#### Contents

- Getting started
- Connecting to the data
- Enabling OData
- Calling your service
- <u>Advanced topics</u>
  - Service operations
  - <u>Stored procedures</u>
  - Securing the service

**Enabling OData** for a DevForce model allows for non-.NET clients to access the same back-end as your DevForce .NET clients. The programming model will be different from DevForce and depends on the OData client implementation on the chosen platform, but it avoids the need for developing yet another back-end or at a minimum some sort of a translation layer in front of the DevForce back-end.

In this topic, we will demonstrate how to expose a DevForce domain model through OData and access it from a client.

## **Getting started**

Let's begin by creating a new project. We'll be using Visual Studio 2012 and creating a new ASP.NET Empty Web Application.

After creating the project, add the *DevForce 2012 Server* NuGet package to install the required DevForce dependencies for a DevForce EntityServer. The *EntityServer* is the middle tier of a typical n-tier DevForce application. At the end of this tutorial it will be able to serve DevForce clients as well as OData clients.

Prerelease	DevForce 2012 Code-First Optional support for Entity Framework Code-First projects.
Prerelease	DevForce 2012 Compatibility Pack Optional limited DevForce 2010 backwards compatiblity support.
Prerelease	DevForce 2012 Client DevForce is a framework for building and operating data-rich business applications for multiple client technologies.

### Connecting to the data

In this tutorial we are going to use our usual suspect, the NorthwindIB database. At this stage we are assuming basic familiarity with how to create a data model in DevForce. If you're new to DevForce you may first want to walk through an introductory sample, such as the <u>Tour of DevForce Silverlight</u> or the <u>Tour of DevForce WPF</u>.

We will name the model NorthwindIB, generate it from the database and select only the Customer and Order tables to keep it small and manageable.

If everything went well, the result should look like this:



This is the data model we'll now expose as a WCF Data Service.

## **Enabling OData**

For all intents and purposes, at this point we have a functioning DevForce back-end and could start building a DevForce client application. Instead, we are going to expose this back-end via OData with a three step process.

#### 1. For the first step, we need to enable OData in our domain model.

Open NorthwindIB.edmx and click anywhere in the whitespace to select the model, then hit **F4** to bring up the model properties. Under the **DevForce Code Generation** category you will find a new option called **OData Enabled**. By default the option is set to **False**. Go ahead and change it to **True**, then save.

Q	Generate Developer Classes	False		
49	Handle Mapping Mismatches/Issue	Fix		
X	Injected Base Type		_	
	Max. Classes Per File	100	=	
	OData Enabled	Тгие		
Â	Tag			
	Validation Attribute Mode	DevForceVerification		
	C 1			

Let's look at what this option did for us. First, it added a couple of attributes to our Entity classes to make them usable in a WCF Data Service. Second, it generated a partial class for our strongly typed *EntityManager*.

Open NorthwindIBEntities\_IUpdatable.cs. In this file, you'll find an implementation of System.Data.Services.IUpdatable. The WCF Data Service calls upon this implementation to handle the CUD operations. This file is generated only once, so you can make customizations without having to worry about them being overwritten.

Now, what's this WCF Data Service? The WCF Data Service is part of the .NET framework and handles all the OData mechanics. In order for the WCF Data Service to do its magic, it requires a compatible data container, which provides the actual data and supports optional updates. Enabling OData in our data model turned the *EntityManager* into just such a container. The implementation of *System.Data.Services.IUpdatable* handles the optional updates.

#### 2. For the second step we need to add a WCF Data Service to our project.

Right-click on the **ODataTour** project and select **Add** | **New Item...** In the **Add New Item** dialog, you will find the WCF Data Service under Web. We'll name ours ODataService and click **Add**.

Add New Item - ODataTour						8 23
Installed Templates	Sort by: Defa	fault 🔹 🗊		Search Instal	ed Templates	٩
✓ Visual C# Code	🛒 ASI	SP.NET Server Control	Visual C#	* Type: Visu A class to pr	al C# rovide an WCF Da	ta Service.
General	Aut	thentication Domain Service	Visual C#			
Windows Forms WPF	💽 Bro	owser File	Visual C#			
Code Generation Reporting	Dor	omain Service Class	Visual C#			
Silverlight Workflow	Dyr	mamic Data Field	Visual C#			
XNA Game Studio 4.0 Online Templates	er 📔	meric Handler	Visual C#			
	🔛 Ste	te Map	Visual C#			
	Ski	in File	Visual C#			
	🚺 wo	CF Data Service	Visual C#			
	wc 🖉	CF Service	Visual C#	8		
	we	eb Configuration File	Visual C#			
	😢 We	eb Service	Visual C#			
				*		
Name: ODataService.sv	¢					
					Add	Cancel

If you're running Visual Studio 2013 the template name will likely be "WCF Data Service 5.6".

If you receive the following message, open the web.config file and change aspNetCompatibilityEnabled to true.



#### 3. Finally, we need to define the WCF Data Service.

Open **ODataService.svc.cs** and modify the code as follows:

public class ODataService : DataService <northwindibentities></northwindibentities>
{ // This method is called only once to initialize service-wide policies. // This method is called only once to initialize service-wide policies.
public static void InitializeService(DataServiceConfiguration config)
// TODO: set rules to indicate which entity sets and service operations are visible, updatable, etc.
// Examples: // config SetEntitySetAccessRule("MyEntityset" EntitySetRights AllRead);
// config.SetServiceOperationAccessRule("MyEntrySet", EntrySetRights.Antead),
config.SetEntitySetAccessRule("Customers", EntitySetRights.All);
config.SetEntitySetAccessRule("Orders", EntitySetRights.All);
config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2; // Or later!
Public Class ODataService
Imperior DataSet Vice(O) I Northwind Delinitative service, wide policies
Public Shared Sub Initialize Service (By Val config As
DataServiceConfiguration)
'TODO: set rules to indicate which entity sets and
' service operations are visible, updatable, etc.
'Examples:
'config.SetEntitySetAccessRule("MyEntitySet", EntitySetRights.AllRead);
config.SetServiceOperationAccessRule("MyServiceOperation",
ServiceOperationKignts.All);
config SetEntitySetAccessRule("Orders" EntitySetRights All)
config DataServiceBehavior MaxProtocolVersion = DataServiceProtocolVersion V2
End Sub
End Class

Let's look at the above code. First, we specified our *EntityManager* as the data container for the service by providing it as the generic data type on the *DataService*<*T*> base class.

Second, we've specified that both of our entity sets, Customers and Orders, are visible and updatable.

Unknown macro: IBNote

The "IBNote" macro is not in the list of registered macros. Verify the spelling or contact your administrator.

# **Calling your service**

At this point we are ready to test our service. Right-click **ODataService.svc** and select **View in Browser**. If everything went well, you should receive the standard service definition response like this:



This response tells us that we have two entity sets as expected. So, let's see if we can actually query some data.

Enter the following URL in the browser's address bar:

http://localhost:9009/ODataService.svc/Customers



Depending on the browser, you may have to *view source* in order to see the raw response.

At this point, our OData service can be consumed by any client that supports OData.

## **Advanced topics**

#### Service operations

You can expose server methods in your WCF Data Service too. Here we'll add a simple <u>service operation</u> named *GetOrdersByCity*, which as you might guess, will accept a *city* argument and return orders.

Adding the operation is easy. First we'll define a method in our *ODataService* class which returns an *IQueryable* and can be called with an HTTP GET request:

```
[WebGet]
public IQueryable<Order> GetOrdersByCity(string city) {
  var mgr = this.CurrentDataSource;
  return mgr.Orders.Where(o => o.ShipCity == city);
}
```

*CurrentDataSource* is an instance of the *EntityManager* we defined for the service. OData has created it for us, so all we need to do is define (and execute if we wish) the query we need. Here we use a simple LINQ query.

Next, we need to set the access to the method, which we can do in the *InitializeService* method we saw above. Here we allow read access for everyone:

config.SetServiceOperationAccessRule("GetOrdersByCity", ServiceOperationRights.AllRead);

We can test this by opening our browser and navigating to a URL such as the following: http://localhost:9009/ODataService.svc/GetOrdersByCity?city='Berlin'

#### Stored procedures

We can also call stored procedures defined in our domain model using the same technique we used for service operations.

You'll first need to define the stored procedure and <u>add it to your EDMX</u>. Once you've done this DevForce will generate a corresponding <u>StoredProcQuery</u> in your model.

The *NorthwindIB* database includes a stored procedure named *Employee\_sales\_by\_country*, so we added that to our model and DevForce obligingly generated the following:

public System.Collections.Generic.IEnumerable<EmployeeSalesByCountry\_Result>
EmployeeSalesByCountry(System.Nullable<System.DateTime> Beginning\_Date, System.Nullable<System.DateTime> Ending\_Date) {

var query=EmployeeSalesByCountryQuery(Beginning\_Date, Ending\_Date); var result=this.ExecuteQuery(query).Cast<EmployeeSalesByCountry\_Result>(); return result;

To expose this from our OData service, we need to follow the same steps as we did for the server method: 1) we define the service operation method in the *ODataService* class, and 2) we set its access.

Here we're calling the service operation *GetEmployeeSalesForPeriod*, since it better describes what it does. Since dates can be hard to pass correctly, we've also changed the arguments to strings. You'll notice we return an *IEnumerable* here too, since that's both returned by execution of the query and acceptable to the WCF Data Service.

```
[WebGet]
public IEnumerable<EmployeeSalesByCountry_Result> GetEmployeeSalesForPeriod(string startDate, string endDate) {
    var mgr = this.CurrentDataSource;
    var dt1 = DateTime.Parse(startDate);
    var dt2 = DateTime.Parse(endDate);
    return mgr.EmployeeSalesByCountry(dt1, dt2);
    }
config.SetServiceOperationAccessRule("GetEmployeeSalesForPeriod", ServiceOperationRights.AllRead);
```

To test the method, open the browser and navigate to a URL such as the following: http://localhost:9009/ODataService.svc/GetEmployeeSalesForPeriod?startDate='1/1/1996'&endDate='12/31/1996'

#### Securing the service

There are several was to secure your OData service. See our <u>sample</u> on how to use basic authentication and authorization headers.