Assignment #3 : An OData REST service

Introduction

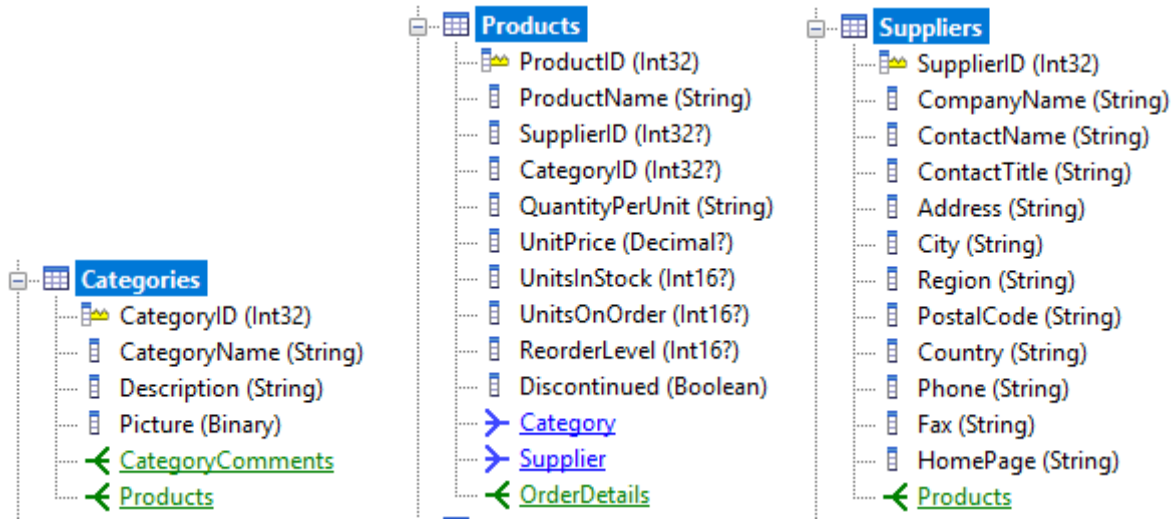
This assignment requires an **OData REST** service - called **OData-XML** – which accepts **OData GET** requests to retrieve data from three **XML** "table" documents: **XCategories.xml**, **XProducts.xml** and **XSuppliers.xml**, in **ATOM+XML** or **JSON** formats.

These XML documents are trimmed (and slightly modified) versions of the similarly named tables – **Categories**, **Products** and **Suppliers** – from the public and well-known **Northwind SQL** database sample.

The service must be written in **C#** and will be compiled with the command-line complier, **CSC**, via a given **batch** file. There is a small bonus if you also submit an **F#** version, compilable with **FSC**. The service will tested by running **IISExpress** via another given **batch** file (we'll typically run this on port 8181, which is available in the labs).

Server data

Our XML documents are not schema based. Therefore, let's also have a look the original SQL types, as Linqpad sees them. This will allow us to properly identify the types.



Note the following **primary keys**: **CategoryID** for **Category**, **ProductID** for **Products**, and **SupplierID** for **Suppliers**. Note also the following **foreign keys**: **CategoryID** and **SupplierID** in **Products**. Together, these define the following "duck's paw" one-to-many and many-to-one relationships:

- **Products**: from **Categories** to **Products**, and **Category**: the other way round
- **Products**: from **Suppliers** to **Products**, and **Supplier**: the other way round

Our XML documents keep only part of the SQL columns:

- **XCategories** keeps, in order: **CategoryID**, **CategoryName**, and **Description**.
- **XProducts** keeps, in order: **ProductID**, **ProductName**, **SupplierID**, **CategoryID**, **UnitPrice**, **UnitsInStock**, and **UnitsOnOrder**. Note that the last 5 of these columns are **nullable** numbers!
- **XSuppliers** keeps, in order: **SupplierID**, **CompanyName**, **ContactName**, and **Country**.

Here follow the first few lines of each of our XML documents.

XCategories

```
<Categories>
 <Category>
  <CategoryID>1</CategoryID>
  <CategoryName>Beverages</CategoryName>
  <Description>Soft drinks, coffees, teas, beers, and ales</Description>
 </Category>
 …
```

XProducts

```
<Products>
 <Product>
  <ProductID>1</ProductID>
  <ProductName>Chai!</ProductName>
  <SupplierID>1</SupplierID>
  <CategoryID>1</CategoryID>
  <UnitPrice>18.0000</UnitPrice>
  <UnitsInStock>39</UnitsInStock>
  <UnitsOnOrder>0</UnitsOnOrder>
 </Product>
 …
```

XSuppliers

```
<Suppliers>
 <Supplier>
  <SupplierID>1</SupplierID>
  <CompanyName>Exotic Liquids</CompanyName>
  <ContactName>Charlotte Cooper</ContactName>
  <Country>UK</Country>
 </Supplier>
 …
```

The markers will initially use the same XML documents as given to you. The **XProducts** document given to you does NOT currently contain any **null** numerical values. However, for full marks, the markers will also test your service with an expanded version of the **XProducts** document which will contain a few **null** numerical values. For example, they may insert such a fictitious product, where **null** number values are represented as **empty** elements:

```
<Product>
 <ProductID>101</ProductID>
 <ProductName>Foo1</ProductName>
 <SupplierID></SupplierID>
 <CategoryID/>
 <UnitPrice></UnitPrice>
 <UnitsInStock/>
 <UnitsOnOrder/>
</Product>
```

Sample GET URL queries and expected results (rudimentary formatted):

Entities: Categories, Suppliers, and Products

http://localhost:8181/WcfDataService1.svc/?$format=json

```
{"odata.metadata":"http://localhost:8181/WcfDataService1.svc/$metadata","value":[
{"name":"Categories","url":"Categories"},
{"name":"Suppliers","url":"Suppliers"},
{"name":"Products","url":"Products"}
]}
```

Top 3 Categories (all fields)

http://localhost:8181/WcfDataService1.svc/Categories()?$format=json&$orderby=CategoryID&$top=3&$select=CategoryID,CategoryName,Description

```
{"odata.metadata":"http://localhost:8181/WcfDataService1.svc/$metadata#Categories&
$select=CategoryID,CategoryName,Description","value":[
{"CategoryID":1,"CategoryName":"Beverages","Description":"Soft drinks, coffees, teas,
beers, and ales"},
{"CategoryID":2,"CategoryName":"Condiments","Description":"Sweet and savory sauces,
relishes, spreads, and seasonings"},
{"CategoryID":3,"CategoryName":"Confections","Description":"Desserts, candies, and sweet
breads"}
]}
```

Top 3 Suppliers (all fields)

http://localhost:8181/WcfDataService1.svc/Suppliers()?$format=json&$orderby=SupplierID&$top=3&$select=SupplierID,CompanyName,ContactName,Country

{"odata.metadata":"http://localhost:8181/WcfDataService1.svc/$metadata#Suppliers&$select=SupplierID,CompanyName,ContactName,Country","value":[

{"SupplierID":1,"CompanyName":"Exotic Liquids","ContactName":"Charlotte Cooper","Country":"UK"},

{"SupplierID":2,"CompanyName":"New Orleans Cajun Delights","ContactName":"Shelley Burke","Country":"USA"},

{"SupplierID":3,"CompanyName":"Grandma Kelly's Homestead","ContactName":"Regina Murphy","Country":"USA"}

]}

Top 3 Products (all fields)

http://localhost:8181/WcfDataService1.svc/Products()?$format=json&$orderby=ProductID&$top=3&$select=ProductID,ProductName,CategoryID,SupplierID,UnitPrice,UnitsInStock,UnitsOnOrder

{"odata.metadata":"http://localhost:8181/WcfDataService1.svc/$metadata#Products&$select=ProductID,ProductName,CategoryID,SupplierID,UnitPrice,UnitsInStock,UnitsOnOrder","value":[

{"ProductID":1,"ProductName":"Chai!","CategoryID":1,"SupplierID":1,"UnitPrice":"18.0000","UnitsInStock":39,"UnitsOnOrder":0},

{"ProductID":2,"ProductName":"Chang","CategoryID":1,"SupplierID":29,"UnitPrice":"19.0000","UnitsInStock":175,"UnitsOnOrder":40},

{"ProductID":3,"ProductName":"Aniseed Syrup","CategoryID":2,"SupplierID":1,"UnitPrice":"10.0000","UnitsInStock":13,"UnitsOnOrder":70}

]}

A join between all three entities (implemented by expansion, i.e. navigation)

http://localhost:8181/WcfDataService1.svc/Products()?$format=json&$orderby=ProductID&$top=3&$expand=Category,Supplier&$select=ProductID,ProductName,CategoryID,SupplierID,Category/CategoryName,Supplier/CompanyName,Supplier/ContactName

{"odata.metadata":"http://localhost:8181/WcfDataService1.svc/$metadata#Products&$select=ProductID,ProductName,CategoryID,SupplierID,Category/CategoryName,Supplier/CompanyName,Supplier/ContactName","value":[

{"Category":{"CategoryName":"Beverages"},"Supplier":{"CompanyName":"Exotic Liquids","ContactName":"Charlotte Cooper"},"ProductID":1,"ProductName":"Chai!","CategoryID":1,"SupplierID":1},

{"Category":{"CategoryName":"Beverages"},"Supplier":{"CompanyName":"For\u00eatsd'\u00e9rables","ContactName":"Chantal Goulet"},"ProductID":2,"ProductName":"Chang","CategoryID":1,"SupplierID":29},

{"Category":{"CategoryName":"Condiments"},"Supplier":{"CompanyName":"Exotic Liquids","ContactName":"Charlotte Cooper"},"ProductID":3,"ProductName":"Aniseed Syrup","CategoryID":2,"SupplierID":1}

]}

More test queries and expected results are given in the online **TestQueriesReponses** archive.

ODATA readings

For those interested: http://www.odata.org/

OData-XML-Demo

This is a fully running demo project, similar to our assignment, based on two simplified sample XML "table" documents, **XMyCustomers** and **XMyOrders** (which will also be used in the lectures).

OData-XML-Project

This is a running but **empty skeleton**, for you to fill, as indicated further below, in the **Development** section.

Project folder structure

Both **OData-XML-Demo** and **OData-XML-Project** share the same structure, which must be kept!

- **bin** : required libraries, Microsoft.*.dll and System.*.dll, and your compilation output, library **WebApplication1.dll**
- **data** : XML documents; **XMyCustomers** and **XMyOrders** in the demo; **XCategories.xml**; **XProducts.xml** and **XSuppliers.xml** in the project skeleton
- **_Compile WcfDataService1.svc.bat** : CSC compilation batch
- **_Compile WcfDataService1.svc.fs.bat** : FSC compilation batch
- **_localhost-8181-WcfDataService1.svc** : URL shortcut to test if the service is running
- **_Service_IISExpress_x64_8181.bat** : starts IISExpress on port 8181 (64 bits)
- **_Service_IISExpress_x86_8181.bat** : starts IISExpress on port 8181 (32 bits)
- **WcfDataService1.svc** : WCF processing directive
- **WcfDataService1.svc.cs** : C# source file – for you to fill
- **WcfDataService1.svc.fs** : F# source file - for you to fill (bonus)
- **Web.config** : service config file

Essentially, you must update the content of the skeleton .**cs** source file (and **.fs** for the optional bonus) but do NOT change anything else from the above list (not even file or folder names)!

Additionally, there are a few disposable **Linqpad** files, which can be used for development or quick tests:

- **odata-xml-test.linq** : could be used to quickly test your service, after compiling it and starting IISExpress
- **odata-xml-dev.linq** : could be used to develop and locally test your code, without starting the service

Running and testing

To run and test your service, you must follow the following steps:

1. Compile your source file with the given **batch**; the resulting **WebApplication1.dll** will be stored in the **bin** folder (where all supporting libraries already are)
2. Start **IISExpress** with the appropriate **batch**
3. Optionally, double click on the **_localhost-8181-WcfDataService1.svc** URL shortcut (which will start the **browser** and display the three entities)
4. Continue testing, using your favourite WCF client: a **browser**, a browser **add-on** (such as **RESTClient** in FF), **Linqpad** (e.g. with the given **odata-xml-test.linq**), …

For an accurate test, you will also receive a test program, the same test program which will be used by the markers!

Development

Of course, we will NOT attempt to develop an ODATA service from scratch, even such a service restricted to GET queries. Instead, we will "piggy-back" on extant libraries, which allow us to fill in just our specific business requests – using various flavours of **LINQ**, i.e. in a crisp **functional way**!

Therefore, you should develop this assignment just by modifying parts of the **WcfDataService1.svc.cs** file, from the provided **OData-XML-Project**, without changing anything else. Following the model given in **OData-XML-Demo**, you should create:

- Classes **Category**, **Products**, and **Supplier** (cf. classes **MyCustomers** and **MyOrders**)
- Enumerable fields **_Categories**, **_Products**, and **_Suppliers** (cf. fields **_MyCustomers** and **_MyOrders**)
- Queryable properties **Categories**, **Products**, and **Suppliers** (cf. properties **MyCustomers** and **MyOrders**)
- Constructor **MyDataSource** to load and parse the given XML documents into these enumerable fields (cf. the skeleton constructor **MyDataSource**)

The provided constructor is **static**, which is executed once only, and might be more useful in our case, as our data does not change. But you can try to change this…

You may also be able to change some names, but please check that you still get exactly the expected results. As received by the client, all fields should appear with the **same names and values**, with the possible exception of field order.

For development, we suggest the following tools: **Notepad++**, **Linqpad** or other light weight editors. You are free to use **VS**, but your submitted project must not contain any trace of this and must still have the indicated structure and behaviour!

Note that the model solution has about 150 well-spaced lines, including blank and skeleton lines, so you probably need to write about 100-120 line – more than this may be a sign of a wrong approach.

Assignment Marks, Deliverables and Submission

This **C#** assignment is worth **5 total course marks**. There will be an additional bonus for an additional **F#** version, worth **1 total course marks**. Please note the strict COMPSCI policy on **plagiarism**.

Submit electronically, to the COMPSCI web dropbox, an **upi.7z** archive containing your **OData-XML-Project** folder, without its **bin** subfolder! The **bin** subfolder contains all required libraries, which are pretty large and are not needed by the marker (which has identical copies). The submission size will be **limited to 1 MB** per student.

Please recheck your projects before submitting, starting from the archive prepared for submission (which is the only thing that the marker will receive from you). Expand this archive on a lab machine, in another location, then restore the **bin** folder, rebuild your solutions and test them again!

You can submit several times, but only the **last submission** will be assessed. Please keep your electronic dropbox **receipt**!

Deadline

Monday 25 Sept 2016, 18:00! Please do not leave it for the last minute. Remember that you can resubmit and, by default, we only consider your **last submission**.

After this deadline, submissions will be still accepted for 4 more days, with gradually increasing penalties, of 0.5% for each hour late. For example:

Monday 25 Sept 2017, 20:00:	-1%,	Tuesday 26 Sept 2017, 18:00:	-12%
Tuesday 26 Sept 2017, 20:00:	-13%,	Wednesday 27 Sept 2017, 18:00:	-24%
Thursday 28 Sept 2017, 18:00:	-36%,	Friday 29 Sept 2017, 18:00:	-48%

After this, no more submissions are accepted!

Appendix 1 – OData queries translated into C#/LINQ and SQL

1. OData:

```
http://localhost:8181/WcfDataService1.svc/Categories()?$format=json&$orderby=CategoryID&$top=3&$select=CategoryID,CategoryName,Description
```

C# LINQ (fluent style):

```
Categories
        .OrderBy(c => c.CategoryID)
        .Select(c => new { c.CategoryID, c.CategoryName, c.Description })
        .Take(3)
```

SQL:

```
SELECT TOP (3) [t0].[CategoryID], [t0].[CategoryName], [t0].[Description]
FROM [Categories] AS [t0]
ORDER BY [t0].[CategoryID]
```

Result as table (via Dump):

IOrderedQueryable<> (3 items)

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads

2. OData:

```
http://localhost:8181/WcfDataService1.svc/Products()?$format=json&$orderby=ProductID&$top=3&$expand=Category,Supplier&$select=ProductID,ProductName,CategoryID,SupplierID,Category/CategoryName,Supplier/CompanyName,Supplier/ContactName
```

C# LINQ (fluent style):

```
Products
        .OrderBy(p => p.ProductID)
        .Select(p => new { p.ProductID, p.ProductName, p.CategoryID, p.SupplierID,
                           p.Category.CategoryName,
                           p.Supplier.CompanyName, p.Supplier.ContactName })
        .Take(3)
```

SQL:

```
SELECT TOP (3) [t0].[ProductID], [t0].[ProductName], [t0].[CategoryID], [t0].[SupplierID],
               [t1].[CategoryName],
               [t2].[CompanyName], [t2].[ContactName]
FROM [Products] AS [t0]
LEFT OUTER JOIN [Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
LEFT OUTER JOIN [Suppliers] AS [t2] ON [t2].[SupplierID] = [t0].[SupplierID]
ORDER BY [t0].[ProductID]
```

Result as table (via Dump):

▲ IOrderedQueryable<> (3 items)

ProductID	ProductName	CategoryID	SupplierID	CategoryName	CompanyName	ContactName
1	Chai!	1	1	Beverages	Exotic Liquids	Charlotte Cooper
2	Chang	1	29	Beverages	Forêts d'érables	Chantal Goulet
3	Aniseed Syrup	2	1	Condiments	Exotic Liquids	Charlotte Cooper