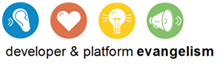
* 1. 

Demo Script

WCF (ADO.NET) Data Services 10-in-1

* 1. Lab version: 1.0.0
  2. Last updated: 12/30/2010
  3. 

1. Contents

[Overview 3](#_Toc281482437)

[Key Messages 3](#_Toc281482438)

[Key Technologies 3](#_Toc281482439)

[Time Estimates 4](#_Toc281482440)

[Setup and Configuration 4](#_Toc281482441)

[Demo Flow 4](#_Toc281482442)

[Opening Statement 5](#_Toc281482443)

[Step-by-Step Walkthrough 6](#_Toc281482444)

[Segment #1 – Exploring Row Count API 7](#_Toc281482445)

[Segment #2 – Setting Server-Driven Paging 10](#_Toc281482446)

[Segment #3 – Configuring Friendly Feeds 12](#_Toc281482447)

[Segment #4 – Surfacing Containment Entities 14](#_Toc281482448)

[Segment #5 – Implementing the Enhanced BLOB Support 17](#_Toc281482449)

[Summary 21](#_Toc281482450)

Overview

* 1. This document provides setup documentation, systematic instructions, and a written script for showing WCF Data Services (formerly known as ADO.NET Data Services) features and code. This document can also serve as a tutorial or walkthrough of the exposed technologies.
  2. The WCF Data Services consists of a combination of patterns and libraries that enable the creation and consumption of data services for the web.
  3. This demo shows the set of new features in WCF Data Services. The demo consists of several scenarios showing a single feature in the easiest possible way.

### Key Messages

* 1. WCF Data Services is a component of the .NET Framework that enables you to create services that use the Open Data Protocol (OData) to expose and consume data over the Web or intranet by using the semantics of representational state transfer (REST). OData exposes data as resources that are addressable by URIs. Data is accessed and changed by using standard HTTP verbs of GET, PUT, POST, and DELETE.
  2. WCF Data Services expose OData feeds regardless of the underlying data source. After WCF Data Services accepts an HTTP request for a resource that a URI identifies, the request is deserialized and a representation of that request is passed to an WCF Data Services provider. This provider translates the request into a data source-specific format and executes the request on the underlying data source. WCF Data Services achieves storage independence by separating the conceptual model that addresses resources prescribed by OData from the specific schema of the underlying data source.

### Key Technologies

This demo uses the following technologies:

* 1. Microsoft Visual Studio 2010

### Time Estimates

* + Estimated time to complete the demo: 60 min

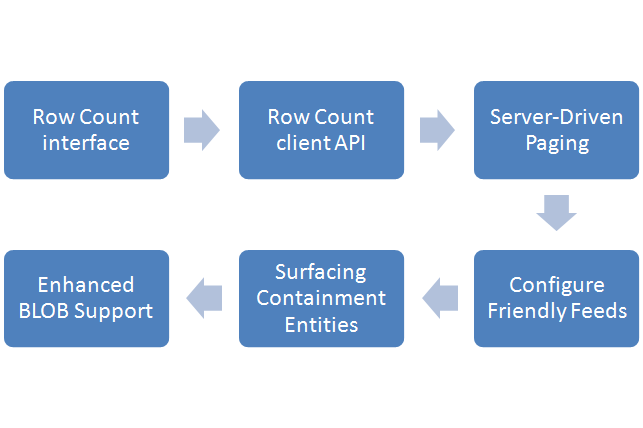
Setup and Configuration

## System Requirements

* + Microsoft Visual Studio 2010
  + Microsoft SQL Server 2005 or Microsoft SQL Server 2008 (Express edition or above)

**Note:** The browser might display a message when navigating the WCF Data Service 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**.

Demo Flow

1. The following diagram illustrates the high-level flow for this demo and the steps involved:
   1. 

Opening Statement

* 1. Today I would like to walk you through several demo applications built to show the new capabilities of WCF Data Services. The demo relies on the .NET Framework 4 release and represents the latest version of the WCF Data Service technology.
  2. The following is a brief description of the enhancements included in the latest release that will be shown during this demo:
  3. Two new capabilities related to row count calculations are included: the $count pseudo-selector and a new query option called $inlinecount. Both options work similarly, but they target different scenarios. When you target an entity-sets $count, you get back only the count of items in that set. This is nice if you already have the data and are only concerned with determining the server-side count. In addition, the new query option called $inlinecount allows you to query for data and then optionally include the count inside the response.
  4. In addition, the client API is enhanced to take advantage of these options by adding new methods and properties to the client libraries.
  5. When a client requests an entity set that potentially contains many items on the server, the service can enforce paging on the request and send back only the records it wants to give the client. This allows the consumer to continue making the same resource-centric requests they are familiar with, but it allows to server to “guide” the client down the path of success for both ends.
  6. WCF Data Services introduces a feature called “friendly feeds” that allows you to map an entity property to an element within the APP entry. This can be either a pre-defined element such a title or author, or a custom element.
  7. You will be able to separate an entity with binary data into two pieces: a media resource and a media link entry. The media resource represents the actual binary data, as well as its content type. The media link entry represents the metadata and additional information that goes along with the binary data. This makes it possible to query the two individually.

Step-by-Step Walkthrough

1. This demo is composed of the following segments:
   1. Exploring Row Count API
   2. Setting Server-Driven Paging
   3. Configuring Friendly-Feeds
   4. Surfacing Containment Entities
   5. Implementing the Enhanced BLOB Support

### Segment #1 – Exploring Row Count API

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **AdoNetDataServices1510In1.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Right click on the **01 - Row Count\RowCountService.svc** file and click **View in Brower**. | * + The row count feature is enabled automatically, so from a server perspective, there is nothing you have to do to your data service to begin using it. There also is no distinction between using row count with either a CLR or EF provider.   + There are, however, two new query parameters that you will need to take advantage of this feature: $count, and $inlinecount. |  |
| * 1. Edit the browser address textbox to browse to **/RowCountService.svc/Products/$count**. | * + $count is actually a pseudo-member (like $value) that allows you to request the record count of a specific entity set. You use it by appending it after an entity set in your request URI.   + The previous example would return the number of products that exist in your data model.   + This option works great if all you want is the record count, but chances are you will likely want both the count and the actual data. |  |
| * 1. Edit the browser address textbox to browse to **/RowCountService.svc/Products?$inlinecount=allpages**. | * + This is where the $inlinecount parameter comes in.   + You can add the $inlinecount query parameter to a query and it will include the count of the targeted entity set along with the response.   + The previous example would return both product data as well as the total count of products.   + This becomes useful when used in conjunction with server-driven paging (as is used in the RowCountService.svc file) because you might only get back 20 products but you want to know how many products exist on the server. |  |
| * 1. Right click on the **01 - Row Count\Client.aspx** file and click **View Code**.   2. Point to the **context.Products.IncludeTotalCount()** call in the LINQ query.   3. Highlight the code below the query to show the use of the **TotalCount** property of the QueryOperationResponse.   4. Edit the service **Uri** in the creation of **RowCountContext** with the correct port number.   5. Right click on the **01 - Row Count\Client.aspx** file and click **View in Browser**. | * + There are client APIs for the row count feature that abstract the need to manually use either $count or $inlinecount when requesting data from a data service.   + The DataServiceQuery class includes the new IncludeTotalCount, which will be equivalent to add the $inlinecount=allpages in the query.   + In addition, the QueryOperationResponse class has a new TotalCount method that allows you to retrieve the count that was returned from a request with an $inlinecount appended.   + Count and LongCount are the client API equivalent to $count, and allow you to immediately execute a query that is requesting the record count of an entity set. |  |

### Segment #2 – Setting Server-Driven Paging

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **AdoNetDataServices1510In1.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Right click on **02 - Server-Driven Paging\ServerDrivenPagingService.svc** file and click **View Code**. | * + The server-driven paging feature revolves around a new method that was added to the DataServiceConfiguration class. In order to "configure" server-driven paging on a specific entity set, you simply call the SetEntitySetPageSize of the DataServiceConfiguration class, within the InitializeService method.   + You can see this in use in the ServerDrivenPagingService.svc.cs file.   + There are no distinctions between how you configure server-driven paging for a CLR or EF provider. From a server perspective, once you've called SetEntitySetPageSize for every entity set you wish to page, you are done |  |
| * 1. Right click on the **02 - Server-Driven Paging\ServerDrivenPagingService.svc** file and click **View in Brower**.   2. Edit the browser address textbox to browse to **/ServerDrivenPagingService.svc/Products**.   3. Collapse all the **<entry>** elements and highlight the **<link>** element at the bottom of the page. | * + If you open a browser and request an entity set that has been page-restricted, you will only be able to get back at most the number of records specified in the server-side page.   + In this case, the limit is 20 entries per page.   + If more pages of data exist then a <link> element (named "next") will be included in the AtomPub that provides a link to the next page |  |
| * 1. Highlight the **$skiptoken** parameter. | * + The **$skiptoken** parameter is simply specifying the key values used to skip ahead to the requested page.   + In the above example, it is requesting all products whose id is higher than **724**.   + The generation of the next links is done automatically for us by the data service. There is no automatic generation of "previous links", so the server-driven paging is forward only. |  |

### Segment #3 – Configuring Friendly Feeds

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **AdoNetDataServices1510In1.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Right click on **03 - Friendly Feeds/CLR/** **FriendlyFeedObjectModel.cs (C#)** or **FriendlyFeedObjectModel.vb (Visual Basic)** file and click **View Code**.   4. Highlight the **EntityPropertyMapping** attributes decorating the **Products** class definition.   5. Highlight the **Name** and **Author** property mappings.   6. Highlight the **Price** property mapping.   7. Highlight the **Category/Name** property mapping. | * + By default, when WCF Data Services exposes an instance of an entity type, it serializes the data using the Atom Publishing Protocol (AtomPub/APP) format. Every public property on the entity type is mapped to an element within the content of the respective entry element.   + While this default behavior works just fine for many situations, there are some oddities when using APP. For instance, the APP format requires that every entry include a **title** and an **author**. WCF Data Services will render these elements, but never actually fill them with content. This could confuse consumers of the service that are APP aware and would expect to be provided with a title and/or author.   + WCF Data Services introduces a feature called “friendly feeds” that allows you to map an entity property to an element within the APP entry.   + This can be either a pre-defined atom element such a title or author, or a custom element, like the price element in this example.   + It can also be a complex object property as well, like the Category example in the following code. |  |
| * 1. Right click on the **03 - Friendly Feeds/CLR/FriendlyFeeds.svc** file and click **View in Brower**.   2. Edit the browser address textbox to browse to **/FriendlyFeeds.svc/Products**.   3. Highlight the **<title>** and **<author>** elements in the beginning of the first product entry.   4. Highlight the two custom elements **<money:price>** and **<category:category>** at the bottom of the first product entry. | * + The ability to map properties to custom elements allows you to add additional information to your data feeds, such as microformats (i.e. GeoRSS), that can be interpreted by understanding clients.   + If you open and request the list of Product entities, you will notice that the **<title>** element is mapped to the **Name** property of the Product class.   + In addition, the **<name>** element inside the **<author>** element is mapped to the Author property of the Product class.   + Custom elements can be generated in the entry payload by using the EntityPropertyMapping attribute and complex type properties.   + You can achieve this by simply decorating the entity class with the EntityPropertyMapping attribute with no changes to the model or the service itself. |  |

### Segment #4 – Surfacing Containment Entities

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **AdoNetDataServices1510In1.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Right click on the **04 - Containment\CLR\ContainmentService.svc** file and click **View in Brower**. | * + WCF Data Services provides a RESTful interface that makes it possible to surface different types of data from different types of clients.   + In this case, we will use a browser as a client and we prepared a WCF Data Service exposing various entity sets based on plain CLR classes related to each other.   + By using WCF Data Services, we are able to publish data in AtomPub Protocol that is a way of exposing a collection of feeds.   + In this example, each entity set is shown as part of the collection. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Products**.   2. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers**. | * + We can query those feeds like **Products** or **Customers** by editing the Uri in the browser and navigate those Products and Authors. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)**. | * + In addition, we can navigate by primary key, so if we want to look for a given **Customer**, we can navigate into that record by adding the Id to the Uri. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/LastName**. | * + Moreover, we can go deeper and query a single Customer property, for instance the **LastName** property. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)**.   2. Highlight the **Customers(1)/Orders** link in the XML output. | * + Now, going back to the **Customer** instance you will notice the **Orders** property has a link to the Orders collection.   + Therefore, we can navigate across that relationship to the **Orders**. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/Orders**. | * + Now there we will see the collection of **Orders** related to the Customer.   + The Framework has done that navigation possible for us. |  |
| * 1. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/Orders(2)**   2. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/Orders(2)/OrderDetails**.   3. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/Orders(2)/OrderDetails(1)**.   4. Edit the browser address textbox to browse to **/ContainmentService.svc/Customers(1)/Orders(2)/OrderDetails(1)/Product**.   5. Highlight the **Name** and **Price** properties of the Product entity. | * + The fact that the containment relationship is being exposed by the service means we can navigate even deeper in the model by using the RESTful interface of WCF Data Services.   + We can query a **single** **Order** inside the Orders collection contained in the Customer entity.   + Then we can query the **OrderDetails** inside it and the **Product** entity inside the details. |  |

### Segment #5 – Implementing the Enhanced BLOB Support

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **AdoNetDataServices1510In1.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Right click on the **05 - BLOB Streams\EF\BLOBStreamService.svc** file and click **View Code**.   4. Highlight the **BLOBStreamService** class definition implementing **IServiceProvider** interface.   5. Highlight the **GetService** implementation, which returns an instance of **ProductStreamProvider** class. | * + WCF Data Services enables you to access binary large object (BLOB) data as a data stream. Streaming defers the loading of binary data until it is needed, and the client is able to more efficiently process this data.   + In order to take advantage of this functionality, the data service must implement the **IDataServiceStreamProvider** provider.   + The BLOBStreamService implements GetService method of the IServiceProvider interface to return a custom implementation of the IDataServiceStreamProvider: the **ProductStreamProvider** class. |  |
| * 1. Scroll down in the **BLOBStreamService.svc.cs file (C#) or BLOBStreamService.svc.vb (Visual Basic)** and expand the ProductStreamProvider class definition, if collapsed.   2. Highlight the **GetReadStream** method definition.   3. Highlight the portion of code that creates a **MemoryStream** instance and returns it. | * + When the client explicitly requests for the binary value, the GetReadStream method will be executed and will be responsible for obtaining the binary data.   + The method will receive an instance of the entity being request that can be used to retrieve new data from a different storage.   + In this case, the Product instance Id is used to retrieve binary data from an Entity Framework entity set, wrap it with a MemoryStream, and return it to the Data Service layer to be streamed back to the client. |  |
| * 1. Right click on the **05 - BLOB Streams\EF\BLOBStreamService.svc** file and click **View in Browser.**   2. Edit the browser address textbox to browse to **/BLOBStreamService.svc/Products**. | * + Now, we will use the browser as a client to show how this works from a client perspective.   + If we browse the Products entity set, we will find an ordinary Atom Pub output from the service. |  |
| * 1. Edit the browser address textbox to browse to **/BLOBStreamService.svc/Products(715)**.   2. Highlight the **edit-media** **<link>** element. | * + We will navigate to a single Product entity to check that no binary data is still being returned.   + But you should notice a <link> element that refers to a Product(#)/$value link. This element links to the binary data that will be streamed if the client requests it. |  |
| * 1. Edit the browser address textbox to browse to **/BLOBStreamService.svc/Products(715)/$value**. | * + If we wanted to retrieve the media resource for the photo (the binary content) we could simply append “$value” to the request.   + As we already saw, at this point it is up to the service to determine where to actually retrieve this media resource and to return it back to the client. |  |

Summary

* 1. In this demo, you saw the new features and capabilities included in the latest WCF Data Services release included in .NET Framework 4. Additionally, you learned the basics of the RESTful interface for data that provides WCF Data Services and how it can be adapted to different representations and data sources.