthermore, because you've just created some additional UI, you need to update the manifest to include this information as well.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the `HappyHourDiscount.manifest` file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInViews.V300.UIExtension.IAddInMenuItemProvider`. When you are done with this step, the value of the `implementedViews` attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Inside the AddinAssembly tag, add the following attribute=value pair:

   exposeUI="true"

4. Save the manifest file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called `HappyHourDiscount` in the `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\` folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\Working\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

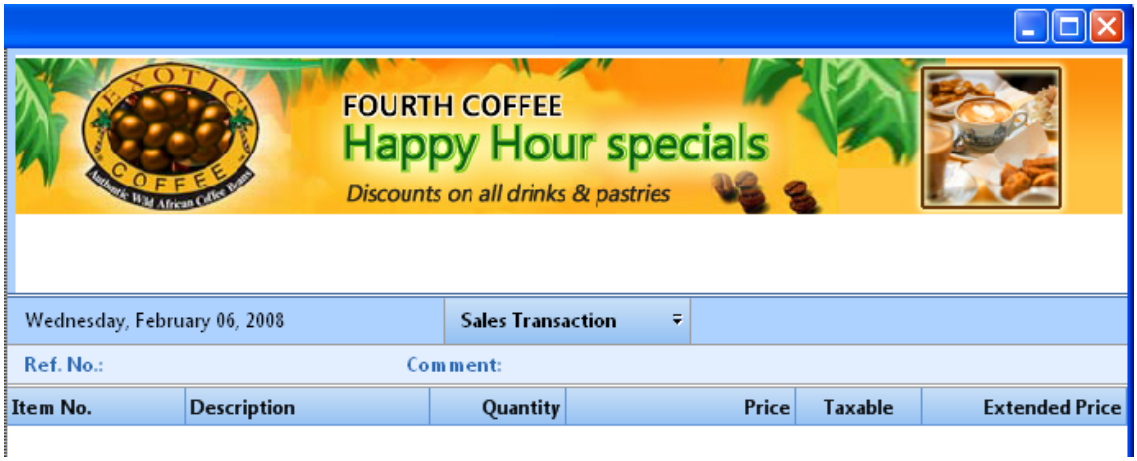
Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

Your Custom Pane will display the Happy Hour image:



Add a status panel to the status bar in POS View

In this section, you will use an `IStatusBarPanelFactory` passed to you by Microsoft Dynamics POS to create a status bar message that displays a brief message that includes the current time, updated every 20 seconds.

Implement IStatusBarPanelProvider

1. Add a class to your project called `StatusBarPanelProvider`.

2. Add using statements for the `System.AddIn` namespace and the add-in views, including the `UIExtension` namespace.
3. Mark `StatusBarPanelProvider` with the `AddIn` attribute.
4. Make `StatusBarPanelProvider` public.
5. Implement the `IStatusBarPanelProvider` interface in the `StatusBarPanelProvider` class.
6. View the `StatusBarPanelProvider.cs` file from the reference project for this lab and copy and paste the code for `_firstPanel`, `FirstPanel`, `UpdateStatusBar`, `UpdateStatusBar_Handler`, `intervalInMillisec`, and the `StatusBarPanelProvider` constructor into your `StatusBarPanelProvider` class.

Briefly study the code you copied in so that you can understand how it works. The timer code in the constructor is similar to code used in `HeaderPaneProvider.cs`. Consequently, much of the code of this section is similar. There is an event handler that responds to timer events by ignores the event parameters and simply calling a helper method.

7. In the `PopulateStatusBar` method, provide an implementation that uses the passed `in IStatusBarPanelFactory` to create a visible status bar with no image, assigns it to `_firstPanel`, and calls `UpdateStatusBar`.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any existing views.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the HappyHourDiscount.manifest file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInViews.V300.UIExtension.IStatusBarPanelProvider`. When you are done with this step, the value of the implementedViews attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Save the manifest file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called `HappyHourDiscount` in the `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\` folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\Working\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

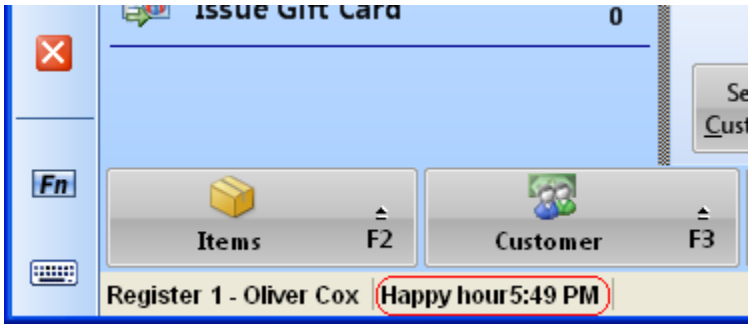
%windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

2. After you log in to Microsoft Dynamics POS, notice the Happy Hour message in the status bar.



Add a function menu in POS View

In this section, you will create a function menu class, a function menu button class, and a function menu provider class, that together present POS users with the options to extend Happy Hour by one hour or by two. (For this lab, you will just display a message box that declares that Happy Hour has been extended.) Note that, previously, function menus were referred to as slider menus. The interfaces related to function menus use the older terminology. Therefore, you will use ISliderMenuButton, ISliderMenu, and ISliderMenuProvider in this lab.

Create the function menu button class

The function menu button class, `SliderMenuButton`, implements `ISliderMenuButton`. It uses an enumeration, `MenuButtonNameEnum`, to determine the action it should take when clicked. Its structure is straightforward and its implementation is simple. Note that that the AddIn attributes does not need to be applied to this piece of UI code.

The function menu class, SliderMenu, implements ISliderMenu, which requires accessors for an array of buttons and an identifier that specifies on which function menu to place the menu. The constructor for the implementation in this lab simply takes two arguments and assigns them to the private fields behind the interface's required properties.

- View the `SliderMenuButton.cs` and `SliderMenu.cs` files in `Session2_UIExtension_1\Reference` subfolder of the `SDKBootcamp` folder.
- Add two new classes to your solution, `SliderMenuButton.cs` and `SliderMenu.cs`, and paste the class and enumeration implementation code from the next session into this file.
- Add the following using statements to `SliderMenuButton.cs` and `SliderMenu.cs`:

```
using System.AddIn;

using Microsoft.Rms.AddInViews.V300.UIExtension;
```

Create the function menu provider class (ISliderMenuProvider)

The function menu provider, `SliderMenuProvider`, implements ISliderMenuProvider. This interface requires one method that returns an array of submenus. Because the SliderMenuProvider class must be discoverable by POS, it needs the AddIn attribute, needs to be public, and needs an entry in the manifest.

1. Add a new class to your project called `SliderMenuProvider`.
2. Make the `SliderMenuProvider` class public.
3. Add the following using statements to `SliderMenuProvider.cs`:

   ```
   using System.AddIn;

   using Microsoft.Rms.AddInViews.V300.UIExtension;
   ```

4. Create a private `ISliderMenu` array member (`sliderMenu`) for the `SliderMenuProvider` class and a no-arg constructor.
5. In the constructor create an array that contains two `SliderMenuButton` objects named "Extend Happy Hour by 1 Hour" and "Extend Happy Hour by 2 Hours," both of which are created with the corresponding `MenuButtonNameEnum` values.
6. Create a new `SliderMenu` object with the key "Tasks" and pass it the array created in Step 5.
7. Assign sliderMenu to a single-element ISliderMenu array that contains the SliderMenu object created in Step 6.
8. If you used temporary variables to perform Steps 5 through 7, clean up your code by removing temporary variables and using anonymous arrays and objects, if desired.
9. Implement the `ISliderMenuProvider` interface in the `SliderMenuProvider` class. Simply return `sliderMenu`.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any existing views.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the `HappyHourDiscount.manifest` file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInViews.V300.UIExtension.ISliderMenuProvider`. When you are done with this step, the value of the implementedViews attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Save the manifest file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called `HappyHourDiscount` in the `C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\` folder, if you have not already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\Working\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.

4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

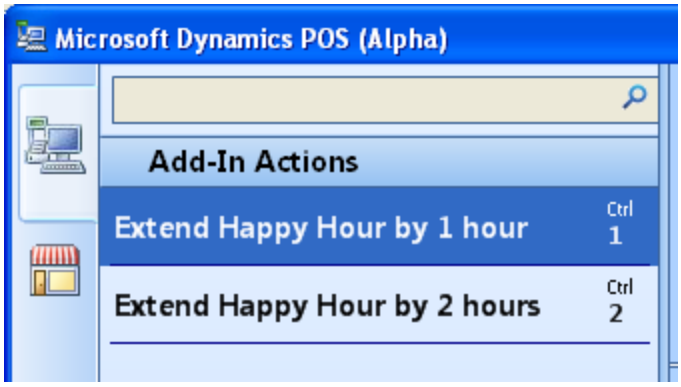
Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

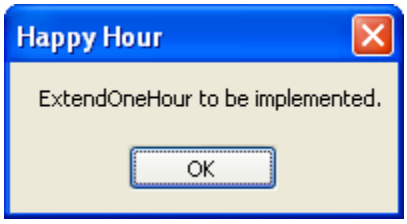
   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

2. After you log in to Microsoft Dynamics POS, click the Tasks function menu in POS View to reveal the **More** task menu:

3. Click the **More** menu to display the task pane menus:



4. Click the Extend Happy Hour by 1 hour menu to display the Happy Hour dialog.



Add a task item to the task pane in POS View

In this section, you will create a task item on the Task Pane that will display a dialog box that contains basic information about the Happy Hour Discount add-in. To do this, you will have to both code the add-in and configure Microsoft Dynamics POS to display the task item you created in the Task Pane.

The add-in code consists of two files, ExposedTask.cs and ExposedTaskProvider.cs. As you may be able to guess by now, the ExposedTasks class will be public, implement an add-in interface, and require the AddIn attribute. Also, as you might have guessed, the ExposedTask class does the actual work of this add-in. Since ExposedTasks depends on ExposedTask, you will implement ExposedTask first.

Create the task class

The ExposedTask class implements the IExposedTask interface. Its implementation for this exercise is very simple; Its properties return uncomplicated values and its action simply displays a message box. Note, however, that the IsConfigurable property returns false, and that the EditTaskConfiguration property therefore returns an empty string. For tasks where the IsConfigurable property returns true, you must supply a configuration dialog that is called from EditTaskConfiguration and returns a configuration string.

1. Add a new class to your solution called ExposedTask.cs.
2. Add the following using statements to ExposedTask.cs:

```
using System.Windows.Forms;

using Microsoft.Rms.AddInViews.V300.UIExtension;
```

3. Declare that the ExposedTask class implements IExposedTask.
4. View the ExposedTask.cs file in the reference solution and copy the body of the ExposedTask class from there into the body of your ExposedTask class.

Create the task provider class

1. Add a new class to your solution called ExposedTaskProvider.cs.
2. Add the following using statements to ExposedTaskProvider.cs:

```
using System.Windows.AddIn;

using Microsoft.Rms.AddInViews.V300.UIExtension;
```

3. Make the ExposedTasks class public.
4. Mark the ExposedTasks class with the AddIn attribute.
5. Implement the IExposedTasks interface in the ExposedTasks class.
6. View the ExposedTaskProvider.cs file in the reference solution and copy the body of the `ExposedTask` class from there into the body of your `ExposedTask` class.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any existing views.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the HappyHourDiscount.manifest file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInViews.V300.UIExtension.IExposedTasks`. When you are done with this step, the value of the `implementedViews` attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Save the manifest file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called HappyHourDiscount in the C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\Working\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session2_UIExtension1\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Configure Microsoft Dynamics POS to display the task

The steps in this section are performed in Microsoft Dynamics POS Alpha.

1. Run Microsoft Dynamics POS Alpha.

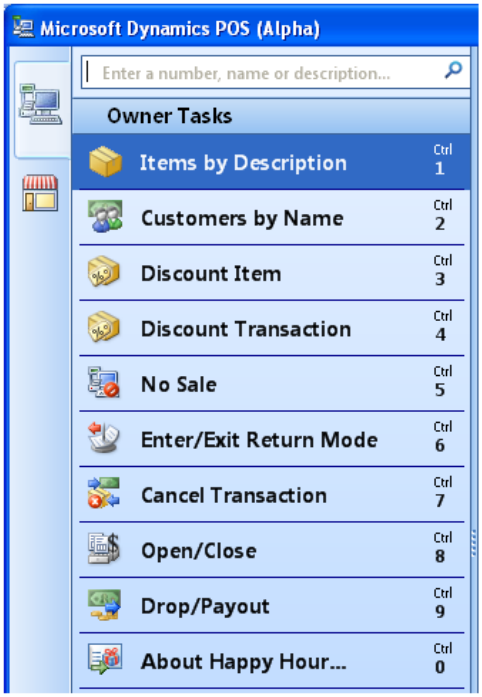
2. Switch to Manager View.
3. On the **Settings** menu, point to **Register Settings**, and then click **Task Pads**.
4. In the **Task Pads** pane, double-click on the **Owner Tasks** task pad.
5. In the **Owner Tasks** dialog box, click **Issue Gift Card** in the **Preview** group.
6. In the **Button type** list in the **Settings for the selected button** group, click **Add-In Action**.
7. In the **Add-InName** list, click **Happy Hour Discount**.
8. In the **Action** list, click **About Happy Hour**.
9. In the **Caption 1** box, type *About Happy Hour...*.
10. Close Microsoft Dynamics POS Alpha.

Debug the add-in

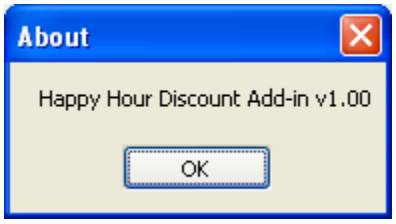
1. On the **Debug** menu in Visual Studio 2008, click **Start Debugging**.

Note
If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

Notice that the **Issue Gift Card** task has been replaced by **About Happy Hour**:



2. Click **About Happy Hour...** to launch the **About** dialog box:



lab 3

User interface extension 2

In this brief lab, you will add a settings dialog for the HappHourDiscount add-in. In the next two labs, you will update this dialog box to interact with the POS database.

For now, since most of the implementation details of this dialog box are either utility functions or designer code, you will copy the code over from the reference section, study it, and then build and debug the application.

Setup

First, we'll set up the project for this lab.

1. Run Visual Studio 2008 as Administrator.
2. From the **File** menu, click **Open**, and then click **Project/Solution**.
3. In the **Favorite Links** pane, click **Desktop**
4. Navigate to Session3_UIExtension_2\Working the folder.
5. Select the **HappyHourDiscount.csproj** file, and click **Open**.

Add the configuration dialog and helper class

Adding the files

1. Copy the HappyHourDiscountControl.cs, HappyHourDiscountControl.Designer.cs, HappyHourDiscountControl.resx, and XmlConfig.cs files from the reference project for this section into your current Working directory.
2. On the **Project** menu, select **Show all files**.
3. Right-click on the **HappyHourDiscountControl.cs** node in **Solution Explorer**, click **Include In Project**.
4. Repeat Step 3 for the **XmlConfig.cs** node.
5. On the **Project** menu, clear **Show all files**.

Add handler logic to AddInMenuProvider.cs

- Open to the AddInMenuProvider.cs file in the reference project for this lab. Copy the body of the DiscountHandler method into the DiscountHandler method in your AddInMenuProvider.cs file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.

2. Create a folder called HappyHourDiscount in the C:\Program Files\Microsoft Dynamics – Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session3_UIExtension2\Working\bin\Debug\ subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics – Point of Sale\Addins\HappyHourDiscount\.
4. Open the SDKBootCamp folder on the Desktop and navigate to the Session3_UIExtension2\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics – Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe –rebuild –PipelineRoot:"C:\Program Files\Microsoft Dynamics – Point of Sale"

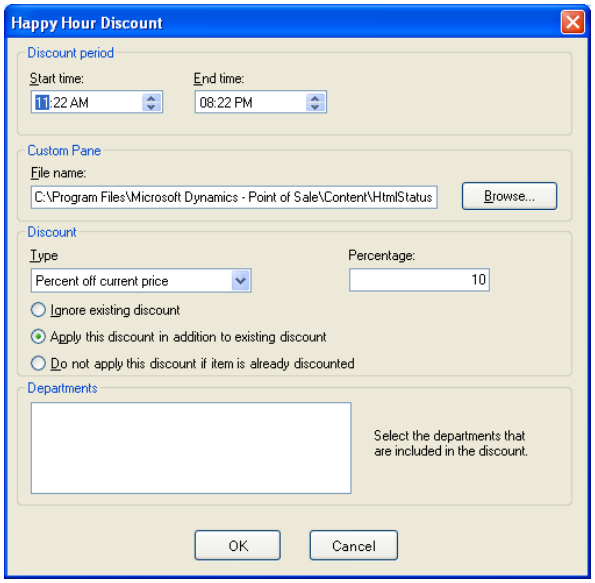
Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

2. Switch to Manager View.
3. On the **Inventory** menu, click **Happy Hour Discount**.

The **Happy Hour Discount** configuration dialog is displayed.



lab 4

Object model and data views 1

In this and the following lab, you will get a connection to the product database and use it to read to and write from the Microsoft Dynamics POS product database. You will also register for events raised by Microsoft Dynamics POS and provide add-in logic that responds to these events.

Note
You can find a finished example of each lab in the **Reference** subfolder in each of the lab folders in the SDKBootCamp folder on your desktop.

Throughout this document, you will be instructed to view these reference projects to study finished code or to copy and paste infrastructure code that is either lengthy or not directly related to the mechanics of writing a successful add-in.

Feel free to view these reference projects at any time. Note also that these projects compile and can be deployed on your computer.

Setup

1. Run Visual Studio 2008 as Administrator.
2. From the **File** menu, click **Open**, and then click **Project/Solution**.
3. In the **Favorite Links** pane, click **Desktop**
4. Navigate to the Session4_DataAndObjects1\Working folder.
5. Select the **HappyHourDiscount.csproj** file, and click **Open**.

Create a department data class

In this section, you will create a class called Department in a new file called Data.cs. The Department class will have three public string fields, DepartmentCode, DepartmentName, and ID. It will have a ToString method tht returns *DepartmentName (DepartmentCode)*.

1. On the **Project** menu, click **Add New Item...**.
2. In the **Add New Item** dialog box, expand the **Visual C# Items** node.
3. In the **Templates** pane, click **Code File**.
4. In the **Name:** text area, type *Department.cs*.
5. Either implement the code described in the opening remarks of this section, or copy it from the reference solution for this lab.

Expose PosApp as a public static readonly property

In the first lab, you created a PosAddIn class that implemented IPosAddIn. That class had an IPosApplication member called PosApp. That member was not marked public nor static. In this section, you will update the PosAddIn class to enable other classes to get a handle to the running Microsoft Dynamics POS application instance.

1. Open PosAddIn.cs in your working project.
2. Change PosApp from a private field to a public static readonly property.
3. Remove the message box from SetPosApplication, if you haven't done so already.

Create the POSDataManager class

In this section, you will create a helper class, POSDataManager, that gets a connection to the product database and then uses that connection to return department data from the store database.

Create a property of the type ISqlConnectionProvider

You get a valid ISqlConnectionProvider instance by first getting a valid IPosApplication instance and calling QueryInterface<T>. Conveniently, in the PosAddIn class , you just exposed PosApp, which receives a valid IPosApplication instance from Microsoft Dynamics POS.

The code that gets the database connection is in the static PosSqlConnection property of type ISqlConnectionProvider. This property calls the PosAddIn.PosApp.QueryInterface<T> generic method, passing the type parameter ISqlConnectionProvider. This causes QueryInterface to return an ISqlConnectionProvider interface that you can use to query the Microsoft Dynamics POS database.

1. Add a new public static class to your project called DataManager.
2. Create a public static read-only property of the type ISqlConnectionProvider for the POSDataManager class. Call the member PosSqlConnection.
3. Implement the get accessor so that checks to see if PosAddIn.PosApp is defined and, if it is, returns an ISqlConnectionProvider that can be used to query the Microsoft Dynamics POS database.

Use the POSDataManager.PosSqlConnection property

Microsoft Dynamics POS exposes a variety of public database view. Querying against these views is the recommended way of interacting with the database. These views represent a stable way to access POS data, and they insulate you from accidentally deleting data.

1. In Visual Studio 2008, choose the **Tools** menu and click **Connect to Database...**.
2. In the **Add Connection** dialog box, change the **Data Source**.
3. In the **Change Data Source** dialog box, choose the **Microsoft SQL Server** data source. Click **OK**.
4. In the **Add Connection** dialog box, type *(local)\msposinstance* in the **Server Name** combo box.
5. Select the **MSPOSSample** database in the **Select or enter a database name** drop down menu.
6. In **Server Explorer**, expand Data Connections, expand the MSPOSSample database, and then expand **Views**

The GetDepartmentforItem and GetListOfDepartments methods both use the PosSqlConnection property that you just created to call the ISqlConnectionProvider.EstablishConnectionAndRun method with the actions shown in the reference file. The GetDepartmentForItem method queries the views provided in the Microsoft Dynamics POS database to return the department code for an item. Similarly, the GetListOfDepartments method calls EstablishConnectionAndRun to query the database for the list of departments in the database and returns them as a list of Department objects. (The Department class was defined earlier in the Data.cs file.)

4. Create a public static method called GetDepartmentForItem that takes a string called itemLookup and returns a Department object that corresponds to the department for the item. Your method should use the EstablishSqlConnectionAndRun method of the PosSqlConnection property created above.

   Note
   The item and department information are in views called PublicItem and PublicDepartment, respectively.

5. Create a public static method called GetListOfDepartments that returns a List<Department> of all the departments in the store.

Update the HappyHourDiscount control

Update btnSave_Click

In the following steps, you will add code to btnSave_Click that adds the departments that are checked in the dialog box to the configuration file.

1. Open the HappyHourDiscountControl.cs file in your working project for this lab.
2. Find the TODO: item in the btnSave_Click method.
3. Open the HappyHourDiscountControl.cs file from the reference project for this lab and locate the code that corresponds to the TODO: item found in Step 2. (Hint: The comments are identical.)
4. Study the reference code until you understand what it does, then copy it in to your btnSave_Click method.

Update PopulateDepartment

This section is similar to the previous section. You will find the two PopulateDepartment methods in the working and reference HappHourDiscountControl class, study the reference code, and then copy it in to your solution.

When you are done, your PopulateDepartment method will populate the dialog box with the list of departments, preserving their checked state. The following steps assume that you still have the two HappyHourDiscountControl.cs files open.

1. Find the TODO: item in the PopulateDepartment method in your working project.
2. Find the code that corresponds to the TODO: item found in Step 1. (Hint: The comments are identical.)
3. Study the reference code until you understand what it does, then copy it in to your PopulateDepartment method.

Create the DiscountHelper utility class

Eventually, you will provide a handler in the PosAddIn class that calculates the discounted price when an item sales charge is added to the transaction. Rather than put that logic directly in the handler, you will create a utility class in this section to perform that work.

The DiscountHelper utility class contains four methods: IsHappyHour, HappyHourMinutesLeft, IsQualifiedForDiscount, and CalculateDiscountedPrice. HappyHourMinutesLeft reads the configuration string for the dialog to determine the start and end times for Happy Hour and determines if the current time is in that range. HappyHourMinutesLeft checks to see if it is Happy Hour, and then returns the time left. IsQualifiedForDiscount checks the department for an item and returns true if the item's department is eligible for discounting. (Implicitly in this example, eligibility depends upon the item's department.) CalculateDiscountedPrice performs some relatively simple tests to determine how to apply the discount, and then does so if required.

Study this class to be sure you understand how it works, and then simply copy it from the reference project for this lab into your working project for this lab.

1. Add a new class to your working project called DiscountHelper.
2. Open the reference project and copy DiscountHelper.cs into the DiscountHelper.cs file in your working project.

Finish updating PosAddIn

Previously in this lab, you updated the PosAddIn class to make the IPosApplication member PosApp public. In the following subsections, you will further update the PosAddIn class by implementing a handler for the ItemSalesChardeAdded event raised by Microsoft Dynamics POS. This example is interesting because you get to see two calls to the static IPosAddIn.QueryInterface<T> generic method. One of these calls specifies the IPosInstance type and returns the currently running instance of POS. The other call specifies the IPosEvents type and returns a collection of all the events that POS can raise.

Study this class to be sure you understand how it works, and then simply copy it from the reference project for this lab into your working project for this lab.

1. Open the PosAddIn.cs file that is in the reference directory for this lab.
2. Open the PosAddIn.cs file that is in the working directory for this lab.
3. Add the following using statements to your PosAddIn.cs file:

```
using Microsoft.Rms.AddInViews.V300.PosTransaction;

using Microsoft.Rms.AddInViews.V300.Events;

using Microsoft.Rms.AddInViews.V300.ServiceModel;
```

4. Copy the `PosAddIn_ItemSalesChargeAdded` event handler from the reference file to your file.
5. Copy the comment line and code line that add an event handler to the `ItemSalesChargeAdded` event in the `SetPosApplication` method to your SetPosApplication method.

Update the Status Bar, Custom Pane, and Function Menus

Until now, the implementations of the Status Bar, Custom Pane, and Function menus have provided a basic demonstration of the kinds of customizations possible with the Microsoft Dynamics POS SDK. Now that you've put most of the pieces in place that are necessary to implement a functioning add-in, it's time to update the code for these items so that they provide the total Happy Hour experience for cashiers and managers.

Update the HeaderPaneControl.SetHtmlPane method

Previously, the SetHtmlPane method always displayed the Happy Hour graphic, no matter the time. In this section, you will update the SetHtmlPane method to only show the Happy Hour graphic during the appropriate time slot. The new SetHtmlPane method uses the IsHappHour method to determine whether or not it is Happy Hour. It then sets the new URL for the HeaderPane based on

whether the configuration is valid and the user's preferences. Note that the StatusBar.htm file contains the HTML that is displayed by POS before it has been customized. That is, the appearance of POS will be unaffected if there are any "soft failures" in SetHtmlPane.

1. Open the HeaderPaneControl.cs file in your working project for this lab.
2. Open the HeaderPaneControl.cs file in the reference project for this lab.
3. Study the code in the SetHtmlPane method to be sure that you understand how it works, then copy its body into the body of your SetHtmlPane method.

Update the SliderMenuButton.Execute method

Previously, the SliderMenuButton.Execute method simply displayed message boxes indicating that the extension functionality had not been implemented. The logic to perform the extension is straightforward. Study the SliderMenuButton.Execute method in the reference file and copy it in to your own.

Update the StatusBarPanelProvider.UpdateStatusBar method

Previously, the UpdateStatusBar method displayed a simple happy hour message that contained the current time. By using IsHappyHour and HappyHourMinutesLeft, it is easy to update UpdateStatusBar to correctly display the number of minutes remaining in the current Happy Hour. Either do this yourself or copy the code in from the reference project.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called HappyHourDiscount in the C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session4_DataAndObjects1\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session4_DataAndObjects1\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, click **Run**.
6. In the **Run** dialog box, type *cmd*. Press Enter.
7. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Debug the add-in

- On the **Debug** menu, click **Start Debugging**.

Note
If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

lab 5

Object model and data views 2

In this lab, you will create a table for the Happy Hour add-in in the Microsoft Dynamics POS database. You will then modify DataManager.cs and PosAddIn.cs to save discount information to the database. The table you create will be used in subsequent labs.

You will also create a new receipt variable that displays a Happy Hour savings message on the customers' receipts. In addition, you will configure Microsoft Dynamics POS to enable this functionality.

Create and modify a table for the Happy Hour add-in

Setup

1. Run Visual Studio 2008 as Administrator.
2. From the **File** menu, click **Open**, and then click **Project/Solution**.
3. In the **Favorite Links** pane, click **Desktop**
4. Navigate to the Session5_DataAndObjects2\Working folder.
5. Select the **HappyHourDiscount.csproj** file, and click **Open**.

Create the SQL script to create the HappyHour table

In this section, you'll create a table in the POS database to store data related to items sold during happy hour. The CreateDBTable.sql file creates a Table called HappyHour that has four columns: ID, ItemLookpCode, DiscountedPrice, and SoldTime. ID and SoldTime do not need to be updated by your code.

1. Copy the CreateDBTable.sql file from the reference project to C:\temp\CreateDBTable.sql.
2. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
3. At the command prompt, type the following:

   sqlcmd -E -S (local)\msposinstance -i "C:\temp\CreateDBTable.sql"

Alternatively, you can add the CreateDBTable.sql file to your working project. In this case, if SQL Server 2005 is installed, you will be able to run the SQL script from inside Visual Studio.

Add POSDataManger.SaveDiscountDetails

Now that you have a table in the POS database, you can save records to it. The code in SaveDiscountDetails in the reference project for this lab runs an INSERT query that saves the lookup code and discounted price. It uses the POSDataManager.SqlConnectionProvider property to call EstablishConnectionAndRun to run the INSERT query.

- Create a public static void method called SaveDiscountDetails that takes the item lookup code as a string and the discounted price as a decimal, and uses

PosSqlConnection.EstablishSqlConnectionAndRun to insert the discounted price into the Microsoft Dynamics POS database.

Update PosAddIn.PosAddIn_ItemSalesChargeAdded

Add a call to SaveDiscountDetails in PosAddIn_ItemSalesChargeAdded to save the discount details for each discounted item. View the reference project if you have any difficulties.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called HappyHourDiscount in the C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session5_DataAndObjects2\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session5_DataAndObjects2\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.
2. Using the Happy Hour UI you have added in previous labs, ensure that it is now Happy Hour and that at least one department has discountable items.
3. Using POS, process a transaction that has at least one discounted item.
4. Exit POS.
5. In Visual Studio 2008, choose the **Tools** menu and click **Connect to Database...**.
6. In the **Add Connection** dialog box, change the **Data Source**.
7. In the **Change Data Source** dialog box, choose the **Microsoft SQL Server** data source. Click **OK**.
8. In the **Add Connection** dialog box, type *(local)\msposinstance* in the **Server Name** combo box.
9. Select the **MSPOSSample** database in the **Select or enter a database name** drop down menu.
10. In **Server Explorer**, expand Data Connections, expand the **MSPOSSample** database, and then expand **Tables**.
11. Right-click on the **HappyHour** table and click **Retrieve Data**.

    Notice the new transaction data in the HappyHour database.

Create a receipt variable

Setup

1. Run Visual Studio 2008 as Administrator.
2. From the **File** menu, click **Open**, and then click **Project/Solution**.
3. In the **Favorite Links** pane, click **Desktop**
4. Navigate to the Session6_ReceiptVariables\Working folder.
5. Select the **HappyHourDiscount.csproj** file, and click **Open**.

Add an event listener and data to track the total discount

Microsoft Dynamics POS makes event available to you through the IPosApplication.QueryInterface<T> generic method. Simply specify the type Microsoft.Rms.AddInViews.V300.Events.IPosEvents when calling QueryInterface. The IPosEvents class has instance variables that expose all the events that can be raised by Microsoft Dynamics POS.

1. Open the DiscountHelper.cs file in your project.
2. Add a public static decimal field to `DiscountHelper` called `TotalDiscount`. Initialize it to 0m.
3. Open PosAddin.cs in your project.
4. Create a void method called `PosAddIn_TransactionInitialized` that takes an IPosTransaction object as its only argument.
5. In the body of `PosAddIn_TranasctionInitialized`, set `DiscountHelper.TotalDiscount` equal to `0m`.
6. In the `SetPosApplication` method in the `PosAddIn` class, add `PosAddIn_TransactionInitialized` to the `TransactionInitialized` delegate immediately below the line that adds a listener to `ItemSalesChargeAdded`. (View that line of code to guide you.)
7. In `PosAddIn_ItemSalesChargeAdded`, add a line of code that keeps track of the amount of the discount total in `DiscountHelper.TotalDiscount` field.

Create a new receipt template

Receipt variables are stored in receipt templates, which are XML files that represent conditional logic for printing information on a receipt.

1. Copy the C:\Program Files\Microsoft Dynamics - Point of Sale\Database\2.50\SeedData\Files\Receipt_40column.xml file to your working directory and add the file to your project.
2. Open, in Visual Studio, the Receipt_40Column.xml file that is in your project.
3. Save the Receipt_40Column.xml file in the same folder as HappyHourReceipt_40Column.xml.
4. On the **Edit** menu, click **Go To...**.
5. In the **Go To Line** dialog box, type 688 and click **OK**.

Note
Visual Studio formats XML files for display. This formatting adds and removes newlines so that, in general, line locations will be different between Visual Studio and Notepad, for example. In these instructions, line 688 is immediately above the first line inside the Transaction Totals section that reads **<FOR each="tender">**.

6. Insert the following code between the **</IF>** and **<FOR each="tender">** lines.

```
<IF>
```

```
<CONDITION>len(addin.POSTestCompany.DiscountAmount)</CONDITION>
          <THEN>
                <ROW>"Happy Hour Discount|"
addin.POSTestCompany.DiscountAmount</ROW>
          </THEN>
   </IF>
```

Note
The above code inserts a row on the receipt with a label and the discount amount if the length of the discount amount is greater than zero.

7. Save the HappyHourReceipt_40Column.xml file.

Add a receipt variable provider class

To use receipt variables in an add-in, you need to implement IReceiptVariableProvider.

1. Add a class called `ReceiptVariableProvider` to your project.
2. Add the following using statements to your ReceiptVariableProvider.cs file:

```
using System.AddIn;
using System.AddIn.Contract;
using System.AddIn.Contract.Collections;
using Microsoft.Rms.AddInViews.V300.PosTransaction;
```

3. Mark the `ReceiptVariableProvider` class with the `AddIn` attribute.
4. Make the `ReceiptVariableProvider` class public.
5. Declare that `ReceiptVariableProvider` implements the `IReceiptVariableProvider` interface.
6. Use Visual Studio's **Implement Interface** right-click menu item to implement the IReceiptVariableProvider interface.
7. Implement `ServiceIdentifier` to return "POSTestCompany".
8. Implement `GetValue` so that it returns DiscountHelper.TotalDiscount, rounded to two decimal places, but only if the parameter for the GetValue method is equal to "DiscountAmount" and DiscountHelper.TotalDiscount is greater than zero.

Update the manifest file

Because you've just implemented another view, the `implementedViews` attribute of the `AddinAssembly` tag in the `HappyHourDiscount.manifest` file needs to specify this new view, in addition to any existing views.

1. In the **Solution Explorer** window, double-click the **HappyHourDiscount.manifest** node to open the HappyHourDiscount.manifest file.
2. Edit the value of the `implementedViews` attribute of the `AddinAssembly` tag so that it includes `Microsoft.Rms.AddInViews.V300.PosTransaction.IReceiptVariableProvider`. When you are done with this step, the value of the `implementedViews` attribute should be a comma-separated list of the previously implemented views and the new view, without white space or line breaks.
3. Save the manifest file.

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called HappyHourDiscount in the C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session6_ReceiptVariables\bin\Debug subfolder. Copy the HappyHourDiscount.dll file and HappyHourDiscount.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session6_ReceiptVariables\ subfolder Copy the HappyHourDiscount.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Configure Microsoft Dynamics POS

In this section, you will configure Microsoft Dynamics POS to use your new receipt template on Register 1.

1. Run Microsoft Dynamics POS.
2. In the **Microsoft Dynamics POS (Alpha)** application window, switch to Manager View.

Add the HappyHour_40Column.xml receipt template:

1. On the **Settings** menu, point to **Register Settings**, and then click **Receipt Formats**.
2. In the **Receipt Formats** pane, click **+Add New Receipt Format**.
3. In the name field in the **Receipt Format** dialog box, type *Happy Hour 40 Column*.
4. In the **Description** field, type *Tells customers how much they saved*.
5. Click the **Select Template...** button.
6. In the **File Center** dialog box, click **Add File...**.
7. In the **Add File** dialog box, navigate to your working project directory, click HappyHourReceipt_40column.xml, and then click **Open**.
8. In the **File Center** dialog box, click **HappyhourReceipt_40Column.xml** and then click **Select**.
9. In the **Receipt Format** dialog box, click **Save and Close**.

Configure printing on the register:

1. On the **Settings** menu, click **Register Settings**, and then click **View/Edit Register**.
2. Double-click **Register 1**.
3. In the **Register 1** dialog box, click **Save electronic copies of receipts**. Then choose **Happy Hour 40 Column** from the **Journaled receipt format** list.
4. Click **Save and Close**.
5. Exit Microsoft Dynamics POS.

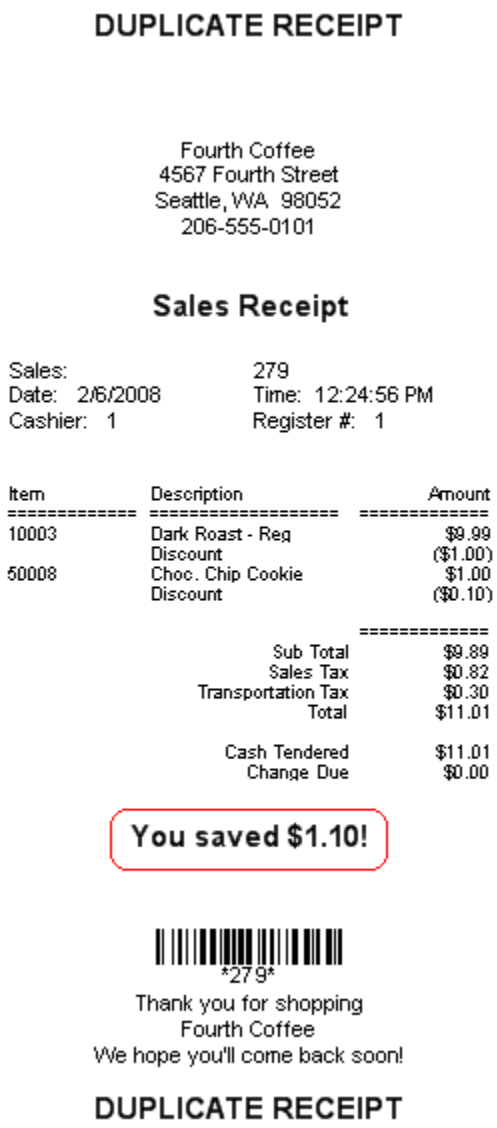
Debug the add-in

1. On the **Debug** menu, click **Start Debugging**.

   Note
   If you get an error message about class library projects needing a debug action, see the section **Debug the add-in**, on page 13.

2. Using the Happy Hour UI you have added in previous labs, ensure that it is now Happy Hour and that at least one department has discountable items.
3. Using POS, process a transaction that has at least one discounted item.
4. Click the **Tasks** function menu, and then click **Find Receipts**.
5. Select the receipt for the transaction you just processed.

Notice the **You Saved $XX.XX!** message at the bottom of the receipt:



DUPLICATE RECEIPT

Fourth Coffee
4567 Fourth Street
Seattle, WA 98052
206-555-0101

Sales Receipt

Sales: 279
Date: 2/6/2008 Time: 12:24:56 PM
Cashier: 1 Register #: 1

Item	Description	Amount
10003	Dark Roast - Reg	$9.99
	Discount	($1.00)
50008	Choc. Chip Cookie	$1.00
	Discount	($0.10)
	Sub Total	$9.89
	Sales Tax	$0.82
	Transportation Tax	$0.30
	Total	$11.01
	Cash Tendered	$11.01
	Change Due	$0.00

You saved $1.10!

279

Thank you for shopping
Fourth Coffee
We hope you'll come back soon!

DUPLICATE RECEIPT

lab 6

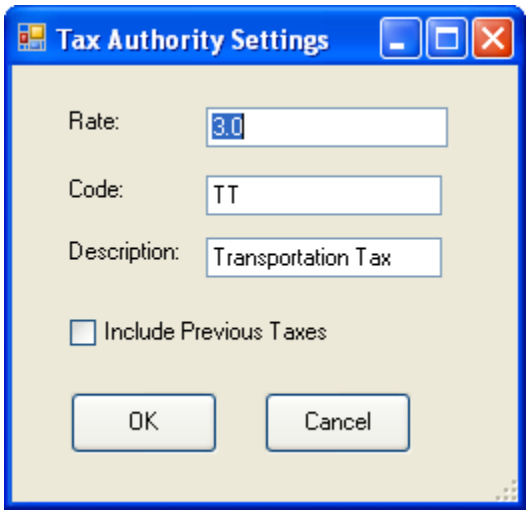
Using taxes

In this lab, you will create a configurable tax authority that implements a transportation tax. In addition, you will create the dialog box that a manager would use to configure this tax and set its rate.

Create a tax rate configuration dialog

The configuration dialog for configuring the tax authority is straightforward Windows Forms code with some sensible tax-related member variables and methods. The TaxRateDialog class has four private variables to store tax-related state: decimal taxRate, string code, string description, and bool includePreviousTaxes. There are four getter methods that simply return these values: GetTaxRate, GetDescription, GetCode, and GetIncludePreviousTaxes. It performs basic input validation as well. You should simply copy TaxRateDialog.cs, TaxRateDialog.resx, and TaxRateDialog.designer.cs from the reference project into your own.

The tax authority configuration dialog is shown below, with sample input:



Create the project

1. Run Microsoft Visual Studio 2008 as Administrator.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Project types:** pane in the **New Project** dialog box, expand **Visual C#**, click **Windows**, and then click **Class Library** in the **Templates:** pane.
4. In the **Name:** text area type, *TaxAuthority*.
5. Click the **Browse** button.

6. In the Project Location dialog box, navigate to C:\Documents and Settings\yourusername\Desktop\SDKBootCamp\Session7_TaxAuthorities\Working.
7. Uncheck the **Create directory for solution** check box.
8. Click **OK**.

Create the TaxAuthority add-in

Add the necessary references

1. On the **Project** menu, click **Add Reference**.
2. In the **Add Reference** dialog box, click the **.NET** tab.
3. Press and hold the **Ctrl** key and click **System.AddIn**, **System.AddIn.Contract**., and **System.Windows.Forms**.
4. Click **OK**.
5. On the **Project** menu, click **Add Reference**.
6. In the **Add Reference** dialog box, click the **Browse** tab.
7. In the **Add Reference** dialog box, navigate to C:\Program Files\Microsoft Dynamics - Point of Sale\AddInViews\.
8. Select **Microsoft.Rms.AddInViews.V300.dll**. Click **OK**.
9. Right-click on **Class1.cs** in **Solution Explorer** and click **Rename**. Rename Class1.cs to TaxAuthority.cs. If you are prompted to change all references, click **Yes**.
10. Open the TaxAuthority.cs file and add the following using statements:

```
using System.AddIn;

using System.Windows.Forms;

using Microsoft.Rms.AddInViews.V300.PosTransaction;

using Microsoft.Rms.AddInViews.V300.Payments.Core;
```

Mark the TaxAuthority class with the AddIn attribute

In the TaxAuthority.cs file, add the following line immediately above the definition of the `TaxAuthority` class:

```
[AddIn ("Boot Camp Tax Authority Add-in", Version="1.0.0.0")]
```

Note
`TaxAuthority` was correctly marked as public by default. When you define your own add-in classes, remember that classes marked with the **AddIn** attribute must be public. **Visual Studio 2008** does not mark as public classes that you add by using the **Project** menu or **Solution Explorer**.

Implement the basic portions of ITaxAuthorityProvider

ITaxAuthorityProvider.GetName simply returns a descriptive string.
ITaxAuthorityProvider.GetAddInId simply returns a GUID.

1. Declare that the TaxAuthority class implements ITaxAuthorityProvider.
2. Mark `TaxAuthority` public.

3. Right-click `ITaxAuthorityProvider` and choose **Implement Interface**. Then click **Implement Interface**.
4. Implement `GetName` so that it returns "Boot Camp Tax Authority Provider".
5. Create a private member variable for the `TaxAuthority` class called `taxAuthorityGuid` and assign it to a new Guid.
6. Implement `GetAddInId` to return `taxAuthorityGuid`.

About the IDictionaryAddInView addInData parameter

Each of ShowConfigurationDialog, LimitedTaxableAmount, and ComputedTaxableAmount have an addInData parameter of type IDictionaryAddInView. Each of these three methods is called by POS to perform tax-related logic. The addInData parameters that are passed to each of these three methods point to the same IDictionaryAddInView instance, which POS maintains internally. This is how tax information is shared between these three methods, as well as taxAuthorityUpdater, which is of the type ITaxAuthorityUpdater. For example, the ShowConfigurationDialog method operates on addInData by adding keyed information to the dictionary, which ComputedTaxRate can then access in order to calculate the tax when POS asks for it.

Implement ITaxAuthorityProvider.ShowConfigurationDialog

The ShowConfigurationDialog method gets the current tax rate from the tax authority, if present, and displays the TaxRateDialog. It then grabs the user's setting and stores them back into the tax authority by using IDictionaryAddInView that it was passed.

- Study the TaxAuthority.cs file in the reference project and then either implement this code on your own or copy it in to your project.

Implement ITaxAuthorityProvider.LimitedTaxableAmount

The LimitedTaxableAmount method is called by POS and simply caps the amount that can be taxed. It compares the taxable amount it receives to the limitedTaxableAmount variable and returns the appropriate value. Again, note that it retrieves the tax rate from the dictionary that it receives.

- Add an implementation to your `LimitedTaxableAmount` method that returns the taxable amount if the taxable amount is less than the limit, but returns the limit otherwise.

Implement ComputedTaxAmount

The ComputedTaxAmount method is called by POS to compute the tax on an item. It retrieves the tax rate from the dictionary that it receives from POS.

- Implement the `ComputedTaxAmount` method to check for the tax rate and, if the rate exists, compute the extended tax rate. Otherwise, it should return 0. View the reference implementation if you get stuck.

Create the manifest file

You will need to create a manifest file that declares that TaxAuthority.dll implements the ITaxAuthorityProvider interface.

Note
In the alpha version of Microsoft Dynamics POS, an incorrect manifest file can cause your add-in not to load or create hard-to-find crashing bugs in your add-in.

1. In the Solution Explorer pane in the **Visual Studio 2008** application window, right-click the **TaxAuthority** node, click **Add**, then click **New Item**.
2. In the **Categories** pane in the **Add New Item** dialog box, click **General**.

3. In the **Templates** pane, select **Application Manifest File**.
4. In the **Name** text area, type *TaxAuthority.manifest*. Click **Add**.
5. Replace the contents of the TaxAuthority.manifest file with the following XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<AddinManifest>
   <AddinAssembly
     assembly="TaxAuthority.dll"

implementedViews="Microsoft.Rms.AddInViews.V300.PosTransaction.ITaxAut
horityProvider"
     namedPermissionSet="FullTrust"
/>
</AddinManifest>
```

Build and deploy the add-in

1. Save your work. Then, on the **Build** menu, click **Build Solution**.
2. Create a folder called TaxAuthority in the C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\ folder, if you haven't already done so.
3. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session7_TaxAuthorities\TaxAuthority\bin\Debug subfolder. Copy the TaxAuthority.dll file and TaxAuthority.pdb file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\HappyHourDiscount\.
4. Open the **SDKBootCamp** folder on the Desktop and navigate to the Session6_ReceiptVariables\TaxAuthority subfolder Copy the TaxAuthority.manifest file from here to C:\Program Files\Microsoft Dynamics - Point of Sale\Addins\TaxAuthority\.
5. On the **Start** menu, point to **All Programs**, then point to **Accessories**, then right-click **Command Prompt**, and then click **Run as administrator**.
6. At the command prompt, type:

   %windir%\Microsoft.Net\Framework\V3.5\AddinUtil.exe -rebuild -PipelineRoot:"C:\Program Files\Microsoft Dynamics - Point of Sale"

Configure Microsoft Dynamics POS

1. Run Microsoft Dynamics POS.
2. Switch to Manager View.

Add the tax authority:

1. On the **Settings** menu, point to **Store Settings**, and then click **Tax Authorities**.
2. In the **Tax Authorities** pane, click **+Add a New Tax Authority**.
3. In the **Select Tax Authority Provider** dialog box, select **Boot Camp Tax Authority Provider**. Click **OK**.
4. In the **Tax Authority Settings** dialog box, enter *3.0* in the **Rate** field, *TT* in the **Code** field, and *Transportation Tax* in the **Description** field. Click **OK**.

Note
The tax rate for this tax authority will not show up in the **Tax Rate (%)** column of the **Tax Authorities** pane.

Include the tax in the calculation:

1. On the **Settings** menu, click **Store Settings**, and then click **Sales Taxes**.
2. Double-click **TAX**.
3. In the **Sales Tax** dialog box, find the first available blank **Tax Authority** in the **Included tax authorities** group.
4. Choose the TT tax authority for the tax authority located in step 3, and check the corresponding **Receipt** checkbox.
5. Click **Save and Close**.
6. Exit Microsoft Dynamics POS.

Debug the add-in

1. On the **Project** menu, click **TaxAuthority Properties**.
2. On the **TaxAuthority**, click the **Debug** tab.
3. In the **Start Action** radio button group, select **Start external program:**.
4. In the **Start external program:** text area, type:

   C:\Program Files\Microsoft Dynamics - Point of Sale\pos.exe
5. In the **Working directory:** text area in the **Start Options** group, type:

   C:\Program Files\Microsoft Dynamics - Point of Sale
6. Close the **TaxAuthority** tab.
7. On the **Debug** menu, click **Start Debugging**.
8. Using POS, process a transaction that has at least one item.

Notice the **Transportation Tax** item near the bottom of the receipt in Microsoft OneNote 2007.

DUPLICATE RECEIPT

Fourth Coffee
4567 Fourth Street
Seattle, WA 98052
206-555-0101

Sales Receipt

Sales:	278
Date: 2/6/2008	Time: 12:22:53 PM
Cashier: 1	Register #: 1

Item	Description	Amount
10205	10 Cup Automatic Drip	$24.99
10204	8 Cup Automatic Drip	$19.99
	Sub Total	$44.98
	Sales Tax	$3.71
	Transportation Tax	$1.35
	Total	$50.04
	Cash Tendered	$50.04
	Change Due	$0.00

Thank you for shopping
Fourth Coffee
We hope you'll come back soon!

DUPLICATE RECEIPT

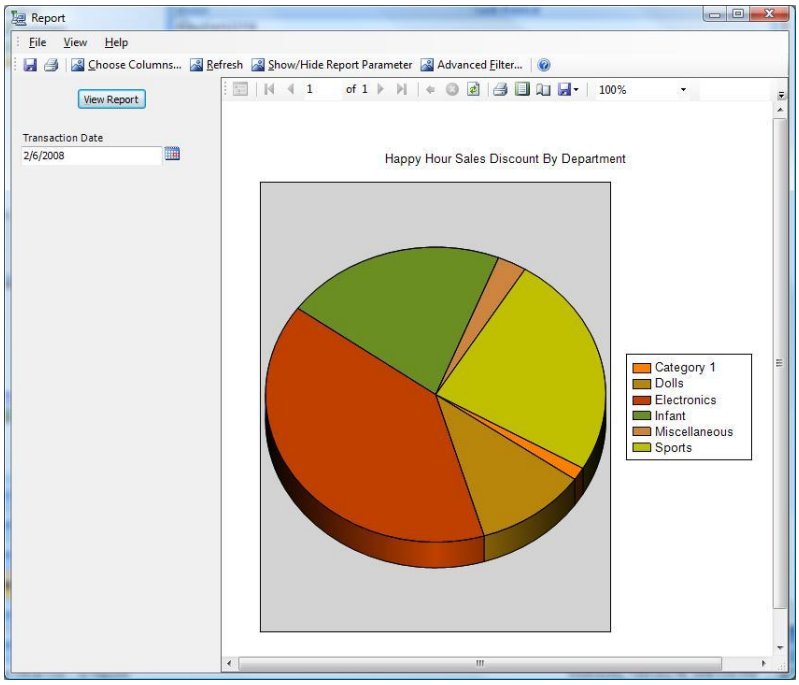
lab 7

Creating custom reports

With Microsoft Dynamics POS and the powerful Microsoft SQL Server reporting services, you can create sophisticated custom reports using a variety of report layouts and tools.

In this lab, you will use the SQL Server Business Intelligence Development Studio to create simple table and pie-chart reports, and then use File Center in Microsoft Dynamics POS to make the reports available to users.

The following image shows one of the reports you will create in this lab. It gives you an idea of the attractive customizations that are possible.



Setup

Since the reports in this lab reference data in the HappyHour table, process a few transactions in Microsoft Dynamics POS to make sure there are Happy Hour transactions in the database.

1. In Microsoft Dynamics POS, check the status bar in POS View to make sure Happy Hour is active.

   Note
   If you need to start Happy Hour, switch to Manager View, click Happy Hour Discount on the Inventory menu, and then change the start and end times as needed.

2. Press F2 (Items), press 4 (By Item Number), and then add to the transaction an item that will receive the Happy Hour discount.

For example, if you included the Baked Goods department in the discount, add a bagel, cookie, or muffin to the transaction.

3. Press F12 (Total), and then press the plus (+) sign.

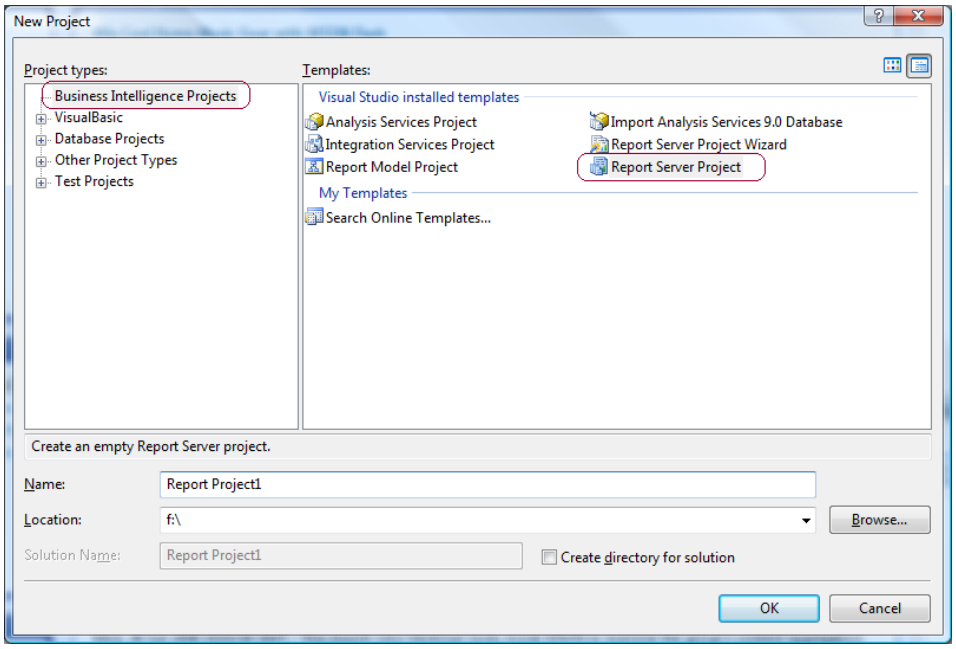
Create a report project

1. Open SQL Server Business Intelligence Development Studio.

On the Start menu, point to Programs or All Programs, point to Microsoft SQL Server 2005, and then click SQL Server Business Intelligence Development Studio.

Note
SQL Server Business Intelligence Development Studio is a plug-in for Visual Studio. To make it available on the Start menu, you must first enable IIS and ASP.NET and install the SQL Server reporting services (an optional component in SQL Server Setup).

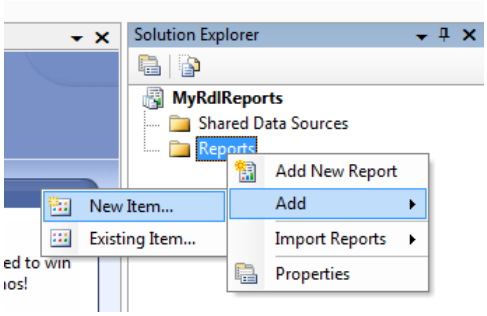
2. In the New Project dialog box, select Report Server Project, type a name and location for the project, and then click OK.



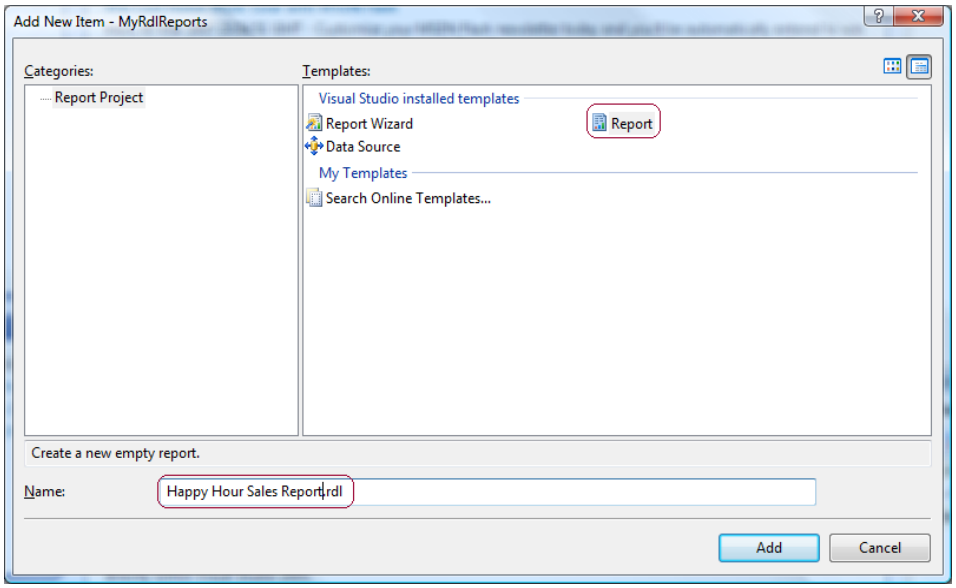
Create a custom table report

Add a report to the project

1. In Solution Explorer, right-click Reports, point to Add, and then click New Item.

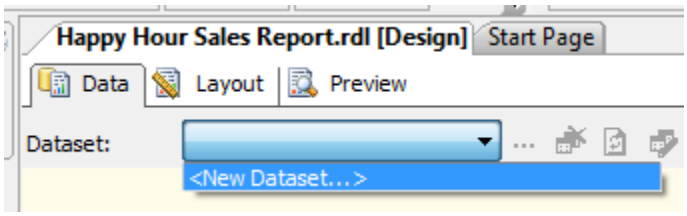


2. In the Add New Item dialog box, select Report under **Templates**, type "Happy Hour Sales Report.rdl" in the Name box, and then click Add.

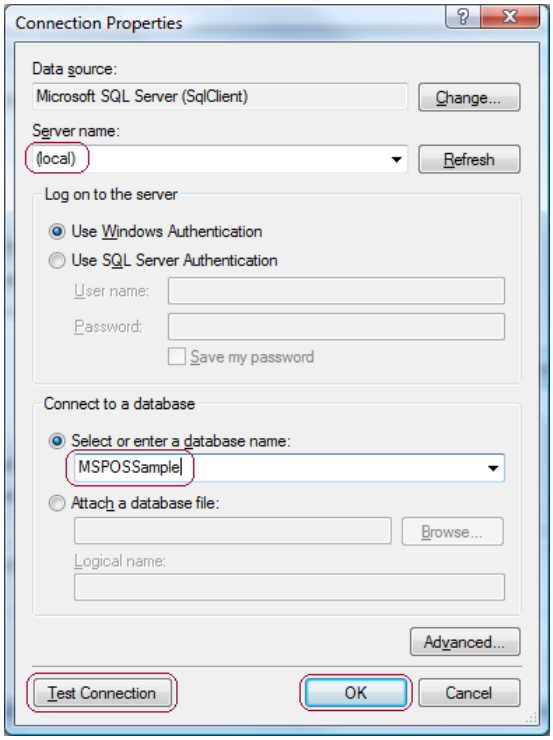


Specify the dataset

3. In the Dataset box, select New Dataset.



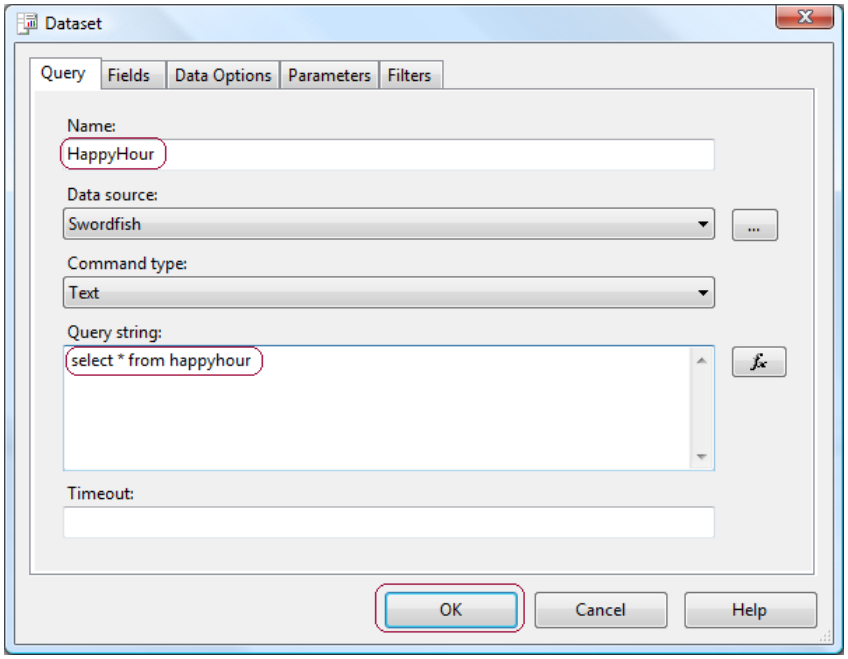
4. In the Data Source dialog box, type "Swordfish" in the Name box, and then click Edit.
5. In the Connection Properties dialog box, type "(local)" in the Server name box, type "MSPOSSample" in the Select or enter a database name box, click Test Connection, and then, if the test was successful, click OK.



6. Back in the Data Source dialog box, click OK.

7. Click the Edit Selected Dataset button on the toolbar. 

8. In the Dataset dialog box, type “HappyHour” in the Name box, type “select * from happyhour” in the Query string box, and then click OK.

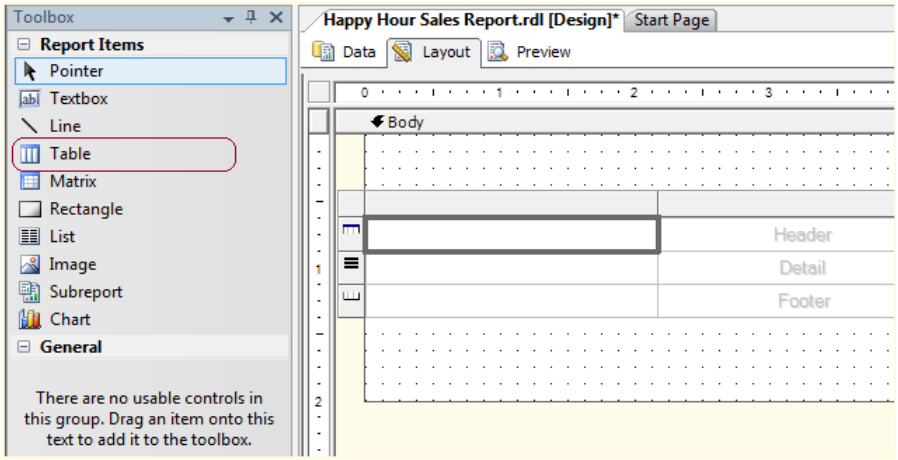


9. Run the HappyHour dataset to make sure data is returned.

On the toolbar in Visual Studio, click the Run button. 

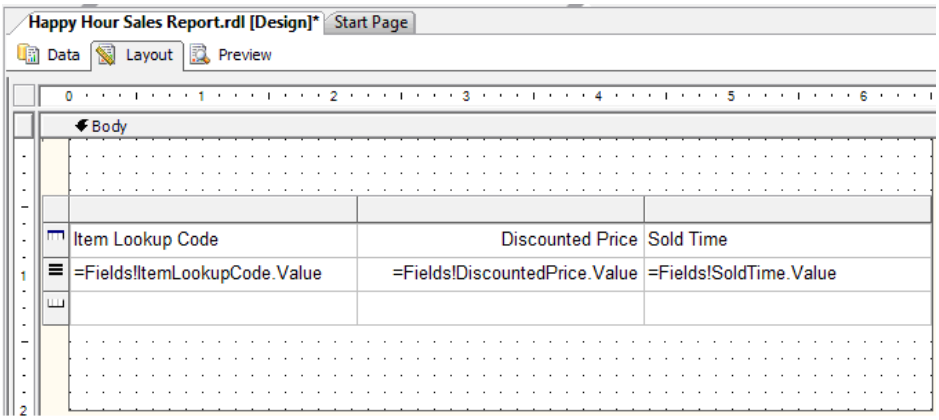
Design the report

10. Switch to the Layout tab, and then, on the View menu, click Toolbox.
11. From the Report Items list in the Toolbox, drag the Table item into the body of the report.

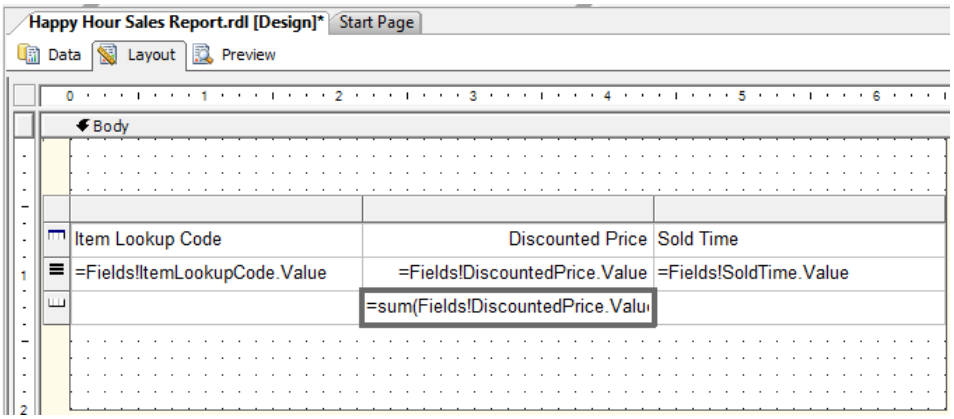


12. On the View menu, click Datasets, and then, one at a time, drag ItemLookupCode, DiscountedPrice, and SoldTime into the cells in the header row of the table.

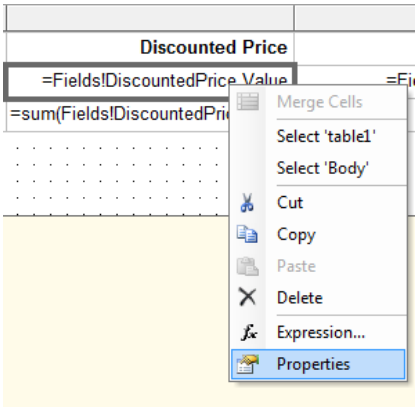
The query for each column will be filled into the detail row automatically.



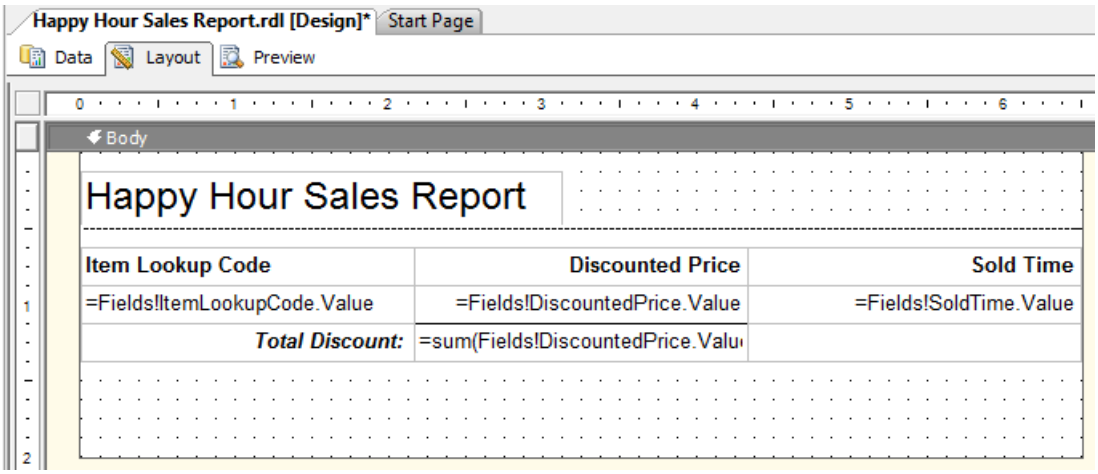
13. Add a total to the report by typing "=sum(Fields!DiscountedPrice.Value)" into the footer cell in the Discounted Price column.



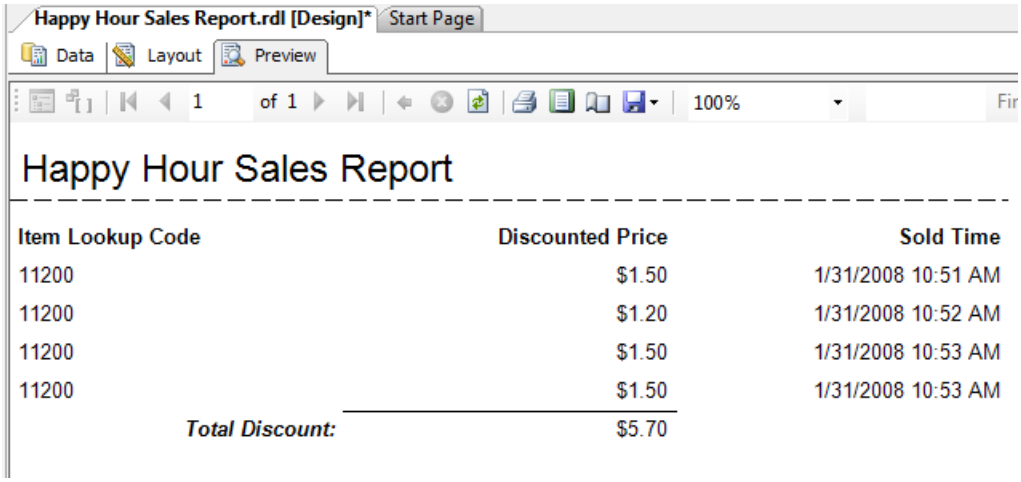
14. Switch to the Preview tab to see how the report looks so far, and then switch back to the Layout tab to begin formatting the report.
15. To add currency formatting to the prices in the report, right-click the Discounted Price query, and then click Properties.



16. In the Choose Format dialog box, select Currency, and then click OK.
17. To add date formatting to the Sold Time values, repeat Steps 15 and 16, selecting Date in the Choose Format dialog box.
18. To add a title to the report, add a textbox above the table, type the title, and then apply any desired formatting. You can also add a label to the total.



19. Switch to the Preview tab again and make sure you are satisfied with how the report looks.



20. On the File menu, click Save.

Create a pie-chart report

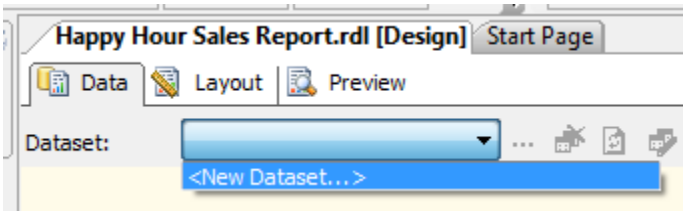
The steps for creating a pie-chart report are very similar to those for creating a table report.

Add a report to the project

1. In Solution Explorer in Visual Studio, right-click Reports, point to Add, and then click New Item.
2. In the Add New Item dialog box, select Report under **Templates**, type "Happy Hour Sales by Department.rdl" in the Name box, and then click Add.

Specify the dataset

3. In the Dataset box, select New Dataset.



4. In the Data Source dialog box, type "Swordfish" in the Name box, and then click Edit.
5. In the Connection Properties dialog box, type "(local)" in the Server name box, type "MSPOSSample" in the Select or enter a database name box, click Test Connection, and then, if the test was successful, click OK.
6. Back in the Data Source dialog box, click OK.
7. Click the Edit Selected Dataset button on the toolbar. 
8. In the Dataset dialog box, in the Name box, type "HappyHourByDepartment".
9. In the Query string box, type the following:

```
Select  department.name as 'Department Name',  sum(HappyHour.DiscountedPrice) as 'Total Discount'

From Department right outer JOIN
```

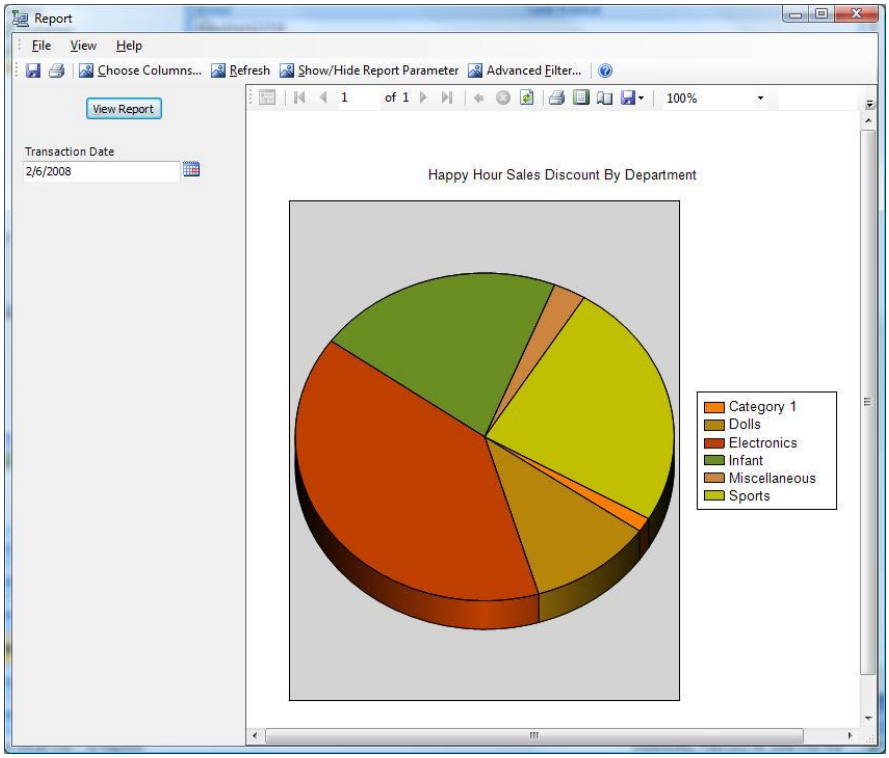
```
                Item ON Department.ID = Item.DepartmentID RIGHT OUTER JOIN

                HappyHour ON Item.itemlookupcode = HappyHour.itemlookupcode

Where happyhour.soldtime >= @TransactionDate and  happyhour.soldtime < DateAdd(day, 1,
@TransactionDate)

Group by  department.name
```

10. Run the HappyHourByDepartment dataset, entering today's date to make sure data is returned.

On the toolbar in Visual Studio, click the Run button. 

Design the report

11. Switch to the Layout tab, and then, on the View menu, click Toolbox.
12. From the Report Items list in the Toolbox, drag the Chart item into the body of the report.
13. On the View menu, click Datasets.
14. Drag Department_Name into the Drop category fields here region of the report, and then drag Total_Discount into the Drop data fields here region.
15. Right-click on the chart, point to Chart Type, point to Pie, and then click Simple Pie.
16. Right-click on the chart, and then click Properties.
17. On the General tab, type "Happy Hour Sales by Department" in the Title box, and then select Earth Tones in the Palette box.
18. On the 3D Effects tab, select Display chart with 3-D visual effects, and then click OK.
19. On the Report menu, click Report Parameters, and then take the following actions:
    a. In the Data type box, select DateTime.
    b. In the Prompt box, type "Transaction Data".
    c. Under Default values, select Non-queried and enter "=Today".
20. Click OK.
21. Switch to the Preview tab to see how the report looks so far.
22. Switch to the Layout tab to continue customizing the report
23. Switch to the Preview tab again and make sure you are satisfied with how the report looks.
24. On the File menu, click Save.



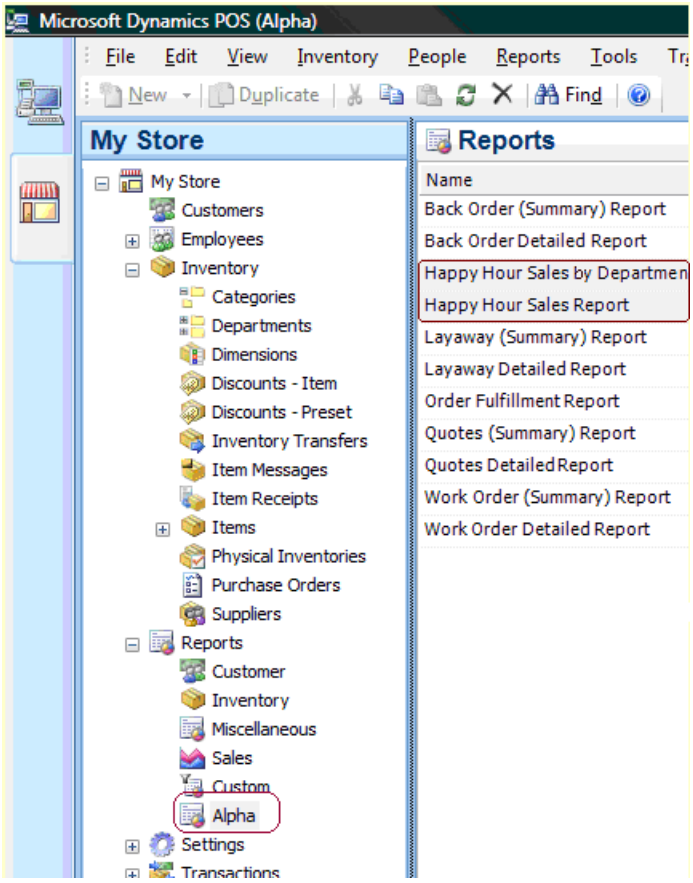
Make a custom report available in Microsoft Dynamics POS

You can make the custom report available in Microsoft Dynamics POS simply by adding it to File Center. And because the custom report uses a data source name instead of a specific database name to connect to the store database, the same report .rdl file can be used in more than one store, without modification.

1. In Microsoft Dynamics POS, switch to Manager View, and then, on the Tools menu, click File Center.
2. In the folder tree, expand the Stingray Reports folder, and then click Miscellaneous.
3. Click Add File, browse to the Happy Hour Sales Report.rdl file, select it, and then click Open.

   Note
   Browse to the location where you previously saved the file.
4. Repeat Step 4 for the Happy Hour Sales by Department.rdl file.
5. Click Close.
6. In the navigation pane, expand the Reports folder, and then click Alpha.

The Happy Hour Sales and Happy Hour Sales by Department reports appear in the list.



7. Double-click the report name to view the report.