

Windows Workflow Foundation

Hands-On Lab

Lab Manual

Lab 01 – Workflow Enabled Services and Other New Features in the .NET Framework 3.5

|  |  |
| --- | --- |
| Lab version: | 2.0.0 |
| Last updated: | 12/3/2008 |

Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Microsoft Corporation.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user.  Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document.  Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2005 Microsoft Corporation.  All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Visual C# and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**Contents**

[Overview 2](#_Toc216700867)

[Objectives 2](#_Toc216700868)

[System Requirements 2](#_Toc216700869)

[Setup 2](#_Toc216700870)

[Physical Folder Structure 2](#_Toc216700871)

[Installing Code Snippets 3](#_Toc216700872)

[Starting Material 6](#_Toc216700873)

[Using Code Snippets 6](#_Toc216700874)

[Acronyms Used in this Lab 7](#_Toc216700875)

[Scenario 8](#_Toc216700876)

[Exercise 1 – Workflow Enabled Services 8](#_Toc216700877)

[Task 1 - Creating a new Workflow project 9](#_Toc216700878)

[Task 2 – Working with the Receive activity 13](#_Toc216700879)

[Task 3 – Hosting the workflow 20](#_Toc216700880)

[Task 4 – Adding a client to the solution 22](#_Toc216700881)

[Exercise Summary 29](#_Toc216700882)

[Exercise 2 - WCF and Syndication 30](#_Toc216700883)

[Task 1 - Create a syndication service 30](#_Toc216700884)

[Task 2 – Add a service client application 36](#_Toc216700885)

[Task 3 – Accessing the ATOM feed 38](#_Toc216700886)

[Task 3 – Accessing the RSS feed 41](#_Toc216700887)

[Exercise Summary 41](#_Toc216700888)

[Exercise 3 - WCF Services and AJAX 42](#_Toc216700889)

[Task 1 – Creating a web site project 42](#_Toc216700890)

[Exercise Summary 46](#_Toc216700891)

[Lab Summary 47](#_Toc216700892)

# Overview

Estimated time to complete this lab: **60 Minutes**

## Objectives

This lab introduces the new features in the **.NET Framework 3.5** for **Windows Communication Foundation (WCF)** and **Windows Workflow Foundation (WF)**. The lab uses **Visual Studio 2008**, **.NET Framework 3.5**, andhas code in both VB and CSharp. The lab demonstrates a developer’s experience working with the newest parts of these technologies for the first time.

Exercises include building workflow enabled services that use WCF for service messaging and WF for service logic. Also RSS, ATOM, and REST style programming are included making use of these new aspects of WCF. Finally, all an AJAX application demonstrates the extremely re-usability of the service model.

## System Requirements

* Microsoft Visual Studio 2008
* .NET Framework 3.5
* Visual Basic
* Visual C#

## Setup

Unzip the lab to your local hard drive. Everything needed to complete the lab is in the zip file.

### Physical Folder Structure

File paths referenced in this lab assume the lab is installed in the following folder:

*C:\WF\WF 3.5 Labs\Lab[Number]*

Within the **Lab[Number]** folder, several child folders are available:

* **Ex1, Ex2, Ex3 –** Workspaces for each exercise in the lab.
* **resources** –Any files referenced in the lab can be found in the Resources subdirectory, including source code for custom assemblies referenced in the exercises.

Within each *Ex#* folders, the following are available:

* **before** – The work area for completing the HOL
* **after** – The fully completed HOL

### Code Snippets

All code required for this lab consisting of more than 2 lines is available as code snippets. To learn more about code snippets including how to install them and how to use them, see the snippet guide document for the language of your choice in the folder:

C:\WF\WF 3.5 Labs\Snippets

**Note:** For all code snippets needed in this lab, install both VB and C# snippets.

## Starting Material

### Acronyms Used in this Lab

* IIS – Internet Information Server
* REST – Representational State Transfer
* RSS – Really Simple Syndication
* WCF – Windows Communication Foundation
* WES - Workflow Enabled Services
* WF – Windows Workflow Foundation

|  |
| --- |
| What is a Workflow? A workflow is a model of a human or system process which is defined as a map of activities. An activity is a step in a workflow and is the unit of execution, re-use and composition for a workflow. A workflow is designed by laying out activities and is compiled to a .NET assembly that is executed on the workflow runtime and the Common Language Runtime (CLR). |

### Scenario

The exercises are built around a service that takes as input a number and returns the English language equivalent of that number. For example: 1,000 results in “One Thousand”. This functionality is exposed in several ways as services and is then called in various ways from various clients.

Also in these exercises is a service that returns

# Exercise 1 – Workflow Enabled Services

In this lab, you will build a simple Windows Workflow Foundation (**WF**) application that exposes a Windows Communication Foundation (**WCF**) Service that converts a number value into its English string presentation like what you use when writing a check.

For example, if you enter 123.45, the service transforms it to One Hundred Twenty-Three and 45/100. After you’ve created the service, you’ll create a Windows Forms client that calls into the workflow and executes the service method.

|  |
| --- |
| Workflow Enabled Services Workflow Enabled Services (**WES**) is a new feature of the .NET Framework 3.5. WES allows developers to easily expose a **WCF Service** from a **WF** enabled solution.  Additionally, **WF** workflows can consume **WCF Services** using the new **Messaging Activities**: **SendActivity** and **ReceiveActivity**. |

## Task 1 - Creating a new Workflow project

#### Using Visual Studio to create a new workflow project

1. Select **Start |** **All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008**.
2. Within Visual Studio, select **File | New Project**.
3. In the **New Project** dialog, under the **Project Types** section, select and expand **Visual C#**, then select **Workflow**.
4. In the **Templates** section, select **Sequential Workflow Console Application**.
5. Fill out the **New Project** dialog like this:

|  |  |
| --- | --- |
| **Property Name** | **Value** |
| Name | PaymentWF |
| Location | C:\WF\WF 3.5 Labs\Lab11\CSharp\Ex1\before |
| Solution Name | Exercise1 |

**Note:** If necessary put a check next to **Create directory for solution**

1. Click **OK**.

#### Renaming the workflow

1. In the **Solution Explorer**, right-click on **Workflow1.cs** and select the **Rename** command. Change the name to **Checks.cs**.
2. Click **Yes** in the rename dialog that Visual Studio presents.

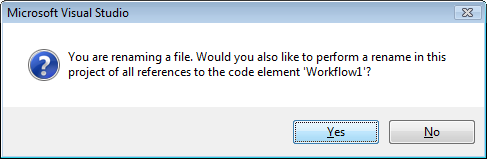
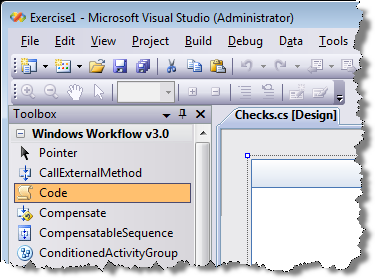


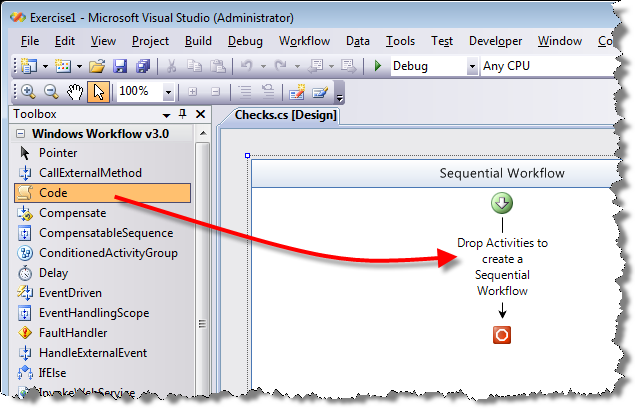
Figure : Click “Yes” to rename in references

#### Adding a Code activity to the workflow

1. From the Toolbox (select **View | Toolbox** if it’s hidden), drop a **Code** activity from the **Windows Workflow v3.0** tab group (see ) onto the drop zone (See ). The Code activity exposes an **ExecuteCode** event which makes it easy to add a bit of code to run as part of the WCF Service’s invocation.



**Figure 8: The Code activity on the toolbox**



**Figure 9: Drag a Code activity to the drop location on the workflow**

1. Using the **Properties** window, change the activity’s name from **codeActivity1** to **CheckLogic**.

#### Implementing the Code activity

The code activity allows developers to add custom code directly into a workflow.

1. In the Workflow designer, right-click the **CheckLogic** activity and select **Generate Handlers**. The code editor will open. You’ll find a single event handler for the **ExecuteCode** event.
2. Add the following two fields to the class, outside the new event handler:

|  |
| --- |
| **Note:** Newly added code is highlighted in yellow. |
| **public Checks()**  **{**  **InitializeComponent();**  **}**  public string checkText = "VOID";  public decimal checkAmount = 0;  **private void CheckLogic\_ExecuteCode(object sender, EventArgs e)**  **{**  **}** |

#### Adding the class that does converts the values

1. **Right-click** the **PaymentWF** project in **Solution Explorer**, select **Add | Existing Item**.
2. Select the following code file and add it to the project:  
   C:\WF\WF 3.5 Labs\Lab11\resources\Code Files – Csharp\Ex1\en\_US\_Translation.cs

This class provides the logic to convert a decimal value to a string.

1. Select **File | Save All**.

#### Using the conversion class in the Code activity

1. Return to the **Checks.cs** code editor and insert the following code within the body of the **CheckLogic\_ExecuteCode** method:

|  |
| --- |
| **private void CheckLogic\_ExecuteCode(object sender, EventArgs e)**  **{**  checkText = en\_US\_Translation.NumToStr(checkAmount);  **}** |

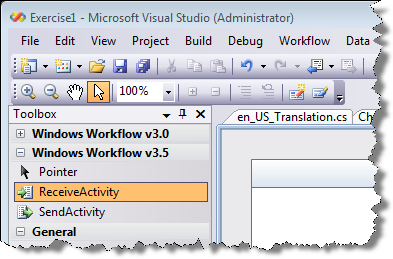
1. Ensure the solution builds properly (**Build | Build Solution** **or CTRL+Shift+B**).

In version 3.0 of Windows Workflow, you would need to write a bit of plumbing code to expose a WCF service from your workflow. In the next few steps, you’ll use the new **ReceiveActivity** to not only expose a server, but you’ll actually define the service contract using the new design-time tools in Visual Studio 2008.

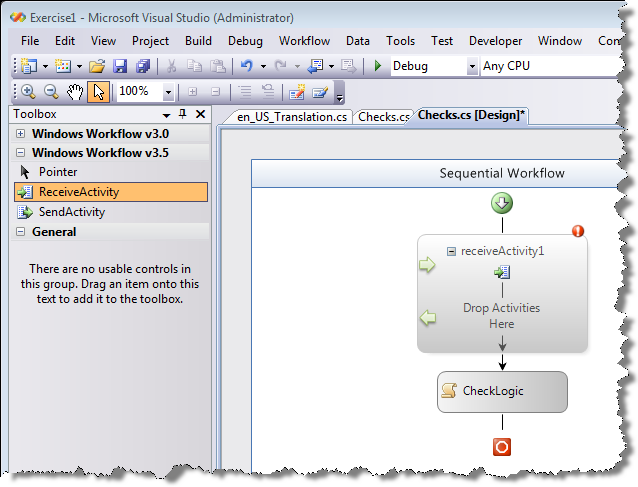
## Task 2 – Working with the Receive activity

The **Receive** WF activity enables a workflow to receive data from outside the workflow by exposing the workflow as a WCF service.

1. Return to the Workflow designer for Checks.cs.
2. From the **Windows Workflow v3.5** tab group in the Toolbox, drop a **Receive** activity (See ) onto the workflow between the green start arrow and the existing **CheckLogic** Code activity so that it runs before the CheckLogic activity. (See ).

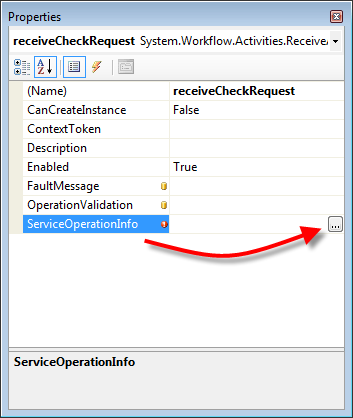
****

**Figure 10: The new WCF-aware ReceiveActivity on the Toolbox**

****

**Figure 11: The ReceiveActivity added to the workflow**

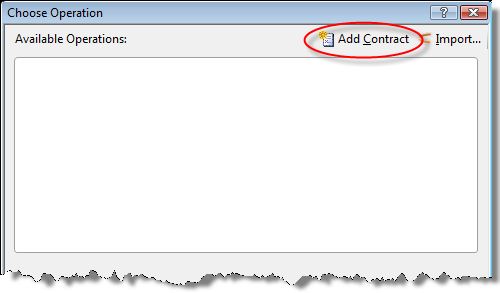
1. Using the **Properties** window, change the name from **receiveActivity1** to **receiveCheckRequest**.
2. In the **Properties** window, select the **ServiceOperationInfo** property and then click the ellipsis button (see ) to open the **Choose Operation** dialog. It is here that you define the service contract that will be exposed by the activity.

****

**Figure 12: Opening the Choose Operation dialog from the Properties window**

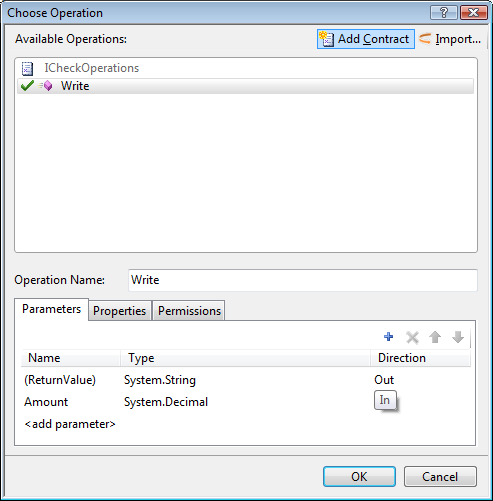
#### Defining an operation contract

1. In the **Choose Operation** dialog, you can either define a new service operation contract or import and existing one. In this exercise, you’ll create a new contract using the tools here. Click the **Add Contract** button (see ).

****

**Figure 13: The Choose Operation dialog box and the Add Contract button**

1. Click **Contract1** in the **Available Operations** list and then change the name from **Contract1** to **ICheckOperations** via the **Contract Name** text box. Service contracts are expressed as interfaces which by convention are named with a capital I.
2. In the **Available Operations** list, click **Operation1** and change its name from **Operation1** to **Write** using the **Operation Name** text box.
3. In the list of **Parameters**, change the **(ReturnValue)** property from **System.Void** to **System.String**.
4. Add a new input parameter named **Amount**, of the type **System.Decimal**. The **Choose Operation** dialog box should resemble once you’ve completed all of the steps.

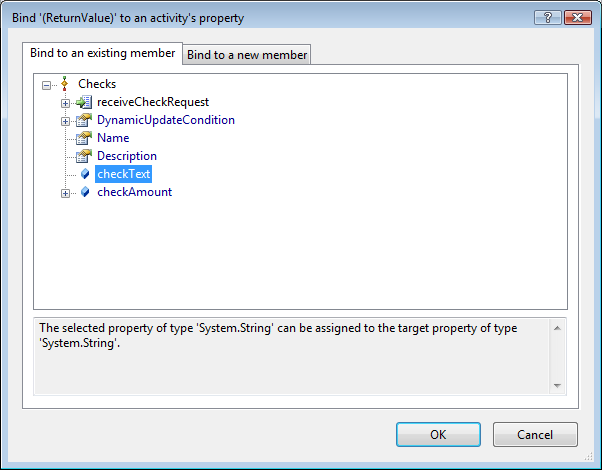
****

**Figure 14: The completed Choose Operation dialog box**

1. Click **OK** to close the dialog box. You’ve now defined a simple operation contract with one method that accepts a decimal value as input and returns a string as output.
2. In the **Properties** window, change the **CanCreateInstance** property to **True**. This is necessary because service invocation is the only facility you’ve exposed to have the workflow run.

#### Setting the value returned by the workflow

1. Select the **(ReturnValue)** property and click the ellipsis button to open the **Bind Property** dialog box.
2. Select **checkText** from the list of options under the **Bind to an existing member** tab (see ) and click **OK**.

****

**Figure 15: Binding the return value to the checkText property**

1. Select the **Amount** property and click the ellipsis button to open the **Bind Property** dialog box.
2. Under the **Bind to an existing member** tab, select **checkAmount** from the list of options and click **OK**.
3. The **receiveCheckRequest** properties should now look like this.

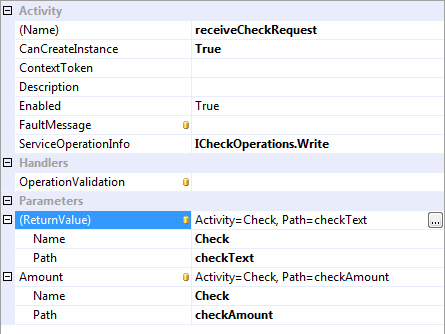
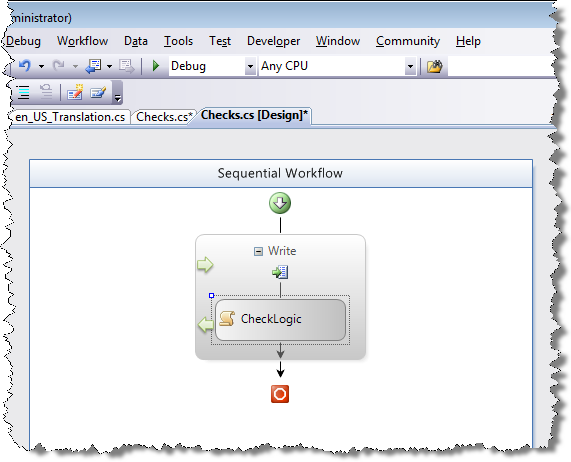


Figure : Configured receiveCheckRequest Properties

#### Finishing the workflow

1. In the Workflow designer, drag and drop the existing **CheckLogic** code activity on to the **Drop Activities Here** drop location in the center of the **receiveCheckRequest** activity. displays the results.

****

**Figure 17: The completed move operation of the CheckLogic activity**

1. Select **File | Save All**.
2. Select **Build | Build Solution** to ensure everything is correct.
3. Select **Window | Close All Documents**.

## Task 3 – Hosting the workflow

Now that the workflow exists, it must be hosted in some way, allowing it to be called and invoked from an outside process.

#### Adding a host for the workflow

1. In **Solution Explorer**, delete the *Program.cs* code file.
2. **Right-click** the **PaymentWF** project in **Solution Explorer**.
3. Select **Add | Existing Item**
4. Select *Program.cs* from the following folder:  
   *C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex1*
5. Open the new *Program.cs* file and inspect its contents.

|  |
| --- |
| Hosting a workflow as a service **WorkflowServiceHost** is a new hosting class that brings together the **WorkflowRuntime** and **ServiceHost** classes. You pass this class a type object representing the workflow service you want exposed. The class does the work of creating the endpoint and handing requests from service clients.  Once the **WorkflowServiceHost** is initialized, the **Open** method starts the listener. In this exercise, you’ll just use a Console application to host the workflow and service. In a production environment, you would likely use a richer host such as **IIS** or your own Windows Service (a daemon). |

#### Configuring the Service Host

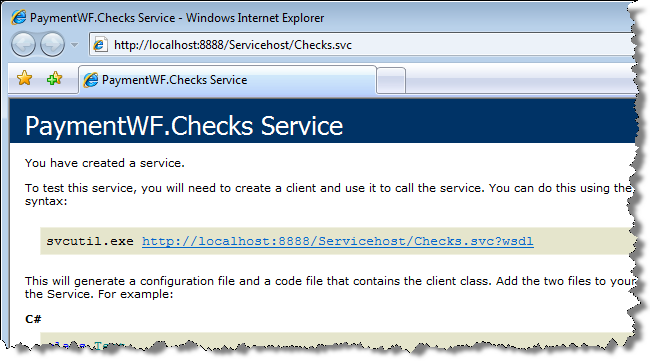
1. **Right-click** the **PaymentWF** project in **Solution Explorer**.
2. Select **Add | Existing Item**
3. Select *App.config* from the following folder:  
   *C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex1*

|  |
| --- |
| Service Host Configuration This configuration file defines the properties of the WCF service such as the service name (**PaymentWF.Checks**), what address clients can access the service at (**baseAddress**), what operations are supported (**contract**), and that the service supports exposing its metadata via **HTTP GET** commands. |

1. Select **File | Save All** to save your work.
2. Perform a build (select **Build | Build Solution** or **CTRL+Shift+B**)
3. Select **Window | Close All Documents**.

#### Running the service in the debugger

1. Press **F5** to start the service. Once you see the message *Service is listening* in the console window, start **Internet Explorer**.
2. Navigate to <http://localhost:8888/Servicehost/Checks.svc> . If a dialog entitled **Information Bar** comes up, check the box **Don’t show this message again** and click **Close**. Then click on the yellow information bar and select **Enable Intranet Settings**. Select **Yes** to the confirmation dialog that appears.
3. You should see a window similar to . This test shows you that your service is running and able to process requests. If you want, you can click on the first link to review the WSDL for the service.

****

**Figure 18: Using Internet Explorer to access the new service**

1. Close Internet Explorer and the console hosting the service and return to Visual Studio.

## Task 4 – Adding a client to the solution

Now that the workflow is running in a host, you need a client to invoke it and demonstrate the workflow actually works as expected. This task uses a simple Windows Forms application to call into the service and workflow.

#### Adding a Windows Forms application that calls the workflow service

1. Select **File | Add | New Project**.
2. In the **New Project** dialog, under the **Project types** section, select and expand **Visual Basic**, then select **Windows**.
3. In the **Templates** section, select **Windows Forms Application**.
4. Set the **Name** property to **CheckWriter**.
5. Under **Solution**, select **Add to Solution**.
6. The New Project dialog should look similar to this:

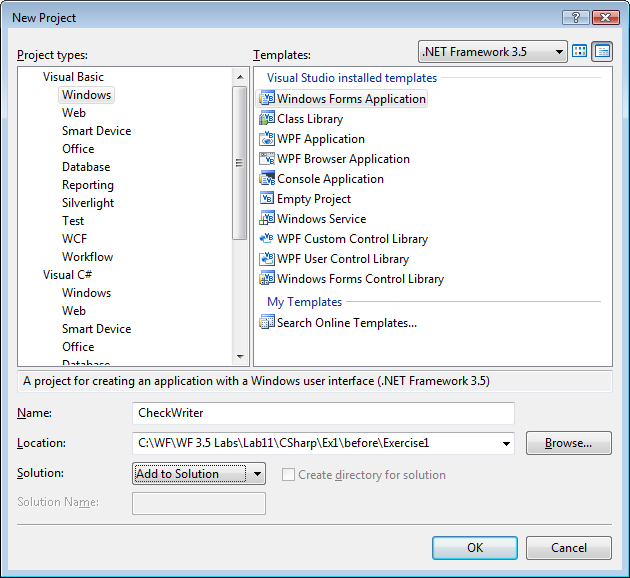


Figure : Adding a new Window Forms project to the solution

1. Click **OK**.

#### Designing the Windows Form

1. From the **Common Controls** section of the **Toolbox**, add **TextBox**, **Label**, and **Button** controls to the form.
2. Lay out the form primitively like this:

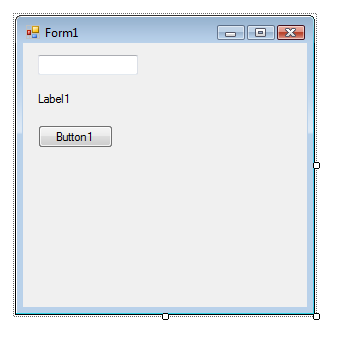
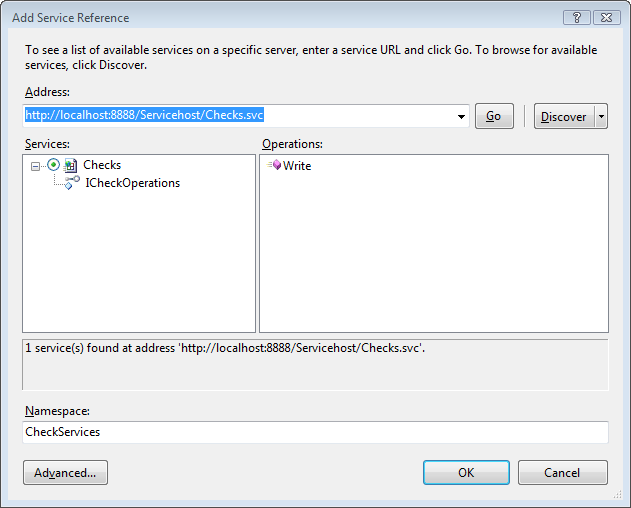


Figure : Simple Windows Form

#### Using the Add Service feature of Visual Studio

|  |
| --- |
| Add Service Feature The new Add Service feature of Visual Studio gets information about a service, making it very easy to call service methods from the Windows Forms application. |

1. Press **Ctrl+F5** to start the service without the Visual Studio debugger. Once the service is started, leave it running and return to Visual Studio.
2. In the **Solution Explorer**, right-click on the **CheckWriter** project and select **Add Service Reference**.
3. In the **Add Service Reference** dialog box (), type <http://localhost:8888/Servicehost/Checks.svc> in the address field.
4. Click **Go**.
5. Change the **Reference namespace** to **CheckServices**.

****

**Figure 21: The Add Service Reference dialog**

1. Click **OK**
2. Return to the console window and close the service.

#### Using the service in the Windows From

1. Back in Visual Studio, select the form and then double-click on the form’s title bar to open the code editor. Visual Studio adds the event handler stub for the form’s Load event.
2. Add the following code inside the form’s class:

|  |
| --- |
| **Public Class Form1**  Private svc As CheckServices.CheckOperationsClient  **Private Sub Form1\_Load(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles MyBase.Load**  svc = New CheckServices.CheckOperationsClient  svc.Open()  **End Sub**  **End Class** |

1. Add an event handler for the form’s **FormClosed** event. To add an event handler, click the **Events** combo-box (the one on the top right) above the code editor as seen below in Figure 22.

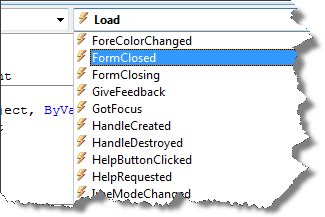


Figure : Adding a FormClosed event handler

1. Add the following code to the **FormClosed** event handler:

|  |
| --- |
| **Private Sub Form1\_FormClosed(ByVal sender As Object, \_**  **ByVal e As System.Windows.Forms.FormClosedEventArgs) \_**  **Handles Me.FormClosed**  svc.Close()  **End Sub** |

#### Creating an event handler for Button1’s click event.

1. Expand the Object combo-box (the one on the left above the code editor) — it currently reads (**Form1 Events**)
2. Select the **Button1** object
3. From the Events combo-box select **Click**



Figure : Selecting the Click handler for Button1

1. The event handler is created automatically.
2. Finally, add the following code to the **Click event handler** for **Button1**

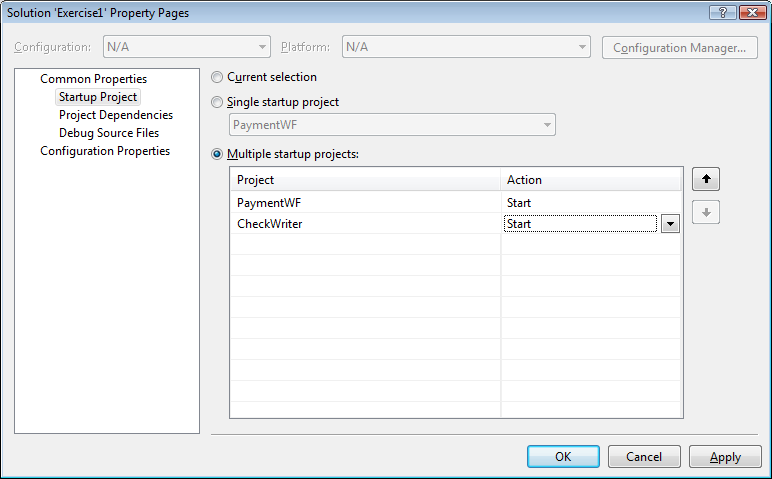
|  |
| --- |
| **Private Sub Button1\_Click(ByVal sender As Object, ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  Label1.Text = svc.Write(CDec(TextBox1.Text))  **End Sub** |

1. Select **File | Save All**.
2. Select **Build | Build Solution** to ensure everything is correct.
3. Select **Window | Close All Documents**.

#### Configuring the solution to run in the debugger

Now there are 2 projects in the solution, both of which need to run. Visual Studio may be configured to launch both projects under the debugger.

1. In the **Solution Explorer**, right-click on the **Solution 'Exercise1' (2 projects)** node and select **Properties**.
2. With the **Startup Project** node selected, choose the **Multiple startup projects** option.
3. Move the **PaymentWF** project to the top of the list by clicking the up arrow to the right of the project list. This ensures the service is fully launched before the WinForm application is started.
4. Change the **Action** property of both projects to **Start** (see ) and click **OK**.

****

**Figure 24: The adjusted Startup Project for the solution**

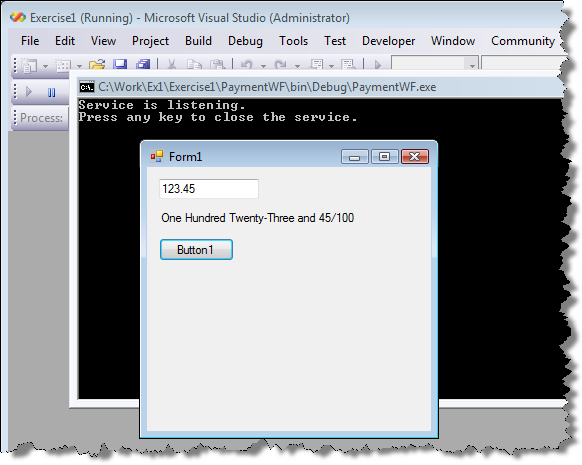
1. Select **File | Save All** to save your work.

#### Running the complete solution

1. Press **F5** to start both projects.
2. In the Windows Forms application, enter **123.45** and click Button1.

|  |
| --- |
| A note on what is happening By clicking the button, you had the Windows Forms code call the exposed Write method of the WCF service you created earlier. The WCF service invokes the workflow which translates the input from a decimal value to a string suitable for printing on a check.  The workflow created earlier is very simple. However, you can easily see how the marriage of WCF and Windows Workflow provides a world of possibilities. |

1. displays the expected results.

****

**Figure 25: The completed solution running from the debugger**

1. Close the form and the console application.
2. Close Visual Studio.

## Exercise Summary

In this exercise, you created a WF workflow and exposed it as a WCF service. Then, you created a client application that accessed that service and ran the workflow.

# Exercise 2 - WCF and Syndication

Another new feature introduced in the .NET Framework 3.5 is the ability to easily generate syndication content such as RSS and ATOM feeds. In this exercise, you’ll build a simple WCF service that renders data about the current system via both an RSS and ATOM feed. In addition to providing text data, you expose application specific data as XML content in the syndication feed. You’ll then build a client to consume the feed data.

## Task 1 - Create a syndication service

#### Creating the basic service project

1. Start Visual Studio by selecting **Start |** **All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008**.
2. In Visual Studio, select **File | New Project**.
3. In the **New Project** dialog, select and expand **Visual Basic** under the **Project types** section, then select **Windows** | **Console Application**.
4. Fill out the **New Project** dialog like this:

|  |  |
| --- | --- |
| **Property Name** | **Value** |
| Name | SyndicationServer |
| Location | C:\WF\WF 3.5 Labs\Lab11\CSharp\Ex2\before |
| Solution Name | Syndication |

1. The **New Project** dialog should look like this.

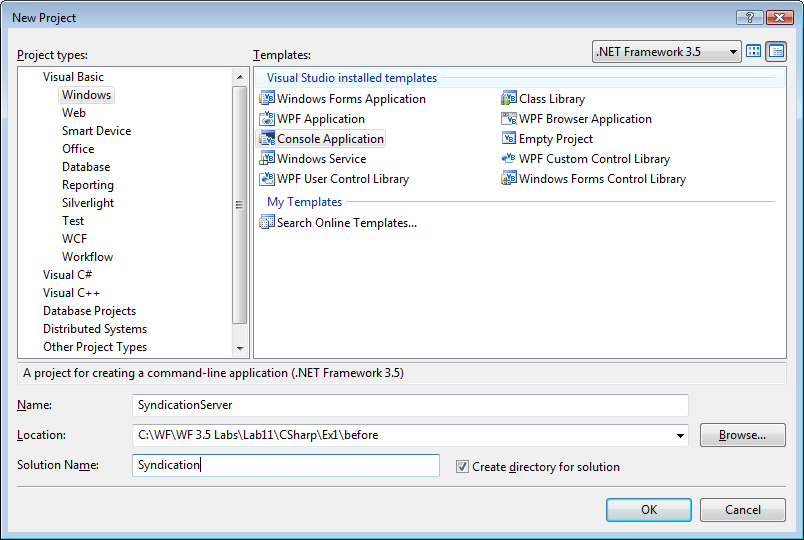


Figure : New Project dialog

1. Click **OK**.

#### Configuring the basic project

1. In the **Solution Explorer**, right-click on **Module1.vb** and select the **Rename** command. Change the name to **Program.vb**. Click **Yes** to the rename dialog that Visual Studio presents.
2. In the **Solution Explorer**, right-click on the **SyndicationServer** project node and select **Add Reference.**
3. With the **.NET** tab selected, using the Ctrl key, select these dependencies as shown in Figure 27.
   1. S**ystem.Runtime.Serialization**
   2. **System.ServiceModel**
   3. **System.ServiceModel.Web**

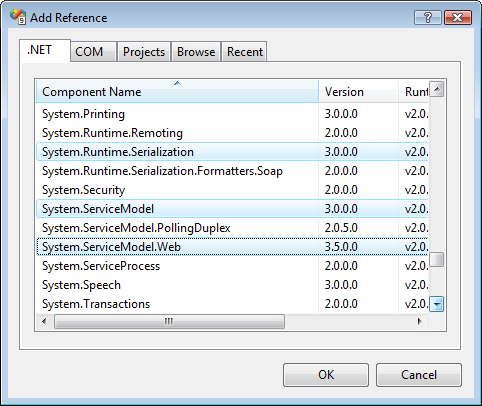


Figure : Adding project references

1. Click **OK**.

#### Adding the SystemInfo class

1. **Right-click** the **SyndicationServer** project in **Solution Explorer**.
2. Select **Add | Existing Item**
3. Select SystemInfo.vb from the following folder:

C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex2

1. Open the class file and familiarize yourself with it.

|  |
| --- |
| **SystemInfo class**  The project needs a class to be used as the data source for the syndication server’s feed. This class exposes some properties that will expose data to the feed, and some functions enabling easy reading of the data members by a person. |

#### Adding the service definition and implementation class

1. **Right-click** the **SyndicationServer** project in **Solution Explorer**.
2. Select **Add | Existing Item**
3. Select IPublishSystemInfo.vb and PublishSystemInfo.vb from the following folder:

C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex2

1. Click **OK**
2. Open the **IPublishSystemInfo** interface and become familiar with it.

|  |
| --- |
| IPublishSystemInfo The **IPublishSystemInfo** interface defines contract for the WCF syndication service. The interface specifies that the service will have a single method, GetSystemInfo. This method expects a string as an argument and returns a SyndicationFeedFormatter object. |

1. Open the **PublishSystemInfo** class and become familiar with it.

|  |
| --- |
| PublishSystemInfo The **PublishSystemInfo** class first creates two objects: si and feed.   * The **SystemInfo** class represents your data source. You could instead use another WCF service or database as your source: It doesn’t really matter. * **SyndicationFeed** is a new class in the .NET Framework 3.5 that handles the low-level work of creating a correctly formed syndication feed.   Once the objects are initialized, the code defines some basic feed setting such as the feed title. This is done with the **SyndicationContent.CreatePlaintextContent** method. Other methods exist to expose richer text content such as HTML and XML. With the feed initialized, the code needs to create **SyndicationItems**.  In this case, the code only creates a single item—similar to a single blog post. You could create many items at one time. It’s up to you.  You’ll note the code exposes three types of data: plain text, HTML, and XML. The XML content represents “application specific” data embedded in to the syndication feed. This allows the feed to be used by more than just humans reading the text. You can enable application behaviors based upon the custom content. Note, you can also send down non-text data.  Once the syndication item is created, the code adds it to the feed and then returns the feed object to the caller. |

1. Select **File | Save All** to save your work.
2. Select **Build | Build Solution** to ensure everything is correct.
3. Select **Window | Close All Documents** to tidy up the IDE.

#### Hosting the service

The service and data it will provide are now defined. The service must now be hosted so it may be called.

1. Delete the current file: *Project.vb* (**right-click | Delete** in **Solution Explorer**).
2. **Right-click** the **SyndicationServer** project in **Solution Explorer**.
3. Select **Add | Existing Item**
4. Select Project.vb from the following folder:

C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex2

1. Click **OK**
2. Open the code file to become familiar with it.

|  |
| --- |
| Program.vb This class initializes the WCF service via the **ServiceHost** class. It then creates two endpoints: one that returns feeds using the **ATOM** syndication format and a second that returns **RSS**.  The data content of the feed is specified via the call to **AddServiceEndpoint**. Once the endpoints are defined, the service is “started” by calling the **Open** command.  The rest of the code is only necessary for this simple demo using a Console application. Production-worthy code would host the service via **IIS** or a custom Windows Service. |

1. Select **Build | Build Solution** to ensure everything is correct.

#### Running the service in the Visual Studio debugger

1. Press **F5** to start the program.
2. Start **Internet Explorer** and navigate to either <http://localhost:8899/sysinfo/atom/> or <http://localhost:8899/sysinfo/rss/> (note you need to include the trailing / ).

|  |
| --- |
| About the service Internet Explorer 7 understands how to parse and render both ATOM and RSS feeds in human- friendly format. Can you see where the data on the screen comes from relative to the code discussed earlier? What do you see if right-click and select **View | Source**? Do you see information that is not being rendered on screen (for example the custom XML data)? |

Once you’re done examining the feeds, close Internet Explorer, exit the service by pressing **Enter** in the console window, and return to Visual Studio.

1. Select **Window | Close All Documents** to tidy things up.

## Task 2 – Add a service client application

Now that a fully function service exists, create a client application that talks to the service.

#### Creating the application project

1. Select **File | Add | New Project**.
2. In the **New Project** dialog, select and expand **Visual Basic** under the **Project types** section, then select **Windows**.
3. Select **Windows Forms Application** in the **Templates** section.
4. Set the **Name** property to **SyndicationClient**.
5. The **New Project** dialog should look like this:

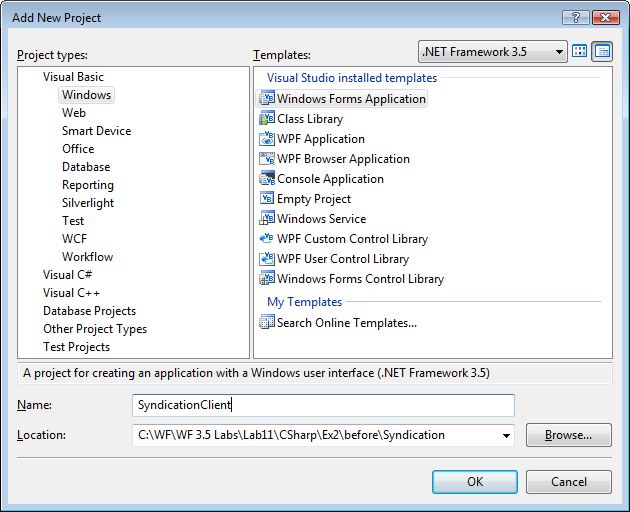


Figure : New project dialog

1. Click **OK**.
2. In the **Solution Explorer**, right-click on the **SyndicationClient** project node and select **Add Reference**.
3. With the **.NET** tab selected, select these dependencies and click **OK**.
   1. **System.ServiceModel**
   2. **System.ServiceModel.Web**
4. Right-click again on the **SyndicationClient** project node and select **Add Reference**.
5. Select the **Projects** tab. The **SyndicationServer** project will already be selected. Click **OK** to add the reference.
6. From the **Common Controls** section of the **Toolbox**, add the following controls to the form:
   1. A **ListBox** control
   2. A **WebBrowser** control
   3. Two **Button** controls
7. Change the **Text** property of the button controls to **XML** and **RSS** respectively.
8. Adjust the **Anchor** property of the **WebBrowser** control to **Top, Bottom, Left, Right**.
9. Once you’re done, your form should resemble the one in .

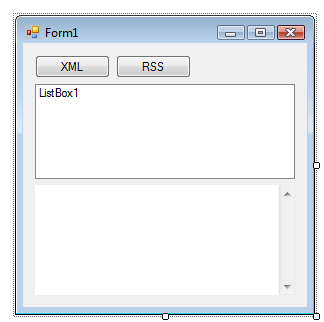


Figure : Basic form layout

## Task 3 – Accessing the ATOM feed

1. **Double-click** on the button labeled **XML** to create a Click event handler.
2. At the top of the form’s class file, add the following Imports statements:

|  |
| --- |
| **Snippet:** WFLab11\_Ex02\_Task02\_Members |
| Imports SyndicationServer  Imports System.ServiceModel.Syndication  Imports System.Xml  **Public Class Form1 ...** |

1. Add the following code to the **Click** event handler for **Button1**:

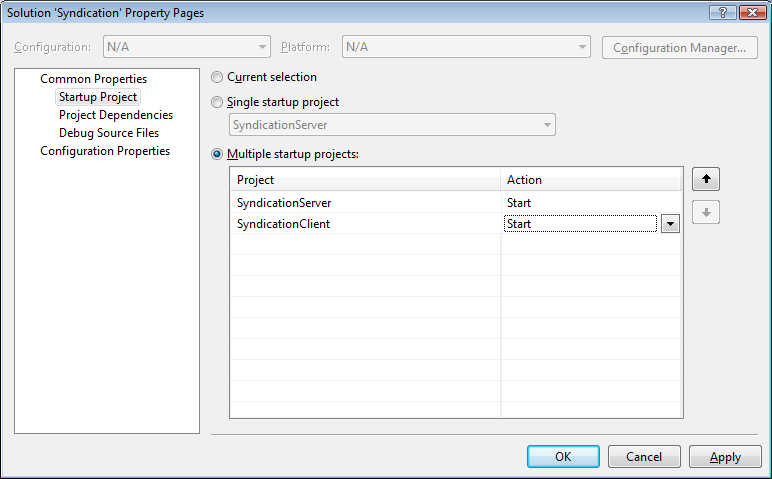
|  |
| --- |
| **Snippet:** WFLab11\_Ex02\_Task02\_Button1 |
| **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  Try  Dim reader As XmlReader  reader = XmlReader.Create("http://localhost:8899/sysinfo/atom/")  Dim feed As SyndicationFeed  feed = SyndicationFeed.Load(reader)  Dim c As XmlSyndicationContent = Nothing  Dim si As SystemInfo = Nothing  For Each i As SyndicationItem In feed.Items  c = CType(i.Content, XmlSyndicationContent)  si = c.ReadContent(Of SystemInfo)()  ListBox1.Items.Add(i.Title.Text)  ListBox1.Items.Add(si.ToString())  ListBox1.Items.Add("\* " & si.ProcessorCount)  ListBox1.Items.Add("\* " & si.OSVersion)  ListBox1.Items.Add("\* " & si.Version)  MsgBox(si.ToString(), MsgBoxStyle.Information, \_  si.MachineName)  Next  Catch ex As Exception  MsgBox(ex.Message, MsgBoxStyle.Exclamation, ex.Source)  End Try  **End Sub** |

|  |
| --- |
| This code uses the new **SyndicationFeed** class to read the syndication feed provided by the WCF service you wrote earlier and parse the custom XML content. Once the feed is loaded, the code uses a **For Each** look to read each item (your sample only creates a single item, but more could be created). Once it has a valid item, it parses the custom XML data from the feed item and then uses the **ReadContent** method to re-hydrate the custom **SystemInfo** type. |

1. Select **File | Save All**.
2. Select **Build | Build Solution** to ensure everything is correct.

#### Preparing the solution to be run by the Visual Studio debugger

1. In the **Solution Explorer**, right-click on the **Solution 'Syndication' (2 projects)** node and select **Properties**.
2. With the **Startup Project** node selected, choose the **Multiple Startup Projects** option.
3. Move the **SyndicationServer** project to the top of the list by clicking the up arrow to the right of the project list.
4. Change the **Action** property of both projects to **Start** (see Figure 15) and click **OK**.

****

**Figure 30: The adjusted Startup Project for the Syndication solution**

1. Select **File | Save All** to save your work.
2. Press **F5** to start the projects.
3. Once the programs have loaded, click the button labeled **XML** on the Windows form.
4. View the results. Notice how the application was able to read access the feed, access the custom XML content and create an object and display its contents in the **ListBox** control and via a **MessageBox**.
5. Close both programs and return to Visual Studio.

## Task 3 – Accessing the RSS feed

1. Add a Click event handler for Button2 (the one labeled RSS).
2. Add the following code to its handler:

|  |
| --- |
| **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  WebBrowser1.Navigate("http://localhost:8899/sysinfo/rss/")  **End Sub** |

1. Select **File | Save All** to save your work.
2. Press **F5** to start the projects.
3. Once the programs have loaded, click the button labeled **RSS** on the Windows forms.
4. If necessary, resize the form so you can see the results in the **WebBrowser** control. Note, how the web browser control simply shows the feed contents as nicely formatted XML as opposed to the view you saw in Internet Explorer 7.0.
5. When done, close the both programs and return to Visual Studio.
6. Back in Visual Studio, select **File | Close Solution**.
7. Close Visual Studio

## Exercise Summary

In this exercise, you created a syndication server using WCF’s new syndication features and exposed plain text, formatted HTML, and custom XML content: all from the same feed. You then created a client that was able to consume the provided data.

# Exercise 3 - WCF Services and AJAX

In this exercise, you’ll create a simple WCF service that can be accessed via AJAX. Using a bit of script in the browser, you’ll translate a number sequence as the user types into the words representing the numbers as if they were writing a check.

## Task 1 – Creating a web site project

#### Creating an empty web site project

1. Start Visual Studio by selecting **Start |** **All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008**.
2. Once Visual Studio is running, select **File | New | Web Site**.
3. Select **Empty Web Site** as the Template
4. Set the dialog options as follows:

|  |  |
| --- | --- |
| Setting | Value |
| **Location** | HTTP |
| Location URL | <http://localhost/hol> |
| Language | Visual C# |

1. The New Web Site dialog should look like Figure 31 below.

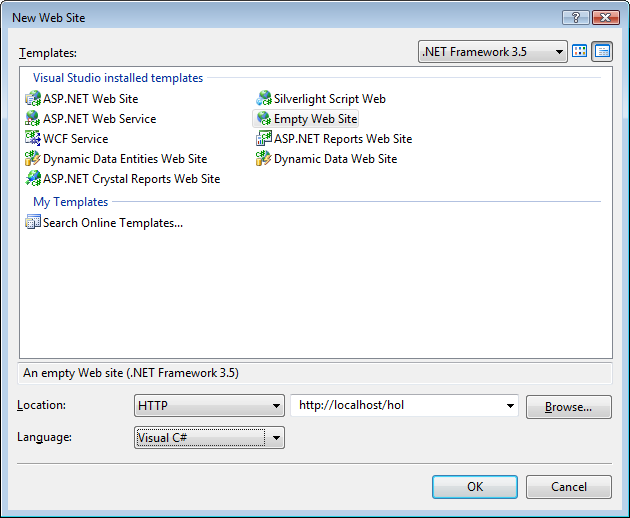


Figure : New Web Site dialog

1. Click **OK**

## Task 2 – Adding the WCF Service

This task adds a WCF service as a part of the web site.

#### Adding the CheckWriter service contract

1. Select **Web Site | Add New Item**.
2. In the **Add New Item** dialog, select **Class** as the template, change the **Name** to **CheckWriter** and click **Add**.
3. When thewarning dialog appears about placing code in the **App\_Code** folder, click **Yes** (thiswill automatically create the **App\_Code** folder for you).
4. Replace the contents of the file with the following code that defines the interface that the WCF service will expose.

|  |
| --- |
| **Snippet:** WFLab11\_Ex03\_Task1\_ICheckWriter |
| **using System;**  **using System.Runtime.Serialization;**  **using System.ServiceModel;**  **using System.ServiceModel.Activation;**  **using System.ServiceModel.Web;**  **namespace Microsoft.Hol.Samples**  **{**  **[ServiceContract(Namespace = "")]**  **public interface ICheckWriter**  **{**  **[OperationContract]**  **[WebGet()]**  **string Write(decimal checkAmount);**  **}**  **}** |

#### Adding the CheckWriter service implementation

1. Now add the implementation class inside the same code file (make sure it is inserted inside the namespace braces):

|  |
| --- |
| **Snippet:** WFLab11\_Ex03\_Task1\_CheckWriter |
| **public class CheckWriter : ICheckWriter**  **{**  **...**  **}** |

This is the same code used earlier that translates a decimal value into a string usable for printing checks.

#### Adding the CheckWriter ServiceHost file

1. Select **WebSite | Add New Item**.
2. In the **Add New Item** dialog, select **Text File** and set the **Name** to **CheckWriter.svc** and click **Add**. Make sure the file goes in the root of the project (not the App\_Code folder). If it’s in the wrong place, drag and drop it into the root.
3. Add the following line to **CheckWriter.svc** (note this is one line of code with no line breaks):

|  |
| --- |
| **<%@ ServiceHost Language="C#" Debug="true"**  **Service="Microsoft.Hol.Samples.CheckWriter"**  **Factory="System.ServiceModel.Activation.WebScriptServiceHostFactory" %>** |

1. Select **File | Save All** to save your work.
2. Select **Window | Close All Documents** to tidy things up.

#### Adding an HTML page that talks to the service

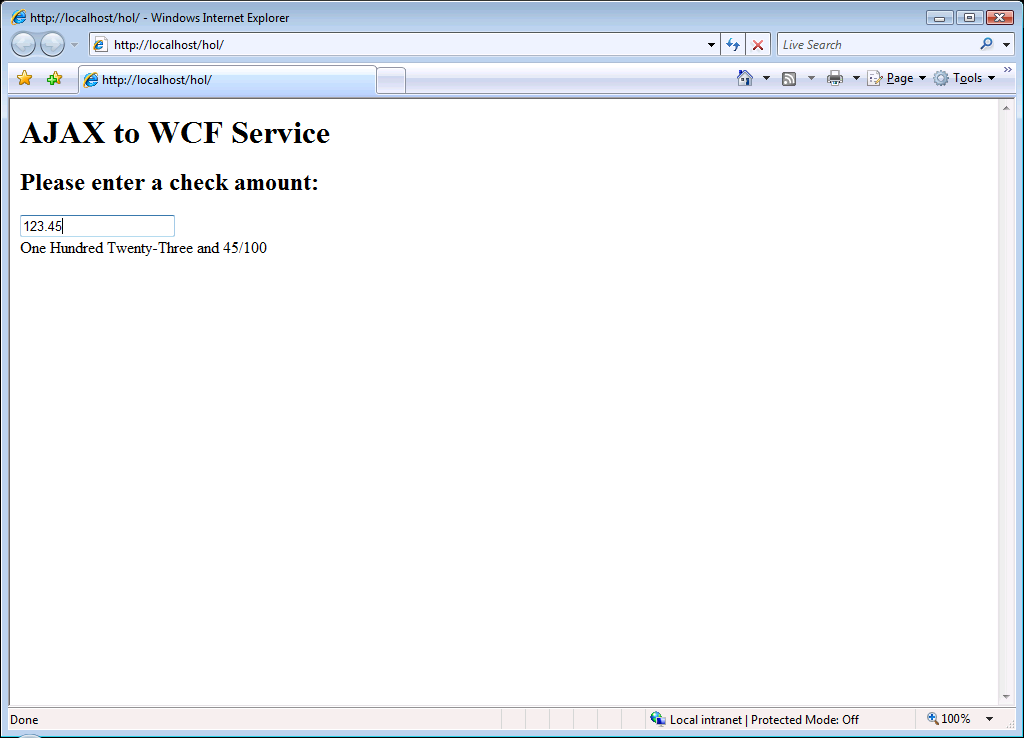
1. Select **Web Site | Add Existing Item**.
2. Select default.htm from the following folder:

C:\WF\WF 3.5 Labs\Lab11\resources\Code Files\Ex3

1. Click **OK**
2. Open the HTML file to become familiar with it.

|  |
| --- |
| Default.htm This HTML file has three major sections.   * The first part of the file defines the basic structure of the file. * The next section defines a form called **dataEntry** that has a input control and a div tag. The **onkeyup** event is bound to a JavaScript function **convertToWords()**. * The third section is a script block that processes the **onkeyup** event and makes the AJAX call to the WCF service suing the XMLHTTP object.   Note the data entered by the user is send via a standard HTTP GET command. The return value is returned and placed in the innerHTML of the div tag defined earlier. |

1. Select **File | Save All** to save your work.
2. Select **Window | Close All Documents** to tidy things up.
3. Select **Build | Build Solution**.
4. Start Internet Explorer.
5. Navigate to <http://localhost/hol/>. If an **Information Bar** warning dialog, appears, click **Close**. Then click the yellow **Information Bar** in Internet Explorer and select **Enabled Intranet Settings** from the menu and select **Yes** from the confirmation dialog box.
6. Type 123.45 in the text box and notice how, without a post back, the amount is displayed below the text box (see **Figure 24**). Note that it might take a few seconds after you first enter a number for the service to load and respond.

****

**Figure 32: The AJAX-enabled web page returning results from a WCF service**

1. Close Internet Explorer and Visual Studio. You’re all done.

## Exercise Summary

In this exercise, you created a simple WCF service that exposed its features in such a way that it could be consumed using simple AJAX code in the browser.

# Lab Summary

This lab introduced several new features available in .NET 3.5 for WF workflows, focusing on WES (Workflow enabled Services).

Exposing WF workflows as WCF services is a straightforward process supported by the Send and Receive messaging activities, both of which are new for 3.5. A service enabled workflow can be hosted by IIS, a Windows service, or other custom service hosts.