|  |
| --- |
|  |
| Procedure Manual |
|  |
| Portable ASP.NET Web Service Sandbox |



**ITDI373 - 1402B - 01: LEGAL AND TECHNICAL REPORT WRITING AND PRESENTATION**

May 25, 2014

Authored by: Adam Baker

This Page is Intentionally Left Blank

Table of Contents

[Project Outline 5](#_Toc388525574)

[Abstract 6](#_Toc388525575)

[Outline 7](#_Toc388525576)

[Proposal 8](#_Toc388525577)

[Procedure Manual Brochure 9](#_Toc388525578)

[Brochure Front Matter 9](#_Toc388525579)

[Brochure Back Matter 10](#_Toc388525580)

[Preparation Plan 11](#_Toc388525581)

[Preparation Checklist 11](#_Toc388525582)

[Obtaining the .Net Framework 11](#_Toc388525583)

[Obtaining SQL Server Compact 4.0 12](#_Toc388525584)

[Recommended System Requirements 12](#_Toc388525585)

[Procedure Manual 13](#_Toc388525586)

[Introduction 13](#_Toc388525587)

[The Purpose of this Manual 13](#_Toc388525588)

[Who Should Read This Manual 13](#_Toc388525589)

[How This Manual Is Organized 13](#_Toc388525590)

[Chapter Overview 14](#_Toc388525591)

[Chapter 1: Creating a Web Project 15](#_Toc388525592)

[The Web Service Project 15](#_Toc388525593)

[Chapter 2: SQL Server Compact 4.0 16](#_Toc388525594)

[Adding the Local Database 16](#_Toc388525595)

[Adding Tables to the Database 17](#_Toc388525596)

[Adding Data to the Database 18](#_Toc388525597)

[Chapter 3: Business Components 19](#_Toc388525598)

[Creating the Class 19](#_Toc388525599)

[Regions and Functionality 20](#_Toc388525600)

[Variables 21](#_Toc388525601)

[Constructors 22](#_Toc388525602)

[Properties 23](#_Toc388525603)

[Private Methods 24](#_Toc388525604)

[Public Methods 25](#_Toc388525605)

[Service Methods 28](#_Toc388525606)

[Testing Your Business Component 29](#_Toc388525607)

[Chapter 4: Web Service 32](#_Toc388525608)

[Adding the Web Service 32](#_Toc388525609)

[Identifying Conventions 33](#_Toc388525610)

[WSDL: A Complete Web Service 34](#_Toc388525611)

[Chapter 5: Implementing the Web Service 35](#_Toc388525612)

[Creating the Web Service Object 35](#_Toc388525613)

[Chapter 6: Points of Interest 37](#_Toc388525614)

[Importance of Learning Web Services 37](#_Toc388525615)

[Taking it a Step Further 37](#_Toc388525616)

[Quick Reference 38](#_Toc388525617)

[Index 41](#_Toc388525618)

[References 42](#_Toc388525619)

**Procedure Manual**

# Project Outline

The following is an outline of the processes required to bring the completed project to fruition.

Week 1

Select the appropriate template and format for developing the procedure manual along with generating a physical outline of each step required in the process.

Week 2

Complete the first draft of the procedures manual and develop a brochure to aid in the promotion of the procedure manual itself.

Week 3

Modify the procedure manual to contain the following elements: a suitable logo, table of contents, Introduction that include a chapter overview, index page(s), and a reference page(s). Amend the procedure manual to include a procedure manual preparation plan with the intended audience in mind.

Week 4

Update the procedures manual based on peer review, and add an Abstract and Procedures manual quick reference section with appropriate content.

Week 5

Produce multimedia training material to be used as a companion within the procedures manual. Submit the final deliverable containing all designed and developed materials during the course of this project.

**Procedure Manual**

# Abstract

As the demand for Web Service experience from software developers in the IT field climbs, more current and future students are beginning to search for academic institutions capable of providing such skills in their curriculum. This procedure manual was developed to remedy this issue by providing the means of creating a Web Service learning resource for undergraduate students with the intention of preparing them with the skills they need to succeed in a future software development career. The procedure manual is expected to reduce the relatively high learning curve associated with creating Web Services by explaining how to create a portable Web Service Sandbox using Microsoft’s ASP.NET, C#, and SQL Server Compact Edition 4.0. The manual covers creating Web projects, a local database, tables and data, business components, and Web services. This manual will give undergraduate software development students the general Web Service experience required of them by future employers with providing the academic institution the curriculum resource it needs to retain current and obtain future student prospects.

**Procedure Manual**

# Outline

**Introduction**:

* Identify the purpose of this manual.
* Identify the intended audience for the manual and identify reader expectation.
* Identify How the manual is organized

**System requirements:**

* Identify the system and software requirements needed to follow the process manual as intended.

**Chapter 1: The Web Project**

* Describe the process of creating a Web project required for the process.

**Chapter 2: SQL Server Compact 4.0**

* Describe the process for adding a SQL Server Compact 4.0 database to a Web project.
* Describe how Add Tables and Data within Visual Studio 2010.

**Chapter 3: Business Components**

* Describe how to tie the Web project and Database together via a custom connection string.
* Describe how to create business components for accessing data within the Database.
* Describe how to test business components via presentation layer.

**Chapter 4: Web Service**

* Describe how to add a Web service to the Web project.
* Describe how to make business component method available via Web service.

**Chapter 5: Accessing Web Service**

* Describe how to access methods of Web services within Web Presentation Layer

**Chapter 6: Points of Interest**

* Identify commonly used Web Serveries.
* Make suggestions for applying learned material to practical applications.

Procedure Manual

# Proposal

**Executive Summary:**

Web services are becoming the de facto method for providing interoperability between enterprise applications. Amazon, Salesforce, and Dropbox APIs provide some of the most widely used Web services for Web applications. As more Web service become in-demand, current and future academic students will be expected to be familiar with such technologies. In order to prepare students with knowledge and skills they will need, a process manual for developing and working with Web services will need to be developed.

**Statement of Problem:**

Aware of the fact organizations are adapting to incorporate Web services over legacy communication methods, there is a driving need for software developers experienced in implementing and testing Web services.

**Objectives:**

The main objective of this project is to create a process manual and associated material for developing an ASP.NET Web Service Testing Environment.

It will be expected that students will be capable of creating a portable development environment using the most common industry standard technologies. The process manual will touch on all aspects of system architecture needed in order for Web services to be experimented with. The result will provide a broad educational software development approach by walking students though the following steps.

* Visual Studio Web Project Creation
* ASP.NET Web Application Development: Implementing Form Elements
* SQL Server Compact 4.0 Development and Implementation: Adding Tables and Data
* Web Application Data Binding: Using Data Views to display data
* Web Service Development and Implementation: Using Methods from Business Components to Access Database Information

**Technical Approach:**

The Process Manual project will be carried out over a five week span beginning in the second quarter of 2014. The manual will be developed based on the actual implementation derived from an individual with experience in developing Web services with the selected technologies. Each process of the manual will be documented with code and/or code snippets and illustrations where appropriate. The manual will be created in a digital format for easy distribution and printing purposes.

**Procedure Manual**

# Procedure Manual Brochure

## Brochure Front Matter



## Brochure Back Matter

****

**Procedure Manual**

# Preparation Plan

As this procedure manual is focused on software development, there are several steps that will be required prior to following along with the steps herein. The processes described within this manual have been created within a specific software development environment. Therefore it is recommended that the same environment is duplicated as accurately as possible in order to prevent issues from arising. The following is a checklist to ensure that you have the required resources and/or tools needed to follow along with each step in the procedure manual.

## Preparation Checklist

* Download and install Microsoft .NET framework 4.0
* Download and install Microsoft SQL Server Compact 4.0
* Download and install the latest Operating System updates and patches
* Download and install Microsoft Visual Studio 2010 Professional/Express
* Verify development environment meets or exceeds minimum system requirements

## Obtaining the .Net Framework

The .Net framework version required to complete the processes in this procedure manual are preinstalled in most modern Windows systems (Windows 7 +). However, if you are using an operating system prior to Windows 7 it is suggested that you download the .NET framework from the following links.

.NET version 3.5 - http://www.microsoft.com/en-us/download/details.aspx?id=21[[1]](#footnote-1)

.NET version 4.0 - http://www.microsoft.com/en-us/download/details.aspx?id=17851[[2]](#footnote-2)

If you are looking to acquire Visual Studio 2010 and the .NET framework in a single download it is recommended that you install Microsoft’s development training kit at the following link.

VS 2010 & .NET 4 - http://www.microsoft.com/en-us/download/details.aspx?id=23507

## Obtaining SQL Server Compact 4.0

SQL Server Compact comes preinstalled in select versions of Microsoft’s Visual Studio. Unfortunately, the local database is not a default component of Visual Studio 2010. Therefore, you will have to visit the following link in order to download and use SQL Server Compact as described within this manual. Use the following link to acquire SQL Server Compact Edition version 4.0.

SQL Server CE 4.0 - http://www.microsoft.com/en-us/download/details.aspx?id=17876 2

## Recommended System Requirements

In order to work though the processes in this manual there are several pieces of software that will be needed. First, Visual Studio 2010 will be required for all activities within this manual. The .NET framework will be required in addition to SQL Server Compact 4.0. The following identifies the system requirements for installing the necessary software.

|  |  |  |
| --- | --- | --- |
| **Visual Studio 2010 System Requirements [[3]](#footnote-3)** | | |
| **Operating System** | **Processor**  **32bit 64bit** | |
| Windows 7 | Yes | Yes |
| Windows Vista | Yes | Yes |
| Windows Server 2008 SP2 | Yes | Yes |
| Windows Server 2008 R2 | N/A | Yes |
| Windows 8 & 8.1 | Yes | Yes |
| **Frameworks** | .Net Framework 3.5+ & Silverlight 4 | |
| **Disk Space** | 11.5 MB | |

|  |  |  |
| --- | --- | --- |
| **SQL Server Compact 4.0 System Requirements [[4]](#footnote-4)** | | |
| **Opiating System** | **Processor**  **32bit 64bit** | |
| Windows XP Home SP1 & Professional SP3 | Yes | N/A |
| Windows Server 2003 SP2 & R2 | Yes | Yes |
| Windows Vista SP2 | Yes | N/A |
| Windows Server 2008 SP2 & R2 | Yes | Yes |
| Windows 7 | Yes | Yes |
| Widows 8 & 8.1 | Yes | Yes |
| **Disk Space** | 23MB | 37MB |

**Procedure Manual**

# Procedure Manual

## Introduction

### The Purpose of this Manual

The purpose of this manual is to identify the procedures necessary for constructing a portable Web Services development sandbox using ASP.NET, C#, and SQL Server Compact 4.0.The process manual will touch on all aspects of system architecture needed in order for Web services to be experimented with. The result will provide a broad educational software development approach by walking students though the following steps.

* Visual Studio Web Project Creation
* ASP.NET Web Application Development
* SQL Server Compact 4.0 Development and Implementation
* Web Application Data Binding
* Web Service Development and Implementation

It will be expected that readers will be capable of creating a portable development environment–focused on Web Services–using the most common industry standard technologies.

### Who Should Read This Manual

This manual is written specifically for academic students with concentrations software development. It is expected that readers of this manual have a basic understanding of C#, SQL commands, and developing applications with Microsoft Visual Studio 2010. However, it is encouraged that hobbyists, software development professionals, and individuals seeking Web Service development experience in ASP.NET and C# read this manual. For software development individuals with more advanced skills in regard to Web Services, this manual can be used as a desktop reference.

### How This Manual Is Organized

This manual contains many direct examples derived from the author's personal experiences as an academic student and Software Developer. It is our hope that this manual will help you acquire a better understanding of Web Service development and that you become comfortable and encouraged to experiment on your own.

The approach of this manual is to walk the read though each step required in the development of a Web Services Sandbox. The first step the manual will include the creation of an ASP.NET Web application and conclude with Web Services. It is advised that individuals that feel comfortable with Developing Web applications in Visual Studio 2010 jump to specific chapters of interest.

### Chapter Overview

The content of the procedure manual is broken down into small digestible chapters. Each chapter focuses on a specific aspect of the development environment that builds onto the next. Therefore, it is recommended students new to creating applications in ASP.NET and C# follow each chapter from beginning to end. The following is a descriptive breakdown what is expected to be learned within each chapter of the process manual.

Chapter 1: Creating a Web Project

Chapter one lays the foundation for creating the portable ASP.NET Web Services Sandbox development environment. This chapter will cover the exact steps for creating a new Web project in Microsoft Visual Studio 2010.

Chapter 2: SQL Server Compact 4.0

Chapter two will discuss the process for adding a local database to your Web project. Additionally, the process of adding tables and data to the SQL database will be discussed.

Chapter 3: Business Components

Business Components are the information broker to all n-tiered Web applications. In chapter three, several aspects of creating Business Components in the C# programming language will be discussed. The chapter will begin with the process for adding a C# class library. Following the process of adding a class, the chapter will discuss the importance of using regions to effectively manage the layout of written code before leading you on to developing specific element needed for accessing data within the local database. The chapter will finish up with a brief description and process for testing the business component(s).

Chapter 4: Web Service

Chapter four identifies the steps necessary for adding a Web Service to you portable Web project. The chapter will identify the conventions used to modify the SOAP Web service such as applying a specific name, namespace, description, and adding public methods.

Chapter 5: Implementing the Web Service

In this chapter you will learn how to create a Web Service object within you Web application. The process will show you how to manipulate a Web applications presentation layer based on the available processes provided within the created Web service.

Chapter 6: Points of Interest

Within this chapter of the procedure manual, the importance for learning how to create a portable development environment such as this will be addressed. Additional steps beyond the scope of this manual will also be addressed within this chapter as well.

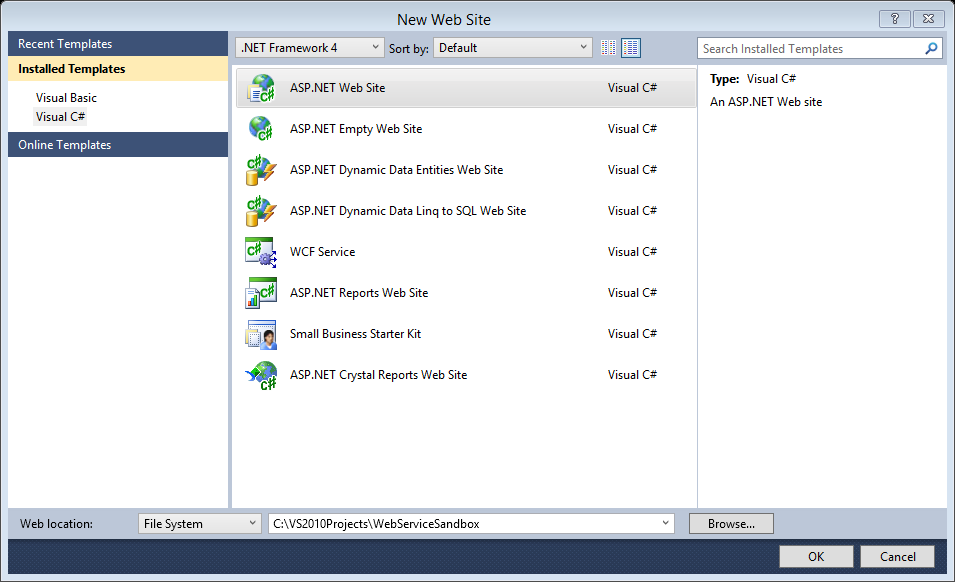
**Procedure Manual**

## Chapter 1: Creating a Web Project

### The Web Service Project

Here we’ll create the Web Services Sandbox application with Visual Studio 2010.

1. Start Visual Studio 2010, and choose **File** -> **New Web Site**. In the dialog that appears, select **ASP.NET Web Site** from the templates panel.
2. For the purpose of this manual we will be using the C# programming language. In the **New Web Site** dialog, be sure to select **Installed Templates** -> **Visual C#.**
3. In the **New Web Site** dialog, select the **File** **System** as the **Web Location** and select a physical location on your disk to store the project.



1. After you have selected the appropriate configuration and identified a location on your disk to save the Web Services Sandbox project, click the **OK** button. Visual Studio will then create a new Web project with pre-generated code including a Master pages, Default pages, Global, and code-behind files.

**Procedure Manual**

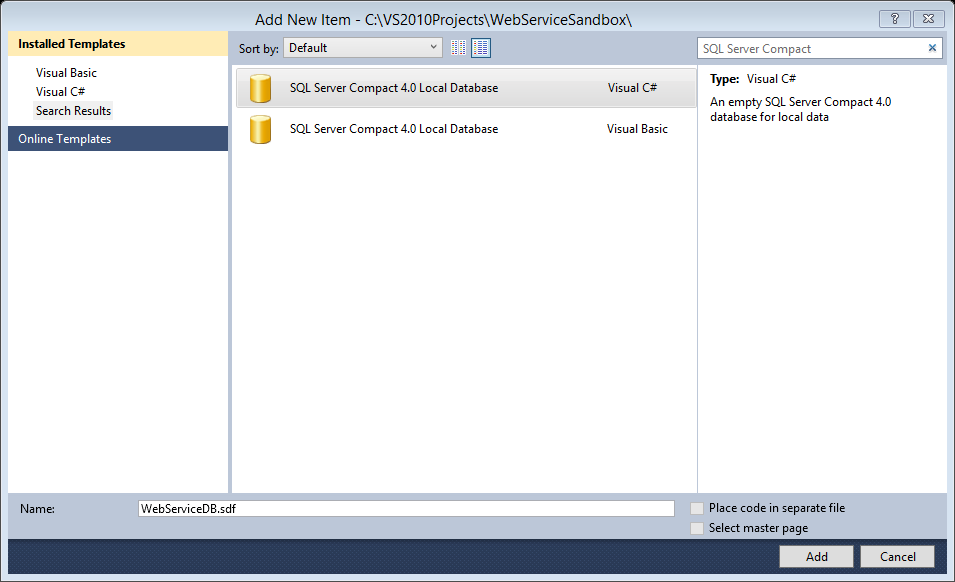
## Chapter 2: SQL Server Compact 4.0

Here we’ll add a SQL Server Compact 4.0 local database to the Web Services Sandbox application. We will cover adding a local database to the project, creating tables, and adding example data.

### Adding the Local Database

Web projects in Visual Studio allow users to store files in dedicated directories. For Databases, files are expected to be stored in the APP\_DATA directory. This is will be the location we will placing our local database. The following steps describe the process for adding a SQL Server Compact 4.0 Database to the Web Services Sandbox project.

1. With the Web Services Sandbox project open, right click on the **APP\_DATA** folder in the **Solution Explorer** window. And select **Add New Item…**
2. In the Add New Items dialog box, click in the Search Installed Templates search box. Type “**SQL Server Compact**” and click the search button.
3. The Add New Items dialog box will automatically update displaying two **SQL Server Compact** database options. Be sure to select **SQL Server Compact 4.0 Local Database Visual C#**.
4. Give the Web Services Sandbox local database an appropriate name (i.e. WebServiceDB) and click the “**ADD**” button.



The local database will be added to the Web Services Sandbox project.

### Adding Tables to the Database

Now that the Web Services Project and local database has been created, it is time to add a table to the database. When creating tables it is important to identify column names, primary keys, and data types. The following will describe the process for adding a single table to the database for experimentation purposes.

1. In the solution explorer, navigate to the local database file in the **APP\_DATA** folder.
2. Double-click or right-click on the database file ->select **open**. This will open Visual Studio’s **Server Explorer** Window.
3. In the **Server Explorer** window, right-click on the **Tables** folder and select “**Create Table**”.
4. The **New Table** dialog box will appear. In the **New Table** dialog box, provide a name for the new **Table**. For the sake of this project, the Table name will be “**Products**”. Enter the following data into the table to identify the requirements of each column within the database.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Length** | **Allow Nulls** | **Unique** | **Primary Key** |
| product\_id | Int | 4 | NO | Yes | Yes |
| product\_name | nvarchar | 100 | NO | NO | NO |
| product\_description | nvarchar | 100 | NO | NO | NO |

1. Select the **product\_id** row and change its identity from **False** to **True**. This will allow the values of the **product\_id** column to be automatically generated by the database itself.
2. Once you are finished configuring the columns for this table, click the “**OK**”button.

Note: this is a simple database created for the purpose of experimenting with Web Services. It is encouraged that more advanced databases be employed or that this database be expanded on in order to provide more functionality later in this manual.

### Adding Data to the Database

Now that the database has been added and a table has been created, it is time to add some data to the SQL database.

1. In the **Server Explorer**, open the Web Service database **Table** folder.
2. Right-click the “**Products**” table and select “**Show Table Data**”.

In order to ensure the focus of this manual stays on the topic of Web Services, we will add general data to the database in or to having something for the Web Service to access and display later. The following is the sample data that will be added to the database table for the sake of creating the Web Service Sandbox project.

|  |  |  |
| --- | --- | --- |
| **Product: Query(…)** | | |
| **product\_id** | **product\_name** | **product\_description** |
| 1 | Red Shoe – M | A red shoe for men |
| 2 | Red Shoe – F | A red shoe for men |
| 3 | Blue Shoe – M | A blue shoe for men |
| 4 | Blue Shoe – F | A blue shoe for men |
| 5 | Green Shoe – M | A green shoe for men |
| 6 | Green Shoe – F | A green shoe for men |
| 7 | Black Shoe – M | A black shoe for men |
| 8 | Black Shoe – F | A black shoe for men |
| 9 | White Shoe – M | A white shoe for men |
| 10 | With Shoe – F | A white shoe for men |

1. After entering all of the data into the database close the table data window.
2. Now it is time to modify the **Web.config** file to incorporate the **connection string** the business components will need in order to access the database. The connection string will look like the following.

|  |
| --- |
| <connectionStrings>  <add name="ConnectionString" connectionString="data source=|DataDirectory|\WebServiceDB.sdf" providerName="System.Data.SqlServerCe.4.0"/>  </connectionStrings> |

1. The process of adding the local database to the Web Services Sandbox project is complete.

**Procedure Manual**

## Chapter 3: Business Components

### Creating the Class

Here is where we will be creating the business components needed to access the data store in the SQL Server Compact database. Business components can also be identified as Data Access objects as they are created using class libraries rather than employing a SQLDataSource object tied directly to the presentation layer of the Web application.

1. Visual Studio 2010 attempts to pace code in an **APP\_CODE** folder. If there is no such folder in the Solution Explorer Window, Right-click the project name in the **Solution Explorer**-> highlight the **Add ASP.NET Folder** ->click **APP\_CODE**.
2. Now we create the business component required to access the database. Right-click **APP\_CODE** folder and click **Add New Item…**
3. The **Add New Item** dialog box will appear. Select **Class Library** ensuring that it is intended for **C#** development.
4. Provide the business component with an appropriate **name**. (i.e. ProductAccess.cs), and click on the **Add** button.
5. Now add the following **C#** code to the ProductAccess Class file. Let’s start by including and organizing our using statements.

**Using Statements**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  **using System.Data**;  **using System.Data.SqlClient;**  **using System.Web.Configuration;**  **using System.Data.SqlServerCe**; |

Within the project several using statements have already been generated. However, take note of the highlighted using statements as they are required to access that local database. Be sure that all using statements presented are included in your Web Services project. Note: Visual studio may identify warnings in regards to using statements that are not being used. This will be corrected automatically when class functionality is implemented.

1. Now it is time to encase the class with a namespace so that the class will be capable of referencing dependencies as needed. Add the following code under the using statements in order to frame the class.

**Framing the Class**

|  |
| --- |
| namespace WebServiceSandbox  {  public class ProductAccess  {  //Class Code/regions will go here  }  } |

### Regions and Functionality

Although this class will be short for demonstration purposes, we will be adding **regions** in order to keep sections of code organized. Each **region** of code begins with the code **#region** followed by a descriptive word or phrase, and ended with the code **#endregion**. For the purpose of the Web Services Sandbox project, we will be using **six regions** to separate our code. The six regions are as follows: **Variables**, **Constructors**, **Properties**, **Private** **Methods**, **Public** **Methods**, and **Service** **Methods**.

With the use of regions the class will look like the following once it is completed.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Data;  using System.Data.SqlClient;  using System.Web.Configuration;  using System.Data.SqlServerCe;  namespace WebServiceSandbox  {  public class ProductAccess  {    Variables  Constructors  Properties  Private Methods  Public Methods  Service Methods  }  } |

### Variables

1. Lest Start by identifying our first region–**Variables**. Add the following code inside the ProductAccess class.

|  |
| --- |
| #region Variables  //connection variables  private String \_connectionString;  **private *SqlCeConnection* \_conn**;  //Private Variables  private long \_productID;  private String \_productName;  private String \_productDesc;  //Reporting Variables  private String \_errorInformation;  #endregion |

Notice that there are three separate sets of variables–connection, private, and reporting. The connection variables will contain information needed for the class to connect to the local SQL Server Compact 4.0 local Database (SQL CE). Take not that the connection variable \_conn uses a **SqlCeConnection** data type rather than a **SqlConnection** data type commonly used with remote and/or server based databases.

Following the connection variables, private variables are created to represent the objects in the database. Private variables are used as it is important to encapsulate such information so that data is not changed by unapproved methods.

Finally, a single reporting variable is declared. The reporting variable is simply used for debugging and purposes as a means of displaying errors that may occur when attempting to access and/or manage the local database.

### Constructors

1. Now let’s look at the **Constructors** region. The class has two constructors needed for creating and accessing data and methods within the class, and for testing purposes. Add the following code to the ProductAccess class file just under the variables region.

|  |
| --- |
| #region Constructors  /// <summary>  /// A Generic class Constructor  /// </summary>  public ProductAccess()  {  //uses connection string in web.config  \_connectionString = WebConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;  \_conn = new SqlCeConnection(\_connectionString);  ClearFields();  }  /// <summary>  /// Class Constructor that allows the passing of connection string  /// </summary>  /// <param name="connString"></param>  public ProductAccess(String connString)  {  \_connectionString = connString;  \_conn = new SqlCeConnection(\_connectionString);  ClearFields();  }  #endregion |

As you can see there are two class **constructors**. The first is a general class **constructor** intended to be used with a predefined **connection string**. However, the second constructor allows the developer to pass an alternate **connection string** to the class in order to provide flexibility and testing as needed. Adding the second constructor for this project will be optional as we will be focusing activities on the default constructor. With the limited code you will receive an error when adding the ClearFields method. This error occurs because the method has not yet been defined. Ignore this error right now as well will be adding its functionality shortly.

### Properties

We now have a means of creating a **ProductAccess** object, and locations within the object to store information. However, we do no not have a means of setting and retrieving any information stored in the defended variables. This is where the **Properties** region comes in. Each pre-defined variable needs methods for getting and setting the value of the class variables.

1. Add the following code under the Constructors region in order to accommodate getting and setting values.

|  |
| --- |
| #region Properties  public long ProductID  {  get { return \_productID;}  set { \_productID = value; }  }  public String ProductName  {  get { return \_productName; }  set { \_productName = value;}  }  public String ProductDescription  {  get { return \_productDesc; }  set { \_productDesc = value; }  }  public String ExtendedErrorInformation  {  get { return \_errorInformation; }  }  #endregion |

With the exception of connection variables, each pre-defined variable has a property with an associated name attached to it for accessing and setting its values. It is important to point out that the **reporting** **variable** is **read only** with means that it does not require a value setting method as its value is set programmatically at runtime.

### Private Methods

Private methods identify functionally that can only be accessed within the class itself. For the purposes of the Web Services Sandbox Project, we will be using three private methods. These methods will be responsible for executing SQL statements directed at our SQL CE database, loading data from a DataSet into the pre-defined class variables, and clearing the data stored in the class variables.

1. Add the following code to the class file to provide data management functionality.

|  |
| --- |
| #region Private Methods  /// <summary>  /// Clear all private variables  /// </summary>  private void ClearFields()  {  \_productID = 0;  \_productName = "";  \_productDesc = "";  \_errorInformation = "";  }  /// <summary>  /// Execute a SQL statement passed to the Method  /// nonquery = update, insert, or delete...not a select statement.  /// </summary>  /// <param name="SQL"></param>  /// <returns></returns>  private bool ExecuteSQLStatment(String SQL)  {  try  {  //SqlCeCommand Command = new SqlCeCommand(SQL);  SqlCeCommand Command = new SqlCeCommand();  Command.Connection = \_conn;  Command.CommandType = CommandType.Text;  Command.CommandText = SQL;  Command.Connection.Open();  Command.ExecuteNonQuery();  return true;  }  catch (Exception ex)  {  \_errorInformation = ex.Message;  return false;  }  }  /// <summary>  /// Find row data and load it into this BL class  /// </summary>  /// <param name="row"></param>  private void LoadData(DataRow row)  {  //load Product ID  try { \_productID = (row["product\_id"] is DBNull) ? 0 : long.Parse(row["product\_id"].ToString()); }  catch { \_productID = 0; }  //load Product Name  try { \_productName = (row["product\_name"] is DBNull) ? "" : row["product\_name"].ToString(); }  catch { \_productName = ""; }  //load Product Description  try { \_productDesc = (row["product\_descritpion"] is DBNull) ? "" : row["product\_descritpion"].ToString(); }  catch { \_productDesc = ""; }  //load Product Price  }  #endregion |

### Public Methods

A class with internal methods is good way of encapsulating information. However, it does not provide a means of interacting with these methods. For this purpose we will need to include a few **Public Methods**. Any ProductAccess object created will have access to public methods. These functions will also give the object a means of accessing class variables, and internal functionality through private methods. The public methods we will be including involve connection testing and data management methods.

1. Now it is time to add the following **Public Methods** code under the **Private Methods** region in order provide to provide functionality to any **ProductAccess** objects created.

|  |
| --- |
| #region Public Methods  /// <summary>  /// Test database connection …helper function  /// </summary>  /// <returns></returns>  public bool TestConnection()  {  try  {  \_conn = new SqlCeConnection(\_connectionString);  \_conn.Open();  \_conn.Close();  return true;  }  catch (Exception ex)  {  \_errorInformation = ex.Message; return false;  }  }  /// <summary>  /// Return a dataset of all data in database  /// </summary>  /// <returns></returns>  public DataSet GetAllProducts()  {  try  {  DataSet ds = new DataSet();  **SqlCeDataAdapter da = new SqlCeDataAdapter**("SELECT \* FROM Products", \_conn);  \_conn.Open();  da.Fill(ds, "Products");  return ds;  }  catch (Exception ex)  {  \_errorInformation = ex.Message; return null;  }  }  /// <summary>  /// Delete product from database via product id  /// </summary>  /// <param name="productID"></param>  /// <returns></returns>  public bool Delete(long productID)  {  try  {  String SQL = "DELETE FROM Products WHERE (product\_id = " + productID + " )";  return this.ExecuteSQLStatment(SQL);  }  catch (Exception ex)  {  \_errorInformation = ex.Message; return false;  }  }  /// <summary>  /// Add product to database  /// </summary>  /// <returns></returns>  public bool Add()  {  try  {  String SQL;  SQL = "INSERT INTO Products ("  + "product\_name, product\_descritpion, product\_price) VALUES (N'"  + \_productName + "', N'"  + \_productDesc + ")";  return ExecuteSQLStatment(SQL);  }  catch (Exception ex)  {  \_errorInformation = ex.Message;  return false;  }  }  /// <summary>  /// Update/Edit rows in database  /// </summary>  /// <returns></returns>  public bool Edit()  {  try  {  String SQL;  if (\_productID == 0)  throw new Exception("Nothing to Update!");  SQL = "UPDATE Products SET product\_name = N'" + \_productName  + "', product\_descritpion = N'" + \_productDesc  + "' WHERE (product\_id = " + \_productID + ")";  return ExecuteSQLStatment(SQL);  }  catch (Exception ex)  {  \_errorInformation = ex.Message; return false;  }  }  /// <summary>  /// Find First item in database based on product\_id  /// </summary>  /// <param name="productID"></param>  /// <returns></returns>  public bool FindFirstByID(long productID)  {  try  {  \_conn.Open();  SqlCeDataAdapter da = new SqlCeDataAdapter("SELECT \* FROM Products WHERE (product\_id =" + productID + " )", \_conn);  DataSet ds = new DataSet();  DataRow row;  da.Fill(ds, "Products");  row = ds.Tables["Products"].Rows[0];  if (row != null)  {  LoadData(row);  return true;  }  else { return false; }  }  catch (Exception ex) { \_errorInformation = ex.Message; return false; }  finally { \_conn.Close(); }  }  #endregion |

### Service Methods

Now let’s wrap up the **ProductAccess** class with a method or two that can be used through a **Web** **Service**. For the sake of simplicity we will be adding a single method to the **ProductAccess** class that will use specifically within the **Web Service** we will create later. Keep in mind that any public method created within the business component class can be used within a developed Web Service. The Service Method that we will use is very generic and simply collects and sorts all data from a table.

Add the following Service Method code to the end of your business component class.

|  |
| --- |
| #region Service methods  /// <summary>  /// Return a dataset of all data in database sorted by product\_id  /// </summary>  /// <returns></returns>  public DataSet SortAllProductsbyProductID(int order)  {  try  {  string loadString = null;  switch (order)  {  case 1:  loadString = "SELECT \* FROM Products ORDER BY product\_id ASC";  break;  case 2:  loadString = "SELECT \* FROM Products ORDER BY product\_id DESC";  break;  }  DataSet ds = new DataSet();  SqlCeDataAdapter da = new SqlCeDataAdapter(loadString, \_conn);  \_conn.Open();  da.Fill(ds, "Products");  \_conn.Close();  return ds;  }  catch (Exception ex)  {  \_errorInformation = ex.Message; return null;  }  }  #endregion |

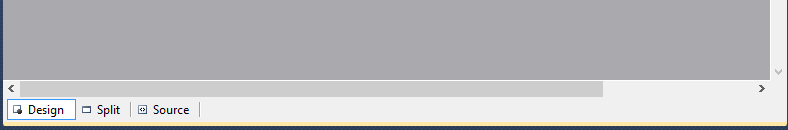
### Testing Your Business Component

Before we go any further we will need to ensure that the **business component** you have created works as intended. The best and fastest means of doing this is to add a data view to the presentation layer, and populate it using methods create in the **business component**. The following steps will describe how to add a **GridView** to the presentation layer and modify its code-behind file to update the contents of the **GridView** by creating a **ProductAccess** object.

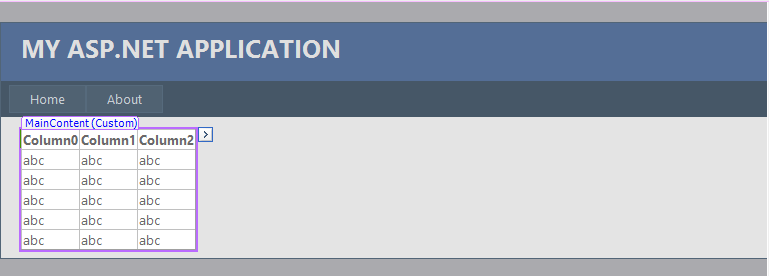
1. The first step is to modify the presentation layer so that it looks like the following:

|  |
| --- |
| <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="\_Default" %>  <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">  </asp:Content>  <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">    <!-- Add GridView Here-->  </asp:Content> |

1. Next, we will add the **GridView** to the presentation layer. Start by clicking on the **Design** button in the lower left corner of the workspace.



1. In the **Toolbox** Window, Select Data. Click and Drag GridView onto the application’s presentation layer in the workspace.



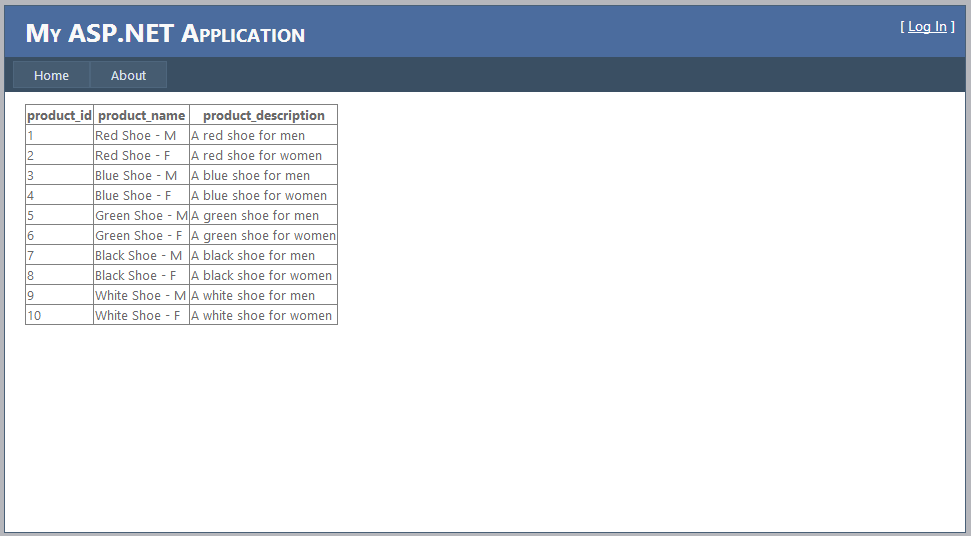
1. Click on the Source button in the bottom left of the workspace to see the code that has been generated from you actions. Take note of the **GridView’s ID** **value**. You will need this value to bind data to the view in the next step. Feel free to give the GridView ID a more memorable or appropriate value. For the sake of simplicity, we will modify the value to “**aaa**”. The Source of the presentation layer should now look like the following:

|  |
| --- |
| <%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="\_Default" %>  <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">  </asp:Content>  <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">    **<asp:GridView ID="aaa" runat="server">**  **</asp:GridView>**  </asp:Content> |

1. Now open up the Default Web page **Code-Behind** file and add the following code. Note that the **using statement** identifying the project namespace may be different depending on the name you have given your project. Modify the code so that it looks like the following:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.UI;  using System.Web.UI.WebControls;  //add this using  using WebServiceSandbox;  public partial class \_Default : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  ProductAccess \_pa = new ProductAccess();  aaa.DataSource = \_pa.GetAllProducts();  aaa.DataBind();  }  } |

As you can see, within the **Page\_Load** method a few things are happening with only three lines of code. First, a **ProductAccess** object is created and named **\_pa**. Secondly, the **data source** of the **GridView** we just added to the presentation layer is set to the returned value of the business components **GetAllProducts** method. Finally, the data retrieved from the business component logic is bound to the GridView. Assuming that you have followed along up to this point, you should now be able to build and run the application without errors. The result of testing the application should look like the following image.



As you can see the result of your Web Service Sandbox application should now be capable of presenting data from the SQL CE database via the ProductAccess business component. As all public methods are available to the ProductAccess object, we encourage you to modify the functionality of the Web application to incorporate additional functionality outside of simply showing all data within the database. As an important note, be sure to modify the GetAllProducts method for larger databases as it is not good practice to query all data from a database table.

**Procedure Manual**

## Chapter 4: Web Service

In order for an application to access Web Services via SOAP, a Web Service Definition Language (WSDL) (pronounced: Wizz-dle) file will need to be referenced [1] [2]. This is the part of the process manual that involves creating and adding such a file to your Web Service Sandbox project. The following steps will describe how to add a Web Service to your Web application project.

### Adding the Web Service

1. First, right-click on the **project** folder in the Solution explorer, and select **Add New Item**.
2. In the **Add New Item** Dialog, type **Web Service** into the **Search Box** in the upper right corner.
3. Be sure to select Web Service C#, and provide a name i.e. **SandboxService.asmx**.
4. Making sure the “place code in separate file” checkbox is ticked, click “**OK**”.
5. Visual Studio 2010 will now populate your web service file with generated code containing a “**hello world**” method that looks like the following.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Services;  /// <summary>  /// Summary description for SandboxService  /// </summary>  [WebService(Namespace = "http://tempuri.org/")]  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]  // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.  // [System.Web.Script.Services.ScriptService]  public class SandboxService : System.Web.Services.WebService {  public SandboxService () {  //Uncomment the following line if using designed components  //InitializeComponent();  }  [WebMethod]  public string HelloWorld() {  return "Hello World";  }  } |

### Identifying Conventions

As with most classes, the Web service file begins with using statements. However it differs in several ways. For example, just under the using statements is a section of code that defines the Web Service attributes such as namespace, name, and description. Following is the WebServiceBinding attribute used to identify the schema the services is intended to conform to. Additionally, each public method is expected to be preceded by a section of code identified as WebMethod in square brackets. Let’s take a closer look and make some simple modifications.

1. First, let’s look at the **Web Service attributes**. The Web Service attributes currently only contain namespace information. We will modify this section of the **WebService** source to accommodate a Service **name** and **description**.

|  |
| --- |
| [WebService(Namespace = "http://WebService.org/", Name = "SandboxService", Description = "This is a sample Web Service build following the 'Portable Web Service Sandbox' process manual")] |

1. Next we will look at the applied method. The hello world method is simply too generic for our purposes. So, we will leave it be and **create a new one** that allows us to utilize functionality within the business component we created previously. In order to apply the Service Method in the **ProductAccess** class, we will modify the code as follows.

|  |
| --- |
| [WebMethod]  /// <summary>  /// Returns a dataset of proudcts ordered by product\_id  /// </summary>  /// <param name="name">1= Acending, 2= Decending</param>  /// <returns>DataSet</returns>  public DataSet SortByPrice(int x)  {  try  {  \_pShare = new ProductAccess();  return \_pShare.SortAllProductsbyProductID(x);  }  catch (Exception ex)  {  return null;  }  } |

You may come to find that after implementing this method **errors** will spring up. This is due to the fact that the Web Service needs to have a reference to **System.Data** and the project **Namespace** (i.e. WebServiceSandbox). Be sure to **add these two references** to your using section in order to eliminate any error messages. The using statements should look like the following.

|  |
| --- |
| using System.Data;  using WebServiceSandbox; |

The completed Web Service source file should now look like the following:

### WSDL: A Complete Web Service

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Services;  //usings added  using System.Data;  using WebServiceSandbox;  /// <summary>  /// Summary description for SandboxService  /// </summary>  [WebService(Namespace = "http://WebService.org/", Name = "SandboxService", Description = "This is a sample Web Service build following the 'Portable Web Service Sandbox' process manual")]  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]  // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.  // [System.Web.Script.Services.ScriptService]  public class SandboxService : System.Web.Services.WebService {  private ProductAccess \_pShare;  public SandboxService () {  //Uncomment the following line if using designed components  //InitializeComponent();  }  [WebMethod]  public string HelloWorld() {  return "Hello World";  }  [WebMethod]  /// <summary>  /// returns a dataset of proudcts ordered by price  /// </summary>  /// <param name="name">1= Acending, 2= Decending</param>  /// <returns>DataSet</returns>  public DataSet SortByPID(int x)  {  try  {  \_pShare = new ProductAccess();  return \_pShare.SortAllProductsbyProductID(x);  }  catch (Exception ex)  {  return null;  }  }  } |

You have just finished creating your portable Web Service Sandbox using ASP.NET, C#, SQL Server Compact 4.0 with Visual Studio 2010.Continue to find out how to implement your web service.

**Procedure Manual**

## Chapter 5: Implementing the Web Service

Now that he Web service has been created and functionality has been added, we are at the point in the process where we can actually implement the Web Service. As with the business component we create and tested earlier, we can implement the Web Service in a similar fashion.

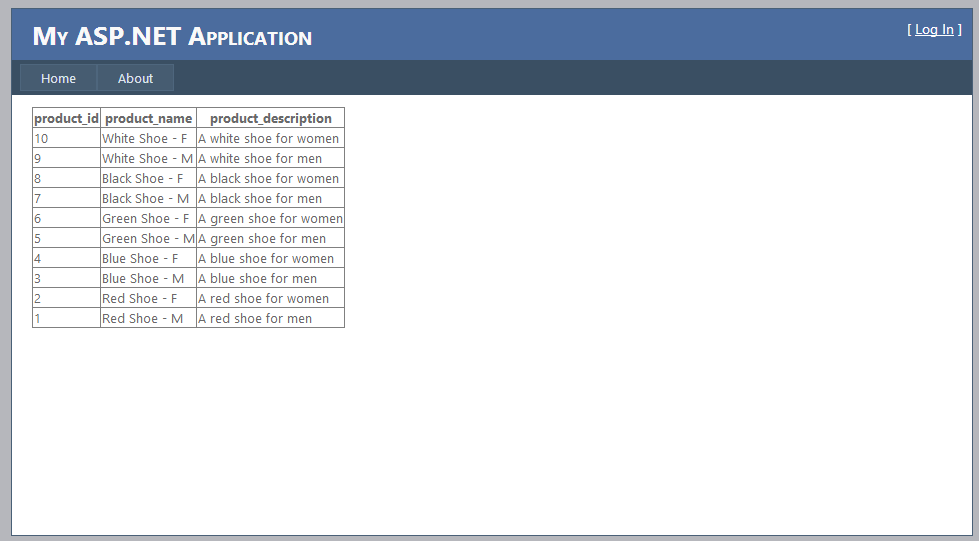
### Creating the Web Service Object

1. The first step will be to open the default.aspx code-behind file in the Solution Explorer.
2. Once the code-behind file is open in the workspace, modify the **Page\_Load** event handler**.** Modify the code so that it looks like the following:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.UI;  using System.Web.UI.WebControls;  //add this using  using WebServiceSandbox;  public partial class \_Default : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  SandboxService \_ss = new SandboxService();  aaa.DataSource = \_ss.SortByPID(2);  aaa.DataBind();    }  } |

Notice how the code is very similar to that of accessing the ProductAccess business component. The first line of code creates the **Sandbox Web Service** object named **\_ss**. The second line of code sets the data source of the **GridView** to the result of the Web Service’s **SortbyPID** method. In order to notice a difference in the presentation layer the value **2** was passed to the method in order to sort the data in **descending order**. Finally, the data returned is bound to the **GridView** data control.

Running the application now should display an image that looks like the following:



**Procedure Manual**

## Chapter 6: Points of Interest

### Importance of Learning Web Services

There are several reasons that Web Services are used today. Additionally, Web Services provide enterprises with benefits that range from enhanced deployability and interoperability to reusability and usability. These facts are apparent as major corporations such as Amazon, Google, Facebook, Twitter and more are employing such technology practices [3] [4]. As Web Services become more of an acceptable enterprise IT method, it will become imperative that IT professionals and students alike become familiar with such concepts. By following this process manual, you have created a portable Web Service Sandbox platform in which you can expand on, create new service, and ultimately experiment with.

### Taking it a Step Further

Now it is your turn. Take your newly developed portable Web Services Sandbox and expand on the concepts presented within this manual. For instance, get a copy of Microsoft’s northwind or another database and plug it into the project rather than creating content from scratch. Create several different business components in order to provide a wider range of functionality for the Web application. More importantly, experiment with adding methods to your newly created Web Service.

**Procedure Manual**

# Quick Reference

|  |
| --- |
|  |

**Step 1.**

**Development Environment Prep**

System Requirements

|  |
| --- |
| If your development environment already meets the system requirements or you have been developing Web applications previously with Visual Studio feel free to skip to step 3.   1. Ensure that your development machine meets and/or exceeds the system requirements needed of the IDE. This can be found on ***page 12***. 2. Update current operating system and include .NET framework 4.0 +. 3. Download and Install Microsoft Visual Studio 2010. 4. Download and Install Microsoft SQL Server Compact Edition 4.0.   ***Note***: This manual can be followed with other version of Visual Studio. However, not all versions contain SQL Server CE. Make sure that your version of Visual Studio requires this step before installing. |

**Step 2.**

**Pouring the Foundation**

Creating a Web Project

|  |
| --- |
| Creating the portable Web Services Sandbox development environment begins by creating a Web application.   1. Start Visual Studio 2010. 2. In the menu bar, choose **File** -> **New Web Site**. 3. Select **ASP.NET Web Site** from the templates panel.      1. In the **New Web Site** dialog, be sure to select **Installed Templates** -> **Visual C#.** 2. In the **New Web Site** dialog, select the **File** **System** as the **Web Location** 3. Designate a physical location on your disk to store the project. i.e. C:\WebService1   ***Note***: the location of your application is recommended to be placed on a portable media such as a thumb drive. This will allow you to take your Web Service Development Sandbox with you wherever you go.   1. Press “OK”   Visual Studio will now auto-generate several files for the Web application. |

**Step 3.**

**Applying Portability**

SQL CE 4.0

|  |
| --- |
| **Adding the Local Database**   1. Open the Web Services Sandbox project in Visual Studio 2010 2. Right click on the **APP\_DATA** folder in the **Solution Explorer** window. 3. Select **Add New Item…** 4. In the Add New Items dialog box, click in the Search Installed Templates search box. 5. Type “**SQL Server Compact**” and click the **search** button. 6. The Add New Items dialog box will automatically update displaying the **SQL Server Compact** database options.      1. Select **SQL Server Compact 4.0 Local Database Visual C#**.      1. Give the Web Services Sandbox local database an appropriate name (i.e. WebServiceDB) and click the “**ADD**” button. |

|  |  |
| --- | --- |
| **Adding a Table** | **Adding Table Data** |
| * Open Server Explorer * Right-Click “Tables” * Select “Create Table” * Name the table “Products” * Provide column names and values * Change product\_id identity to “true” * Click “**OK**” | * Open Server Explorer * Open Table Directory * Right-Click “Products” * Select “Show Table Data” * Provide required data   Reference Page 16-18 |

|  |
| --- |
| **Modifying the Web.config File**   1. Open the Solution Explorer 2. Open the file named Web.config 3. Navigate to the section heading called “connectionString” 4. Add and/or modify the code to incorporate the following within the connectionString heading:   <connectionStrings>  <add name="ConnectionString" connectionString="data source=|DataDirectory|\WebServiceDB.sdf" providerName="System.Data.SqlServerCe.4.0"/>  </connectionStrings> |

**Step 4.**

**Providing Functionality**

Creating Business Components

|  |
| --- |
| * Right-click **APP\_CODE** folder. * Click **Add New Item…** * Select **Class Library** for **C#** development. * Provide the name “ProductAccess” * Click **Add** * Add the required elements to the C# library * Variables * Class Constructors * Properties * Public and Private Methods * Services Methods |

**Step 5.**

**Implementing SOAP**

Web Service with WSDL

|  |
| --- |
| **Create the WSDL Document**   1. Open the Solution Explorer 2. Right-click on the project folder 3. Select **Add New Item**. 4. Search “Web Service” 5. Select Web Service C# 6. Name the Web Service **SandboxService** 7. Tick “place code in separate file” **checkbox** 8. Click “**OK**”   Visual Studio 2010 will now auto generate a **WSDL** document for managing your services.  Once the code-behind file is open in the workspace, modify the Page\_Load event handler. Modify the code so that it looks like the following: |

|  |
| --- |
| **Configuring the Web Service**   1. Modify Web Service attributes to include the name SandboxService 2. Modify the Web Service attributes to include the description “This is a sample Web Service build following the 'Portable Web Service Sandbox' process manual ” 3. Remove the “Hello World” default method |

|  |
| --- |
| **Adding Functionality**   * Create a new Web Service Method. Note: Be sure to precede the method with the following: [WebMethod] * Apply a method(s) that have been identified within the ProduceAccess business component under Services Methods. For more detailed instructions review chapter 4.   Apply the methods created in the WSDL file to the presentation layer code-behind by following the following steps.   1. Open the **default.aspx** code-behind file 2. Create a new Web service object 3. Modify the Page\_Load event handler to bind data to a GridView in the Presentation Layer with methods used in the Web Service.   Refer to **Chapter 6** for detailed instructions. |

**Procedure Manual**

PowerPoint Deck



**Procedure Manual**

# Index

Add New Item… 13, 16

APP\_CODE 16

APP\_DATA 13, 14

ASP.NET 7, 8, 12, 16, 29, 31

business components 6, 15, 16, 27, 34

Business Components 6, 7, 16

C# 8, 12, 13, 16, 26, 27, 29, 31

code-behind 12, 26, 32

connection string 6, 15, 19

Constructors 17, 19

Data Binding 7, 8

Default page 12

endregion 17, 18, 19, 20, 21, 23, 25

GridView 26, 27, 32

Installed Templates 12, 13

Master page 12

Microsoft Visual Studio 2010 8, 11

New Table 14

Presentation Layer 6

Private Methods 17

Properties 17, 20

Public Methods 17, 22

region 17, 18, 19, 20, 21, 22, 25

Server Explorer 14, 15

Service Methods 17, 25

software requirements 6

Solution Explorer 13, 16, 32

SQL 6, 7, 8, 11, 13, 15, 16, 18, 21, 23, 28, 31, 36

SQL Server Compact 4.0 6, 7, 8, 11, 13, 18, 31, 36

SqlCeConnection 18, 19, 22

SqlConnection 18

Table 14, 15

Tables 6, 7, 14, 23

templates 12

Variables 17, 18

Visual Studio 2010 6, 8, 11, 12, 16, 29, 31, 36

Web project 6, 12

Web Service attributes 30

Web Service Definition Language (WSDL) 29

Web services 6, 7, 8

WebService 29, 30, 31

**Procedure Manual**

# References

|  |  |
| --- | --- |
| [1] | Microsoft Corporation, "Understanding WSDL," n.d.. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms996486.aspx. [Accessed 8 May 2014]. |
| [2] | TutorialsPoint, "WSDL," 2014. [Online]. Available: http://www.tutorialspoint.com/wsdl/. [Accessed 8 May 2014]. |
| [3] | WebDam, "Top 10 Web APIs — Bridging Today’s Technology," n.d.. [Online]. Available: http://www.webdam.com/blog/top-10-web-apis\_bridging-todays-technology/. [Accessed 8 May 2014]. |
| [4] | Web Developers, "40 useful APIs for web designers and developers," 22 July 2011. [Online]. Available: http://www.webdesignerdepot.com/2011/07/40-useful-apis-for-web-designers-and-developers/. [Accessed 8 May 2014]. |
| [5] | Microsoft Corporation, "Microsoft SQL Server Compact 4.0," n.d.. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=17876. [Accessed 6 May 2014]. |
| [6] | Microsoft Corporation, "Visual Studio 2010 SDK," n.d.. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=2680. [Accessed 6 May 2014]. |
| [7] | Microsoft, "Microsoft .NET Framework 3.5," n.d.. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=21. [Accessed 15 May 2014]. |
| [8] | Microsoft, "Microsoft .NET Framework 4 (Web Installer)," n.d.. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=17851. [Accessed 15 May 2014]. |

1. .NET 3.5 [7] [↑](#footnote-ref-1)
2. .NET 4.0 [8] [↑](#footnote-ref-2)
3. Microsoft Visual Studio 2010 [2] [↑](#footnote-ref-3)
4. Microsoft SQL Server Compact 4.0 [1] [↑](#footnote-ref-4)