**OBA Sample Application Kit for PeopleSoft:**

**Hands-on Lab**

**Abstract**: This hands-on lab was developed in conjunction with the “OBA Sample Application Kit for PeopleSoft” and demonstrates the key PeopleSoft/Office development features presented by this sample application. These development features include exposing a PeopleSoft component as a web service, consuming this web service in .NET, exposing the information retrieved from this web service in a custom Outlook task pane containing a WPF/XAML control, invoking the custom task pane from a custom Outlook menu, creating a custom form region that will be used to write to PeopleSoft via a web service, deploying the application using Click-Once technology and accessing PeopleSoft through a custom wrapper web service accessed by Microsoft Office SharePoint Server 2007 via the Business Data Catalog.

|  |  |
| --- | --- |
| mslogo.jpg | 872.gif |

# Table of Contents

[Table of Contents 2](#_Toc215008603)

[Applies To 3](#_Toc215008604)

[Introduction 3](#_Toc215008605)

[Overview 4](#_Toc215008606)

[Labs 4](#_Toc215008607)

[Creating a Component Interface and Exposing an Web Service Form PeopleSoft 4](#_Toc215008608)

[Background 4](#_Toc215008609)

[Explore PeopleSoft HRMS and Component Interaces 5](#_Toc215008610)

[Create Component Interface 11](#_Toc215008611)

[Associating Security with the Component Interface 14](#_Toc215008612)

[Exporting Component Interfaces as Web Services 20](#_Toc215008613)

[Provide Web Service 22](#_Toc215008614)

[Associating Security with the Web Service 25](#_Toc215008615)

[Consuming a PeopleSoft Web Service 31](#_Toc215008616)

[Background 31](#_Toc215008617)

[Consuming a PeopleSoft Web Service with Visual Studio 2008 31](#_Toc215008618)

[Encapsulate the web service in a class 41](#_Toc215008619)

[Creating an Outlook Add-in 47](#_Toc215008620)

[Creating an Outlook 2007 Task Pane using XAML 47](#_Toc215008621)

[Hooking an Outlook 2007 Task Pane with a Menu 57](#_Toc215008622)

[Creating a Form Region in Outlook 2007 65](#_Toc215008623)

[Sending the Evaluation Form 75](#_Toc215008624)

[ClickOnce Installation 78](#_Toc215008625)

[Viewing PeopleSoft data to MOSS 2007 84](#_Toc215008626)

[Business Data Catalog 84](#_Toc215008627)

[Creating Application Definition Files using the BDC Editor 84](#_Toc215008628)

[Displaying Data from PeopleSoft in MOSS 2007 Web parts 88](#_Toc215008629)

[Appendix A: Wrapper Web service for BDC/PeopleSoft Integration 93](#_Toc215008630)

[Interacting with PeopleSoft 93](#_Toc215008631)

[Web Method Design 93](#_Toc215008632)

[Appendix B: Progress Display Bar 97](#_Toc215008633)

# Applies To

Visual Studio 2008 including Service Pack 1, Visual Studio Tools for Office 3.0, Outlook 2007, Microsoft Office SharePoint Server 2007 and PeopleSoft Human Resource Management System.

# Introduction

This hands-on lab was developed in conjunction with the OBA Sample Application Kit for PeopleSoft and demonstrate the key PeopleSoft/Office development features presented by this sample application. These include:

1. Exposing a PeopleSoft component as a web service by exposing PeopleSoft components using component interfaces. PeopleTools and the PeopleSoft web client are used to perform this task. Security issues related to PeopleSoft and component interface and web service development are addressed.
2. Consuming the PeopleSoft web service in .NET using a class library to mask the complexity of the underlying PeopleSoft web service.
3. Developing an Outlook task pane containing a WPF/XAML control will be used to access this web service encapsulation.
4. Invoking the custom task pane from a custom Outlook menu
5. Creating a custom form region that will be used to write to PeopleSoft,
6. Using Click-Once technology to deploy the application thus allowing the code to auto-update.
7. Accessing PeopleSoft information through a custom wrapper web service accessed Moss2007 via the Business Data Catalog.

This sample lab reiterates how the OBA Sample Application Kit for PeopleSoft can assist developers in creating .NET applications that integrate with Outlook 2007 using Visual Studio 2008 and Visual Studio Tools Office (VSTO) 3.0. The Office Business Application (OBA) developed interacts (read, create, update and delete) with PeopleSoft Human Resource Management System 9.0\. This sample application also includes a guideline on integrating PeopleSoft with Microsoft Office SharePoint Server 2007 (MOSS 2007) through the Business Data Catalog using an Application Definition File (ADF). Also demonstrated is ClickOnce installation and technologies introduced as of .NET 3.0, namely Windows Presentation Foundation (WPF) and Extensible Application Markup Language (XAML).

# Overview

The sample application implements a portion of a human resources (HR) management system and focuses on the recruitment and evaluation of candidates. Developers can make use of this hands-on lab as a guideline in developing applications that integrate enterprise services with Office 2007 and integrating enterprise services with Microsoft Office SharePoint Server (MOSS 2007).

The Outlook 2007 module of this sample application showcases the following functionality:

* C# 3.0
* Visual Studio 2008
* Visual Studio Tools for Office 3.0
* Windows Presentation Foundation (WPF)
* Extensible Application Markup Language (XAML)
* Developing Outlook add-ins including
  + Custom task panes
  + Messages containing custom forms regions
  + Extending Outlook menus

The MOSS 2007 module of this sample application showcases the following functionality:

* Access PeopleSoft through a Web service exposed by the developer in order to access a PeopleSoft component

The complete functionality of the sample application this lab seeks to augment can be found at: <http://code.msdn.microsoft.com/obapsftsak>. The following papers were developed with this lab and should be in conjunction with this hands-on lab:

* OBA Sample Application Kit for PeopleSoft: Application Overview & Installation
* OBA Sample Application Kit for PeopleSoft: Application Walkthrough
* OBA Sample Application Kit for PeopleSoft: Deep Dive Technical Whitepaper

Note: In order to run this lab, you must set have your environment set up as is outlined in the OBA Sample Application Kit for PeopleSoft. You can find the installation guide for the software off of this link: <http://code.msdn.microsoft.com/obapsftsak/Release/ProjectReleases.aspx?ReleaseId=845>.

# Labs

## Creating a Component Interface and Exposing an Web Service from PeopleSoft

This section describes how a component (core implementation element) is ultimately exposed as a web service.

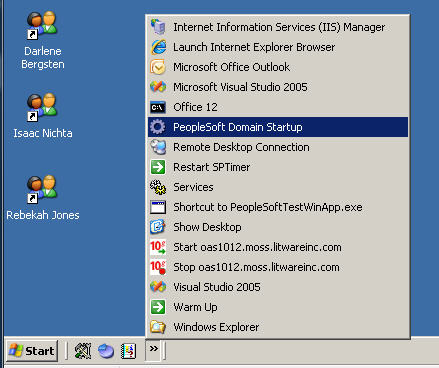
### Background

A Component Interface (CI) is an object within PeopleSoft that supports integration with external technologies such as a .NET application that will access PeopleSoft. The purpose of a component interface is to provide access to a component (a web page or pages within PeopleSoft) while maintaining the component’s business logic and data integrity. Core system logic inside PeopleSoft is implemented as components, or a PeopleSoft artifact. Components are then exposed as component interfaces, which can then be exposed as Web services. These Web services are described using Web services Description Language (WSDL). Basically, CI can be used to extract or enter information with business logic enforced automatically.

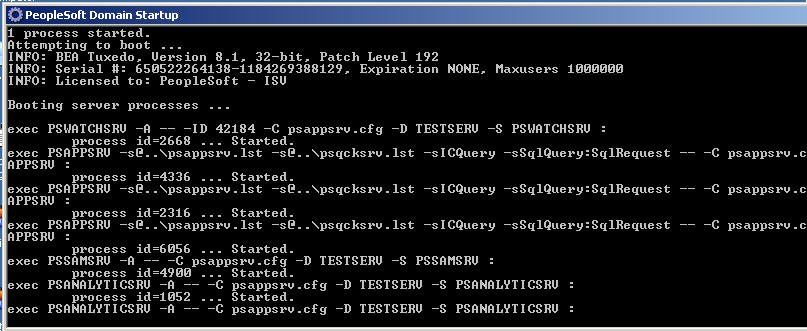
### Explore PeopleSoft HRMS and Component Interfaces

The steps required to create a component interface are as follows

1. From the VPC start PeopleSoft using the shortcut provided:



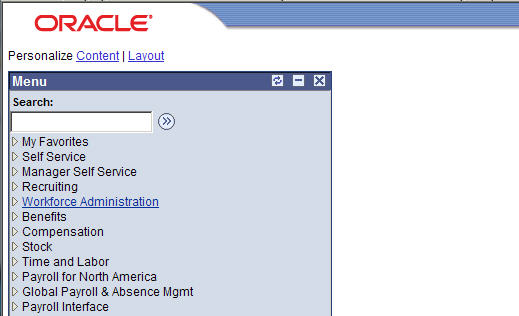
1. Allow the console window displayed to complete the start up of the PeopleSoft Domain (once the window closes, continue):



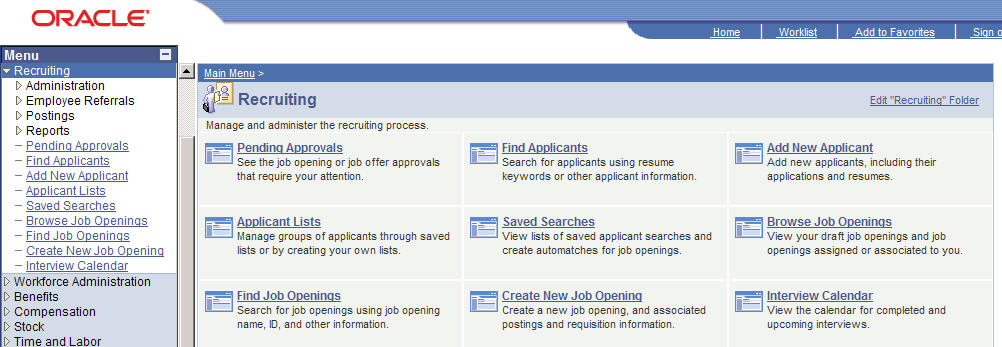
1. PeopleSoft HRMS is we based application that can be invoked by running Internet Explorer and selecting the second tab (the first tab is MOSS 2007):



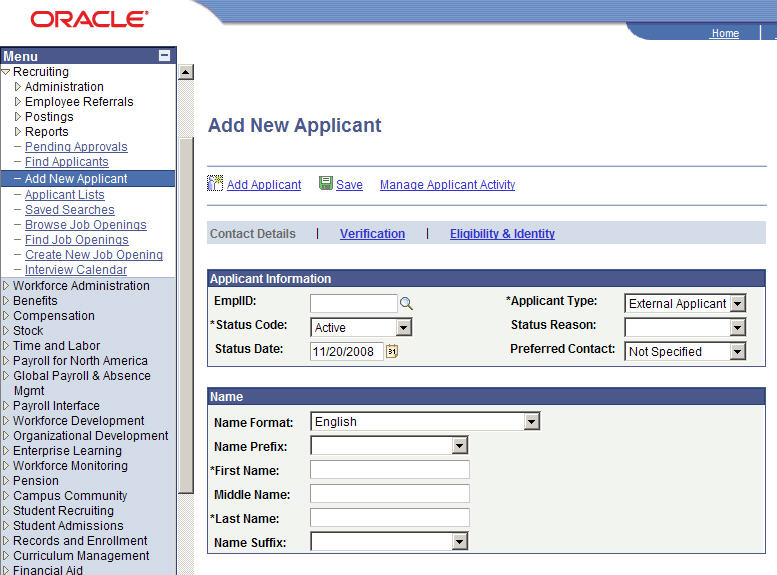
1. Login using the following credentials and then click Sign In:
   1. User ID: PS
   2. Password: PS
2. Once logged in the following screen is displayed:



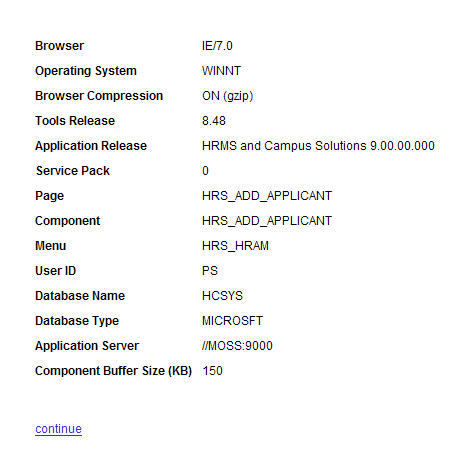
1. Click on the Recruiting link:



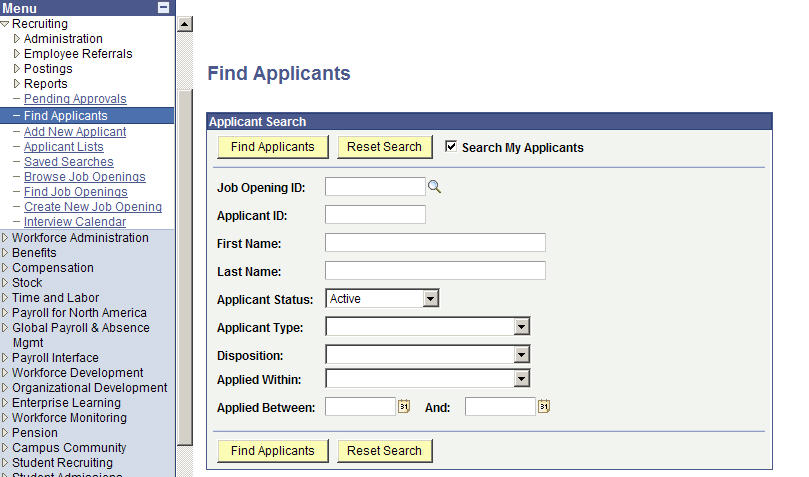
1. Click on the Add New Applicant link (either from the left-navigation pane or from the body of the recruiting page):



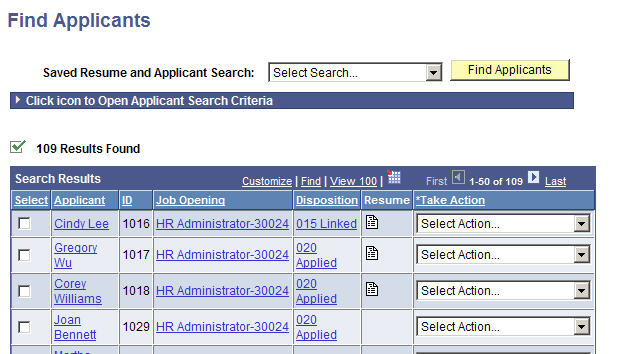
1. The Add New Applicant screen is fairly intuitive but PeopleSoft developers recognize this page is actually an HTML veneer on the HRS\_ADD\_APPLICANT component. The component is ultimately what will be exposed via a component interface and then exposed as a web service. In order to see the underlying component enter CTRL-J on the page which will reveal the following screen:



1. Notice in the previous screenshot the component name is HRS\_ADD\_APPLICANT.
2. Before exposing HRS\_ADD\_APPLICANT as a component interface and then a web service, we will take a look at the sample data exposed by PeopleSoft with respect to recruiting. In order to demonstrate this click on Recruiting | Find Applicants in the left navigation which displays the following:



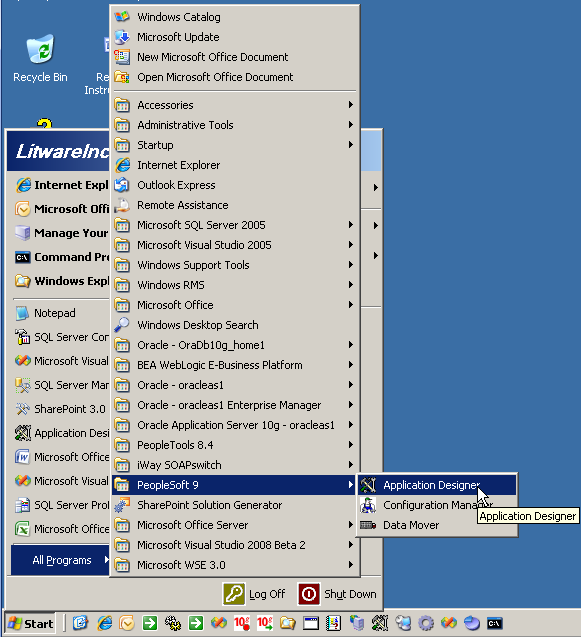
1. From the Find Applicants screen click on the Find Applicants button Find Applicants:



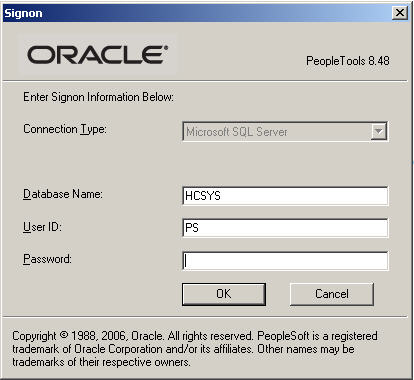
1. Note in the list of Applicants returned Cindy Lee has an applicant ID of 1016. This ID will be used in later portions of this lab to retrieve data specific to Cindy Lee.

### Create Component Interface

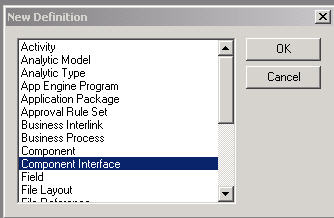
1. The core development tool of PeopleSoft is Application Designer. This tool can be used to create component interfaces such as a component interface that exposes the HRS\_ADD\_APPLICANT component. Launch Application Designer by Start | All Programs | PeopleSoft 9 | Application Designer:



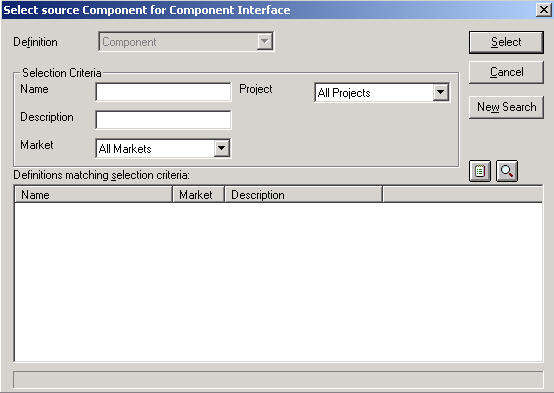
1. The will display the Sign-on screen for the Application designer:



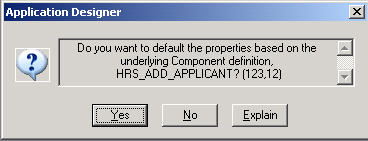
1. The credentials user ID = PS and Password = PS can be entered following by clicking on OK to display the Application Designer.
2. From Application Designer select File | New



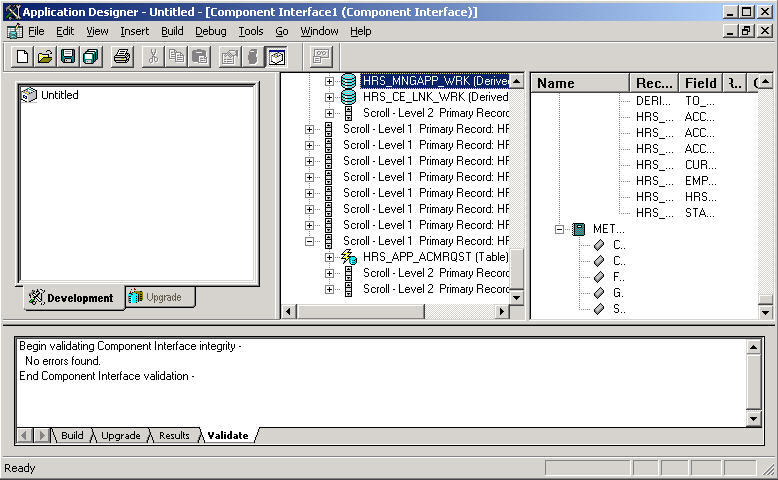
1. From The New Definition dialog select Component Interface and select OK



1. Under Name, enter the name of the component interface: HRS\_ADD\_APPLICANT and click on Select.



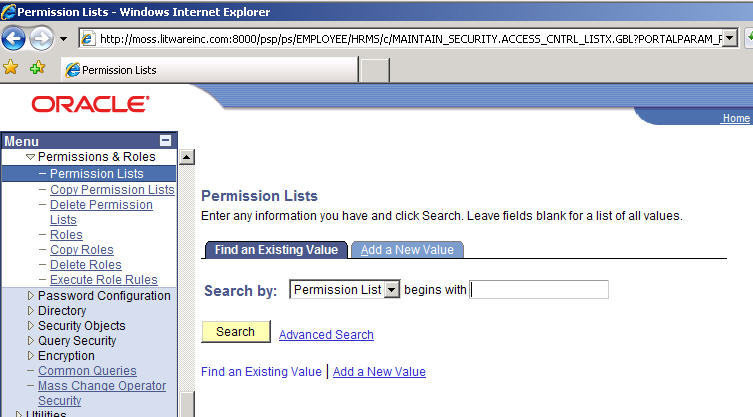
1. From the Application Designer confirm Yes:



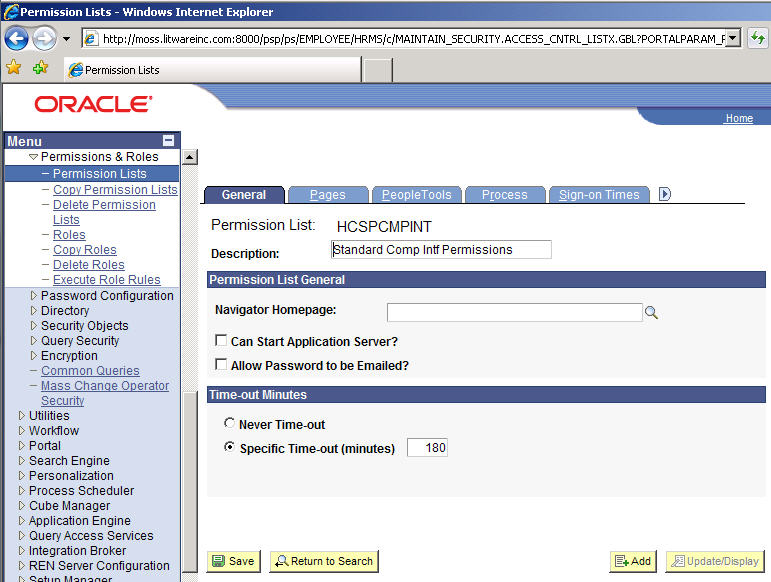
1. Select File | Save As to save the newly created component interface. Name the component interface HOL\_ADD\_APPLICANT.

### Associating Security with the Component Interface

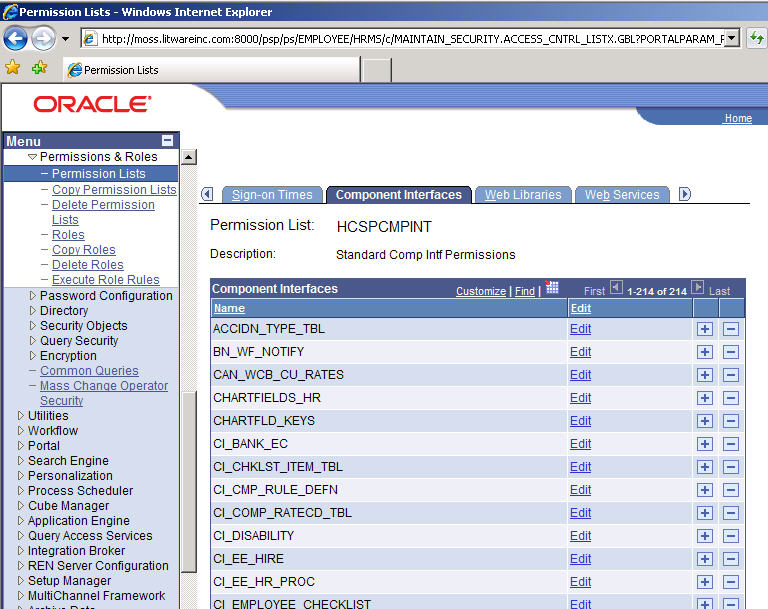
1. Open the PeopleSoft web client and login (username: PS, password: PS for the test system).
2. From the PeopleSoft web client navigate to PeopleTools | Security | Permission & Roles | Permission Lists.



1. Enter HCSPCMPINT in the textbox next to “begins with” and click on Search:



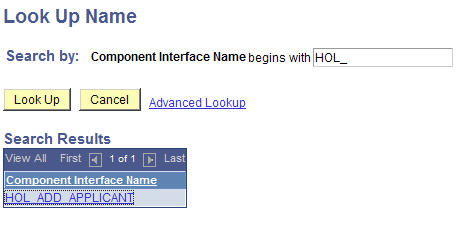
1. Click on the FI1.jpgbutton and select the Component Interfaces tab:



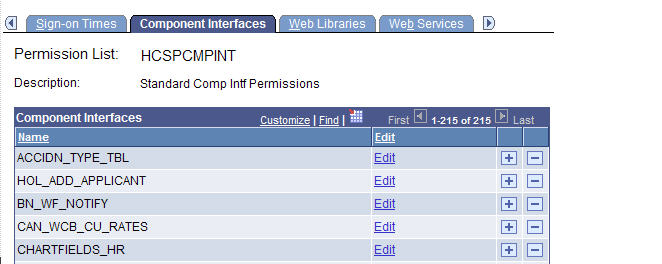
1. Click on the + button of any component interface such as the one next to ACCIDN\_TYPE\_TBL:



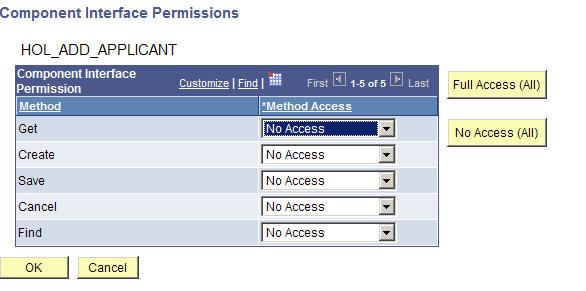
1. Enter HOL\_ into the textbox and click on the magnify glass:



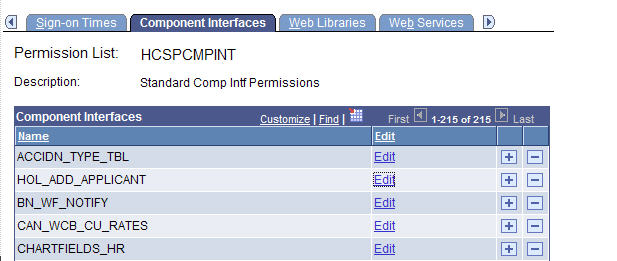
1. Click on the HOL\_ADD\_APPLICANT link which displays:



1. Click on the Edit link next to HOL\_ADD\_APPLICANT:



1. Click on the Full Access (All) button which will set Get, Create, Save, Cancel and Find to Full Access. Then click on OK which displays:



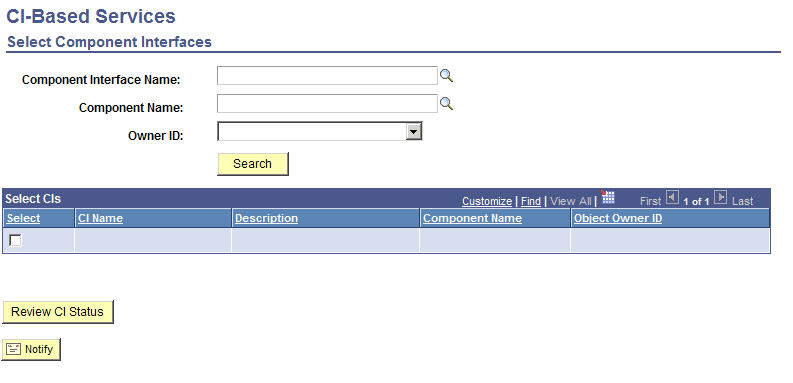
1. Scroll down to the bottom of the previous screen.



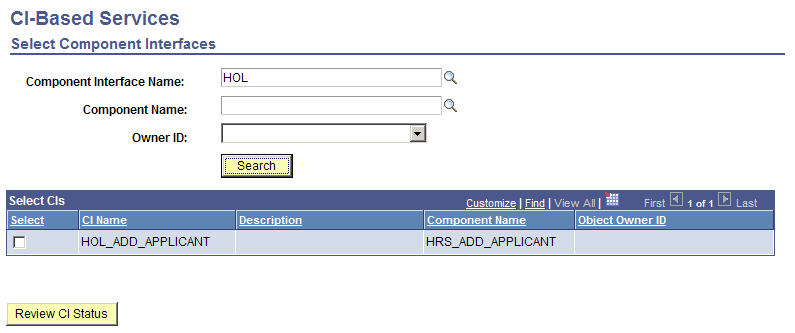
1. Click on the Save button.

### Exporting Component Interfaces as Web Services

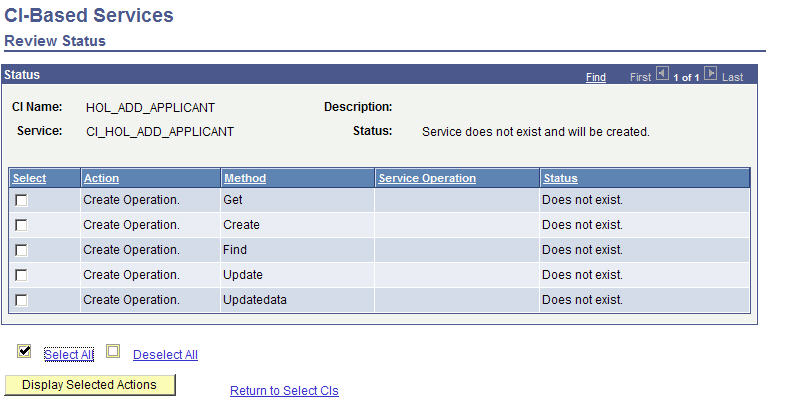
1. Open the PeopleSoft web client and login (username: PS, password: PS for the test system).
2. From the PeopleSoft web client navigate to PeopleTools | Integration Broker | Web Services | CI-Based Services.



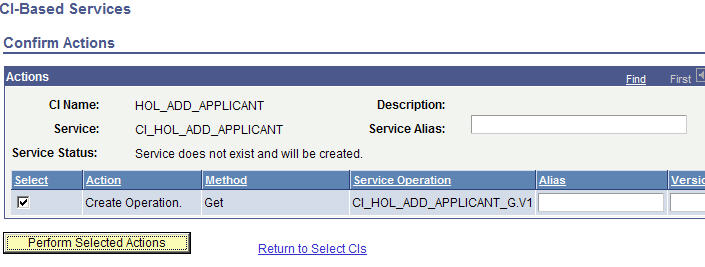
1. In the Component Interface Name textbox, enter HOL\_ADD\_APPLICANT and then click on the Search button.



1. Click on the checkbox to the left of HOL\_ADD\_APPLICANT and select Review CI Status:



1. Click on the checkbox next to the Get method and then click on Display Selected Actions



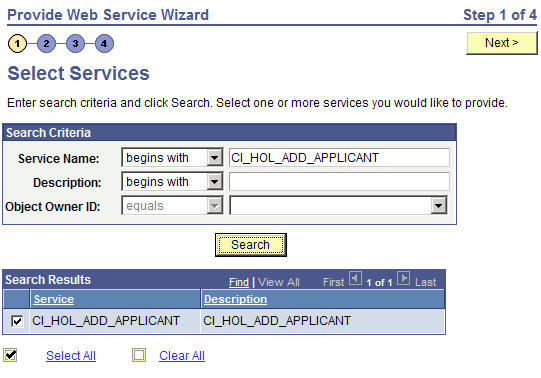
1. In the Alias checkbox enter, Get.
2. Click on Perform Selected Actions.
3. Set up the Create method
   1. Click on the checkbox next to the Create method:
   2. Click on Display Selected Actions.
   3. In the Alias checkbox enter “Create.”
   4. Click on Perform Selected Actions.
4. Set up the Find method
   1. Click on the checkbox next to the Find method:
   2. Click on Display Selected Actions.
   3. In the Alias checkbox enter “ Find.”
   4. Click on Perform Selected Actions.
5. Set up the Update method
   1. Click on the checkbox next to the Update method:
   2. Click on Display Selected Actions.
   3. In the Alias checkbox enter “Update.”
   4. Click on Perform Selected Actions.
6. Set up the UpdateData method
   1. Click on the checkbox next to the UpdateData method:
   2. Click on Display Selected Actions.
   3. In the Alias checkbox enter “UpdateData.”
   4. Click on Perform Selected Actions.

### Provide Web Service

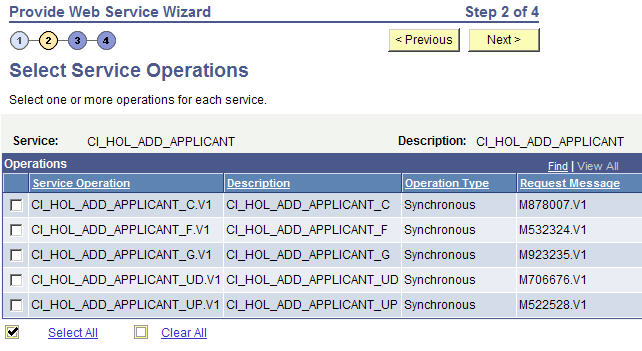
1. Navigate to: People Tools | Integration Broker | Web Services | Provide Web Service.



1. Enter CI\_HOL\_ADD\_APPLICANT in the Service Name textbox and click Search



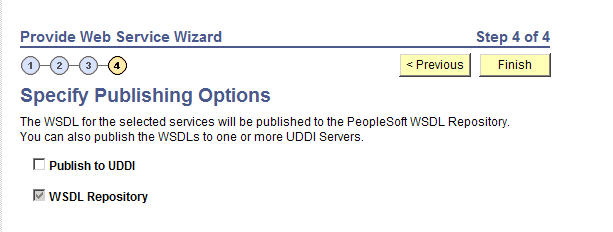
1. Make sure CI\_HOL\_ADD\_APPLICANT is checked and click Next:



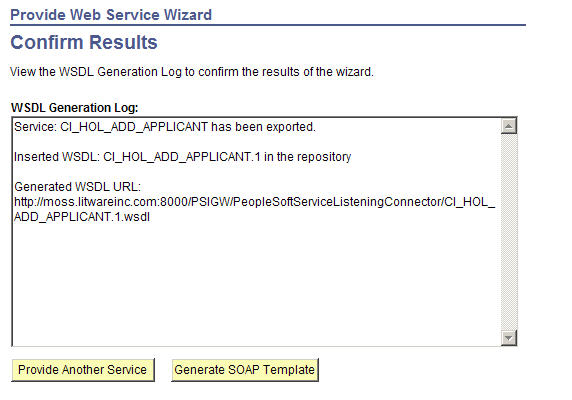
1. Click on Select All which checks all Service Operation values then click on Next.



1. Click on Next:



1. Click Finish which displays the following screen that includes the URL used to access the web service:

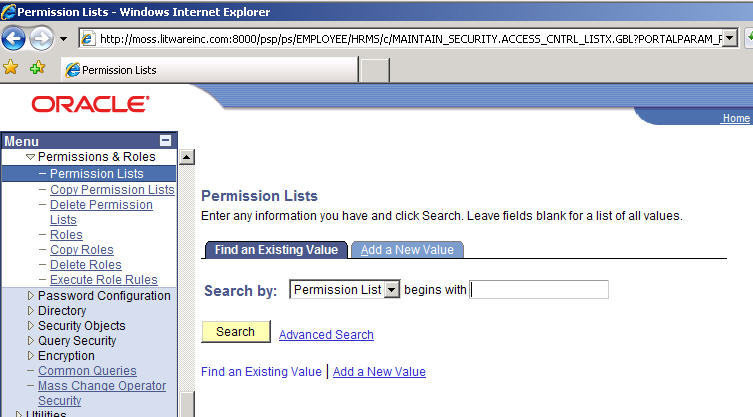


1. The URL from the previous screenshot is as follows. This will be used to create a service reference inside of a .NET development project:

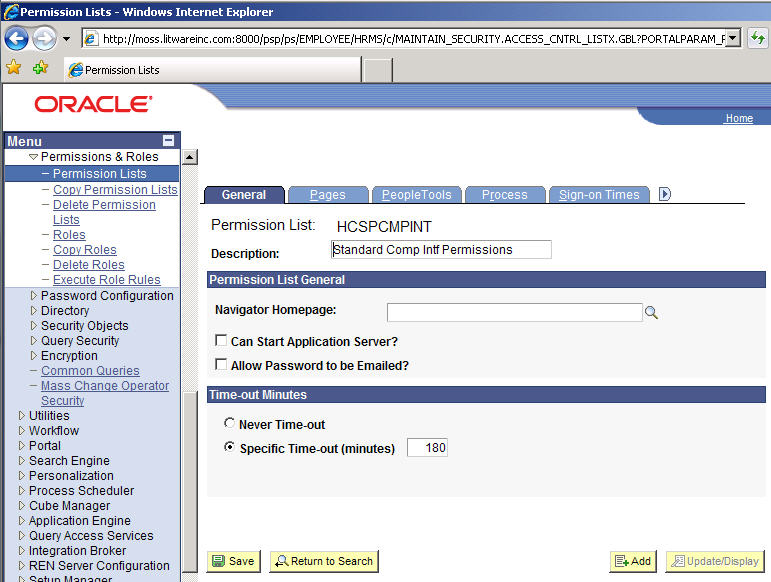
<http://moss.litwareinc.com:8000/PSIGW/PeopleSoftServiceListeningConnector/CI_HOL_ADD_APPLICANT.1.wsdl>

### Associating Security with the Web Service

1. From the PeopleSoft web client navigate to PeopleTools | Security | Permission & Roles | Permission Lists.



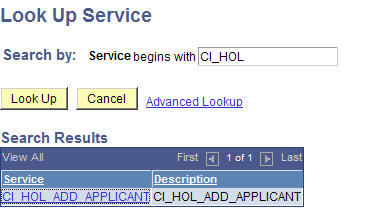
1. Enter HCSPCMPINT in the textbox next to “begins with” and click on Search:



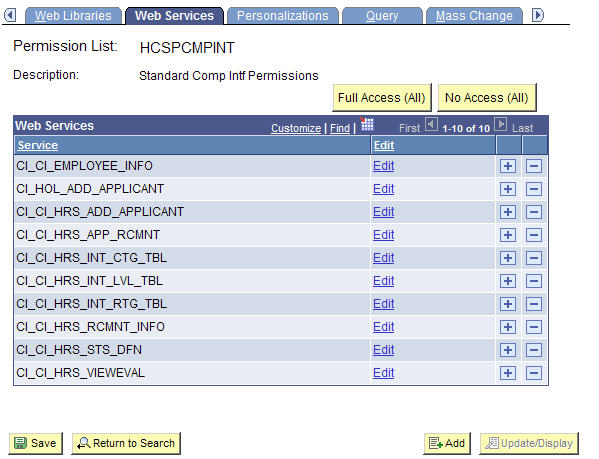
1. Click on the FI1.jpgbutton and select the Web Services tab:



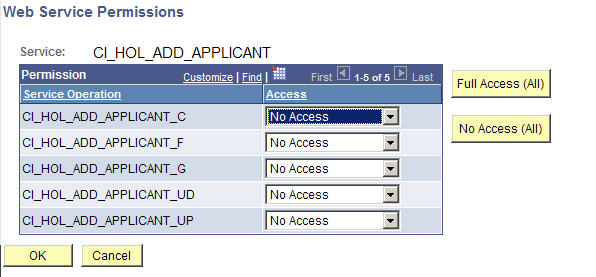
1. Enter CI\_HOL into the textbox and click on the magnify glass:



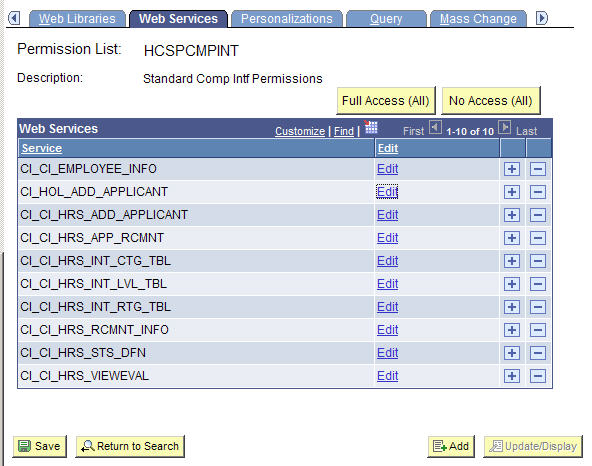
1. Click on the CI\_HOL\_ADD\_APPLICANT link which displays:



1. Click on the Edit link next to CI\_HOL\_ADD\_APPLICANT:



1. Click on the Full Access (All) button which will set Get, Create, Save, Cancel and Find to Full Access. Then click on OK which displays:



1. Click on the Save button at the bottom of the screen.
2. It is possible to view the WSDL associated with the web service by entering the following URL in Internet Explorer:

<http://moss.litwareinc.com:8000/PSIGW/PeopleSoftServiceListeningConnector/CI_HOL_ADD_APPLICANT.1.wsdl>

1. When the previous URL is entered, Internet Explorer displays the following:



Having completed this lab, you have created a web service that can access PeopleSoft. The next section will present how to consume a web service using Visual Studio .Net 2008.

## Consuming a PeopleSoft Web Service

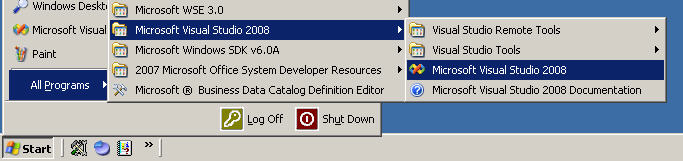
### Background

If you’ve set up your environment as per the OBA Sample Application Kit for PeopleSoft and have installed all of the source code that ships with the kit, you will have existing web services in the source code that you can use. If you have not got the environment set up, you can use the same principles that are described in this HOL using the following section as guidance. Thus, once you’ve created a web service within your PeopleSoft environment you can then consume that service from within the Visual Studio 2008 environment—and subsequently from your custom solution that integrates with PeopleSoft and Office.

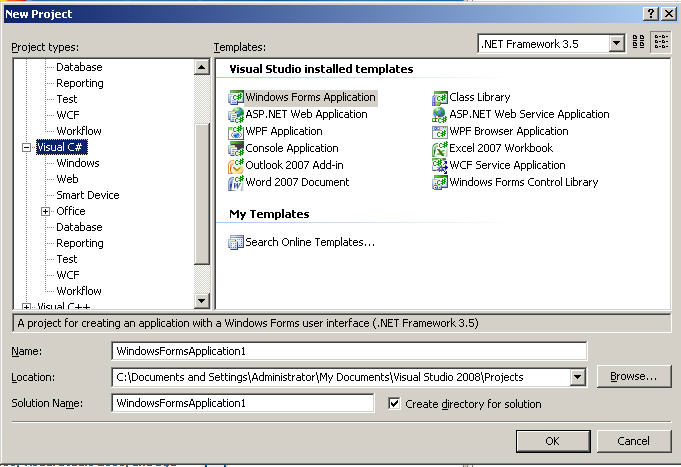
The web service used in the HOL retrieves interviewers and the work they need to perform today, but more importantly illustrates how you can consume a PeopleSoft web service within Visual Studio 2008. By providing their employee ID, information can be received such as the total number of job posting associated with the interviewer or the total number of interviews to be performed today.

### Consuming a PeopleSoft Web Service with Visual Studio 2008

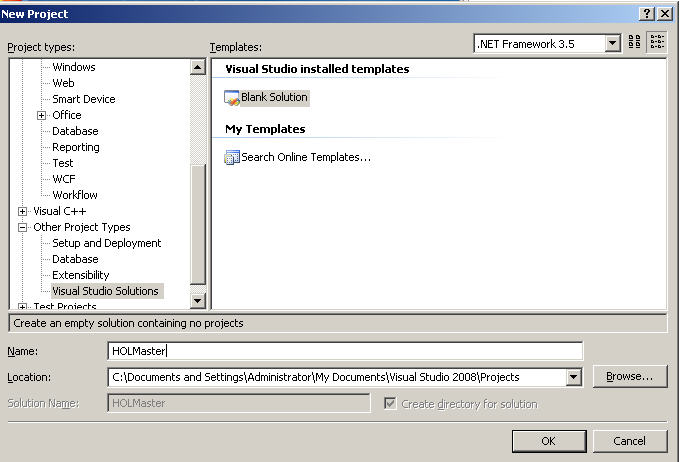
1. Launch Visual Studio 2008 (Start | All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008):



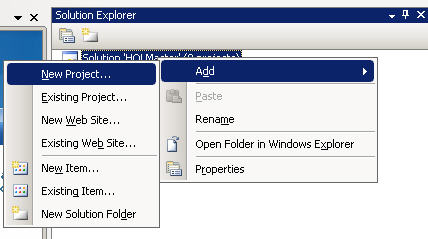
1. Within Visual Studio 2008 we will create a new solution by opening the New Project dialog (File | New Project):



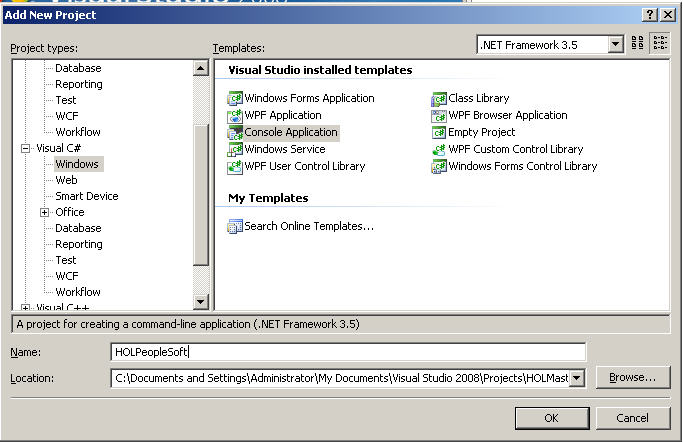
1. From inside the Project types panel select Other Project Types | Visual Studio Solutions and enter HOLMaster in the Name textbox:



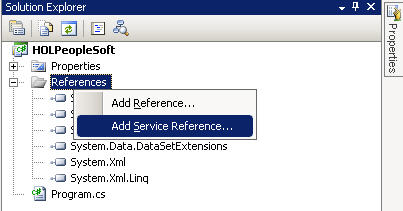
1. Click on OK thus creating a Visual Studio solution name HOLMaster.
2. Within Solution Explorer right click on the HOLMaster solution name and select Add | New Project:



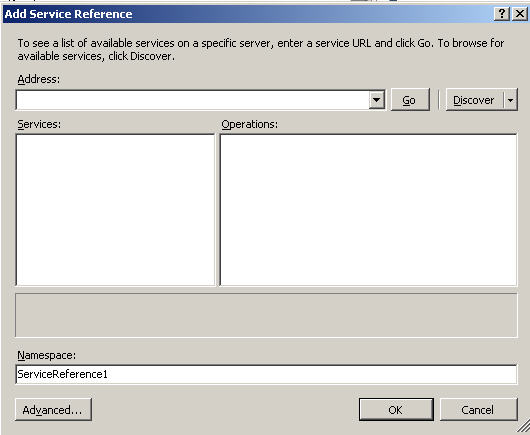
1. From within the New Project dialog select the follow and enter the name HOLPeopleSoft:
   1. Project Type: Visual C# | Windows
   2. Templates: Console Application



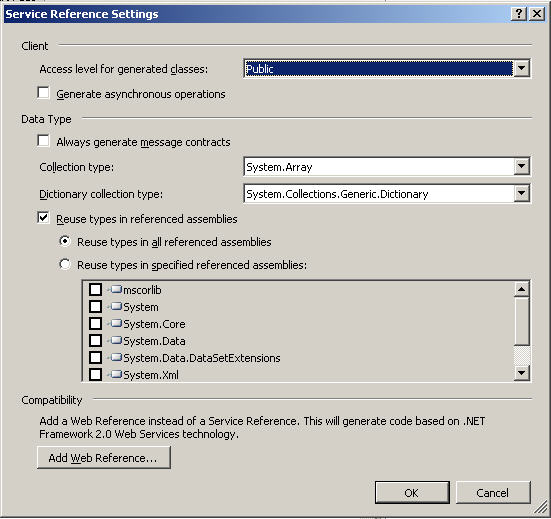
1. Click on OK thus adding the HOLPeopleSoft console application to the HOLMaster Solution.
2. This code will access a PeopleSoft web service. In order to reference the web service click on the HOLPeopleSoft project’s Add Service Reference menu item:



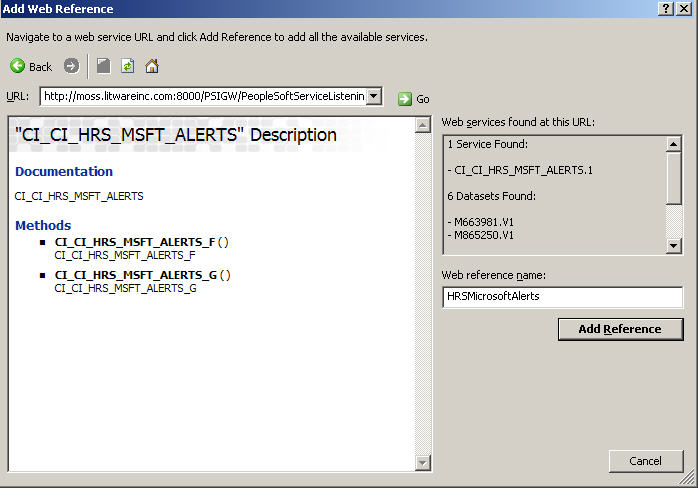
1. This display the Add Service Reference Dialog which is used to consume services using Windows Communication Foundation (WCF):



1. The web services generated by PeopleSoft do not always behave properly when accessed via WCF infrastructure. The web service from PeopleSoft will be instead consumed using the same technology as is exposed in Visual Studio 2005. This is accomplished by clicking on the Add Service Reference dialog’s Advanced button:



1. From the Service Reference Settings dialog click on Add Web Reference thus displaying the Add Web Reference dialog Visual Studio 2005 developers will find familiar:



1. In the Add Web Reference dialog
   1. Set the URL textbox to the following and click on 25.jpg which will display the methods exposed by

<http://moss.litwareinc.com:8000/PSIGW/PeopleSoftServiceListeningConnector/CI_CI_HRS_MSFT_ALERTS.1.wsdl>

* 1. Set the Web Reference name text box to HRSMicrosoftAlerts

1. Click on Add Reference to create a web reference to the HOLPeopleSoft project.
2. Open the Program.cs file for the HOLPeopleSoft application and create the following code constructions (the updated Program.cs follows):
   1. \*\* 1 \*\* : add a using statement in order to more easily access the types associated with the referenced web service:

using HOLPeopleSoft.HRSMicrosoftAlerts;

* 1. \*\* 2 \*\*: add a stub function into which more code will be added to find an interviewer to retrieve:

static string GetFirstInterviewer() // \*\* 2 \*\*

{

return String.Empty;

}

* 1. \*\* 3 \*\*: add a try/catch construct to simplify any errors encountered.

try

{

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

The updated Program.cs in its entirety is as follows:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using HOLPeopleSoft.HRSMicrosoftAlerts; // \*\* 1 \*\*

namespace HOLPeopleSoft

{

class Program

{

static string GetFirstInterviewer() // \*\* 2 \*\*

{

return String.Empty;

}

static void Main(string[] args)

{

try // \*\* 3 \*\*\*

{

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

1. Within the try/catch protected region add the following code in order to access the underlying PeopleSoft web service (the updated Program.cs follows).
   1. Create an instance of the PeopleSoft alert service

CI\_CI\_HRS\_MSFT\_ALERTS alerts = new CI\_CI\_HRS\_MSFT\_ALERTS();

* 1. Create the variable used by the Get operator to return results:

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSResponseTypeShape results;

* 1. Create the variable used a parameter to the Get method. This variable represents the applicant for whom information is being requested and is created and seeded with data as follows:

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape request =

new Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape();

request.OPRID = new OPRIDTypeShape2();

request.OPRID.Value = GetFirstInterviewer();

* 1. Invoke the Get method of the alerts web service:

results = alerts.CI\_CI\_HRS\_MSFT\_ALERTS\_G(request);

* 1. Display the results return by the alerts web services Get method being invoked for Cindy Lee:

Console.WriteLine("AUTOMATCH: {0} ",

results.AUTO\_MATCH.Value);

Console.WriteLine("EMPLID: {0} ",

results.EMPLID.Value);

Console.WriteLine("INTERVIEW\_EVALS: {0} ",

results.INTERVIEW\_EVALS.Value);

Console.WriteLine("INTERVIEWS\_TODAY: {0} ",

results.INTERVIEWS\_TODAY.Value);

Console.WriteLine("JOB\_POSTINGS: {0} ",

results.JOB\_POSTINGS.Value);

Console.WriteLine("JOBOPENING\_APPROVAL: {0} ",

results.JOBOPENING\_APPROVAL.Value);

Console.WriteLine("NEW\_APPLICANTS: {0} ",

results.NEW\_APPLICANTS.Value);

Console.WriteLine("OFFERS\_FOR\_APPROVAL: {0} ",

results.OFFERS\_FOR\_APPROVAL.Value);

Console.WriteLine("PREP\_FOR\_HIRE: {0} ",

results.PREP\_FOR\_HIRE.Value);

Console.WriteLine("PREPARE\_OFFERS: {0} ",

results.PREPARE\_OFFERS.Value);

The code in its entirety is as follows:

try

{

CI\_CI\_HRS\_MSFT\_ALERTS alerts = new CI\_CI\_HRS\_MSFT\_ALERTS();

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSResponseTypeShape results;

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape request =

new Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape();

request.OPRID = new OPRIDTypeShape2();

request.OPRID.Value = GetFirstInterviewer();

results = alerts.CI\_CI\_HRS\_MSFT\_ALERTS\_G(request);

Console.WriteLine("AUTOMATCH: {0} ",

results.AUTO\_MATCH.Value);

Console.WriteLine("EMPLID: {0} ",

results.EMPLID.Value);

Console.WriteLine("INTERVIEW\_EVALS: {0} ",

results.INTERVIEW\_EVALS.Value);

Console.WriteLine("INTERVIEWS\_TODAY: {0} ",

results.INTERVIEWS\_TODAY.Value);

Console.WriteLine("JOB\_POSTINGS: {0} ",

results.JOB\_POSTINGS.Value);

Console.WriteLine("JOBOPENING\_APPROVAL: {0} ",

results.JOBOPENING\_APPROVAL.Value);

Console.WriteLine("NEW\_APPLICANTS: {0} ",

results.NEW\_APPLICANTS.Value);

Console.WriteLine("OFFERS\_FOR\_APPROVAL: {0} ",

results.OFFERS\_FOR\_APPROVAL.Value);

Console.WriteLine("PREP\_FOR\_HIRE: {0} ",

results.PREP\_FOR\_HIRE.Value);

Console.WriteLine("PREPARE\_OFFERS: {0} ",

results.PREPARE\_OFFERS.Value);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

1. The code to substitute for the GetFirstInterviewer method should also be pasted into the application:

static string GetFirstInterviewer() // \*\* 2 \*\*

{

CI\_CI\_HRS\_MSFT\_ALERTS alerts = new CI\_CI\_HRS\_MSFT\_ALERTS();

CI\_HRS\_MSFT\_ALERTSTypeShape[] response;

Find\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape request =

new Find\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape();

request.OPRID = new OPRIDTypeShape();

//Limit the Operators in US only

request.OPRID.Value = "HCRUSA" + "%";

response = alerts.CI\_CI\_HRS\_MSFT\_ALERTS\_F(request);

if ((response == null) || (response.Length == 0))

{

throw new Exception("No U.S. interviewers found.");

}

return response[0].OPRID.Value;

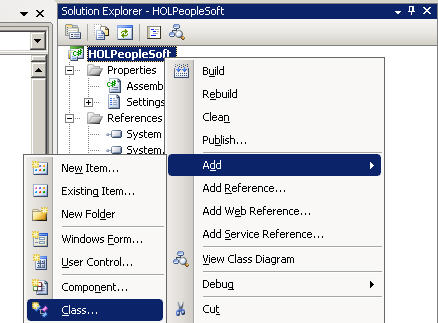
}

1. The GetFirstInterviewer method will not be covered in detail but again it is a request input parameter to a PeopleSoft method associated with the alerts variable. The response is called, response and it returns a list of ID’s associated with interviewers.
2. Run the code and demonstrate it can access the Alerts web service exposed by PeopleSoft.

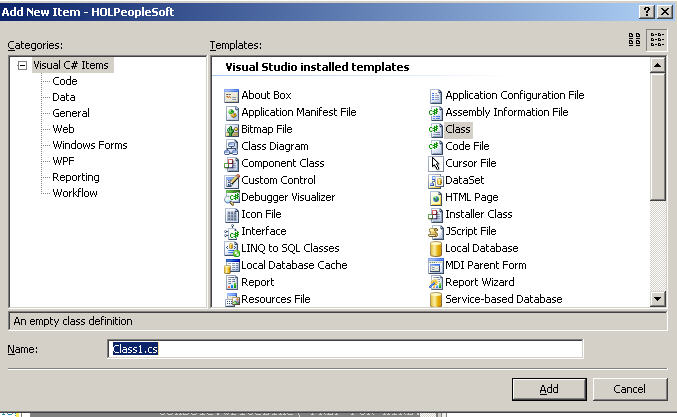
### Encapsulate the web service in a class

Our ultimate goal is to expose PeopleSoft web service to Microsoft Office. In support of this a PSFTAlerts class will be created:

1. From within Solution Explorer right click on HOLPeopleSoft and select Add | Class:



1. This display the Add New item dialog:



1. Enter the class name of PSFTAlerts and click on OK which will create the PSFTAlerts.cs source file which is as follows:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace HOLPeopleSoft

{

class PSFTAlerts

{

}

}

1. Replace the contents of the PSFTAlerts.cs with the following code adapted from Program.cs:

using System;

using HOLPeopleSoft.HRSMicrosoftAlerts;

namespace HOLPeopleSoft

{

public class PSFTAlerts

{

private string \_autoMatch;

private string \_employeeID;

private string \_interviewEvals;

private string \_interviewsToday;

private string \_jobPostings;

private string \_jobOpeningApproval;

private string \_newApplicants;

private string \_offersForApproval;

private string \_prepForHire;

private string \_prepareOffers;

public PSFTAlerts(string employeeID)

{

CI\_CI\_HRS\_MSFT\_ALERTS alerts =

new CI\_CI\_HRS\_MSFT\_ALERTS();

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSResponseTypeShape

results;

Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape request =

new Get\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape();

request.OPRID = new OPRIDTypeShape2();

request.OPRID.Value = employeeID;

results = alerts.CI\_CI\_HRS\_MSFT\_ALERTS\_G(request);

\_autoMatch = results.AUTO\_MATCH.Value;

\_employeeID = results.EMPLID.Value;

\_interviewEvals = results.INTERVIEW\_EVALS.Value;

\_interviewsToday = results.INTERVIEWS\_TODAY.Value;

\_jobPostings = results.JOB\_POSTINGS.Value;

\_jobOpeningApproval = results.JOBOPENING\_APPROVAL.Value;

\_newApplicants = results.NEW\_APPLICANTS.Value;

\_offersForApproval = results.OFFERS\_FOR\_APPROVAL.Value;

\_prepForHire = results.PREP\_FOR\_HIRE.Value;

\_prepareOffers = results.PREPARE\_OFFERS.Value;

}

public string AutoMatch

{

get { return \_autoMatch; }

}

public string EmployeeID

{

get { return \_employeeID; }

}

public string InterviewEvals

{

get { return \_interviewEvals; }

}

public string InterviewsToday

{

get { return \_interviewsToday; }

}

public string JobPostings

{

get { return \_jobPostings; }

}

public string JobOpeningApproval

{

get { return \_jobOpeningApproval; }

}

public string NewApplicants

{

get { return \_newApplicants; }

}

public string OffersForApproval

{

get { return \_offersForApproval; }

}

public string PrepForHire

{

get { return \_prepForHire; }

}

public string PrepareOffers

{

get { return \_prepareOffers; }

}

public static string[] GetInterviewerIDs()

{

CI\_CI\_HRS\_MSFT\_ALERTS alerts =

new CI\_CI\_HRS\_MSFT\_ALERTS();

CI\_HRS\_MSFT\_ALERTSTypeShape[] response;

Find\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape request =

new Find\_\_CompIntfc\_\_CI\_HRS\_MSFT\_ALERTSTypeShape();

string[] interviewerIDs;

request.OPRID = new OPRIDTypeShape();

//Limit the Operators in US only

request.OPRID.Value = "HCRUSA" + "%";

response = alerts.CI\_CI\_HRS\_MSFT\_ALERTS\_F(request);

if (response == null)

{

throw new Exception("No U.S. interviewers found.");

}

interviewerIDs = new string[response.Length];

for (int index = 0;

index < interviewerIDs.Length;

index++)

{

interviewerIDs[index] = response[index].OPRID.Value;

}

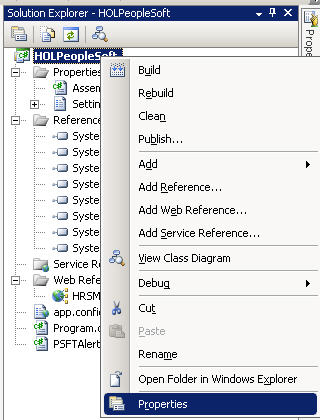
return interviewerIDs;

}

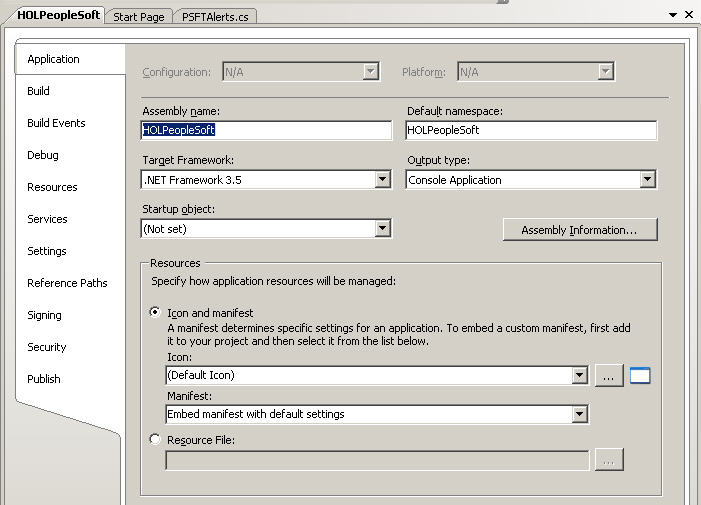
}

}

1. The GetInterviewerIDs method in the previous snippet is just another example of invoking a PeopleSoft web service. In this case it returns all the interviews whose current interviewing statistics can be retrieved (local to the U.S. only).
2. Our goal is to allow Office application to consume the PSFTAlerts class. This could be simpler if the HOLPeopleSoft application were a DLL rather than a console application (an executable). This is achieved by first right clicking on the HOLPeopleSoft project in Solution Explorer and selecting properties:



1. The properties window for the project is as follows:



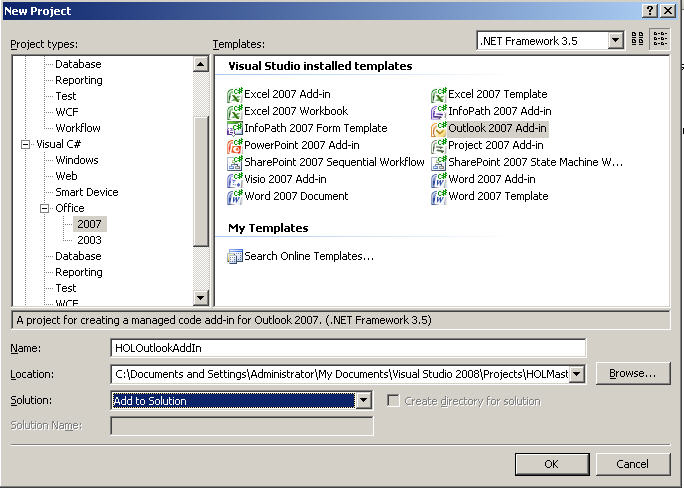
1. Change the output type from Console Application (an EXE) to Class Library (a DLL) and click on the X in the upper right corner to close and save the properties.

From this point forward the HOLPeopleSoft assembly will not run as a console application as it is a new a DLL. To convert back to a console application, open the project properties and simply change the output type back to Console Application

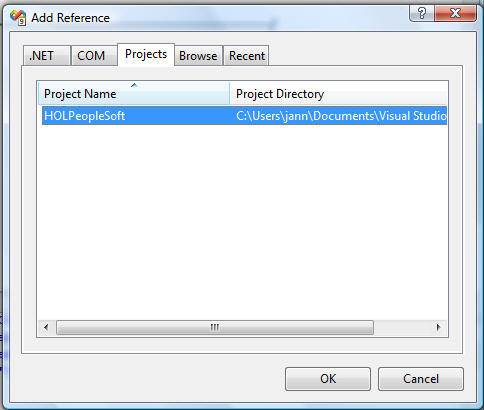
## Creating an Outlook Add-in

### Creating an Outlook 2007 Task Pane using XAML

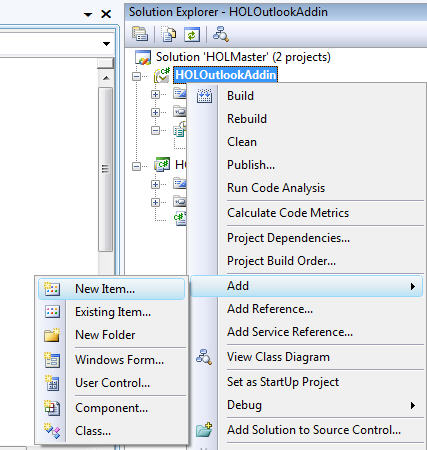
1. With the HOLMaster solution still open create an Outlook project by first select File | New Project and navigating Projects types | Visual C# | Office | 2007 and high lighting an Outlook 2007 Add-in project:



1. On the New Project dialog fill in the controls as follows (make sure you select “Add to Solution” so that the project created is added to HOLMaster):
   1. Name: HOLOutlookAddIn
   2. Solution: Add to Solution
2. Click on OK to create the Outlook 2007 Add-in project to add this project to the HOLMaster solution.
3. Our Outlook Add-in will be access the HOLPeopleSoft project so in Solution Explorer for the HOLOutlookAddIn project right click on references and click on Add Reference.
4. From the Add Reference dialog select the Projects tab:



1. Select the HOLPeopleSoft project and click on OK thus allow the HOLOutlookAddin project to reference the HOLPeopleSoft project.
2. Back to the primary task. An Outlook task pane contains a control so before creating the task pane, a control is needed, in this case a WPF/XAML control. This control is created by clicking on the the HOLOutlookAddIn project in Solution Explorer
3. In order to create an Outlook 2007 task pane right click on the HOLOutlookAddIn project in Solution Explorer and selecting Add | New Item:



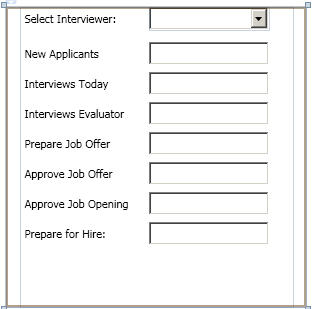
1. This displays the Add New Item dialog on which you should navigate to Categories WPF and Templates User Control (WPF):



1. On the New Item dialog enter the name HOLXAMLDashboard and click on Add. The control added to the HOLOutlookAddin project will appears as follows:



1. Visual Studio 2008 ships with an XAML designer, which enables the developer to simply drag and drop objects onto the designer from the Toolbox (View | Toolbox). Along with the visual designer, the XAML code can also be edited for those developers who want a hands-on experience on what XAML is really like – for those developers who like to code raw XML by hand as the XAML language is an XML-based language. XAML describes the visual user interface. Use the visual designer in conjunction with the Toolbox to layout the following controls on the XMLDashboard user control:



1. The controls laid out are as follows:
   1. Label containing text Select Interviewer
   2. Under the Select interviewer a ComboBox named ComboBoxInterviewers.
   3. The remainder of the controls are label/textbox pairs (the LabelNewApplicants label should reside to the left of the TextBoxNewApplicants control).
2. Once the controls are laid out double click on the combobox in order to generated a method for handling the selection changed event. The code in HOLXAMLDashboard.xaml.cs will look as follows:

namespace HOLOutlookAddIn

{

public partial class HOLXAMLDashboard : UserControl

{

public HOLXAMLDashboard()

{

InitializeComponent();

}

private void comboBoxInterviewer\_SelectionChanged(

object sender,

SelectionChangedEventArgs e)

{

}

}

}

1. In order to populate the combobox and handle the selection changed event the previous code will have to be augmented as follows (the code in boldface should be added to the CS file):

**using HOLPeopleSoft;**

namespace HOLOutlookAddIn

{

/// <summary>

/// Interaction logic for HOLXAMLDashboard.xaml

/// </summary>

public partial class HOLXAMLDashboard : UserControl

{

public HOLXAMLDashboard()

{

InitializeComponent();

**comboBoxInterviewer.ItemsSource =**

**PSFTAlerts.GetInterviewerIDs();**

}

private void comboBoxInterviewer\_SelectionChanged(

object sender,

SelectionChangedEventArgs e)

{

**PSFTAlerts alerts;**

**if (comboBoxInterviewer.SelectedValue == null)**

**{**

**return;**

**}**

**alerts = new PSFTAlerts(**

**comboBoxInterviewer.SelectedValue.ToString());**

**textBoxApproveJobOffer.Text = alerts.OffersForApproval;**

**textBoxApproveJobOpening.Text = alerts.JobOpeningApproval;**

**textBoxInterviewEvaluator.Text = alerts.InterviewEvals;**

**textBoxInterviewsToday.Text = alerts.InterviewsToday;**

**textBoxNewApplicants.Text = alerts.NewApplicants;**

**textBoxPrepareForHire.Text = alerts.PrepForHire;**

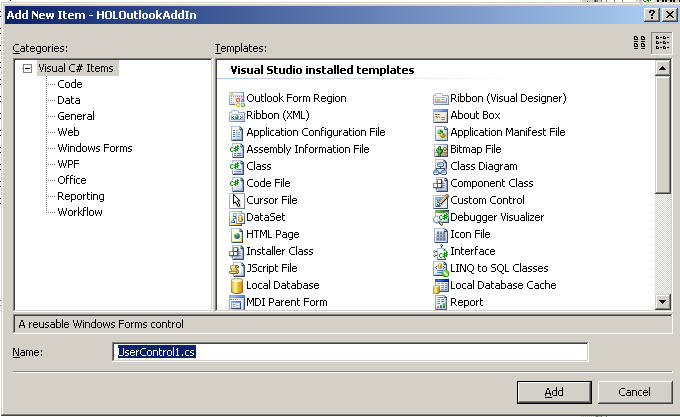
**textBoxPrepareJobOffer.Text = alerts.PrepareOffers;**

}

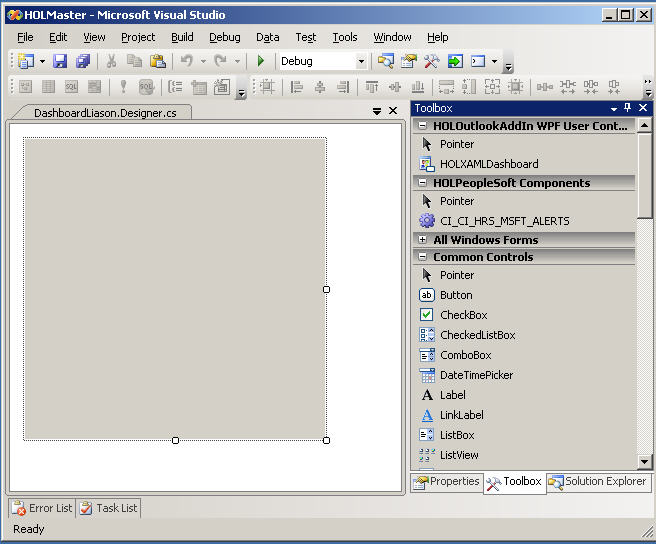
}

}

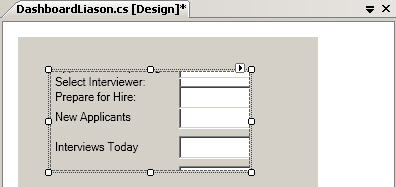
1. It is likely that the names of the combox and textboxes may differ in the ones you laid out by hand. Make the combobox name and the textbox names match the appropriate portion of the code.
2. In order for the XAML user control to be displayed in a task pane, the XAML user must be hosted in a user control using an ElementHost control. This is handled by Visual Studio when a WPF user control is dragged onto Windows Forms user control. To add this Windows Forms user control right click on the HOLOutlookAddIn project within Solution Explorer and select Add | UserControl. This displays the Add New Item dialog:



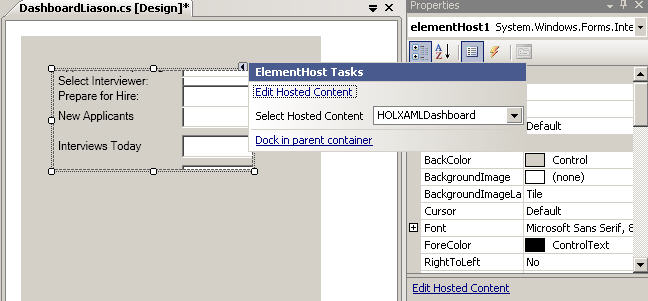
1. Change the name of the control to DashboardLiason.cs and click on Add.
2. Select View | Properties and use the Properties window to set the DashboardLiason control’s size to 300,300.
3. Display the Toolbox (View | Toolbox):



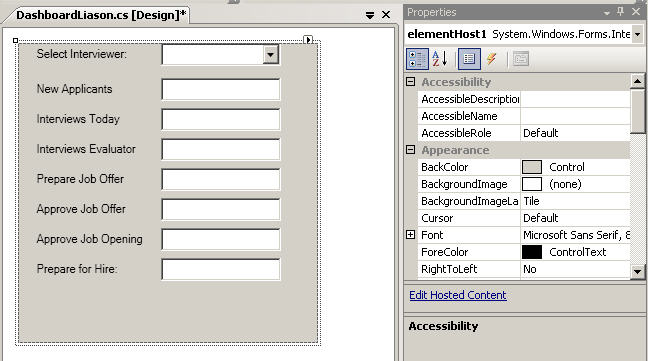
1. Notice that the HOLXAMLDashboard control resides on the Toolbox. Drag HOLXAMLDashboard from the Toolbox and drop it onto the design surface of the DashboardLiason Windows Forms user control.
2. When the XAML control is dropped on DashboardLiason left click on the XAML control and notice the small arrow in the upper right corner of the instance of the HOLXAMLDashboard control:



1. Click on the error displaying the ElementHost Tasks tasks menu:



1. From the ElementHost Tasks menu select Dock in parent container. This expands the XAML control to fill the area associated with the Windows Forms user control (see the following):

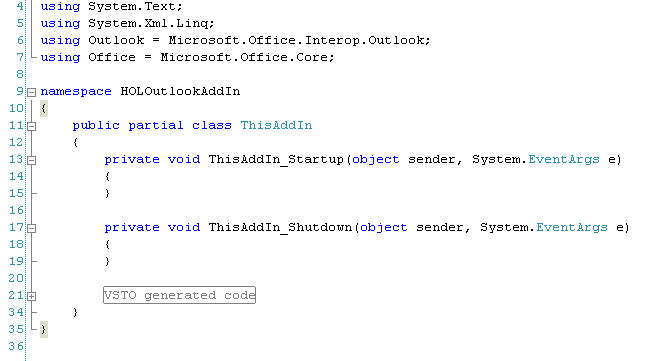


### Hooking an Outlook 2007 Task Pane with a Menu

When a developer creates an add-in for any Office 2007 application that supports VSTO, the add-in’s functionality must be triggered. For Excel 2007, PowerPoint and Word 2007 the common mechanism for incorporating add-in functionality is by extending the ribbon. Outlook 2007 does not display a ribbon control in its main interface hence the ribbon cannot be used to trigger the functionality associated with an Outlook 2007 add-in. To expose Outlook 2007 functionality, developers can create new command bars or new elements in the Outlooks menu. In this application, the latter was used.

The steps to accomplish this task are as follows

1. First consider the code that Visual Studio creates when an new add is created (see the following):



1. Visual Studio 2008 does not come with a visual designer for the menu. In order for a developer to introduce a new element in the Outlook 2007 menu, the new element is created in programmatically. To facilitate this add the code in bold to the ThisAddIn.cs source file:

**// Defines CustomTaskPane**

**using Microsoft.Office.Tools;**

namespace HOLOutlookAddIn

{

public partial class ThisAddIn

{

**private Office.CommandBar \_menuBar;**

**private Office.CommandBarPopup \_menuBarPopup;**

**private Office.CommandBarButton \_menuBarButton;**

**private string \_menuBarTag = "HOLTag";**

**private void RemoveMenuBar()**

**{**

**}**

**private void AddMenuBar()**

**{**

**}**

**private void menuBarButton\_Click(**

**Office.CommandBarButton Ctrl,**

**ref bool CancelDefault)**

**{**

**}**

private void ThisAddIn\_Startup(

object sender, System.EventArgs e)

{

}

private void ThisAddIn\_Shutdown(

object sender, System.EventArgs e)

{

}

1. The code added is defined as follows:
   1. Insurces the CustomTaskPane time can be instantiated to demonstrate our XAML control running in a custom task pane:

// Defines CustomTaskPane

using Microsoft.Office.Tools;

* 1. Fields used to define the programming elements of the custom menu:

private Office.CommandBar \_menuBar;

private Office.CommandBarPopup \_menuBarPopup;

private Office.CommandBarButton \_menuBarButton;

* 1. Name associated with the custom menu bar:

private const string \_menuBarTag = "HOLTag";

* 1. Method that will hold the code remove the custom menu bar:

private void RemoveMenuBar()

{

}

* 1. Method that will hold the code add the custom menu bar:

private void AddMenuBar()

{

}

* 1. Method that will handle when the menu bar’s menu item is clicked:

private void menuBarButton\_Click(

Office.CommandBarButton Ctrl,

ref bool CancelDefault)

{

}

1. In order to add new elements in the Outlook 2007 menu bar, we implement the RemoveMenuBar method so that this method checks if the new menu item already exists and deletes it if it does:

private void RemoveMenuBar()

{

Office.CommandBarPopup foundMenu = (Office.CommandBarPopup)

Application.ActiveExplorer().CommandBars.ActiveMenuBar.FindControl(

Office.MsoControlType.msoControlPopup,

missing,

menuBarTag,

true,

true);

if (foundMenu != null)

{

foundMenu.Delete(true);

}

}

1. The AddMenuBar method is implemented as following creating a menuBarPopup and associating with it a menuBarButton contain the caption “Interviewer Task Pane”:

private void AddMenuBar()

{

\_menuBar = Application.ActiveExplorer().CommandBars.ActiveMenuBar;

\_menuBarPopup = (Office.CommandBarPopup)\_menuBar.Controls.Add(

Office.MsoControlType.msoControlPopup,

missing,

missing,

missing,

false);

if (\_menuBarPopup != null)

{

\_menuBarPopup.Caption = "&HOL Add-In";

\_menuBarPopup.Tag = \_menuBarTag;

\_menuBarButton =

(Office.CommandBarButton)\_menuBarPopup.Controls.Add(

Office.MsoControlType.msoControlButton,

missing,

missing,

missing,

false);

\_menuBarButton.Style =

Office.MsoButtonStyle.msoButtonIconAndCaption;

\_menuBarButton.Caption = "Interviewer Task Pane";

\_menuBarButton.Click +=

new Office.\_CommandBarButtonEvents\_ClickEventHandler(

menuBarButton\_Click);

}

}

1. The add-in’s start up method, ThisAddIn\_Startup, is augmented to include the removal and the adding of the custom menu:

private void ThisAddIn\_Startup(object sender, System.EventArgs e)

{

RemoveMenuBar();

AddMenuBar();

}

1. When the menu bar button is clicked the the menuBarButton\_Click method is invoked and the custom task pane is displayed:

private void menuBarButton\_Click(

Office.CommandBarButton Ctrl,

ref bool CancelDefault)

{

CustomTaskPane customTaskPane =

Globals.ThisAddIn.CustomTaskPanes.Add(

new DashboardLiason(),

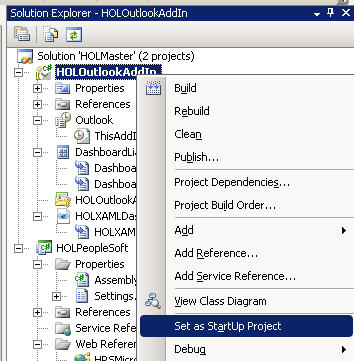
"Recruitment DashBoard");

customTaskPane.Width = 220;

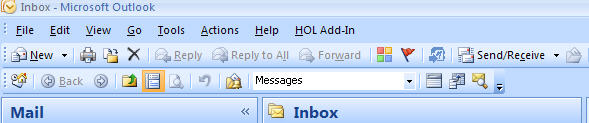
customTaskPane.Visible = true;

}

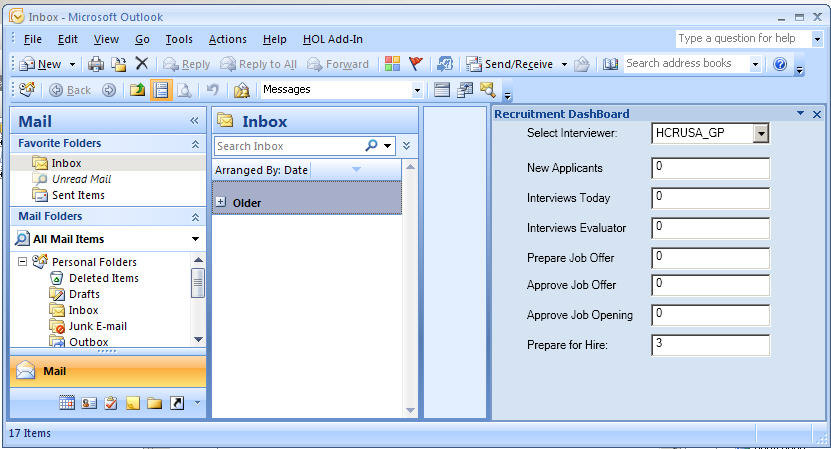
1. The time has come to show off our work. Right click on the HOLOutlookAddIn project and select Set as StartUp Project:



1. Select Debug | Start Debugging which will launch Outlook:



1. Notice the custom menu has been included. The menu item with this menu can be invoked thus displaying our custom task pane:



1. In the previous screenshot an interview has been selected and their relevant information displayed.

The previous code is simpler than that contain the OBA Sample Application Kit for PeopleSoft. If you during the lab you are ahead of your lab-mates consider augmented the code as follows. In the lab the code to retrieve the interviewers from the PeopleSoft web service is as follows:

public partial class HOLXAMLDashboard : UserControl

{

public HOLXAMLDashboard()

{

InitializeComponent();

comboBoxInterviewer.ItemsSource =

PSFTAlerts.GetInterviewerIDs();

}

If the time to access the PeopleSoft web service is too long, Outlook will become unresponsive. The OBA Sample Application Kit for PeopleSoft hid this unresponsiveness by creating a PseudoProgessForm dialog (see the following section for a more complete write up if help is required, “Appendix B: Progress Display Bar”):



This dialog was displayed while the OBA Sample Application Kit for PeopleSoft used a background worker thread to retrieve the interviewers (the BackgroundWorker implementation is highlighted in boldface):

public XAMLDashBoard()

{

InitializeComponent();

FillInterviewers();

comboBoxInterviewers.DropDownClosed +=

new EventHandler(comboBoxInterviewers\_DropDownClosed);

}

private void FillInterviewers()

{

comboBoxInterviewers.ItemsSource = GetInterviewers();

comboBoxInterviewers.DisplayMemberPath = "Name";

comboBoxInterviewers.SelectedValuePath = "EmployeeID";

}

private List<Interviewer> GetInterviewers()

{

bool done = false;

List<Interviewer> interviewers = new List<Interviewer>();

PseudoProgressForm progress = new PseudoProgressForm();

progress.ProgressLabel = "Querying PeopleSoft...";

**BackgroundWorker backgroundWorker = new BackgroundWorker();**

**backgroundWorker.DoWork +=**

**delegate(object obj, DoWorkEventArgs dwev)**

**{**

**interviewers = \_alertService.GetInterviewers();**

**};**

**backgroundWorker.RunWorkerCompleted +=**

**delegate(object obj, RunWorkerCompletedEventArgs rwcev)**

**{**

**progress.Close();**

**};**

**backgroundWorker.RunWorkerAsync();**

progress.ShowDialog();

return interviewers;

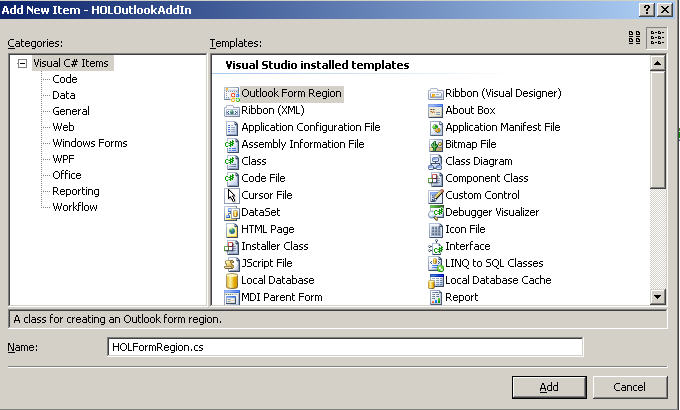
}

A similar scheme could be adopted for this lab. Also note that the OBA Sample Application Kit for PeopleSoft returned more than just a List<string> containing ID’s with respect to interviewers. The OBA Sample Application Kit for PeopleSoft returned a structure containing more detailed information. This information had to retrieved one interviewer at a time, hence the slower speed and the need for the BackgroundWorker thread.

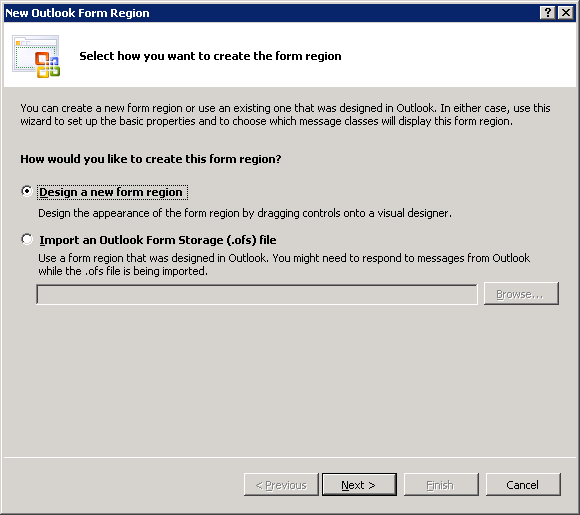
### Creating a Form Region in Outlook 2007

The Evaluation Form in the OBA Sample Application Kit for PeopleSoft was implemented using a customized form region that replaces the standard Outlook message template. Inherited from the *IPM.Note* Outlook message class, the custom form region is designed in the Visual Studio 2008 Designer IDE much like how an ordinary Windows Form is designed. The purpose behind this is simple. Place an editable form in an email message. This form can contain standard controls that are populated by reading data sources such as PeopleSoft and/or SQL Server. When the email message is sent, the form region can programmatically save the information in the email message to data stores such as PeopleSoft and/or SQL Server.

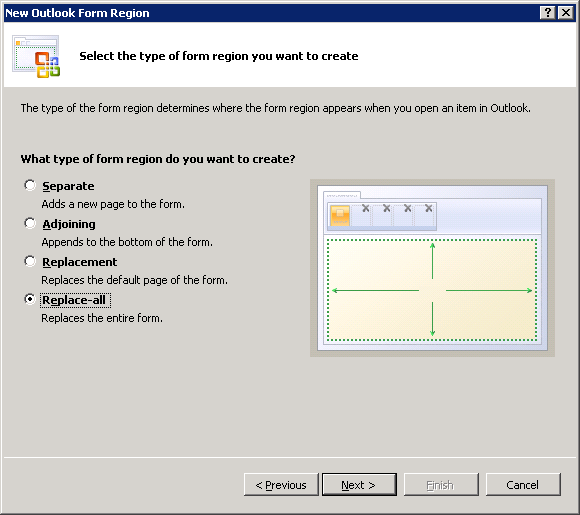
1. To create the custom form region, right-click the add-in project in Solution Explorer, and then choose Add | New Item. From the list of templates that appear, select Outlook Form Region, specify a name for it, HOLFormRegion, and then click Add.



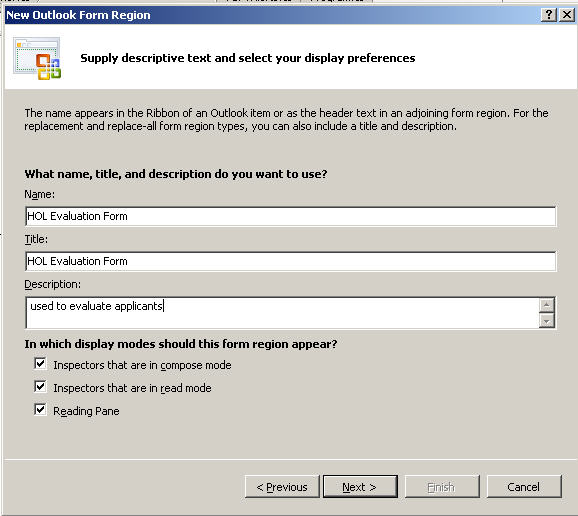
1. Upon adding, the New Outlook Form Region wizard appears.



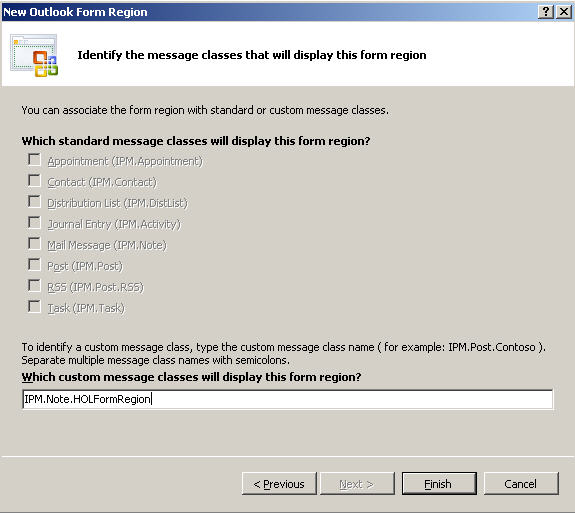
1. Since the form region will be created in a visual designer, leave the default option as is, and then click Next. The following screen will present you with four options on how the form region will be laid out in Outlook. Each option is described as shown in the figure below:



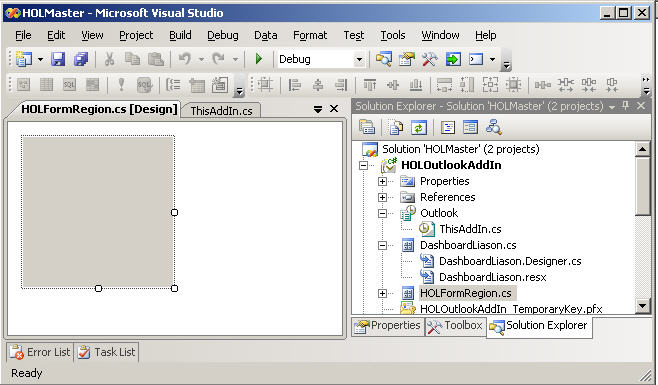
1. The Evaluation Form for this sample application kit will take the structure of a standard Outlook message, and derive from it. In order to do this, select the fourth option, Replace-All.
2. Clicking Next will proceed to the following step where you have the option to indicate a name, a title and a description for the form region, as well as define its display modes.



1. Lastly, the final step allows you to specify the custom class that will display the form region. Because the Evaluation Form is inheriting from the Mail Message class, type “*IPM.Note.”* followed by the class name (HOLFormRegion) of the form region, as seen in the screenshot below:



1. Click Finish to end the wizard. At this point, controls can now be laid out on the designer to come up with this Evaluation Form:



Before continue the lab let’s examine the for region implemented by the OBA Sample Application Kit for PeopleSoft. This class, FormRegionEvaluation.cs, within the sample application

* Contains over 700 lines of code
* Uploads files such as a work samples (PDF’s for a designer, photos for actors/models or code snippets for a developer) for a candidate applying for a position.
* Reads eight separate PeopleSoft web services to populate controls on the screen.
* Write to a PeopleSoft web service
* Write to a SQL Server data store (the files uploaded representing the applicant’s samples).

Within the context of a sample application, the FormRegionEvaluation class is a fantastic development resource. Within the context of a hand’s on lab the FormRegionEvaluation can serve as a model for a simpler example, a hybrid of the current lab exercises so far and the original sample application.

1. Close the current solution.
2. Open the HOLMasterPlus solution found under c:\HOL\HOLMasterPlus.
3. The HOLMasterPlus solutons’s HOLOutlookAddIn project contains support for creating a interviewee review using a form region via an extend custom menu in Outlook. In order to see this support open HOLOutlookAddIn’s ThisAddIn.cs file can consider (note: this code does not need to be added to ThisAddIn.cs as it is already part of the solution):
   1. The CommandBarButton \_menuBarButtonCreateEvaluation which represents the menu item that triggers the evaluation creation:

namespace HOLOutlookAddIn

{

public partial class ThisAddIn

{

**private Office.CommandBarButton**

**\_menuBarButtonCreateEvaluation;**

private Office.CommandBar \_menuBar;

private Office.CommandBarPopup \_menuBarPopup;

private Office.CommandBarButton \_menuBarButton;

private const string \_menuBarTag = "HOLTag";

* 1. The AddMenuBar method has been extended to support this menu:

private void AddMenuBar()

{

\_menuBar = Application.ActiveExplorer().CommandBars.ActiveMenuBar;

\_menuBarPopup = (Office.CommandBarPopup)\_menuBar.Controls.Add(

Office.MsoControlType.msoControlPopup,

missing,

missing,

missing,

false);

if (\_menuBarPopup != null)

{

\_menuBarPopup.Caption = "&HOL Add-In";

\_menuBarPopup.Tag = \_menuBarTag;

\_menuBarButton =

(Office.CommandBarButton)\_menuBarPopup.Controls.Add(

Office.MsoControlType.msoControlButton,

missing,

missing,

missing,

false);

\_menuBarButton.Style =

Office.MsoButtonStyle.msoButtonIconAndCaption;

\_menuBarButton.Caption = "Interviewer Task Pane";

\_menuBarButton.Click +=

new Office.\_CommandBarButtonEvents\_ClickEventHandler(

menuBarButton\_Click);

**\_menuBarButtonCreateEvaluation =**

**(Office.CommandBarButton)\_menuBarPopup.Controls.Add(**

**Office.MsoControlType.msoControlButton,**

**missing,**

**missing,**

**missing,**

**false);**

**\_menuBarButtonCreateEvaluation.Style =**

**Office.MsoButtonStyle.msoButtonIconAndCaption;**

**\_menuBarButtonCreateEvaluation.Caption = "Create Evaluation";**

**//This is the Icon of the menu item**

**\_menuBarButtonCreateEvaluation.FaceId = 1979;**

**\_menuBarButtonCreateEvaluation.Click +=**

**new \_CommandBarButtonEvents\_ClickEventHandler(**

**menuBarButtonCreateEvaluation\_Click);**

}

}

* 1. The menuBarButtonCreateEvaluation\_Click method is invoked when the evaluation menu item is clicked on. This method is responsible for creating the form region:

private void menuBarButtonCreateEvaluation\_Click(

CommandBarButton Ctrl, ref bool CancelDefault)

{

try

{

Outlook.Application application =

new Outlook.ApplicationClass();

Outlook.NameSpace ns = application.GetNamespace("MAPI");

Outlook.MAPIFolder mapi = ns.GetDefaultFolder(

Outlook.OlDefaultFolders.olFolderNotes);

Outlook.MailItem mailItem = mapi.Items.Add(

FormRegionEvaluation.\_formRegionMessageClassName) as

Outlook.MailItem;

mailItem.Display(false);

}

catch

{

string errorMessage = String.Concat(

"An error was encountered.",

Environment.NewLine,

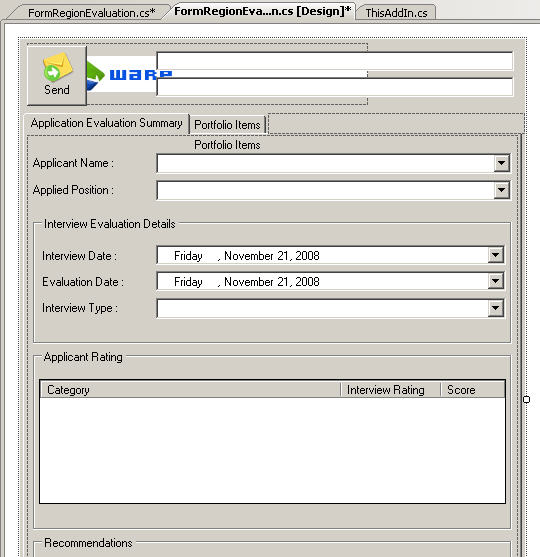
"Please contact your system administrator.");

MessageHelper.DisplayError(errorMessage);

}

}

1. The menuBarButtonCreateEvaluation\_Click method behaves as follows: the application obtains an instance of a Namespace object from the Message Application Programming Interface (MAPI) namespace. Using this object, a MAPIFolder instance is created from which the form region is loaded.
2. Upon showing, the form region queries the *PeopleSoft* Web services for the list of applicants to populate one of the *ComboBoxes*. The data for the other *ComboBoxes* are obtained the same way but upon the startup of the Outlook application. For faster retrieval, the data is cached in a temporary folder using XML serialization. When an evaluator selects an applicant from the list, the Web services are again called to provide the list of positions being applied for by the selected candidate.
3. The forms region created as part of the enhanced solution is FormRegionEvaluation.cs seen here is design view:



1. The FormRegionEvaluation.cs file contains the following code that shows the retrieval of the list of Interview Levels from the Web service, saving to and reading it from the cache, and binding it to a ComboBox control:

OBASampleAppKitForPSFT.PSFTWebServices.LookUpTableServices lookUpTbl =

new OBASampleAppKitForPSFT.PSFTWebServices.LookUpTableServices();

CacheHelper<InterviewLevel>.SaveToCache(

lookUpTable.GetInterviewLevels());

List<InterviewLevel> intLevels =

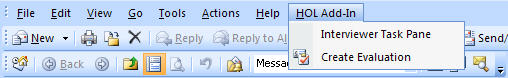
CacheHelper<InterviewLevel>.RetrieveFromCache();

cbInterviewType.DataSource = intLevels;

cbInterviewType.DisplayMember = "LevelDescription";

cbInterviewType.ValueMember = "LevelID";

1. Evaluations are created by running the HOLMasterPlus solutions HOLOutlookAddIn project from Visual Studio and selected HOL Add-In | Create Evaluation from the Outlook custom menu:



### Sending the Evaluation Form

1. Because it is derived from an Outlook message class, the evaluation form has an associated MailItem object. An important property of this object is the PropertyAccessor, which can hold the evaluator’s inputs. It supports three namespaces under the MAPI namespace: the proptag namespace, the id namespace and the string namespace. Since all inputs can be accessed through string-identified properties, the third option is used.
   1. More information on *Referencing Properties by Namespace* can be found at the following URL:

<http://msdn2.microsoft.com/en-us/library/bb147567.aspx>

1. To map each input to a unique property, the method PropertyAccessor.SetProperty() is called, with the property name in correct format, and the value with which it is assigned. The following code demonstrates this (using the Applicant Name field):

string SCHEMA\_APPLICANT\_NAME =

"http://schemas.microsoft.com/mapi/string/

{d7c99dbe-3ca3-4cc1-89ec-ef22ebc58c86}/ApplicantName";

\_mailItem.PropertyAccessor.SetProperty(

SCHEMA\_APPLICANT\_NAME,

cbApplicantName.Text);

1. After the properties have been set, the portfolio items specified are attached along with the evaluation summary. This is simply done through the MailItem.Attachments.Add() method:

foreach (DataGridViewRow row in gridAttachments.Rows)

{

\_mailItem.Attachments.Add(

row.Tag, // Tag contains the full path of the file

Outlook.OlAttachmentType.olByValue,

1,

attachmentTitle);

}

1. Before the evaluation is sent, it is saved to the PeopleSoft web services.
2. The FormRegionEvaluation.cs class relies on cached data retrieved from PeopleSoft that si setup when the add-in is loaded (the following snippet from ThisAddIn.cs):

private void LoadEvaluationDataToCache()

{

LookUpTableServices lookUpTable =

new LookUpTableServices();

CacheHelper<InterviewLevel>.SaveToCache(

lookUpTable.GetInterviewLevels());

CacheHelper<InterviewRating>.SaveToCache(

lookUpTable.GetInterviewRatings());

CacheHelper<EvaluationCategory>.SaveToCache(

lookUpTable.GetEvaluationCategories());

CacheHelper<Recommendation>.SaveToCache(

lookUpTable.GetRecommendationStatus());

}

1. This code is accessed from cache when the form region is loaded (FormRegionEvaluation.cs):

List<InterviewLevel> intLevels =

CacheHelper<InterviewLevel>.RetrieveFromCache();

List<InterviewRating> intRatings =

CacheHelper<InterviewRating>.RetrieveFromCache();

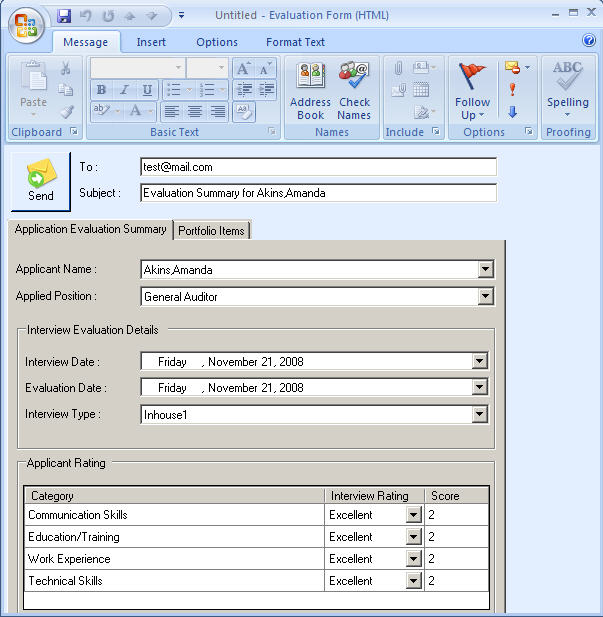
List<EvaluationCategory> evalCategories =

CacheHelper<EvaluationCategory>.RetrieveFromCache();

List<Recommendation> recommendations =

CacheHelper<Recommendation>.RetrieveFromCache();

1. As an exercise in accessing data from PeopleSoft remove the calls to CacheHelper found in FormRegionEvaluation.cs and replace them with direct calls to PeoleSoft using the code that originally setup the cache in ThisAddIn.cs.
   1. Make sure the form region still populates and sends correctly. You will likely notice degradation in performance when the email containing the form region is created.
2. An exercise in user interface development consider the following screenshot of the custom form region:



1. Notice that there is no scrollbar along the right side. This means if such an email is created on a machine with a lower resolution monitor, there is no way to set the controls at the bottom of the form region. Fixing this demonstrates the form regions are simply Windows Forms UI and are developed just like any other Windows Form application (except the UI resides in an email message). Compete the exercise to fix the form but either:
   1. Option 1: add a scroll bar to the right side of the form and enable scrolling so that scroll can be used to access all controls on the form.
   2. Option 2: change the dimensions of the form to be no more than 500 pixels in heigh.

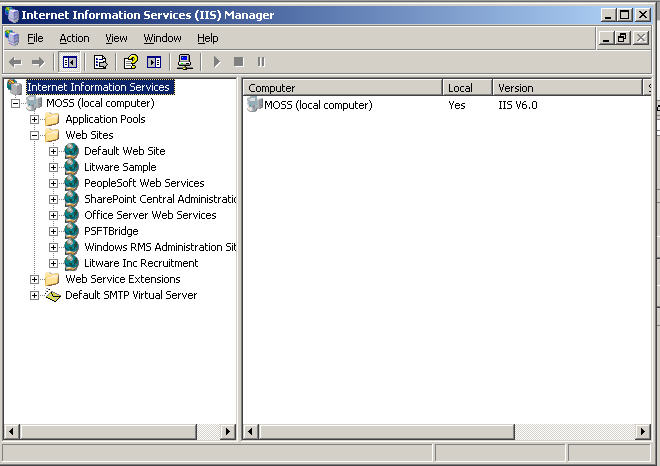
## ClickOnce Installation

ClickOnce is an incredibly useful technology that allows install software to automatically update without requiring any programming on the part of a developer. The lab differs significantly from the full OBA Sample Application Kit for PeopleSoft implementation in that the lab solution does not make use of a database. The OBA Sample Application Kit for PeopleSoft installation was complicated because:

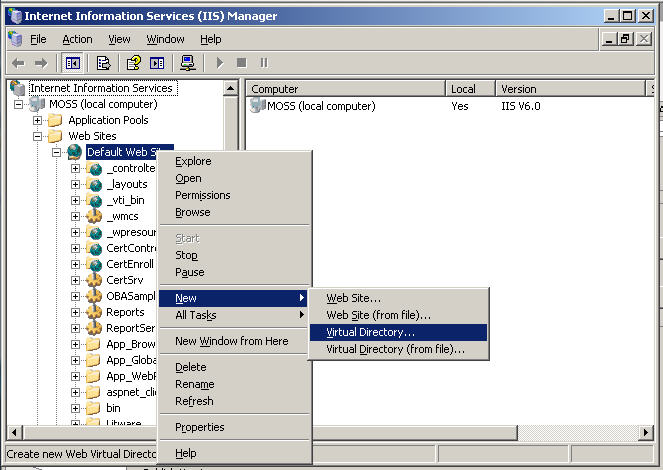
1. A separate install package had to be developed to install the database files.
2. A boot strap component had to be used in order to allow ClickOnce to launch the install package.

The HOLMasterPlus solution does not have any such issue because it does not make use of a database file that needs to be installed with the application. Conventional ClickOnce can therefore be used. In order to support ClickOnce:

1. A virtual directory needs to be created.
2. First create the following director using Windows Explorer C:\Inetpub\wwwroot\HOLOutlookAddIn
3. On the local machine (localhost) using the administrative interface for Internet Information Server (IIS) associated an alias name HOLOutlookAddIn with the directory path just created. In order to this launch IIS Manage ( Start | Administrative Tools | Internet Information Services (IIS) Manager)
   * 1. This entire installation process is also described in the sample application’s install document “Installation Guide: OBA Sample Application Kit for PeopleSoft.”



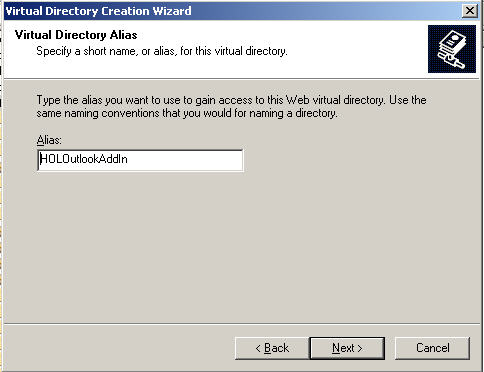
1. Open Web Sites and right click on the Default Web Site and select New | Virtual Directory:



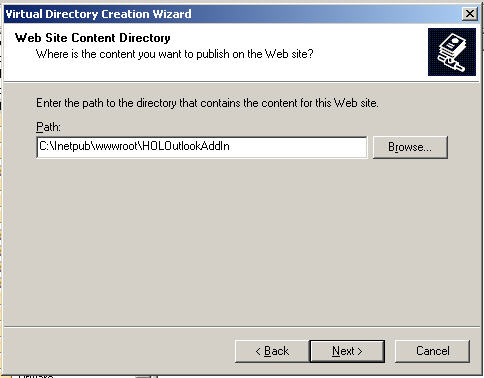
1. This will display:



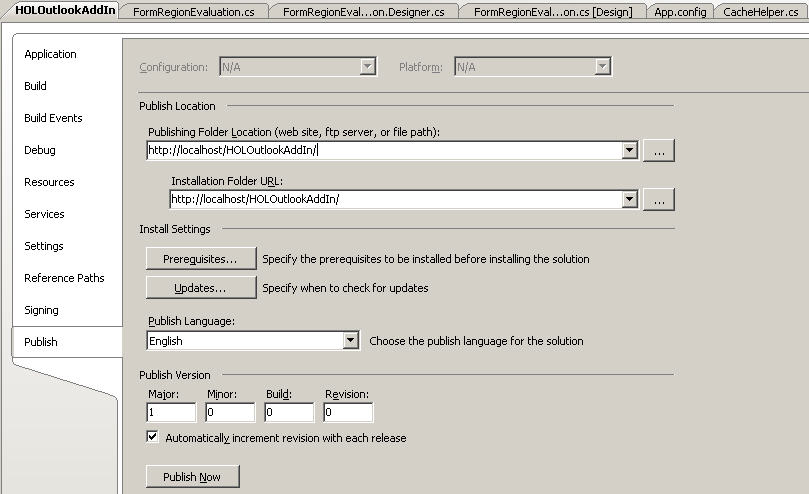
1. Click Next



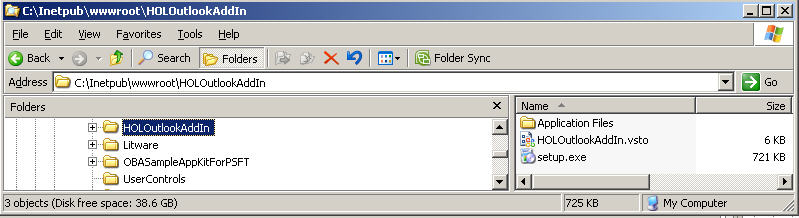
1. Enter Alias name, HOLOutlookAddIn, and click Next.



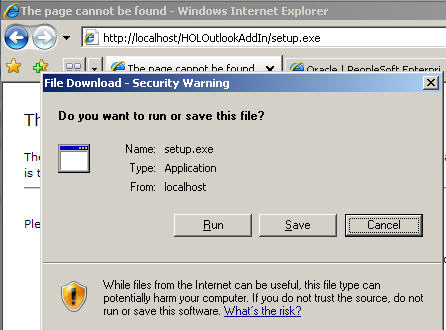
1. Enter a Path of C:\Inetpub\wwwroot\HOLOutlookAddIn and click Next.
2. Click Next on the Virtual Directory Access Permissions screen.
3. Click Finish.
   1. The HOLOutlookAddIn web site on localhost will be used for publishing the application. In the real world a company would have a web site for deploying their ClickOnce-based applications. Localhost is just for demonstration purposes.
4. Once the virtual directory is setup, publishing installers for ClickOnce can be done by right-clicking on the HOLOutlookAddIn project, then clicking at Properties. Select the Publish tab to configure the installer, as shown below:



1. Click on Publish Now and the physical directory will receive the current build (version 1.0.0.0):

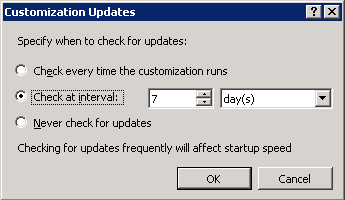


1. To install simply navigate to the <http://localhost/HOLOutlookAddIn/setup.exe> url in internet Explorer:



1. To install click Run followed by Run at the next prompt. Then click on Install.

Every time the add-in starts it checks the version available in the URL provided with the installed version. If mismatch occurs, the update is installed. An elegant feature of ClickOnce is that it can be configured to update at a specific time. This can be engaged by clicking the Updates button in the Publish tab, which displays the Customization Updates dialog:



The Customization Updates dialog controls how updates are to be checked. The update can be checked every time the application runs, check updates after a given amount of time or updates cannot be checked for at all. After selecting an appropriate setting for update, click Publish to create the ClickOnce installer. The created files would look like the image shown below:

## Viewing PeopleSoft data to MOSS 2007

### Business Data Catalog

This lab will demonstrate integrating PeopleSoft Microsoft Office SharePoint Server (MOSS) 2007. This can be done using a new business integration technology that ships with MOSS 2007: Business Data Catalog (BDC). The BDC is a shared service that enables MOSS 2007 to surface business data from back-end server applications requiring little to no custom code to manage connections and retrieve data. The BDC bridges the gap between the portal site and your business applications, and enables you to bring in key data from various business applications to MOSS 2007 Web parts, lists, search, user profiles, and custom applications.

The following sections will demonstrate how to integrate PeopleSoft with MOSS 2007 by creating an application definition file for PeopleSoft using the Business Data Catalog Editor.

### Creating Application Definition Files using the BDC Editor

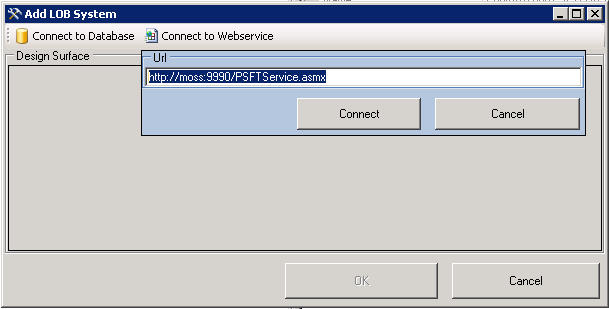
An ADF is an XML file that describes the metadata model of the backend system that you are connecting to. In order for the BDC to function correctly, the application definition file must be free of errors. Creating an ADF used to be a long and tedious process because developers had to create the XML schema manually using text editors. MOSS 2007 introduced the Business Data Catalog Editor, which simplifies the creation of ADFs. This is a free tool that comes with the SharePoint Server 2007 SDK, which can be downloaded at this site: <http://www.microsoft.com/downloads/>. The ADFs that are used in this application are all created using the BDC Editor.

PeopleSoft is one of the most popular Human Resource ERP system used by many medium/large corporations. Microsoft Office SharePoint Server comes with built-in Business Data Catalog Web parts that are used to provide Enterprise Application Integration services in order to establish communication between systems such as PeopleSoft and a SharePoint Portal. Web parts use Web services to connect to many other applications as well, such as SAP, Oracle Applications, Siebel and other line-of-business (LOB) applications.

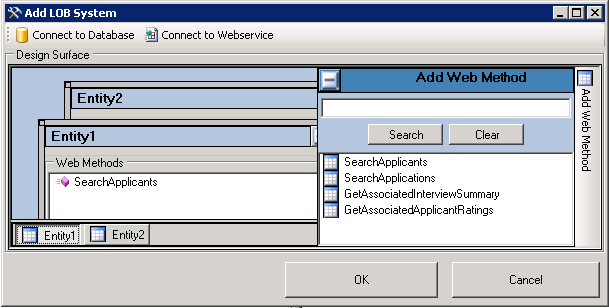
PeopleSoft’s API, like any ERP system, is designed in a granular way to allow maximum flexibility for application developers. Because of this, there is a general expectation that multiple APIs must be invoked as a set to accomplish a single unit of business functionality. With this in mind, attempting to bring PeopleSoft data into MOSS 2007, by working directly against PeopleSoft’s API, is not recommended.

Before PeopleSoft’s component interfaces can be consumed by Microsoft Office SharePoint Server 2007’s BDC, it is necessary to compose user-targeted business entities out of these discrete APIs and expose these resultant business entities as Web services. Being one of the most flexible solutions, creating a wrapper Web service that shall act as a bridge between PeopleSoft and BDC is the solution implemented in the sample application. A discussion on creating custom Web services is discussed in the “Appendix A: Wrapper Web service for BDC/PeopleSoft Integration” section of this document, and publishing it for MOSS 2007’s use is discussed separately in the installation document. This part of the documentation assumes that the custom Web service is up and running and ready to be used by the BDC Editor. Discussed below are the steps that demonstrate how an application definition file can be created to communicate with PeopleSoft via BDC as follows:

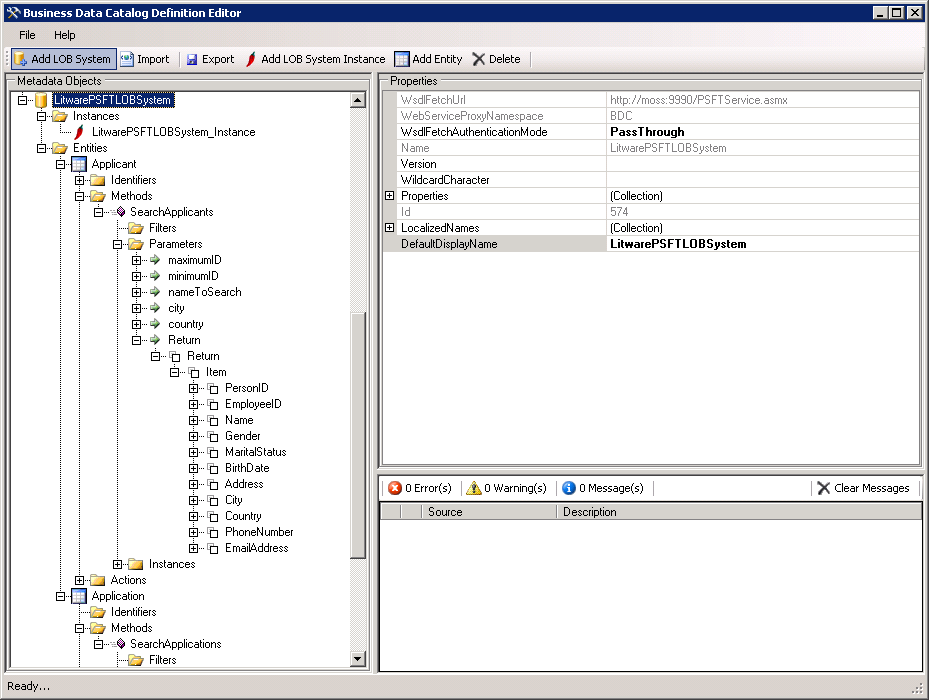
1. Open the Microsoft Business Data Catalog Editor and connect to an LOB system by clicking the Add LOB System button from the toolbar. This will open an Add LOB system dialog box. To connect to the Web service, click the Connect to Webservice button. Enter the complete URL of the published custom Web service and click Connect to continue.



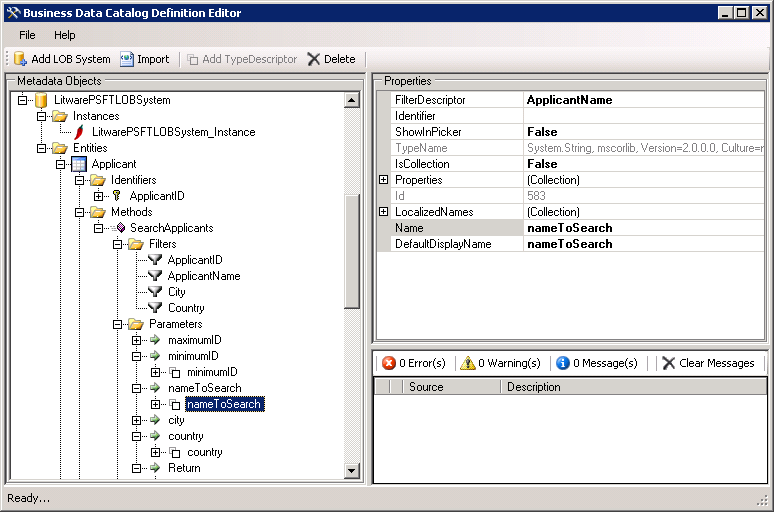
1. Once connected, the BDC editor retrieves all web methods in the Web service. These methods allow communication between BDC and SAP. Drag the available methods to the designer surface. Click Ok to continue, and then provide a meaningful name for the new LOB system (i.e. LitwarePSFTLOBSystem).



1. The BDC editor will generate the selected entities similar to the image below. Take note that the names of the entities are already modified to reflect meaningful names.

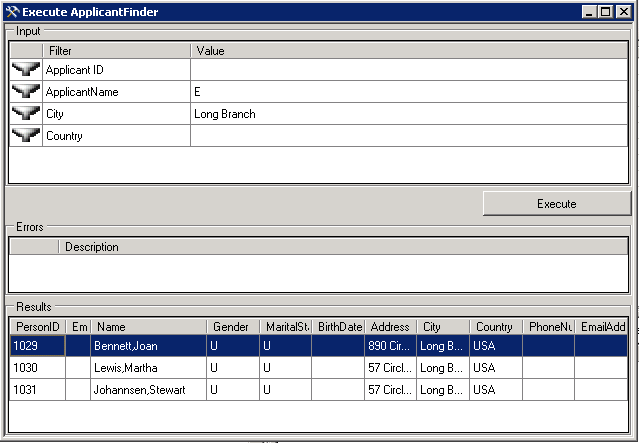


1. Configure the wildcard property of the new LOB system to reflect the wildcard character that is being used by the Web service during comparison (i.e. ‘\*’).
2. The editor-generated entities will not work immediately. Identifiers, filters and method instances should be added manually. The customer entity has CustomerID as its identifier and the flight entity has CarrierID as its identifier. After adding these identifiers in the editor, make sure that the CarrierID and CustomerID in the return parameters are mapped to the newly created identifiers.
3. The editor-generated entities will not work immediately. Identifiers, filters and method instances should be added manually. The applicant entity has ApplicantID as its identifier and the application entity has ApplicantID , RecruitmentID and InterviewID as its identifiers. After adding these identifiers in the editor, make sure that the identifier fields in the return parameters are mapped to the newly created identifiers.
4. To add filter descriptors, select the Filters folder and click the Add Filter button in the toolbar. This creates a new filter with default name of FilterDescriptor0. Provide a meaningful name for the filter, and modify its FilterType property based on the design of the Web service (i.e. ApplicantName filter makes use of Wildcard searches). Remember to map the parameters to the newly created filter descriptors. Shown below is an example of a parameter in customer entity that is mapped to a newly created filter ApplicantName.



Make sure that all parameters needed to be mapped to filter descriptors are updated to ensure a working application definition file.

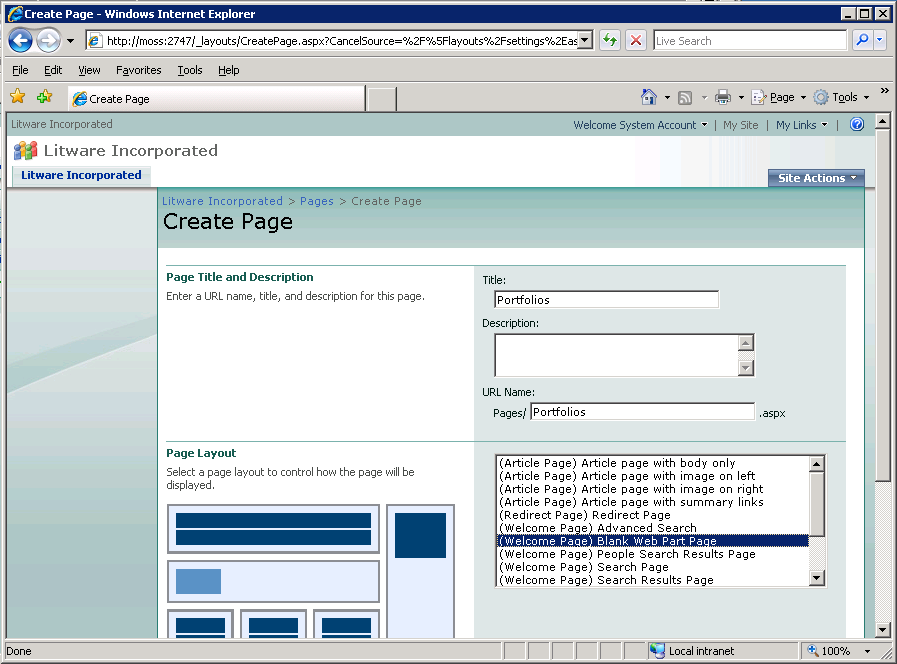
1. In order to execute the methods, a method instance must be created. The web methods in the wrapper Web service were built to support both the FinderMethod and the SpecificFinderMethod. Adding method instances of these types can be done in the same manner as with BDC for SQL 2005. Shown below is an image showing the finder method in action:



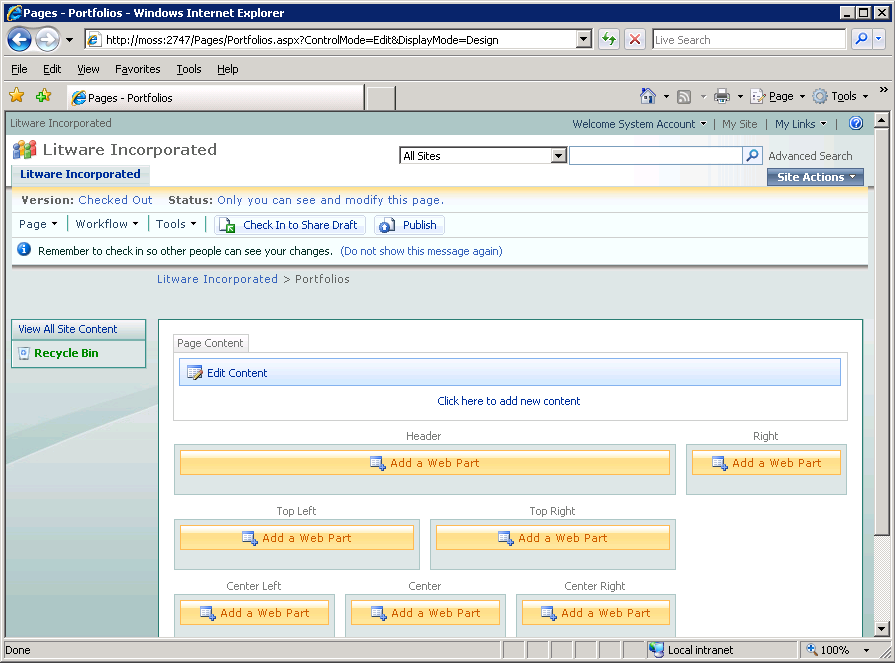
### Displaying Data from PeopleSoft in MOSS 2007 Web parts

Using MOSS 2007’s BDC, users can display data from LOB systems like PeopleSoft with little to no code required. The primary step to accomplish this is to create an application definition file for the database and import it to MOSS 2007. Creating an application definition file, using Microsoft Business Data Catalog Editor, was discussed in the previous section of this paper, while importing application definition files to SharePoint was discussed in the installation document of this sample application. Thus, this assumes that MOSS 2007’s BDC is already configured, and the only task that is left is how to display these data from LOB systems using Web parts. On this demo application, business data list and business data related lists are used. The following section will discuss briefly how to do it in MOSS 2007:

1. In order to display Web parts, we create an empty page in the site that contains empty Web parts. We can do this by clicking the Site Actions menu in the home page and selecting the Create Page menu. Configure the new page and be sure to select the Blank Web part Page as page layout. Click the Create button to continue.



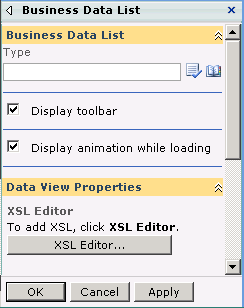
1. The new page will be opened in edit mode. The designer can begin inserting Web parts that displays data from the LOB system through to the BDC.



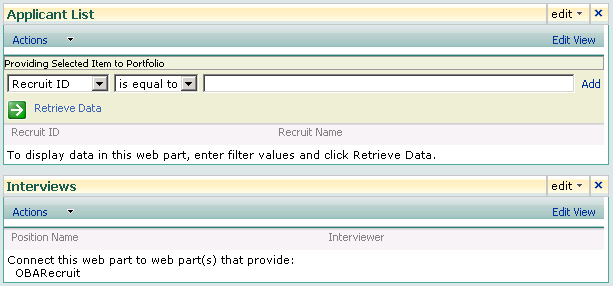
1. Inserting Web parts in the new page can be done easily by following simple steps. For this example, two types of Web parts will be used: the Business Data List Web part and the Business Data Related list Web part. The Business Data List displays a list of items from a data source in the BDC, while Business Data Related list displays a list of items related to one or more parent items from a data source in the BDC. Take note that in order to display a business data related list, an association method should be supported by the application definition file of the data source.

The package entity in the LitwareSQLLOBSystem application contains three viewable lists, the list of recruits, related positions and related portfolios. To be able to display these, we need to insert a business data list for the list of packages and business data related list for the related lists.

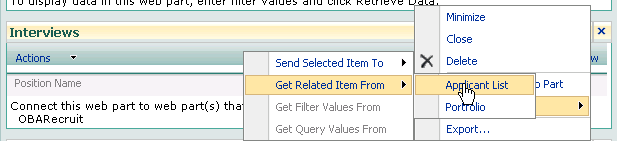
On the newly created page, select the location where the Web part will be placed and click the Add a Web part button. In the Add a Web part dialog box, scroll down to the Business Data collections and select Business Data List and Business Data Related List. Click Add to continue. This will add two empty Web parts in the page. To configure the Web part, click the link that reads “Open the tool pane.” This shows a Business Data List task pane. For the type requirement, browse the data list by clicking on the browse button. From the Business Data Type Picker web page dialog box select the Packages entity of the LitwareSQLLOBSystem and click Ok. To add the data list in the web page, click the Ok button in the Business Data List task pane.



Adding a Business Data Related list is also similar to adding Business Data list. The only difference is that only those entities that support the association method will be displayed in the Business Data Type Picker web page dialog box. Initially, the two Web parts are not connected as is shown below:

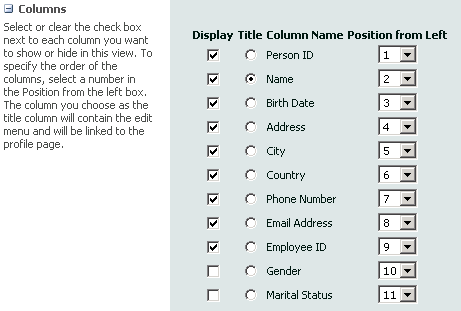


To relate the table, click the Edit | Connections | Get Related Item From | Applicant List in the business data related item Web part. Below is a screenshot showing a business data list and business data related list in action in design mode:

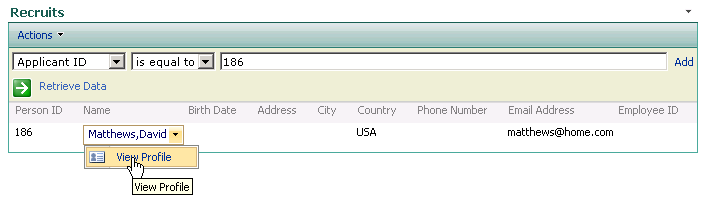


To commit the changes to the page, click the publish button in the Page Editing Toolbar.

1. Every entity that supports a specific finder method has a view profile action created by default. Initially, the view profile action is not displayed in the list. To enable the action, click the Edit View link in the Business Data list Web part. In the Edit View page, scroll down to the Columns property and select which among the columns is the title column.



1. Once clicked, the data in the title column will display the view profile menu. Shown below is a Web part displaying the view profile action in the title column:



1. Other entities can be added in a similar way as that which was discussed above. Just do not forget to publish the page after editing.

# Appendix A: Wrapper Web service for BDC/PeopleSoft Integration

As discussed in an earlier section, it is recommended to pre-compose business entities out of PeopleSoft web method prior to using them in the BDC. This is performed through the creation of an ASP.NET Web service to interact with the PeopleSoft system. To distinguish the ASP.NET Web service that wraps PeoplseSoft data from the PeopleSoft Web services used, the ASP.NET Web service accessed by the BDC will be referred to as the wrapper Web service. This web service is implemented in both the OBA Sample Application Kit for PeopleSoft code-base and in the HOLMasterPlus solution used in this lab.

The functionality of the PeopleSoft BDC is determined by the methods of the wrapper Web service. Because of this, there are some important points to be considered in creating a custom Web service like the wrapper Web service to communicate with PeopleSoft. The discussion in this section of the document takes into consideration the techniques for extracting data from PeopleSoft through component interfaces that was exposed as Web services, which was previously presented. Thus, this section will concentrate on the design on the wrapper Web service’s web methods. Take note that the BDC editor uses these web methods in the application definition file design process.

A good Web service design should take note of the basic structure of a common BDC. These include support for filters, identifiers, finder methods, specific finder methods and associations. Even the wildcard character that will be used for wildcard searches is dictated by the custom Web service. All of these entities are incorporated into the design of the wrapper Web service.

## Interacting with PeopleSoft

Similar to its Outlook 2007 add-in counterpart, the wrapper Web service communicates with PeopleSoft via PeopleSoft’s component interfaces exposed as Web services. This enables the wrapper Web service to retrieve data from PeopleSoft. For simplicity of design, the wrapper Web services makes use of the same library that is used by Outlook 2007 in accessing PeopleSoft .

Consuming these PeopleSoft Web services is performed in the same manner as the Outlook 2007 add– in. Refer to the previous Outlook add-in discussion for an in-depth discussion.

## Web Method Design

The most important part of the wrapper Web service design takes place in creating the web methods. Filters, identifiers and method instances must be included in the design for the wrapper Web service to be compatible with the BDC. To illustrate how BDC elements are included in the design, this section of the document will describe how the implemented wrapper Web service accesses PeopleSoft recruitment data.

Identifiers are critical parts of the application definition file. If a developer plans to integrate a specific finder method to its BDC, an identifier should be incorporated. Identifiers, as discussed earlier, are similar to primary keys and can be used to search for a specific record in the database. Identifiers are unique. For example, in the case of recruitment data, the ApplicantID field is the identifier.

Filters are user-visible search criteria in a MOSS 2007 site. Users filter their searches based on the keywords entered and, internally, these keywords are passed to the wrapper Web service’s methods. User filtering is achieved by mapping the filters to the parameters of the wrapper Web services methods. In order for the BDC to support filters, the wrapper Web service must contain parameters that support filtering. To demonstrate how the wrapper Web service supports filters, consider the applicant page on the MOSS site, which allows users to search for applicants by ID, name, city or country. In order for this to be possible, the application definition file must be configured to have four filter descriptors. These expose the requisite search criteria to the BDC as these filter descriptors are mapped to the parameters of the applicant Web service’s web methods. The wrapper Web service to access the applicant was designed to accept two parameters that allow filtering by ID, name, city and country. Shown below is the construction of the web method that returns a list of applicants.

public List<PSFTWebServices.BusinessObjects.Applicant>

SearchApplicants(

int maximumID,

int minimumID,

string nameToSearch,

string city,

string country)

Notice the five parameters in the method. The maximumID and the minimumID refer to the applicant’s ID. The reason why two parameters are specified for the single filter is for the method to support both the SpecificFinder method and Finder method. When used in the SpecificFinder method, passing the same value to the two parameters will return a single record that points to a single applicant. However, in the Finder method, we pass a defined ID range to allow it to return applicant records based on other criteria aside from the ID filter. Similar to the web method SearchApplicants, the SearchApplications web method takes the same logic in the parameter design.

public List<Application> SearchApplications(

int minimumID,

int maximumID,

string applicantName,

string positionTitle,

int minRecID,

int maxRecID,

int minRecIntID,

int maxRecIntID)

When designing the web methods, it is important to take the search algorithm into consideration. Unlike SQL Server 2005 databases, where queries naturally support search including wildcard-based searches, the web method should support this type of search programmatically. To illustrate how search is performed inside the wrapper Web service, consider the CompareString method. The code below implements a private method named CompareString that takes the strings to be compared as parameters and returns a Boolean value to the calling method. The CompareString method takes care of determining if a wildcard character is included in the search. The method supports common wildcard searches, the same search scenarios supported by the BDC. These include the following: *equals, starts with, end with and contains*. Below is the implementation of the CompareString method. Notice how the wildcard characters are defined in the method.

private string \_wildcard=”\*”;

private bool CompareString(

string paramFromList,

string paramFromParameter)

{

int lenghtOfSearchString = 0;

if (String.Compare(paramFromParameter, \_wildcard) == 0)

{

return true;

}

if (paramFromParameter.StartsWith(\_wildcard) &&

paramFromParameter.EndsWith(\_wildcard))

{

paramFromParameter = paramFromParameter.Replace(

\_wildcard, " ");

paramFromParameter = paramFromParameter.Trim();

if (paramFromList.ToLower().IndexOf(

paramFromParameter.ToLower()) >= 0)

{

return true;

}

}

else if (paramFromParameter.EndsWith(\_wildcard))

{

lenghtOfSearchString = paramFromParameter.Length - 1;

if (paramFromList.Length >= lenghtOfSearchString)

{

if (String.Compare(

paramFromList.Substring(

0,

lenghtOfSearchString).ToLower(),

paramFromParameter.Substring(0,

lenghtOfSearchString).ToLower()) == 0)

{

return true;

}

}

}

else if (paramFromParameter.StartsWith(\_wildcard))

{

lenghtOfSearchString = paramFromParameter.Length - 1;

int startIndex = 0;

if (paramFromList.Length > lenghtOfSearchString)

{

startIndex = paramFromList.Length - lenghtOfSearchString;

if (String.Compare(

paramFromList.Substring(startIndex).ToLower(),

paramFromParameter.Substring(1).ToLower()) == 0)

{

return true;

}

}

}

else

{

if (String.Compare(paramFromList.ToLower(),

paramFromParameter.ToLower()) == 0)

{

return true;

}

}

return false;

}

The CompareMethod is used by the web methods in the wrapper Web service. The searching is performed using LINQ as exemplified by the sample search code below.

var applicantList =

from list in applicants

where list.PersonID >= minimumID &&

list.PersonID <= maximumID &&

CompareString(list.Name, nameToSearch) == true &&

CompareString(list.City, city) == true &&

CompareString(list.Country, country) == true

select list;

Shown below is the snippet of the web method, GetCustomerList, demonstrating how the CompareString method is used and how the search algorithm, in its entirety, is implemented.

[WebMethod]

public List<PSFTWebServices.BusinessObjects.Applicant> SearchApplicants(

int maximumID,

int minimumID,

string nameToSearch,

string city,

string country)

{

PSFTWebServices.ApplicantServices appService = new ApplicantServices();

List<Applicant> applicants = new List<Applicant>();

applicants = appService.GetAllApplicants();

var applicantList =

from list in applicants

where list.PersonID >= minimumID &&

list.PersonID <= maximumID &&

CompareString(list.Name, nameToSearch) == true &&

CompareString(list.City, city) == true &&

CompareString(list.Country, country) == true

select list;

return applicantList.ToList<Applicant>();

}

The design of the Web service is essential in the operation of the BDC. For a developer to be able to develop a working BDC that supports most of the search functionality in Microsoft Office SharePoint Server 2007, the custom Web service design should be based on the guidelines discussed above. For a more in depth discussion regarding modeling of Web services that interact with MOSS 2007’s BDC visit <http://msdn2.microsoft.com/en-us/library/bb447533.aspx>.

# Appendix B: Progress Display Bar

This section presents how the progress bar is displayed to inform the user that PeopleSoft is being accessed. This section demonstrates how to:

1. Create a pseudo progress bar form.
2. Use the BackgroundWorker class to process time-consuming task while displaying a pseudo progress form.

There are various points in the sample application’s application that we need to access data from PeopleSoft via Web services. Such operation takes a noticeable amount of time to complete, given that PeopleSoft is being deployed as a background process on a machine dedicated for development (the environment in which the sample application was developed). The progress form provides a feedback mechanism while PeopleSoft is accessed. Since operations like that do not render to exact computation as to its duration or time of completion, a pseudo progress bar is more appropriate. A pseudo progress bar indicates progress of what seems to be an ongoing process but it’s really nothing more than an eye candy to give user a sense of the word. The contraption is merely a form with a Timer control and a ProgressBar control with code designed to fit the applications common processes and their duration. The code below shows the heuristic algorithm used.

private void timer\_Tick(object sender, EventArgs e)

{

timer.Stop();

if (progressBar.Value < 95)

{

progressBar.Value += 1;

label1.Text = string.Format("{0}%", progressBar.Value);

if (progressBar.Value > 80)

{

timer.Interval += (50 \* \_multiplier);

}

else if (progressBar.Value > 60)

{

timer.Interval += (25 \* \_multiplier);

}

else

{

timer.Interval += (7 \* \_multiplier);

}

timer.Start();

}

}

protected override void OnClosing(CancelEventArgs e)

{

timer.Stop();

while(progressBar.Value < 100)

{

progressBar.Value += 1;

label1.Text = string.Format("{0}%", progressBar.Value);

Application.DoEvents();

Thread.Sleep(1);

}

base.OnClosing(e);

}

Displaying the form while a process in the background is running requires a bit of work. Using the code pattern below, we delegate the call to the time-consuming task to a BackgroundWorker object. When the task is finished, we close the pseudo progress bar form. If it happens that the task finishes first and the progress bar is half-filled, logic in the progress bar form fills the progress bar control until it is fully `filled. Now, if it happens that the task took longer than expected, the progress bar will stop at ninety five percent while it waits for the task to finish.

PseudoProgressForm progress = new PseudoProgressForm();

progress.ProgressLabel = "Querying PeopleSoft...";

BackgroundWorker backgroundWorker = new BackgroundWorker();

backgroundWorker.DoWork +=

delegate(object sender, DoWorkEventArgs e)

{

‘

‘time consuming task here

‘

};

backgroundWorker.RunWorkerCompleted +=

delegate(object sender, RunWorkerCompletedEventArgs e)

{

‘

‘called when time consuming task is done

‘

progress.Close();

};

backgroundWorker.RunWorkerAsync();

progress.ShowDialog();

When displayed, the progress bar appears as follows which serves to emphasize that the application used may be Outlook, but under the covers it is accessing PeopleSoft:

