**Tutorial**

**Target audience**

Java developers that want to access OData services in Java

**Author**

Thierry Boileau, co-founder of [Noelios Technologies](http://www.noelios.com) and core committer on the [Restlet open source project](http://www.restlet.org).

**Topics covered**

* Accessing an OData service in Java with Restlet
* Handling associations between entities (if any)
* Handling queries
* Accessing secured services

**Table of contents**

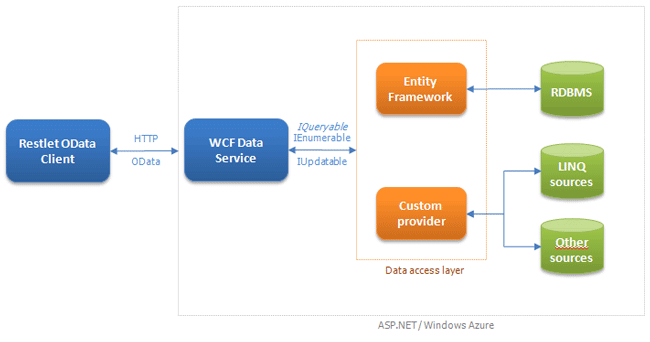
* Introduction: getting started
* [Code generation](http://wiki.restlet.org/docs_2.0/13-restlet/28-restlet/287-restlet/288-restlet.html#dsy288-restlet_codeGeneration)
* Get the current set of Cafes and Items Get a single entity
* Add a new entity
* Update an entity
* Delete an entity
* Get a single Entity and its associated entities
* Other kinds of queries
* Support of projections
* Server-side paging
* Get the row count
* Support of customizable feeds
* Handling blobs (aka media link entries)
* Access secured services
* Support of capability negotiation based on protocol versions

**Resources**

* [Javadocs of the Restlet extension for OData Services](http://www.restlet.org/documentation/2.0/ext/org/restlet/ext/odata/package-summary.html)
* [MSDN documentation on WCF Data Services](http://msdn.microsoft.com/en-us/library/cc907912.aspx)

**Introduction**

REST can play a key role in order to facilitate the interoperability between Java and Microsoft environments. To demonstrate this, the Restlet team Restlet team worked with Microsoft in order to provide a new Restlet extension that will provide several high level features for accessing WCF Data Services.



**Figure 1 - The WCF Data Services Framework Architecture**

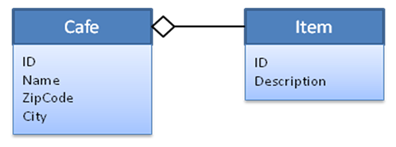
Initially known as “Astoria” then ADO.NET Data Services, the WCF Data Services technology has become the preferred way to RESTfully expose data sources (relational databases, in-memory data, XML files, etc.) in the .NET framework.

The “org.restlet.ext.odata” extension for the Restlet Framework provides a client API to access remote WCF Data Services or other services supporting the OData protocol. It allows access to the exposed entities, to their properties and their associations (when an entity is linked to other entities). We will illustrate this with a sample WCF Data Service with the following root URI: <http://restlet.cloudapp.net/TestAssociationOneToOne.svc/>

This service defines two kinds of entities:

* “Cafe”, and
* “Item”.

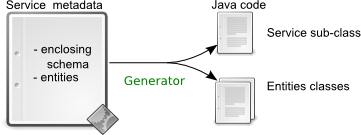
A “Cafe” defines several properties such as “ID”, “Name”, “City”, and an Item is simply defined by an “ID” and a “Description”. Cafe and Item are linked, for intentional simplification, in a “one to one” association. That is to say, a “Cafe” has one attribute called “Item”.



**Figure 2 - Class diagram of the TestAssociationOneToOne service**

**Code generation**

From the client perspective, if you want to handle the declared entities, you will have to create a class for each entity, defines their attributes, and pray that you have correctly spelled them and defined their type. Thanks to the Restlet extension, a generation tool will make your life easier. It will take care of this task for you, and generate the whole set of Java classes with correct types.



**Figure 3 - Overview of code generation**

Just note the URI of the target service, and specify the directory where you would like to generate the code:

java -jar org.restlet.ext.odata.jar http://restlet.cloudapp.net/TestAssociationOneToOne.svc/ ~/workspace/testADO

Please note that this feature requires the use of the core Restlet, and additional dependencies such as Atom (used by OData services for all exchanges of data), XML (required by the Atom extension) and FreeMarker (used for the files generation). The following jars (take care of the names) must be present on the current directory:

* org.restlet.jar (core Restlet)
* org.restlet.ext.odata.jar (OData extension)
* org.restlet.ext.atom.jar (Atom extension)
* org.restlet.ext.xml.jar (XML extension)
* org.restlet.ext.freemarker.jar (Freemarker extension)
* org.freemarker.jar (Freemarker dependency)

You can also used the full command line that includes the list of required archives for the class path argument (nb: take care of the OS specific classpath separator) and the name of the main class:

java -cp org.restlet.jar:org.restlet.ext.xml.jar:org.restlet.ext.atom.jar:org.restlet.ext.freemarker.jar:org.restlet.ext.odata.jar:org.freemarker.jar org.restlet.ext.odata.Generator

http://restlet.cloudapp.net/TestAssociationOneToOne.svc/

 ~/workspace/testADO

This will generate the following java classes and directory:

testAssociationOneToOne/

+-- Cafe.java

+-- Item.java

TestAssociationOneToOneQuery.java

The classes that correspond to entities are generated in their corresponding package (in our case: “testAssociationOneToOne”), as defined by the meta data of the target Data Service. The last class (“TestAssociationOneToOneService”) is what we call a service object. Such object is able to handle the communication with the data service, and is able to store the state of the latest executed request and the corresponding response. The communication between the client and the server is still stateless.

Before using the generated classes, let’s explain how the Java code is generated from the metadata of the target OData service. Actually, the Generator class extracts a few elements of the metadata such as the schema and the entities as follow.

|  |  |
| --- | --- |
| **WCF concept** | **Java concept** |
| Entity | Class |
| Entity name | Class name |
| Enclosing schema namespace | Package name |
| Entity property | Member variable with getter and setter |
| EDM data type | Java primitive types and classes |

**Transformation table from WCF concepts to Java equivalent concepts**

Regarding the conversion of the data type, an equivalence table has been established as follow:

|  |  |
| --- | --- |
| **EDM data type** | **Java data type** |
| Binary | byte[] |
| Boolean | boolean |
| DateTime | java.util.Date |
| Decimal | long |
| Single | float |
| Double | double |
| Guid | java.lang.String |
| Int16 | short |
| Int32 | int |
| Int64 | long |
| Byte | byte |
| String | java.lang.String |

**Data type conversion table**

We have finished for now of the theoretical aspects; let's see how to use the generated classes.

**Get the current set of Cafes and Items**

The code below gets the whole set of Cafe entities and displays some of their properties. It will display this kind of output on the console:

id: 1

name: Le Café Louis

id: 2

name: Le Petit Marly

TestAssociationOneToOneService service = new TestAssociationOneToOneService();

Query<Cafe> query = service.createCafeQuery("/Cafes");

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

}

**Retrieve the set of “Cafe” entities.**

The first step is the creation of a new service. This is the only required action, and it must be done once, but prior to any other one. Then, as, we want to get a set of “Cafe”, we just create a new query and specify the desired data. Under the hood, it actually makes a GET request to the “/Cafes” resource (relatively to the service's URI), and receive as a result a AtomXML feed document. This document is parsed by the query which provides the result as an Iterator. Finally, we can loop over the iterator and access to each “Cafe” instance.

Here is the content of the HTTP request:

GET /TestAssociationOneToOne.svc/Cafes HTTP/1.1

Host: restlet.cloudapp.net

User-Agent: Noelios-Restlet/2.0snapshot

Accept: \*/\*

Connection: close

And here is the response of the server including both response headers and entity:

HTTP/1.1 200 OK

Cache-Control: no-cache

Content-Type: application/atom+xml;charset=utf-8

Server: Microsoft-IIS/7.0

DataServiceVersion: 1.0;

X-AspNet-Version: 2.0.50727

X-Powered-By: ASP.NET

Date: Fri, 24 Jul 2009 14:21:20 GMT

Connection: close

Content-Length: 2221

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base="http://restlet.cloudapp.net/TestAssociationOneToOne.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">

<title type="text">Cafes</title>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes</id>

<updated>2009-07-24T14:21:20Z</updated>

<link rel="self" title="Cafes" href="Cafes" />

<entry>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes('1')</id>

<title type="text"></title>

<updated>2009-07-24T14:21:20Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Cafe" href="Cafes('1')" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Item"

type="application/atom+xml;type=entry" title="Item" href="Cafes('1')/Item" />

<category term="TestAssociationOneToOne.Cafe" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID>1</d:ID>

<d:Name>Le Café Louis</d:Name>

<d:ZipCode m:type="Edm.Int32">92300</d:ZipCode>

<d:City>Levallois-Peret</d:City>

</m:properties>

</content>

</entry>

<entry>

<id>http://restlet.cloudapp.net/TestAssociationOneToOne.svc/Cafes('2')</id>

<title type="text"></title>

<updated>2009-07-24T14:21:20Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Cafe" href="Cafes('2')" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Item"

type="application/atom+xml;type=entry" title="Item" href="Cafes('2')/Item" />

<category term="TestAssociationOneToOne.Cafe" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID>2</d:ID>

<d:Name>Le Petit Marly</d:Name>

<d:ZipCode m:type="Edm.Int32">78310</d:ZipCode>

<d:City>Marly Le Roi</d:City>

</m:properties>

</content>

</entry>

</feed>

Getting the set of defined “Item” is quite similar:

Query<Item> queryItem = service.createItemQuery("/Items");

for (Item Item : queryItem) {

System.out.println(“id: ” + Item.getID());

System.out.println(“desc.: ” + Item.getDescription());

}

**Retrieve the set of “Item” entities.**

Please note that for the rest of the document, we assume the “service” object has already been instantiated.

**Get a single Entity**

Imagine we want to retrieve the first “Cafe” of the list, that is to say, the one which identifier is equal to “1”. As for the set of entities, you just have to create a new query, with a new parameter. The code below should produce this output:

id: 1

name: Le Café Louis

Query<Cafe> query = service.createCafeQuery("/Cafes('1')");

Cafe Cafe = query.iterator().next();

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

**Retrieve the “Cafe” by its identifier.**

As for a set of entities, you have to create a new query, and precise the identifier of the target resource. WCF adopts its own naming convention. You surely notice that the identifier is enclosed between “(“ and “)” characters.   
Now, you can ask the query to return the entity with a call to the “next” method, and if the entity exists, you can print its properties. An additional call to “next” will provide a “null” result.

**Add a new Entity**

Let's complete the current list of entities and add a new one. This process is quite simple and just requires you to firstly create and complete the new entity, then invoke the “addEntity” method as follow.

Cafe Cafe = new Cafe();

Cafe.setID("3");

Cafe.setZipCode(12345);

Cafe.setName("Bar des sports");

Cafe.setCity("Paris");

service.addEntity(Cafe);

**Add a new Cafe.**

The generated subclass of Service contains a dedicated method for each declared entities. In our case, there are two of them, one for the Cafe class, and the other for the Item class. Such method actually sends a POST request to the corresponding entity set resource. For example, adding a new Café sends a POST request to the “/Cafes” resource. Here is the sample content of such generated request:

POST /TestAssociationOneToOne.svc/Cafes HTTP/1.1

Host: restlet.cloudapp.net

User-Agent: Noelios-Restlet/2.0snapshot

Accept: \*/\*

Content-Type: application/atom+xml

Transfer-Encoding: chunked

Connection: close

281

<?xml version="1.0" standalone='yes'?>

<entry xmlns="http://www.w3.org/2005/Atom">

<content type="application/xml"><properties xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">

<ZipCode xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">12345</ZipCode>

<ID xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">3</ID>

<Name xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Bar des sports</Name>

<City xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Paris</City>

<Article xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices"/></properties></content>

</entry>

0

Before using this feature, ensure that you provide a correctly identified object, especially, don’t try to add an entity with a null value. It appears that the remote service does not prevent such cases which could lead to irremediably mess the content of the entity set.

**Update an Entity**

Once you have retrieved an entity, imagine that you want to update one of its properties. The sample code below illustrates this with the “Nom” property. It simply uses the “updateEntity” method. You can check that the value has really been taken into account by making a new query.

Query<Cafe> query = service.createCafeQuery("/Cafes('1')");

Cafe Cafe = query.iterator().next();

Cafe.setNom("Bar des sports");

service.updateEntity(Cafe);

**Update a Cafe.**

Under the hood, a PUT request is sent to the corresponding Web resource.

**Delete an Entity**

Let's finish the tour of the basic operations with the deletion of an entity. You just need to wall the deleteEntity method as shown just below.

Query<Cafe> query = service.createCafeQuery("/Cafes('1')");

Cafe Cafe = query.iterator().next();

service.deleteEntity(Cafe);

**Delete a Cafe.**

Basically, this operation requires a valid Java Object instance correctly identified: that is to say, the attribute that serves as identifier must be correctly valuated.

**Get a single Entity and its associated entities**

Until now, we looked at entities and their basic properties. However, we pointed out at the beginning of this Item that a “Cafe” also aggregates “Item” entities. You may have noticed, during your own tests, that the Item property was always null. By default, associations are not expanded, but they can be. If you run the following code, you will get this kind of trace at the console.

Cafe

id: 1

name: Le Café Louis

Item

id: 1

Description: Poulet au curry

Query<Cafe> query = service.createCafeQuery("/Cafes('1')").expand("Item");

Cafe Cafe = query.next();

System.out.println("Cafe");

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

System.out.println("Item");

System.out.println(“id: ” + Cafe.getItem().getID());

System.out.println(“Description: ” + Cafe.getItem().getDescription());

**Retrieve a “Cafe” by its identifier, with the associated Item in one request.**

Instead of just creating a query as seen above, you can complete it by calling the “expand” method. It takes one parameter which is the name of the entity attribute you want to expand. Please note that you can also invoke this when requesting the “/Cafes” entity set.

**Other kinds of queries**

The expansion of query is one of the features provided by the OData protocol. Other ones are available that aim at ordering, filtering, and limiting the set of the results returned by a query. This section is an exhaustive list of those available features.

**Order**

Let's say you want to order a list of ItemItems by their “description” attribute. For example, once applied to the set of all Items provided by our sample OData service, the following code should display at the console this king of trace:

id: 2

description: Pâté

id: 1

description: Poulet au curry

Query<Item> query = service.createItemQuery("/Items").orderby("Description");

for (Item Item : query) {

System.out.println(“id: ” + Item.getID());

System.out.println(“description: ” + Item.getDescription());

}

**Order a set of entities.**

Just as the “expand” method, it takes one parameter which is the name of the property used to order the set of results.

**Filter**

Do you want to filter the set of results? Using the “filter” method, you will be able to specify one or more criteria that the entities returned by the query must share. You will express these constraints using LINQ  (Language-Integrated Query). LINQ has been originally designed to bridge the gap between the world of data and the world of objects (in the context of C# and Visual Basic programs). In the context of OData services, it helps filtering a set of data. Its syntax is very close to SQL's one.

Let's illustrate its use by limiting the set of Cafe objects to the one (there should be only one) having its ID equals to “1”. Applied to the list of current Cafes, the following code will produce this display on the console:

id: 1

nom: Le Café Louis

Query<Cafe> query = service.createCafeQuery("/Cafes").filter("Name eq 'Le Café Louis'");

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“nom: ” + Cafe.getNom());

}

**Filter a set of entities**

**Skip**

The “skip” method takes one parameter which is a number, and simply allows you to omit the first elements of the set theoretically returned by the query.

Let's say you want to omit the first “Cafe” of the list, just call the “skip” method as shown in the sample code below. It should display this kind of trace at the console.

id: 2

name: Le Petit Marly

Query<Cafe> query = service.createCafeQuery("/Cafes").skip(1);

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

}

**Skip the first entity**

**Top**

Just as the “skip” method, “top” takes a number parameter which represents the maximum number of results that the query will return. Its use is very simple as shown below:

id: 1

name: Le Café Louis

Query<Cafe> query = service.createCafeQuery("/Cafes").top(1);

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

}

**Limit the number of returned entities.**

The “skip” and “top” can be used together for paging a set of results.

**Support of projections**

Projections allow you to retrieve a subset of the properties of queried entities. It is a way to reduce the amount of data transfered from the server.   
You can restrict your query to set of properties, either simple or complex or even navigation properties (links to other entities). The OData protocol specifies the usage of the **$select** query parameter [here](http://msdn.microsoft.com/en-us/library/ee525214%28PROT.10%29.aspx).   
The result of such projection is a list of entities as usual, except that these entities are only populated with the selected properties.  
For example, the following code will only get the name of the Cafe entities:

id: null

name: Le Café Louis

Query<Cafe> query = service.createCafeQuery("/Cafes").top(1).select("Name");

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

}

**Limit the number of returned properties.**

This applies also to associated entities, as far as the association is expanded:

id: null

name: Le Café Louis

Item

id: null

Description: Poulet au curry

Query<Cafe> query = service.createCafeQuery("/Cafes").top(1).expand("Item").select("Name,Item.Description");

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“name: ” + Cafe.getName());

System.out.println("Item");

System.out.println(“id: ” + Cafe.getItem().getID());

System.out.println(“Description: ” + Cafe.getItem().getDescription());

}

**Limit the number of returned properties.**

**Server-side paging**

This mechanism also the server to limit the amount of data to transfer in the response to a query made on an entity set. This is very interesting if the set can contain a large set of entities. Unfortunately, this feature has no impact on the way you use the Query object of the OData extension. This is made transparent for the user.

**Get the row count**

By invoking the **getCount** method on the Query object, you can retrieve the number of entities contained by the target entity set. having said that, you must be aware that there are two ways to get the number of entities of an entity set (once filtering, if any, have been applied). If the service supports the [inlinecount](http://msdn.microsoft.com/en-us/library/dd942040%28PROT.10%29.aspx) feature, this count is obtained from the Feed document itself. This allows to retrieve both count data and entries in the same request. Another way is to send a dedicated request using the [$count segment](http://msdn.microsoft.com/en-us/library/cc716656%28VS.100%29.aspx). You can control this behaviour by invoking the **inlineCount** method on the Query object. It accepts as a parameter a boolean value indicating that you want to retrieve the count number using the *inlinecount* query parameter (set the parameter to "true") or directly using the *$count* segment (set the parameter to "false", this is the value by default). The following sample code illustrates how to get the count using the *inlinecount* query parameter.

Query<Cafe> query = service.createCafeQuery("/Cafes").inlineCount(true);

System.out.println("Number of entities: " + query.getCount());

**Get the number of Cafes.**

Please note that the *inlinecount* query parameter is not supported by every service.

**Support of customizable feeds**

[Customizable feeds](http://msdn.microsoft.com/en-us/library/ee373839%28VS.100%29.aspx) is a server-side feature that address specific use-case that requires that the Atom data feeds produced by the data service will use both standard Atom elements or custom feed elements. This of course has an impact on the way the Atom document must be parsed, but this has no impact on the way you use the OData extension.

**Handling blobs (aka media link entries)**

[Media link entries](http://msdn.microsoft.com/en-us/library/ee473426.aspx) are specific entities that allow to handle binary content such as images, documents, etc. Each entity can be seen as collection of metadata about the binary content, plus the binary content itself. Since the metadata are stored as basic attribute members of the entity class (as usual), there must be a specific way to handle the binary content. At that point the Service class introduces a set of new methods to handle the "value" of a blob entity:

* getValueRef(Object) that returns the URI of the target binary content
* getValue(Object) and getValue(Object, \*) methods that allow you to retrieve as a Representation the binary content of a given entity.

**Access to secured services**

WCF Data Services can choose their security option. In order to access those kinds of services, you must have been given the following information:

* login and password
* security scheme such as HTTP BASIC, Windows Shared Key and Shared Key Lite, etc.

These data must be supplied to the service instance which takes care of all communications with the target service. This is generally done once, after the instantiation of the service. Here is an illustration of how to set the credentials:

service.setCredentials(

new ChallengeResponse(ChallengeScheme.HTTP\_BASIC,

"login",

"password")

);

Query<Cafe> query = service.createCafeQuery("/Cafes").top(1);

for (Cafe Cafe : query) {

System.out.println(“id: ” + Cafe.getID());

System.out.println(“nname: ” + Cafe.getName());

}

**Add credentials to access a secured service.**

Note that Restlet framework supports a wide set of security schemes, including HTTP BASIC, HTTP DIGEST, Windows Shared Key and Shared Key Lite.

**Access to NTLM secured services**

Please refer to [this document](http://wiki.restlet.org/docs_2.0/13-restlet/27-restlet/46-restlet/364-restlet.html) for how to access NTML secured services.

**Support of capability negotiation based on protocol versions**

As stated by the WCF documentation ([here](http://msdn.microsoft.com/en-us/library/dd541202%28PROT.10%29.aspx)), client and server may talk different dialects of the OData protocol. The Service class provides several methods that allow you to control these aspects:

* getClientVersion and setClientVersion to handle the supported version of OData protocol.
* getMaxClientVersion and setMaxClientVersion to handle the max number version od supported version of the OData protocol
* getServerVersion to get the version of the dialect talked by the server.

The values setted by setClientVersion and setMaxClientVersion are not controlled by the framework, but they must follow a format specified [here](http://msdn.microsoft.com/en-us/library/dd541109%28PROT.10%29.aspx).

**Conclusion**

This article illustrates what can be done with the Restlet extension for OData services. We hope that you found it simple and useful to follow to read. It is a good demonstration of how adopting of REST and related standards such as HTTP and Atom facilitates the interoperability across programming languages and executions environments.