Hands-On Lab

Introduction to WCF (ADO.NET) Data Services

Lab version: 1.0.0
Last updated: 12/10/2010
# Contents

**OVERVIEW**.................................................................................................................................................. 3

**EXERCISE 1: CREATING AND CONSUMING WCF DATA SERVICES**................................................................. 5
  Task 1 – Creating the Data Service Web Application Project ............................................................................. 6
  Task 2 – Creating the Entity Data Model ........................................................................................................... 8
  Task 3 – Creating the Data Service .................................................................................................................. 12
  Task 4 – Using the .NET Client API to Consume the Service ........................................................................... 15
  Verification ...................................................................................................................................................... 21

**EXERCISE 2: CONSUMING WCF DATA SERVICES USING ASP.NET AJAX LIBRARY BETA** ............ 28
  Task 1 – Configuring a Client to Consume the Data Service ........................................................................... 28
  Task 2 – Creating a Query to Consume the Data Service ............................................................................... 29
  Task 3 – Inserting Entities Using the Data Service ......................................................................................... 31
  Task 4 – Updating Entities Using the Data Service ......................................................................................... 33
  Task 5 – Removing Entities Using the Data Service ......................................................................................... 35
  Verification ...................................................................................................................................................... 36

**EXERCISE 3: EXTENDING DATA SERVICES WITH SERVICE OPERATIONS AND INTERCEPTORS** 44
  Task 1 – Creating a Service Operation ........................................................................................................... 44
  Task 2 – Creating a Service Interceptor .......................................................................................................... 49
  Task 3 – Using Service Interceptors to Add Custom Query Constraints .......................................................... 54

**EXERCISE 4: ADDING CLIENT-SIDE PAGING WITH ROW COUNT** ......................................................... 56
  Task 1 – Invoking the Count Method ............................................................................................................. 57
  Task 2 – Displaying the Product Count on the User Interface ....................................................................... 61
  Task 3 – Adding Paging to Existing Product List Query ................................................................................. 64
  Task 4 – Updating the User Interface to Show the Paged ListView ............................................................... 66
  Verification ...................................................................................................................................................... 72

**SUMMARY**.................................................................................................................................................. 75
Overview

The WCF Data Services (formerly known as ADO.NET Data Services) framework consists of a combination of patterns and libraries that enable the creation and consumption of data services for the web.

Its goal is to enable applications to expose data as a *data service* that can be consumed by web clients within corporate networks and across the internet. WCF Data Services uses URIs to point to pieces of data and simple, well-known formats to represent that data, such as ATOM and JSON.

Also, the use of web-friendly technologies make it ideal as a data back-end for AJAX-style applications, Rich Interactive Applications and other applications that need to operate against data that is across the web.

The new WCF Data Services release is a standalone release that represents a work in progress towards the next version of the WCF Data Service technology. This release targets the .NET Framework 3.5 SP1 and Silverlight 2 platforms providing new Client and Server side features for data service developers. However, in this lab you will see how to adapt it to Visual Studio 2010. The following is a brief description of the enhancements and new features included:

- **Data Binding**: Support two-way data binding for WPF and Silverlight. It is possible to generate client side types that implement the `INotifyPropertyChanged` and `INotifyCollectionChanged` interfaces.
- **Row Count**: Support for determine the total number of entities in a set without retrieving them all.
- **Feed Customization (aka "Web Friendly Feeds")**: Ability to customize how entities are mapped into the various elements of an Atom feed.
- **Server Driven Paging**: Allows setting, per collection, limits on the total number of entities returned for each request and providing a "next link" specifying how to continue retrieving the rest of the entities.
- **Enhanced BLOB Support**: Enables data services to stream large blobs, store binary content separate from its metadata, etc.
- **New "Data Service Provider" Interface for Custom Provider Writers**: Introduces a way to write a "custom" provider for cases when the supported provider models don't meet your needs.

Objectives

In this Hands-On Lab, you will learn how to:
• Create a WCF data service and consume it using the .NET Client API
• Consume a WCF data service using ASP.NET Ajax Library Beta
• Add Service Operations and Interceptors to the Data Service
• Add Client-Side Paging with Row Count

**System Requirements**

You must have the following items to complete this lab:

- Microsoft Visual Studio 2010
- Microsoft SQL Server 2005 or Microsoft SQL Server 2008 (Express edition or above)
- Adventure Works sample database

**Note:** the Dependency Checker will copy the Adventure Works sample database file on each exercise solution folder. If you wish you can copy it manually from the Assets folder.

---

**Setup**

All the requisites for this lab are verified using the **Configuration Wizard**. To make sure that everything is correctly configured, follow these steps.

**Note:** To perform the setup steps you need to run the scripts in a command window with administrator privileges.

1. Run the **Configuration Wizard** for the Training Kit if you have not done it previously. To do this, browse to Source\Setup folder of this lab, and run the **CheckDependencies.cmd** script. Install any pre-requisites that are missing (rescanning if necessary) and complete the wizard.

**Note:** For convenience, much of the code you will be managing along this lab is available as Visual Studio code snippets. The **CheckDependencies.cmd** file launches the Visual Studio installer file that installs the code snippets.
Exercises

This Hands-On Lab is comprised by the following exercises:

- Creating and consuming WCF data services
- Consuming WCF data services using ASP.NET Ajax Library Beta
- Extending Data Services with Service Operations and Interceptors
- Adding Client-Side Paging with Row Count

Estimated time to complete this lab: **90 minutes**.

**Note:** Each exercise is accompanied by a starting solution. Some code sections are missing from these solutions, which will be completed through each exercise. Therefore the starting solutions will not work if you run them directly.

Inside each exercise you will also find an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

**Note:** Each exercise contains a Visual Basic and a C# version; Inside the End/Begin solution folder you will find two folders: VB, containing the Visual Basic version of the exercise, and C#, containing the C# version of it.

Next Step:

**Exercise 1: Creating and Consuming WCF Data Services**

In this exercise you will learn how to create a data service and consume it with an existing WPF user interface using the .NET Data Services Client API.

**Note:** To verify that each step is correctly performed, it is recommended to build the solution at the end of each task.
Task 1 – Creating the Data Service Web Application Project

In this task you will create an ASP.NET Web application project that will host and expose the data service.


2. Open the solution file ADONETDataServiceSample.sln located under the Source\Ex1-CreatingDataServices\begin folder of this lab (choosing the folder that matches the language of your preference.) This is an empty solution for exercise 1.

3. In Solution Explorer, right-click ADONETDataServiceSample solution, point to Add and select New Project.

4. In the Add New Project dialog, select the Visual C# language or Visual Basic language and then the Web project type. Make sure that .NET Framework 4.0 is selected, and then select the ASP.NET Web Application template.

5. You may set the location to the provided Source\Ex1-CreatingDataServices\begin folder of this lab (and choosing the folder that matches the language of your preference) which is the provided folder.

6. Change the Name to WebSite and click OK.
Figure 1

Creating a new Website (C#)
Task 2 – Creating the Entity Data Model

In this task you will create the mapping specification that connects programmable classes to storage structures.

The Entity Data Model (EDM) is a specification for defining the data used by applications built on the Entity Framework. Applications using the EDM define entities and relationships in the domain of the application in a design schema.

1. Create the AdventureWorks Entity Data Model. To do this, in Solution Explorer, right-click the WebSite project, point to Add, and click New Item.

2. In the Add New Item dialog box select ADO.NET Entity Data Model. Specify a Name value of AdventureWorks.edmx, and then click Add.
Figure 3
Adding the ADO.NET Entity Data Model (C#)
3. After the **Entity Data Model Wizard** opens, select **Generate From Database** and click **Next**.

4. Specify the Database connection. To do this, follow these steps:
   
   a. Click **New Connection**
   
   b. In **Choose Data Source** dialog, select **Microsoft SQL Server Database File (SqlClient)** as **Data Source** and click **Continue**.
   
   c. Click the **Browse** button to set the database file name to use.
   
   d. Browse to the **Source\Assets** folder and select the **AdventureWorksLT.mdf** file.
   
   e. Click **OK** to save the connection.
Figure 5
*Specify the database connection*


6. You will be prompted if you want to copy the file inside your project directory and change the connection string accordingly. Click Yes to proceed.

7. Include only the **Tables** objects from all the proposed database objects, leave the **Model Namespace** by default, clear both check boxes, and click Finish.
Task 3 – Creating the Data Service

In this task you will create the WCF Data Service that expose data by using the Entity Data Model as specified by the ADO.NET Entity Framework.

1. Add the Data Service. To do this, in Solution Explorer, right-click the WebSite project, point to Add, and click New Item.

2. In the Add New Item dialog box select WCF Data Service. Specify a Name value of AdventureWorks.svc, and then click Add.
Figure 7
Add the WCF Data Service item (C#)

Figure 8
Add the WCF Data Service item (Visual Basic)

3. Locate the class definition and replace it with the following code. This will indicate the service to use the entity framework data model as a data source.

   **Note:** Visual Studio will open the code file for the new service by default. You can also find the file in the Solution Explorer by right clicking on the *AdventureWorks.svc* file.

   (Code Snippet – *Data Services Lab - AdventureWorks Data Service Definition C#*)

   **C#**
   ```csharp
   public class AdventureWorks : DataService<AdventureWorksLTEntities>
   ```

   (Code Snippet – *Data Services Lab - AdventureWorks Data Service Definition VB*)

   **Visual Basic**
   ```vbnet
   Public Class AdventureWorks
       Inherits DataService(Of AdventureWorksLTEntities)
   ```

4. Enable read and write access to all resources in the Entity Data Model associated with the service by replacing the code in the `InitializeService` method with the following:

   (Code Snippet – *Data Services Lab - AdventureWorks.svc InitializeService C#*)

   **C#**
   ```csharp
   public static void InitializeService(DataServiceConfiguration config)
   {
       config.SetEntitySetAccessRule("*", EntitySetRights.All);
   }
   ```

   (Code Snippet – *Data Services Lab - AdventureWorks.svc InitializeService VB*)

   **Visual Basic**
   ```vbnet
   Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
       config.SetEntitySetAccessRule("*", EntitySetRights.All)
   End Sub
   ```
Note: By default a data service does not expose any resources. Access to resources need to be explicitly enabled before any resources or associations are accessible. For more information, see `IDataServiceConfiguration.SetEntitySetAccessRule Method (System.Data.Services)`.

5. Configure the web site to use port 50000. To do this:
   a. In Solution Explorer, right-click WebSite project and in the context menu select Properties.
   b. In the Properties page open the Web tab.
   c. In the Servers section select Specific Port.
   d. Set the port number to 50000.
   e. Press Ctrl + S to save changes.

![Figure 9](specifying-a-port-number.png)

Task 4 – Using the .NET Client API to Consume the Service

WCF Data Services includes a minimum-footprint client library that presents a more natural programming model for applications written using the .NET Framework targeting data services. The client library can return results in terms of .NET objects.

In this task you will consume the previously created data service using the .NET Client library. This library provides a programming model for applications using the .NET Framework and Internet data services.
You will interact with the data services using a WPF client application that allows querying, adding, deleting and updating records from the database. These operations are performed through a class that acts as gateway to access the Data Service, and return entity objects using the **WCF Data Services** Client API.

1. Include the WPF application project to the solution. To do this, right-click the **ADONETDataServiceSample** solution in **Solution Explorer**, point to **Add** and click **Existing Project**. Browse the **Source\Assets** folder of this lab (choosing the folder that matches the language of your preference.), and select **UserInterface.csproj** under the **UserInterface** folder and click **Open**.

2. Right click the **WebSite** project node and click **Build**. You can also test the project configuration is correct by starting a new instance of the Web project, but this is not required in this step.

3. Create the .NET objects to represent each of the entities of the data service definition. To do this, right click on the **UserInterface** project and select **Add Service Reference...**

4. Click **Discover**, choose the **AdventureWorks.svc** service and name the reference **AdventureWorks** then click **OK** to save and dismiss the Add Service Reference dialog.

   **Note:** In order to represent each of the entities defined in the data service as .NET objects in the client; corresponding classes need to be defined for the client application. One option is to define the classes manually. Another option, more automatic, is using the **Add Service Reference** tool that ships with Visual Studio 2010.

   If any problem arises in the previous step you can also find the **AdventureWorks.cs** (C#) or **AdventureWorks.vb** (Visual Basic) file, generated with the **Add Service Reference** tool, in the **Source\Assets** folder of this lab and choosing the folder that matches the language of your preference.

5. Start to implement the **ProductGateway** operations using WCF Data Service URI Syntax and LINQ to Data Services. In **Solution Explorer**, open **ProductGateway.cs (C#)** or **ProductGateway.vb** (Visual Basic). To do this, double-click the **ProductGateway.cs** (C#) or **ProductGateway.vb** (Visual Basic) file under the **Gateway** folder of **UserInterface** project.

6. Retrieve all the products that match with the name and category passed as parameter using WCF Data Service URI syntax. To do this, replace the **GetProducts** method with the following content.

   (Code Snippet – **Data Services Lab - ProductGateway GetProducts CSharp**)
public IList<Product> GetProducts(string productName, ProductCategory category)
{
    int categoryId = category.ProductCategoryID;

    IEnumerable<Product> products = this.context.Execute<Product>(
        new Uri(this.context.BaseUri.ToString() +
            "/ProductCategory(" + categoryId + ")/Product?$filter=indexof(Name," + productName + ") gt -1 or '' eq '' + productName + ")");

    List<Product> productsSet = new List<Product>();
    foreach (Product p in products)
    {
        this.context.LoadProperty(p, "ProductCategory");
        productsSet.Add(p);
    }

    return productsSet;
}

Visual Basic

Public Function GetProducts(ByVal productName As String, ByVal category As ProductCategory) As IList(Of Product) Implements IProductGateway.GetProducts
    Dim categoryId = category.ProductCategoryID

    Dim products = Me.context.Execute(Of Product)(
        New Uri(Me.context.BaseUri.ToString() &
            "/ProductCategory(" & categoryId & ")/Product?$filter=indexof(Name," & productName & ") gt -1 or '' eq '' & productName & ")");

    Dim productsSet As New List(Of Product)()
    For Each p In products
        Me.context.LoadProperty(p, "ProductCategory")
        productsSet.Add(p)
    Next

    Return productsSet
End Function
**Note:** This query selects all products belonging to a specified category and also executes a filter that checks the product name. This filter makes use of the `indexof` function and use the `gt` (greater than), `or` and `eq` (equals) operations of the WCF Data Services URI syntax. For more information, see [URI Format for Addressing Resources (ADO.NET Data Services Framework)](#).

7. Retrieve a list of all the product categories stored in the `ProductCategory` table using LINQ to WCF Data Services. The client library handles the details of mapping the LINQ statement to a URI in the target data service and retrieving the specified resources as .NET objects. To do this, replace the `GetCategories` method with the following content.

(Code Snippet – Data Services Lab - ProductGateway GetCategories CSharp)

```csharp
public IList<ProductCategory> GetCategories()
{
    var productCategories = from c in this.context.ProductCategory
                            orderby c.Name
                            select c;

    return productCategories.ToList();
}
```

(Code Snippet – Data Services Lab - ProductGateway GetCategories VB)

```visualbasic
Public Function GetCategories() As IList(Of ProductCategory) Implements IProductGateway.GetCategories
    Dim productCategories = From c In Me.context.ProductCategory
                            Order By c.Name
                            Select c
    Return productCategories.ToList()
End Function
```

**Note:** The context object used in this method is an instance of `AdventureWorksLTEntities`. This class has properties that represent each table of the `AdventureWorksLT` database as a `DataServiceQuery` class that implements `IQueryable` interface and allow LINQ to work with these entities.
8. Delete the specified product from the database and then call the `SaveChanges` method to submit the changes to the data service. To do this, add the following code to the `DeleteProduct` method.

(Code Snippet – *Data Services Lab - ProductGateway DeleteProduct CSharp*)

```
C#
public void DeleteProduct(Product product)
{
    this.context.AttachTo("Product", product);
    this.context.DeleteObject(product);
    this.context.SaveChanges();
}
```

(Code Snippet – *Data Services Lab - ProductGateway DeleteProduct VB*)

```
Visual Basic
Public Sub DeleteProduct(ByVal product As Product) Implements IProductGateway.DeleteProduct
    Me.context.AttachTo("Product", product)
    Me.context.DeleteObject(product)
    Me.context.SaveChanges()
End Sub
```

9. Update the product properties including relationships to objects and single values. To do this, add the following code to the `UpdateProduct` method.

```
Note: This code receives the product with the new `ProductCategory` object associated, and stores it in a temporary variable. Then calling the `LoadProperty` method, it re-loads the `ProductCategory` object from the database to remove the old binding. Finally the new `ProductCategory` and its binding are added using the context to be able to save the changes. Also notice that the context should attach the objects first to perform operations over them.
```

(Code Snippet – *Data Services Lab - ProductGateway UpdateProduct CSharp*)

```
C#
public void UpdateProduct(Product product)
{
    ProductCategory newCategory = product.ProductCategory;
    this.context.AttachTo("Product", product);
```
this.context.LoadProperty(product, "ProductCategory");
if (newCategory.Name != product.ProductCategory.Name)
{
    this.context.DeleteLink(product, "ProductCategory",
    product.ProductCategory);
    this.context.AttachTo("ProductCategory", newCategory);
    this.context.AddLink(product, "ProductCategory", newCategory);
}

this.context.UpdateObject(product);
this.context.SaveChanges();

(Code Snippet – Data Services Lab - ProductGateway UpdateProduct VB)

Visual Basic
Public Sub UpdateProduct(ByVal product As Product) Implements IProductGateway.UpdateProduct
    Dim newCategory = product.ProductCategory
    Me.context.AttachTo("Product", product)
    Me.context.LoadProperty(product, "ProductCategory")
    If newCategory.Name <> product.ProductCategory.Name Then
        Me.context.DeleteLink(product, "ProductCategory",
        product.ProductCategory)
        Me.context.AttachTo("ProductCategory", newCategory)
        Me.context.AddLink(product, "ProductCategory", newCategory)
    End If
    Me.context.UpdateObject(product)
    Me.context.SaveChanges()
End Sub

Note: When updating an associated property you have to delete the old binding to that property, and create the new binding before calling to the UpdateObject method.

10. Add a new product into the database. To do this, add the following code to the AddProduct method.

(Code Snippet – Data Services Lab - ProductGateway AddProduct CSharp)

C#

public void AddProduct(Product product)
{ 
  product.rowguid = Guid.NewGuid();
  this.context.AddObject("Product", product);
  product.ProductCategory.Product.Add(product);
  this.context.AttachTo("ProductCategory", product.ProductCategory);
  this.context.AddLink(product.ProductCategory, "Product", product);
  this.context.SaveChanges();
}

(Code Snippet – Data Services Lab - ProductGateway AddProduct VB)

Visual Basic

Public Sub AddProduct(ByVal product As Product) Implements IProductGateway.AddProduct
    product.rowguid = Guid.NewGuid()
    Me.context.AddObject("Product", product)
    product.ProductCategory.Product.Add(product)
    Me.context.AttachTo("ProductCategory", product.ProductCategory)
    Me.context.AddLink(product.ProductCategory, "Product", product)
    Me.context.SaveChanges()
End Sub

Note: When adding an object related to another object, you also have to bind its object property using the AddLink method.

Next Step:
Verification

Verification

In order to verify that you have correctly performed all steps of the exercises, proceed as follows:

Verification 1

In this verification you will use a web browser to consume the Data Service and check its XML response.

Note: The easiest way to try a data service is to simply access it using a web browser. While this is probably not the way you will ultimately use the data service (it is more likely that a program will
interact with it), it is an easy way to understand how requests work, what results look like, and other details surrounding the implementation of the service.

1. To interact with the Data Service you need to start a new instance of the **WebSite** project. To do this, in **Solution Explorer** right-click the **WebSite** project, point to **Debug** and select **Start New Instance**.

   **Note:** If the dialog **Debugging Not Enabled** appears, select **Modify the Web.config file to enable debugging** and click **OK**.

2. Navigate to **http://localhost:50000/AdventureWorks.svc**. You will notice that the XML response of the data service is a list of entity-sets. The entity-sets in the Entity Data Model (EDM) style represent the database tables that are exposed by the data service. The output will be similar to what is shown below.

   **Note:** The XML document returned by default is an Atom service document since the default serialization used by the data service is Atom.
3. To browse a specific product, use the following address in the web browser

http://localhost:50000/AdventureWorks.svc/Product(680) (where 680 is the product id). The following output should appear.
Figure 11

**Data Service XML response for a specific product**

**Note:** If the browser displays a message saying it cannot display the feed, try turning off the feed reading view. In Internet Explorer, this option is located under **Tools | Internet Options | Content Tab | Feeds Section | Settings**, uncheck **Turn on feed reading view**.

4. In Visual Studio press **Shift + F5 (C#) or Ctrl + Alt + Break (Visual Basic)** to stop debugging.

**Note:** The WCF Data Services framework implements a data addressing scheme that uses URIs. It is based on the Entity Data Model (EDM) specification for data types and data sets. For building more complex requests to the service using URIs check **Simple Addressing Scheme for Data with Uniform URLs**.
Verification 2

In this verification you will use the WPF application to retrieve a list of products from the database and perform an update of a product’s details.

1. Set the UserInterface project as StartUp project. To do this, right-click UserInterface project in Solution Explorer and select Set as StartUp Project.

2. Press F5 to run the application. The following window should appear. You can list, create, update, and delete products from this window.

![Application main window](image)

**Figure 12**

*Application main window*

3. On the **Product Category** combo select **Brakes** and click **Search**. The application will retrieve from the data service all products from the **Brakes** category and display them in the grid.
4. Edit the first product on the list details. To do this, double-click the first product on the product list.
5. Change the product color from **Silver** to **Gold** and click **Save**. The application will update the product entity and the record will be stored in the database.

6. You can also try the *creating and deleting* the operations.

7. Close the application to end this verification.

**Next Step:**

[Exercise 2: Consuming WCF Data Services Using ASP.NET Ajax Library Beta](#)
Exercise 2: Consuming WCF Data Services Using ASP.NET Ajax Library Beta

WCF Data Services provides a client script class for simplifying the interaction between an ASP.NET Ajax application and a WCF data service. By using this class, you can create Web applications that interact with data through a data service on the Web site and that can update the Web page without a full postback to the Web server.

In this exercise you will learn how to consume the WCF data service from an ASP.NET Ajax application using this script class to query, delete or modify data through the data service.

Note: To verify that each step is correctly performed, it is recommended to build the solution at the end of each task.

Task 1 – Configuring a Client to Consume the Data Service

In this task you will modify some configuration on the client side that will allow you to consume the data service using the ASP.NET Ajax Library Beta.


2. Open the solution file ADONETDataServiceSample.sln located under the Source\Ex02-ConsumingDataServicesUsingAspNetAjax\begin folder of this lab (choosing the folder that matches the language of your preference.)

3. Add the data service client library to the project. To do this, right-click on the Scripts folder under the WebSite project and select Add, then Existing Item. From the Add Existing Item dialog, navigate to the Source\Assets folder of this lab and select the MicrosoftAjaxAdoNet.debug.js file under the MicrosoftAjax folder.

Note: In past releases, this AJAX client library was embedded into the System.Web.Extensions assembly. It was removed in .NET 3.5 SP1, and it is now part of the ASP.NET Ajax Library Beta. We’ve packaged the file along with the training kit for convenience, but it can be downloaded with the rest of the ASP.NET Ajax Library Beta scripts from here.

4. Add a ScriptManager that manages client script for Microsoft ASP.NET AJAX pages. To do this, add to the Default.aspx web page the following content.
5. Implement the method that retrieves an instance of the AdoNetServiceProxy class. This class provides functions that you can call in order to interact with a data service from an ASP.NET AJAX application. To do this, replace the getService() function (located in ProductGateway.js file of the WebSite project) with the following content.

JavaScript

```javascript
function getService()
{
    return new Sys.Data.AdoNetServiceProxy("/AdventureWorks.svc");
}
```

Task 2 – Creating a Query to Consume the Data Service

In this task you will create a REST query using some of the data service query string options, call the query method of the AdoNetServiceProxy class to retrieve some products and display them in a web page.
1. Write the query in a REST form. To do this, add to the `getProducts()` function (located in `ProductGateway.js` file) the following content (shown in **bold**).

```javascript
function getProducts()
{
    var ServiceGateway = getService();
    var categoryID = $get("cmbProductCategory").value;
    var productName = $get("txtProductName").value;
    if( categoryID )
    {
        var queryStr = "/ProductCategory(" + categoryID + ")/Product";
        if (productName)
        {
            queryStr += "?$filter=indexof(Name,'" + productName + ") gt -1 or '' eq "" + productName + """;
        }
        //TODO: Call the data service
    }
    else
    {
        alert('Please select a category');
    }
}
```

2. Retrieve the Products from the data service by calling the query method of the data service class. To do this, add to `getProducts()` function the following content (shown in **bold**).

```javascript
function getProducts()
{
    var ServiceGateway = getService();
    var categoryID = $get("cmbProductCategory").value;
    var productName = $get("txtProductName").value;
    if( categoryID )
    {
        var queryStr = "/ProductCategory(" + categoryID + ")/Product";
        if( productName )
        {
            queryStr += "?$filter=indexof(Name,'" + productName + ") gt -1 or '' eq "" + productName + """;
        }
        ServiceGateway.query(queryStr, getProductsSuccess, genericFailure);
    }
    else
    {
```
alert('Please select a category');
}
}

**Note:** The query function signature is:

\[
\text{query: function(query, succeededCallback, failedCallback, userContext, webRequest)}
\]

And the arguments detail is:

- **query:** Path to query.
- **succeededCallback:** Callback to execute upon successful completion of the operation.
- **failedCallback:** Callback to execute upon unsuccessful completion of the operation.
- **userContext:** A context object associated with this operation.
- **webRequest:** A WebRequest object to use for this operation.

**Note:** Notice that the `getCategories()` function in `ProductGateway.js` also makes a query and also applies the `orderby` query option: "/[ProductCategory?$orderby=Name". This will retrieve all the `ProductCategory` items ordered by the `Name` attribute.

---

**Task 3 – Inserting Entities Using the Data Service**

In this task you will create a product object that will be sent to the data service to make an insert in the database. You will also implement a callback function that is executed every time the insert method succeeds.

1. Create the object representing the new product that will be sent to the data service. To do this, add to the `insertProduct()` function (located in `ProductGateway.js` file) the following code.

```javascript
function insertProduct()
{
    var ServiceGateway = getService();
    var newProduct =
    {
        Name: $get("txtName").value,
        ProductNumber: $get("txtProductNumber").value,
        Color: $get("txtColor").value,
        StandardCost: $get("txtStandardCost").value,
```
2. Insert the new product by calling the insert method of the data service gateway. To do this, add to the `insertProduct()` function the following code.

```javascript
function insertProduct()
{
  var ServiceGateway = getService();
  var newProduct = {
    Name: $get("txtName").value,
    ProductNumber: $get("txtProductNumber").value,
    Color: $get("txtColor").value,
    StandardCost: $get("txtStandardCost").value,
    ListPrice: $get("txtListPrice").value,
    Size: $get("txtSize").value,
    Weight: $get("txtWeight").value,
    ProductCategory: {__metadata: {uri: "ProductCategory(" + $get("cmbPDCategory").value + ")"}},
    SellStartDate: new Date(),
    SellEndDate: new Date(),
    ModifiedDate: new Date()
  };

  //TODO: Call the data service
}
```

**Note:** The `Product` entity has a foreign key to `ProductCategory` that needs to be treated differently from the other attributes. Instead of setting a value to the foreign key `ProductCategoryId`, you had to define a pointer to the `ProductCategory` related entity in the form of an URI:

```
uri: "ProductCategory(" ProductCategoryId")"
```

This must be sent to the data service inside a `__metadata` element.
ServiceGateway.insert(newProduct, "/Product", insertProductSuccess, genericFailure);

Note: The insert function signature is:

    insert: function(item, resourceSetUri, succeededCallback, failedCallback, userContext, webRequest)

And the arguments detail is:

item: Item to insert.

resourceSetUri: Resource set into which the item should be inserted.

succeededCallback: Callback to execute upon successful completion of the operation.

failedCallback: Callback to execute upon unsuccessful completion of the operation.

userContext: A context object associated with this operation.

webRequest: A WebRequest object to use for this operation.

3. Implement the function that is executed every time a new product insertion succeeds. In this case you will refresh the products grid with the new items. To do this, replace the insertProductSuccess function with the following content.

   JavaScript
   function insertProductSuccess(result, context, operation)
   {
       getProducts();
   }

Task 4 – Updating Entities Using the Data Service

In this task you will make changes to an existing product that will then be sent to the data service to make an update in the database.

1. Get the full object to send and update its properties. To do this, add to updateProduct() function (located in ProductGateway.js file) the following content.

   JavaScript
function updateProduct()
{
    var ServiceGateway = getService();

    updatedProduct = Records[$get("txtSelectedIndex").value];
    updatedProduct.Name = $get("txtName").value;
    updatedProduct.ProductNumber = $get("txtProductNumber").value;
    updatedProduct.Color = $get("txtColor").value;
    updatedProduct.StandardCost = $get("txtStandardCost").value;
    updatedProduct.ListPrice = $get("txtListPrice").value;
    updatedProduct.Size = $get("txtSize").value;
    updatedProduct.Weight = $get("txtWeight").value;
    updatedProduct.ModifiedDate = new Date();

    //TODO: Call the data service
}

**Note:** In this case you are sending the full object to the data service, although there may be some fields that don't need to be updated, remaining unchanged during the transaction. There is also another approach, in which you can send the object only with the attributes that will be updated.

2. Update the product by calling the update method of the data service gateway. To do this, add to `updateProduct()` function the following content.

**JavaScript**

```javascript
function updateProduct()
{
    var ServiceGateway = getService();

    updatedProduct = Records[$get("txtSelectedIndex").value];
    updatedProduct.Name = $get("txtName").value;
    updatedProduct.ProductNumber = $get("txtProductNumber").value;
    updatedProduct.Color = $get("txtColor").value;
    updatedProduct.StandardCost = $get("txtStandardCost").value;
    updatedProduct.ListPrice = $get("txtListPrice").value;
    updatedProduct.Size = $get("txtSize").value;
    updatedProduct.Weight = $get("txtWeight").value;
    updatedProduct.ModifiedDate = new Date();

    ServiceGateway.update(updatedProduct, updateProductSuccess, genericFailure);
```
Task 5 – Removing Entities Using the Data Service

In this task you will delete products from the database using the data service.

1. Loop over the product list to retrieve all the selected rows and delete them by calling the remove method of the data service gateway. To do this, add to deleteProduct() function (located in ProductGateway.js file) the following content (shown in bold).

```javascript
function deleteProduct()
{
    var ServiceGateway = getService();
    for (i=0;i<Records.length;i++)
        {
            var chk = $get("chk_product_" + i);
            if (chk.checked)
                {
                    var product = Records[i];
                    ServiceGateway.remove(product, deleteProductSuccess, genericFailure);
                }
        }
}
```

**Note:** The remove function signature is:

```javascript
remove: function(item, succeededCallback, failedCallback, userContext, webRequest)
```
remove: function(item, succeededCallback, failedCallback, userContext, webRequest)

And the arguments detail is:

item: Item to remove.
succeededCallback: Callback to execute upon successful completion of the operation.
failedCallback: Callback to execute upon unsuccessful completion of the operation.
userContext: A context object associated with this operation.
webRequest: A WebRequest object to use for this operation.

**Next Step:**

**Verification**

To verify that you have correctly performed all steps of the exercises, proceed as follows.

In this verification you will use a web page to consume the Data Service. You will query, insert, update and delete products via the ASP.NET Ajax Library Beta’s WCF Data Services API.

1. To interact with the Data Service you need to start a new instance of the WebSite project. To do this, in Solution Explorer right-click the WebSite project, point to Debug and select Start New Instance. The following web page should appear:

   **Note:** If the solution throws a JScript error when using Internet Explorer 8, omit the exception and turn on Internet Explorer 8 Compatibility View.

   If the dialog Debugging Not Enabled appears, select Modify the Web.config file to enable debugging and click OK.
2. Query for products corresponding to Brakes category. To do this, select Brakes value in the Product Category drop down and click Search. The following page should appear:
3. Insert a new product. To do this, click **New Product** and type the following data in the modal popup extender.

   a. **Product Number**: ABS-9444  
   b. **Color**: Black  
   c. **Standard Cost**: 150  
   d. **Size**: 2  
   e. **Name**: ABS Brakes  
   f. **Category**: Brakes  
   g. **List Price**: 150  
   h. **Weight**: 320
Click **Save** to call the data service and insert the product. Notice how the product list refreshes so you can see the new inserted product.
4. Update the recently inserted product. To do this, click the row of the **ABS Brakes** product you have inserted in the previous step and wait for the popup modal extender to display the selected information. Update the following fields:

   a. **Product Number**: RIM-9444
   b. **Name**: Rim Brakes
Figure 20

*Updating a product*

Click **Save** to call the data service and update the product. You should see that when the product list refreshes, the recently updated product is updated.

**Note:** If you try adding a second Product, you will receive an error. This is because the product item has a GUID identifier that will be duplicated (it’s set to 0 by default.) On the next exercise, you will add a service interceptor to add the rowguid value.
5. Delete the inserted product. To do this, check the **Select** checkbox of the row to delete and click **Delete Product**.

*Figure 21*  
*Showing the updated product*
Deleting the product

After deleting the product list will be refreshed and the product will disappear.

Next Step:

Exercise 3: Extending Data Services with Service Operations and Interceptors
Exercise 3: Extending Data Services with Service Operations and Interceptors

There are scenarios where you need to add data validation rules or custom behaviors. In this exercise you will use **Service Interceptors** to add validation support and **Service Operations** to perform custom queries.

The scenario used in this exercise is very similar to the one in the previous exercise. You will retrieve a product list, but this time, using a **Service Operation** instead of a REST query and then add **Service Interceptors** to upgrade the *create* and *update* product functionality.

**Task 1 – Creating a Service Operation**

In this task you will create a service operation to retrieve a product list, add an access rule and call it from a JS script.

2. Open the solution file **DataServiceOperationsAndInterceptors.sln** located under the **Source\Ex03-ServiceInterceptors\begin** folder of this lab (choosing the folder that matches the language of your preference.)
3. Open the **AdventureWorks.svc.cs (C#)** or **AdventureWorks.svc.vb (Visual Basic)** file. To do this, in Solution Explorer, double-click the **AdventureWorks.svc.cs (C#)** or **AdventureWorks.svc.vb (Visual Basic)** file.
4. Add a service operation to retrieve products. To do this, in the **AdventureWorks.svc.cs** (C#) or **AdventureWorks.svc.vb** (Visual Basic) file add the following method.
**Note:** This method has a **WebGet** attribute because this API works over WCF. The Service Operation receives two parameters and returns an **IQueryable** interface. The method uses the **CurrentDataSource** property as data source and performs a LINQ query to retrieve the products within the specified category and with the defined product name, if the parameter is distinct from an empty string.

(Code Snippet – *Data Services Interceptors Lab – Service Operation CSharp*)

**C#**

```
[WebGet]
public IQueryable<Product> GetProducts(string productName, int productCategoryId)
{
    return from p in this.CurrentDataSource.Product
            where p.ProductCategory.ProductCategoryID == productCategoryId
            && (p.Name == productName || String.IsNullOrEmpty(productName))
            select p;
}
```

(Code Snippet – *Data Services Interceptors Lab – Service Operation VB*)

**Visual Basic**

```
<WebGet()> _
Public Function GetProducts(ByVal productName As String, ByVal productCategoryId As Integer) As IQueryable(Of Product)
    Return From p In Me.CurrentDataSource.Product _
            Where p.ProductCategory.ProductCategoryID = productCategoryId AndAlso
            (p.Name = productName OrElse [String].IsNullOrEmpty(productName)) _
            Select p
End Function
```

**Note:** To access the data you should use the **CurrentDataSource** property of the Service, that is the generic type received when declaring the Data Service class. Because when received as the generic type, the service registers the serializers through WCF. If you use a different data source, the answer will not be able to be serialized.
**Note:** By returning a query instead of the final data, the Data Service engine is able to apply additional operators (such as orderby or filter) to the query. If you want to avoid this, return an `IEnumerable` interface.

5. To be able to use the **Service Operation** you need to specify an **access rule**. To do so, place the following bolded code inside the `InitializeService` method (located in the `AdventureWorks.svc.cs` (C#) or `AdventureWorks.svc.vb` (Visual Basic) file).

   (Code Snippet – *Data Services Interceptors Lab – Access Rule C#*)

```
C#
public static void InitializeService(DataServiceConfiguration config)
{
    config.SetEntitySetAccessRule("*", EntitySetRights.All);
    config.SetServiceOperationAccessRule("GetProducts", ServiceOperationRights.All);
}
```

   (Code Snippet – *Data Services Interceptors Lab – Access Rule VB*)

```
Visual Basic
Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
    config.SetEntitySetAccessRule("*", EntitySetRights.All)
    config.SetServiceOperationAccessRule("GetProducts", ServiceOperationRights.All)
End Sub
```

You have configured the Service Operation *GetProducts* to have **All** access rights.

**Note:** Other possible access rights are **AllRead**, **None**, **ReadMultiple** and **ReadSingle**.

6. Add JavaScript code to call this **Service Operation**. To do this, in **Solution Explorer**, double-click `ProductGateway.js`, remove the code inside `getProducts` function and add the following code inside it.

**JavaScript**
```javascript
function getProducts() {
    var ServiceGateway = getService();
    var sb = new Sys.StringBuilder();
    sb.append("GetProducts?\n    sb.append("productName='");
    sb.append($get("txtProductName").value);
    sb.append("'&");
    sb.append("productCategoryId=");
    sb.append($get("cmbProductCategory").value);
    ServiceGateway.query(sb.toString(), getProductsSuccess, genericFailure);
}
```

**Note:** This method obtains a service gateway that is an instance of the `Sys.Data.AdoNetServiceProxy` class, and then builds a string to call the `GetProducts` Service Operation, adding the two parameters in the query string URL encoded. Finally it executes a query using the query string and two callbacks functions. One for success and the other one to be called in case of an error.

7. To verify this task, in solution explorer, press **F5** to run the solution. Then select the **Brakes** category from the **Product Category** drop down list and click **Search**. You should see an output similar to the following image.

![Product Search](image-url)

<table>
<thead>
<tr>
<th>Select</th>
<th>Name</th>
<th>ProductNumber</th>
<th>Color</th>
<th>ListPrice</th>
<th>Size</th>
<th>Weight</th>
<th>ModifiedDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rear Brakes</td>
<td>RB-9231</td>
<td>Silver</td>
<td>106.5000</td>
<td>317.00</td>
<td>2004-03-11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Front Brakes</td>
<td>FB-9873</td>
<td>Silver</td>
<td>106.5000</td>
<td>317.00</td>
<td>2004-03-11</td>
<td></td>
</tr>
</tbody>
</table>
**Figure 25**
*Using the Service Operation*

**Note:** If the solution throws a **JScript** error when using Internet Explorer 8, omit the exception and turn on Internet Explorer 8 **Compatibility View**.

8. Close the web browser to stop the application.

**Task 2 – Creating a Service Interceptor**

In this task you will create a Service Interceptor to add a required database field. It will add a row GUID to every updated product to be saved into the database.

1. Add a change interceptor to add a row GUID to every new product created. To do this, in the `AdventureWorks.svc.cs` file add the following method below the `GetProducts` method.

   (Code Snippet – *Data Services Interceptors Lab – Change Interceptor Add rowGuid CSharp*

   ```csharp
   [ChangeInterceptor("Product")]
   public void OnChangeProduct(Product product, UpdateOperations action)
   {
      if (action == UpdateOperations.Add)
      {
         product.rowguid = Guid.NewGuid();
      }
   }
   ```

   (Code Snippet – *Data Services Interceptors Lab – Change Interceptor Add rowGuid VB*

   ```vb
   <ChangeInterceptor("Product")> 
   Public Sub OnChangeProduct(ByVal product As Product, ByVal action As UpdateOperations)
      If action = UpdateOperations.Add Then
         product.rowguid = Guid.NewGuid()
      End If
   End Sub
   ```
Note: This method has a ChangeInterceptor attribute with a Product string as parameter to indicate that it will intercept all the changes to the Product entities. The Service Interceptor receives two parameters, one is the entity that will intercept and the other is the change action performed. It has no return value. The method checks the action and if it is an Add action, it adds a new GUID to the current Product entity. This fixes the duplicate products issue you have identified in the previous exercise.

2. Now you will add validation logic in the Change Interceptor you’ve inserted in the previous step, which will be applied to new and updated entities. You will also add a modified date to every product added or updated. To do this, add the following code in the OnChangeProduct method.

(Code Snippet – Data Services Interceptors Lab – Change Interceptor Product Color Validation CSharp)

**C#**

```csharp
[ChangeInterceptor("Product")] public void OnChangeProduct(Product product, UpdateOperations action) {
    if (action == UpdateOperations.Add || action == UpdateOperations.Change) {
        if (String.IsNullOrEmpty(product.Color))
            throw new DataServiceException("Product must have a color specified");

        DateTime fakeToday = new DateTime(2009, 03, 26);
        product.ModifiedDate = fakeToday;
    }

    if (action == UpdateOperations.Add) {
        product.rowguid = Guid.NewGuid();
    }
}
```

(Code Snippet – Data Services Interceptors Lab – Change Interceptor Product Color Validation VB)

**Visual Basic**

```vbnet
<ChangeInterceptor("Product")> _
Public Sub OnChangeProduct(ByVal product As Product, ByVal action As UpdateOperations)
```
If action = UpdateOperations.Add OrElse action = UpdateOperations.Change Then

    If String.IsNullOrEmpty(product.Color) Then
        Throw New DataServiceException("Product must have a color specified")
    End If

    Dim fakeToday = New DateTime(2009, 3, 26)
    product.ModifiedDate = fakeToday

End If

If action = UpdateOperations.Add Then
    product.rowguid = Guid.NewGuid()
End If
End Sub

Note: There exists a validation that the Color attribute from the Product entity cannot be null or an empty string. If so, an exception is thrown and the operation is aborted. The method also adds a modified date to every added or modified product.

3. To verify the above steps, press F5 to run the solution.

4. Edit an existing product to check for the color validation. To do this, select Brakes from the Product Category drop down list and then click Search.

Adventure Works

Product Search

Product Category
Brakes

Product Name

Search

<table>
<thead>
<tr>
<th>Select</th>
<th>Name</th>
<th>ProductNumber</th>
<th>Color</th>
<th>ListPrice</th>
<th>Size</th>
<th>Weight</th>
<th>ModifiedDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rear Brakes</td>
<td>RB-9231</td>
<td>Silver</td>
<td>106.5000</td>
<td>317.00</td>
<td></td>
<td>2004-03-11</td>
</tr>
<tr>
<td></td>
<td>Front Brakes</td>
<td>FB-9873</td>
<td>Silver</td>
<td>106.5000</td>
<td>317.00</td>
<td></td>
<td>2004-03-11</td>
</tr>
</tbody>
</table>

New Product  Delete Product
5. Click the first row listed where the Name is Rear Brakes. You should see the following popup window.

![Figure 26]
Listing brakes

![Figure 27]
Editing Rear Brakes

6. Empty the Color field and click Save. The following alert should appear.
Figure 28

*Update method is failing*

**Note:** The alert is showing the error message that is received as parameter from the callback event handler in the script.

When using Internet Explorer 8 in Windows 7, the exception may not be caught by JavaScript and will be displayed in Visual Studio as an unhandled exception.

7. Edit the same existing brake from the grid. To do this, click on the first row listed. Set the *Size* to 3 and the *Color* to *Red*. Click *Save* to commit the changes. Notice that the grid that has been updated. The modified date is 2009-03-26; that is the value set in the *Service Interceptor*. 

Page 53
8. Close the web browser to stop the application.

**Task 3 – Using Service Interceptors to Add Custom Query Constraints**

In this task you will add a custom constraint when querying Product Category entities. The product categories are retrieved to fill the Product Category drop down list.

1. Add the following using directive in the **AdventureWorks.svc.cs (C#)** or **AdventureWorks.svc.vb (Visual Basic)** file

   **C#**
   ```
   using System.Linq.Expressions;
   ```

   **Visual Basic**
   ```
   Imports System.Linq.Expressions
   ```

2. Add a query interceptor to just let retrieve the categories that start with the ‘B’ letter. To do this, add the following code below the **OnChangeProduct** method in the **AdventureWorks.svc.cs (C#)** or **AdventureWorks.svc.vb (Visual Basic)** file.
3. To verify this step, press **F5** to run the solution.

4. Expand the **Product Category** listing and check that only the product categories starting with the 'B' letter are shown.
5. Close the web browser to stop the application.

Next Step:

Exercise 4: Adding Client-Side Paging with Row Count

Exercise 4: Adding Client-Side Paging with Row Count

In this exercise you will learn how to add client-side paging to a list of items.

The client-side differs from the Server Driven Paging on who is responsible for retrieving the items in a paginated fashion. On the client-side paging, it is the client application the one that controls the quantity of records retrieved from the server and how many should be skipped on each request, limiting the server responsibility only to the retrieval of the records requested on the client's queries. On the other hand, the server-side paging is done entirely by the server application, which limits the quantity of records returned on each request and providing the client with an URI for it to know how to continue retrieving the following pages.
Task 1 – Invoking the Count Method

In this task you will add a new method to the ProductGateway that will retrieve the count of products that matches the criteria entered by the user, using the now supported Count extension method.


2. Open the solution file DataServicesRowCount.sln located under the Source\Ex04-RowCount\begin folder of this lab (choosing the folder that matches the language of your preference.) The solution file contains the UserInterface and WebSite projects created in Exercise 1 of the WCF Data Service Lab.

   Note: Alternatively, you may continue working with the solution obtained after completing the Exercise 1 of the Lab.

3. Navigate to the IProductGateway.cs (C#) or IProductGateway.vb (Visual Basic) file located in the UserInterface project under the Gateways folder.
Figure 31

IProductGateway.cs file (C#)
4. Add a new method that will retrieve the count of products that matches the criteria entered by the user. To do this, add the following code to the `IProductGateway` interface.

**C#**

```csharp
int GetProductsCount(string productName, ProductCategory category);
```

**Visual Basic**

```vbnet
Function GetProductsCount(ByVal productName As String, ByVal category As ProductCategory) As Integer
```

5. Implement the new method on the `ProductGateway` class. To do this, open the `ProductGateway.cs (C#)` or `ProductGateway.vb (Visual Basic)` file and add the following code.

**Note:** In previous versions of WCF Data Services, the invocation to the `Count` method on a `DataServiceQuery` object would have caused a `NotSupportedException` being thrown. From
version 1.5 **Count** and **LongCount** are implemented and mapped to the $count segment on the corresponding entity set. It is worth to highlight that in this case, no entity set is retrieved from the database, just the count operation being addressed by the database engine.

In previous versions, client-side paging was achieved by retrieving the complete set, and invoking the **Count** method on the resulting enumeration. This was far more expensive as the whole set needed to be retrieved from the database in order to execute the Count operation directly to that enumeration, which in change, is less performant than the Count operation addressed by the database engine.

(Code Snippet – *Data Services Lab - ProductGateway GetProductsCount method CSharp*)

```csharp
public int GetProductsCount(string productName, ProductCategory category) {
    var query = (DataServiceQuery<Product>)
        (from p in this.context.CreateQuery<Product>("Product")
         where ((p.Name.IndexOf(productName) > -1 || string.IsNullOrEmpty(productName)) &&
         select p);

    return query.Count();
}
```

(Code Snippet – *Data Services Lab - ProductGateway GetProductsCount method VB*)

```visualbasic
Public Function GetProductsCount(ByVal productName As String, ByVal category As ProductCategory) As Integer Implements IProductGateway.GetProductsCount
    Dim query = DirectCast(
        (From p In Me.context.CreateQuery(Of Product)("Product")
         Where ((p.Name.IndexOf(productName) > -1 OrElse String.IsNullOrEmpty(productName)) AndAlso p.ProductCategory.ProductCategoryID = category.ProductCategoryID)
         Select p), DataServiceQuery(Of Product))

    Return query.Count()
End Function
```
Note: You may notice that the `GetProductsCount` method is managing the query differently than the `GetProducts` method, using a LINQ query to retrieve the products set instead of creating the query URI directly. The queries can be done using both approaches, as LINQ queries will be transformed into an URI by the WCF Data Services framework.

Task 2 – Displaying the Product Count on the User Interface

In this task you will add a `TextBlock` to the product list window to display the product count using the method created in the previous task.

1. Add a new `TextBlock` to the `ProductList` window to show the count of products that matches the filter criteria. To do this, open the `ProductList.xaml` file using XML Editor, and add the following highlighted code below the `<ListView>` control.

   **XAML**
   ```xml
   ...            
   <GridViewColumn Header="Weight"        
      DisplayMemberBinding="{Binding Path=Weight}" />
   </GridView>
   </ListView.View>
</ListView>
<TextBlock Name="TotalProductsCountLabel" Grid.Row="3" 
Margin="15,0,16,5" VerticalAlignment="Center" HorizontalAlignment="Left" />
</Listview>  
<Button Grid.Row="3" HorizontalAlignment="Right" Margin="0,0,120,10" 
   x:Name="btnNewProduct" Click="BtnNewProduct_Click" VerticalAlignment="Bottom" 
   d:LayoutOverrides="Height" Width="100" Content="New Product"/>
<Button HorizontalAlignment="Right" Margin="0,0,10,10" 
   x:Name="btnDeleteProduct" Width="100" Grid.Row="3" 
   Click="BtnDeleteProduct_Click" VerticalAlignment="Bottom" Content="Delete Product"/>
</Grid>
...```

2. Add a new field to store the products count. To do this, open the `ProductList.xaml.cs (C#)` or `ProductList.xaml.cs (Visual Basic)` file, and add a new field inside the `ProductList` class as shown in the following code.

   **C#**
   ```csharp
   private int productSetSize;
   ```

   **Visual Basic**
   ```vb
   Private Integer productSetSize
   ```
3. Add a new method for retrieving the products count and showing it on the new **TextBlock**. To do this, add the following method to the **ProductList** class.

(Code Snippet – **Data Services Lab - ProductList RecalculateProductsSetSize method CSharp**)

```csharp
private void RecalculateProductsSetSize()
{
    ProductGateway gateway = new ProductGateway();
    this.productSetSize = gateway.GetProductsCount(NameTextBox.Text, CategoryComboBox.SelectedItem as ProductCategory);
    this.TotalProductsCountLabel.Text = string.Format(CultureInfo.CurrentCulture, "Total Products in Category: {0}", this.productSetSize);
}
```

(Code Snippet – **Data Services Lab - ProductList RecalculateProductsSetSize method VB**)

```vbnet
Private Sub RecalculateProductsSetSize()
    Dim gateway As New ProductGateway()
    Me.productSetSize = gateway.GetProductsCount(NameTextBox.Text, TryCast(CategoryComboBox.SelectedItem, ProductCategory))
    Me.TotalProductsCountLabel.Text = String.Format(CultureInfo.CurrentCulture, "Total Products in Category: {0}", Me.productSetSize)
End Sub
```

4. Add an invocation for the new method. To do this, add the following code to the existing **BtnSearch_Click**, **BtnDeleteProduct_Click** and **Window_Closed** methods.

(CODE SNIPPET – **Data Services Lab - ProductList RecalculateProductsSetSize method CSharp**)

```csharp
private void BtnSearch_Click(object sender, RoutedEventArgs e)
{
    this.RestartItems();
    this.BindProducts();
}
```

```vbnet
Private Sub BtnSearch_Click(ByVal sender As System.Object, ByVal e As RoutedEventArgs)
    Me.RestartItems()
    Me.BindProducts()
End Sub
```
if (p != null)
{
    ProductGateway gateway = new ProductGateway();
    gateway.DeleteProduct(p);
    this.BindProducts();
    this.RecalculateProductsSetSize();
}
}
private void Window_Closed(object sender, EventArgs e)
{
    this.BindProducts();
    this.RecalculateProductsSetSize();
}

Visual Basic
Private Sub BtnSearch_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Me.RecalculateProductsSetSize()
        Me.BindProducts()
    End Sub

Private Sub BtnDeleteProduct_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim p As Product = TryCast(ProductsListView.SelectedItem, Product)
    If p IsNot Nothing Then
        Dim gateway As New ProductGateway()
        gateway.DeleteProduct(p)
        Me.BindProducts()
        Me.RecalculateProductsSetSize()
    End If
End Sub

Private Sub Window_Closed(ByVal sender As Object, ByVal e As EventArgs)
    Me.BindProducts()
    Me.RecalculateProductsSetSize()
End Sub

5. Verify that the products count is being shown on the UI. To do this, set the UserInterface as StartUp Project, and press F5 to run the application. Select a category on the Product Category combo box and click on Search. You should see something similar to the following.
Task 3 - Adding Paging to Existing Product List Query

In this task you will modify the method that retrieves the product list to support pagination.

1. Add `pageSize` and `pageNumber` arguments to the existing `GetProducts` method in the `IProductGateway` interface. To do this, navigate once again to the `IProductGateway.cs (C#) or IProductGateway.vb (Visual Basic)` file and replace the `GetProducts` method declaration with the following code.

(Code Snippet – Data Services Lab - IProductGateway GetProducts method CSharp)

```csharp
IList<Product> GetProducts(string productName, ProductCategory category, int pageSize, int pageNumber);
```

(Code Snippet – Data Services Lab - IProductGateway GetProducts method VB)
2. Add `pageSize` and `pageNumber` arguments to the existing implementation of the `GetProducts` method in the `ProductGateway` class. To do this, replace the current method's signature with the following code.

   ```csharp
   public IList<Product> GetProducts(string productName, ProductCategory category, int pageSize, int pageNumber)
   ```

3. Use page size and number arguments in the query. To do this, change the implementation of the `GetProducts` method in the `ProductGateway` class as follows.

   ```csharp
   public IList<Product> GetProducts(string productName, ProductCategory category, int pageSize, int pageNumber)
   {
       int categoryId = category.ProductCategoryID;

       IEnumerable<Product> products = this.context.Execute<Product>(
           new Uri(this.context.BaseUri.ToString() +
           '/../ProductCategory(' + categoryId +
           ')/Product?$filter=indexof(Name,"' + productName + '") gt -1 or '' eq '' +
           productName + ''&$top=' + pageSize + ''&$skip=' + pageSize * pageNumber));

       List<Product> productsSet = new List<Product>();
       foreach (Product p in products)
       {
           this.context.LoadProperty(p, "ProductCategory");
           productsSet.Add(p);
       }

       return productsSet;
   }
   ```
Visual Basic

Public Function GetProducts(ByVal productName As String, ByVal category As ProductCategory, ByVal pageSize As Integer, ByVal pageNumber As Integer) As IList(Of Product) Implements IProductGateway.GetProducts
    Dim categoryId = category.ProductCategoryID
    Dim products = Me.context.Execute(Of Product)(
        New Uri(Me.context.BaseUri.ToString() & "/ProductCategory(" & categoryId & ")/Product?$filter=indexof(Name,’" & productName & ") gt -1 or ’’ eq ’’ & productName & ’’&$top=" & pageSize & ”&$skip=" & pageSize * pageNumber))
    Dim productsSet As New List(Of Product)()
    For Each p In products
        Me.context.LoadProperty(p, "ProductCategory")
        productsSet.Add(p)
    Next
    Return productsSet
End Function

Note: Paging is done by setting the $top and $skip query string options. When setting the $top=n option the number of entities returned is restricted to the first n rows, while with the $skip=m option will cause that the first m records should be skipped from the result set.

Task 4 - Updating the User Interface to Show the Paged ListView

In this task you will add the User Interface controls that will allow the user to navigate through the products list pages.

1. Add a new row to the root Grid in ProductList.xaml.

XAML

```xaml
<Grid x:Name="LayoutRoot">
    <Grid.RowDefinitions>
        <RowDefinition Height="55"/>
        <RowDefinition Height="90"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
</Grid>
```
2. Add the **Previous** and **Next** controls that will allow the user to navigate through the product list pages, and place it within the new row. To do this, open **ProductList.xaml** file with **XML Editor**, and add the following highlighted code below the **TextBlock** control.

### XAML

```xml
...<TextBlock Name="TotalProductsCountLabel" Grid.Row="3" Margin="15,0,16,5" VerticalAlignment="Center" HorizontalAlignment="Left" />
<StackPanel Orientation="Horizontal" Grid.Row="4" Height="23" Margin="0,0,16,5" HorizontalAlignment="Right">
  <Button Name="PreviousPageButton" Width="75" Click="previousPageButton_Click">Previous</Button>
  <TextBlock Name="CurrentPageLabel" Width="90" Margin="15,0,15,0" VerticalAlignment="Center" Text="Current Page: 1" />
  <Button Name="NextPageButton" Width="75" Click="nextPageButton_Click">Next</Button>
</StackPanel>
...```

3. Add a new constant to fix the size of the pages on the **ProductList** class. To do this, add the following code to the **ProductList.xaml.cs (C#)** or **ProductList.xaml.vb (Visual Basic)** file.

### C#

```csharp
private const int PageSize = 8;
```

### Visual Basic

```vbnet
Private Const PageSize As Integer = 8
```

**Note:** The page size for this exercise was arbitrarily set and can be changed to verify different paging configurations.

4. Add a new field to store the current page on the **ProductList** class. To do this, add the following code.
5. Add a new read-only property to calculate the total number of pages to the **ProductList** class. To do this, add the following code.

6. (Code Snippet – *Data Services Lab - ProductList TotalPages property CSharp*)

```csharp
private int TotalPages
{
    get
    {
        return (this.productSetSize / PageSize) + (this.productSetSize % PageSize > 0 ? 1 : 0);
    }
}
```

(Code Snippet – *Data Services Lab - ProductList TotalPages property VB*)

```vbnet
Private ReadOnly Property TotalPages As Integer
    Get
        Return (Me.productSetSize / PageSize) + (If(Me.productSetSize Mod PageSize > 0, 1, 0))
    End Get
End Property
```

7. Add the handler methods for the **Click** event of the new buttons. To do this, add the following code to the **ProductList** class.

(Code Snippet – *Data Services Lab - ProductList Navigation Buttons Handlers CSharp*)

```csharp
private void previousPageButton_Click(object sender, RoutedEventArgs e)
{
```
8. Add the logic to handle whereas the navigation buttons are enabled or disabled depending on the current page. To do this, add the following method to the `ProductList` class.

(CODE SNIPPET – Data Services Lab - ProductList UpdateNavigationButtons method CSharp)

```csharp
private void UpdateNavigationButtons()
{
    this.PreviousPageButton.IsEnabled = (this.currentPageNumber > 0);
    this.CurrentPageLabel.Text = string.Format(CultureInfo.CurrentUICulture, "Current Page: {0}", this.currentPageNumber + 1);
    this.NextPageButton.IsEnabled = this.currentPageNumber < (this.TotalPages - 1);
}
```

(CODE SNIPPET – Data Services Lab - ProductList UpdateNavigationButtons method VB)

```vbnet
Private Sub UpdateNavigationButtons()
    Me.PreviousPageButton.IsEnabled = (Me.currentPageNumber > 0)
    Me.CurrentPageLabel.Text = Me.currentPageNumber + 1
    Me.NextPageButton.IsEnabled = Me.currentPageNumber < (Me.TotalPages - 1)
End Sub
```
9. Update the **BindProducts** method in **ProductList** class to invoke the new **GetProducts**, and **UpdateNavigationButtons** methods. To do this, replace the current implementation of the **BindProducts** method with the following highlighted code.

**C#**

```csharp
private void BindProducts()
{
    if (CategoryComboBox.SelectedIndex > -1)
    {
        ProductGateway gateway = new ProductGateway();
        ProductsListView.ItemsSource = gateway.GetProducts(NameTextBox.Text,
                                     CategoryComboBox.SelectedItem as ProductCategory,
                                     PageSize,
                                     this.currentPageNumber);
        this.UpdateNavigationButtons();
    }
}
```

**Visual Basic**

```vbnet
Private Sub BindProducts()
    If CategoryComboBox.SelectedIndex > -1 Then
        Dim gateway As New ProductGateway()
        ProductsListView.ItemsSource = gateway.GetProducts(NameTextBox.Text,
                                                   TryCast(CategoryComboBox.SelectedItem, ProductCategory), PageSize,
                                                   Me.currentPageNumber)
        Me.UpdateNavigationButtons()
    End If
End Sub
```

10. Set the initial state of the navigation buttons by invoking the **UpdateNavigationButtons** method on the **Window_Loaded** handler method as follows.

**C#**

```csharp
private void UpdateNavigationButtons()
{
    Me.PreviousPageButton.IsEnabled = (Me.currentPageNumber > 0)

    Me_CURRENTPAGELABEL.Text = String.Format(CultureInfo.CurrentUICulture,
                                      "Current Page: {0}", Me.currentPageNumber + 1)

    Me.NextPageButton.IsEnabled = Me.currentPageNumber < (Me.TotalPages - 1)
}
```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    this.BindCategories();
    this.UpdateNavigationButtons();
}

Visual Basic
Private Sub Window_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Me.BindCategories()
    Me.UpdateNavigationButtons()
End Sub

11. Add the code to reset the current page field when a new search is done. To do this, add the following code to the  **BtnSearch_Click**  method.

C#
private void BtnSearch_Click(object sender, RoutedEventArgs e)
{
    this.currentPageNumber = 0;
    this.RecalculateProductsSetSize();
    this.BindProducts();
}

Visual Basic
Private Sub BtnSearch_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Me.currentPageNumber = 0
    Me.RecalculateProductsSetSize()
    Me.BindProducts()
End Sub

12. Invoke the new  **GetProducts**  method in  **ProductView**  class. To do this, in  **Solution Explorer**  double-click the  **ProductView.xaml.cs (C#)**  or  **ProductView.xaml.vb (Visual Basic)**, and add the following highlighted code to the  **UpdateProduct**  method.

C#
public void UpdateProduct(Product product)
{
    ProductGateway gateway = new ProductGateway();
}
this.Product = gateway.GetProducts(product.Name, product.ProductCategory, 1, 0)[0];
this.FormCreateMode = false;
this.Title = "Edit " + product.Name;
}

Visual Basic
Public Sub UpdateProduct(ByVal product As Product)
    Dim gateway As New ProductGateway()
    Me.Product = gateway.GetProducts(product.Name, product.ProductCategory, 1, 0)(0)
    Me.FormCreateMode = False
    Me.Title = "Edit " & product.Name
End Sub

Note: On a real application, a new method would be created when querying for a single product. However, to keep the exercise code simple, the GetProducts method is invoked here with a page size equal to one, and a current page equal to zero (where 0 represents the first page number), and the first element of the returned set is being taken.

Next Step:
Verification

Verification
In this verification you will see how paging works, allowing the user to navigate through the list of products that matches the search criteria entered by the user. You will also see how the products count label and the navigation buttons behave when adding and removing products.

1. Start a new instance of the UserInterface project. To do this, in Solution Explorer right-click UserInterface project, point to Debug and select Start New Instance.
2. Select a category that has several products (i.e. Mountain Bikes) and click Search.

3. Verify that the total products count label shows up and displays the number of products that belong to the selected category.
4. Verify that navigation is functional by clicking **Next** and **Previous** buttons.

5. Verify that when adding a new product pages and the count label are updated. To do this, click on **New Product**, complete the required fields and click on **Save**.

**Note:** The application is not validating user input, so make sure to type coherent values, specially the date (for example: **1/1/2009 12:00:00 AM**).
6. Verify that when deleting a product the pages and the count label are updated. To do this, delete the newly created product by selecting it from the list and clicking on **Delete Product**.

**Next Step:**

**Summary**

In this lab you have created and consumed WCF (ADO.NET) Data Services using the .NET API and the ASP.NET Ajax Library Beta API. You have also added Service Operation and Interceptors into the WCF
Data Service. Additionally, you have been introduced to the new **Row Count** feature in the WCF Data Services release.