



WebSphere. software



WebSphere eXtreme Scale REST data service

Version 7.0.0.0

User Guide

Document version [7.0.0.0]

CONTENTS

List of Figures.....	vi
List of Tables.....	vii
Revision History	viii
1 Introduction.....	1
1.1 Available features.....	2
1.2 Known problems and limitations	3
2 Directory conventions	3
3 Getting started	5
3.1 Getting started sample overview	6
3.1.1 Creating a scalable data model in eXtreme Scale.....	6
3.1.2 Retrieving and updating data	9
3.2 Starting the sample eXtreme Scale grid	11
3.2.1 Starting the sample grid for a stand-alone deployment	11
3.2.2 Starting the sample grid for a WebSphere Application Server integrated deployment	12
3.3 Configuring and starting your web server	14
3.3.1 Getting started with WebSphere Application Server version 7.0.....	14
3.3.2 Getting started with WebSphere eXtreme Scale integrated with WebSphere Application Server version 7.0.....	16
3.3.3 Getting started with WebSphere Application Server Community Edition	17
3.3.4 Getting started with Tomcat.....	19
3.4 Adding data with the sample Java client application	21
3.4.1 Java client command syntax.....	22

3.4.2	Running and building the sample grid and Java client with Eclipse	23
3.5	Using a web browser to view sample data.....	23
3.5.1	Configuring Internet Explorer Version 8	23
3.5.2	Configuring Firefox Version 3	24
3.5.3	Example URLs	25
3.6	Using the sample Visual Studio 2008 WCF Data Services client application	26
3.6.1	Software requirements.....	26
3.6.2	Building and running the getting started client.....	27
3.6.3	WCF Data Services C# client command syntax.....	27
4	Installing the REST data service	28
4.1	Software requirements	28
4.2	Packaging overview	29
4.3	Packaging and deploying the REST data service	29
4.4	Deploying on WebSphere Application Server	31
4.5	Deploying on WebSphere Application Server Community Edition	34
4.6	Deploying on Apache Tomcat	37
5	Configuring the REST data service	39
5.1	Configuring the REST data service properties file	39
5.2	Configuring WebSphere eXtreme Scale	40
5.2.1	Creating an entity model.....	40
6	Using the REST data service	50
6.1	Service root URI.....	50

6.2	Request types	51
6.2.1	Insert request types	51
6.2.2	Update request types.....	51
6.2.3	Delete request types	51
6.2.4	Retrieve request types	52
6.2.5	Invoke request	53
6.2.6	Batch request.....	53
6.2.7	Tunneled requests	53
6.3	Request Protocols and Examples.....	53
6.3.1	Retrieve requests.....	53
6.3.2	Insert Request.....	70
6.3.3	Update Requests	75
6.3.4	Delete Requests	81

LIST OF FIGURES

Figure 1:	Microsoft WCF Data Services	1
Figure 2:	WebSphere eXtreme Scale REST data service.....	2
Figure 3:	Getting started sample topology.....	6
Figure 4:	The Microsoft SQL Server Northwind sample schema diagram	7
Figure 5:	The Customer and Order entity schema diagram	8
Figure 6:	The Category and Product entity schema diagram.....	9
Figure 7:	Configuring Internet Explorer 8 to display ATOM feeds as XML	24
Figure 8:	Firefox Application Chooser Window	24
Figure 9:	WebSphere eXtreme Scale REST Data Service Files.....	30

LIST OF TABLES

Table 1	Java Type to EDM Type Mapping for Retrieve Requests	42
Table 2	Compatible EDM type to Java type	44

REVISION HISTORY

Date	Version	Revised By	Comments
7/28/2009	1.0.0.0	cdjohnson	Initial version for Technology Preview drop 1.0
9/17/2009	2.0.0.0	cdjohnson	Added new information for Technology Preview drop 2
10/29/2009	3.0.0.0	cdjohnson	Added new information for Technology Preview drop 3
12/17/2009	7.0.0.0	cdjohnson	First release, available in eXtreme Scale 7.0.0.0 cumulative fix 2.

1 Introduction

This document describes the WebSphere eXtreme Scale REST data service, Version 7.0.0.0.

The WebSphere eXtreme Scale REST data service is a Java HTTP service that is compatible with Microsoft WCF Data Services (formally ADO.NET Data Services) and implements the [Open Data Protocol](#) (OData). The REST data service allows any HTTP client to access a WebSphere eXtreme Scale Version 7.0.0.0 with cumulative fix 2 or later grid, and is compatible with the WCF Data Services support that is supplied with the Microsoft .NET Framework 3.5 SP1. RESTful applications can be developed with the rich tooling provided by Microsoft Visual Studio 2008 SP1. Figure 1 shows a general overview of how WCF Data Services interacts with clients and databases.

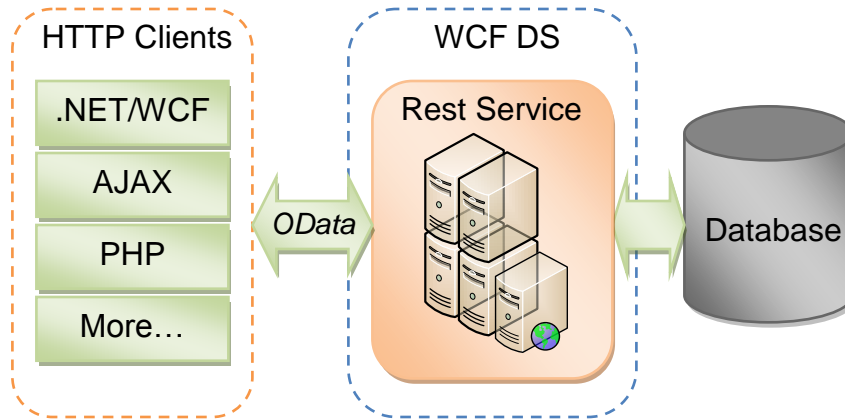


Figure 1: Microsoft WCF Data Services

WebSphere eXtreme Scale includes a function-rich API set for Java clients. As shown in Figure 2, the REST data service is a gateway between HTTP clients and the eXtreme Scale grid, communicating with the grid through an eXtreme Scale client. The REST data service is a Java servlet, which allows flexible deployments for common Java Platform, Enterprise Edition (JEE) platforms such as WebSphere Application Server. The REST data service communicates with the eXtreme Scale grid using the eXtreme Scale Java APIs and allows WCF Data Services clients or any other client that can communicate with HTTP and XML.

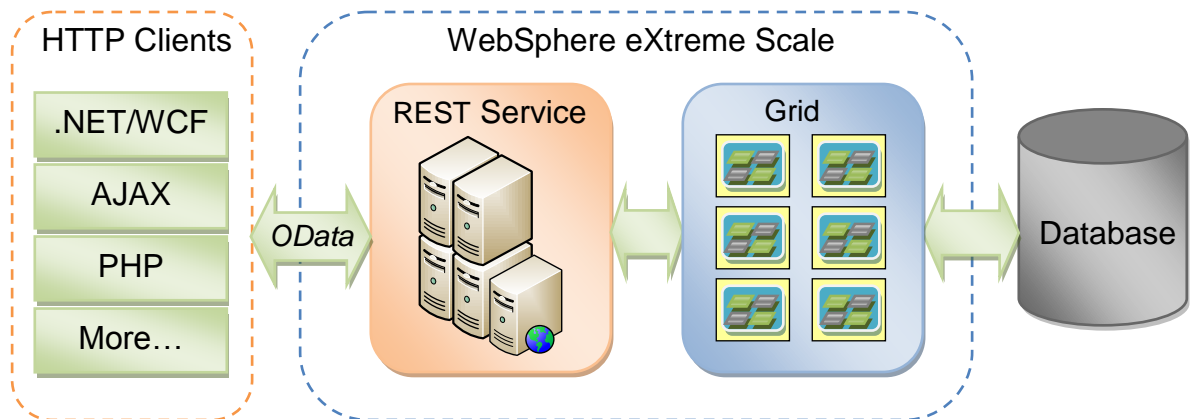


Figure 2: WebSphere eXtreme Scale REST data service

Use the following links to find additional information about WCF Data Services:

- [Microsoft WCF Data Services Developer Center](#)
- [ADO.NET Data Services overview on MSDN](#)
- [Whitepaper: Using ADO.NET Data Services](#)
- [Atom Publish Protocol: Data Services URI and Payload Extensions](#)
- [Conceptual Schema Definition File Format](#)
- [Entity Data Model for Data Services Packaging Format](#)
- [Open Data Protocol](#)
- [Open Data Protocol FAQ](#)

1.1 Available features

The eXtreme Scale REST data service, Version 7.0.0.0 implements a subset of the Microsoft [Atom Publishing Protocol: Data Services URI and Payload Extensions](#) specification, Version 1.0 which is part of the OData protocol. Microsoft WCF Data Services is compatible with this specification when using Visual Studio 2008 SP1 and the .NET Framework 3.5 SP1.

This version of the eXtreme Scale REST data service supports the following features:

- Automatic modeling of eXtreme Scale EntityManager API entities as WCF Data Services entities which includes (see section 5.2.1 for additional detail):
 - Java data type to Entity Data Model type conversion.
 - Entity association support.
 - Schema root and key association support which is required for partitioned grids.

-
- Atom Publish Protocol (AtomPub or APP) XML and JavaScript Object Notation (JSON) data payload format.
 - Create, Read, Update and Delete (CRUD) operations using the respective HTTP request methods: POST, GET, PUT and DELETE. In addition, the Microsoft extension: MERGE is supported.
 - Simple queries are supported using filters. See section 6.2.4 for details on which query features are supported.
 - Batch retrieval and change set requests are supported. See section 6.2.6.
 - Partitioned grid support for high availability.
 - Interoperability with eXtreme Scale EntityManager API clients.
 - Support for standard JEE web servers.

1.2 Known problems and limitations

- Security is not supported. User authorization and authentication is not supported between the REST data service and the eXtreme Scale grid. Java SE security is also not supported.
- Mixed character sets are not supported in batches. Each batch part must contain the same character set as the request.
- ETags are not supported. Concurrent update collisions are only detected if using entity versioning and the version field is supplied in the data payload.
- Tunneled requests are not supported.

Additional details can be found in section 5.2.1: Creating an entity model.

2 Directory conventions

This document describes many examples and command-line syntax that must reference special directories such as *wxs_install_root*, and *wxs_home*. These directories are defined as follows and are displayed throughout this document in italics:

wxs_install_root

The *wxs_install_root* directory is the root directory where WebSphere eXtreme Scale product files are installed. This can be the directory in which the trial zip file is extracted or the directory in which the eXtreme Scale product is installed.

Example when extracting the trial:

```
/opt/IBM/WebSphere/eXtremeScale
```

Example when eXtreme Scale is installed to a stand-alone directory:

```
/opt/IBM/eXtremeScale
```

Example when eXtreme Scale is integrated with WebSphere Application Server:

```
/opt/IBM/WebSphere/AppServer
```

wxs_home

The *wxs_home* directory is the root directory of the WebSphere eXtreme Scale product libraries, samples and components. This is the same as the *wxs_install_root* directory when the trial is extracted. For standalone installations, this is the *ObjectGrid* sub-directory within the *wxs_install_root*. For installations that are integrated with WebSphere Application Server, this directory is the *optionalLibraries/ObjectGrid* directory within *wxs_install_root*.

Example when extracting the trial:

```
/opt/IBM/WebSphere/eXtremeScale
```

Example when eXtreme Scale is installed to a stand-alone directory:

```
/opt/IBM/eXtremeScale/ObjectGrid
```

Example when eXtreme Scale is integrated with WebSphere Application Server:

```
/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid
```

was_root

The *was_root* directory is the root directory of a WebSphere Application Server installation.

Example /opt/IBM/WebSphere/AppServer

restservice_home

The *restservice_home* directory is the directory in which the eXtreme Scale REST data service libraries and samples are located. This directory is named *restservice* and is a sub-directory under the *wxs_home* directory.

Example for stand-alone deployments:

```
/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice
```

Example for WebSphere Application Server integrated deployments:

```
/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/  
restservice
```

tomcat_root

The *tomcat_root* is the root directory of the Apache Tomcat installation.

Example /opt/tomcat5.5

wasce_root

The *wasce_root* is the root directory of the WebSphere Application Server Community Edition installation.

Example /opt/IBM/WebSphere/AppServerCE

java_home

The *java_home* is the root directory of a Java SE Development Kit (JDK) installation.

Example /opt/java

3 Getting started

This topic describes how to quickly get started with the WebSphere eXtreme Scale REST data service. Instructions are provided for WebSphere Application Server version 7.0, WebSphere Application Server Community Edition and Apache Tomcat.

The included sample has source code and compiled binaries to run a partitioned eXtreme Scale grid. This sample demonstrates how to create a simple grid, model the data using eXtreme Scale entities and provides two command-line client applications that allow adding and querying entities using Java or C# (see Figure 3).

The sample Java client uses the eXtreme Scale Java EntityManager API to persist and query data in the grid. This client can be run in Eclipse or using a command-line script. Note that the sample Java client does not demonstrate the REST data service, but allows updating data in the grid, so a web browser or other clients can read the data. The sample Java client and web browser, as shown in Figure 3, illustrate HTTP clients using the REST data service and eXtreme Scale Java clients using the same eXtreme Scale grid and data contained therein.

The sample Microsoft WCF Data Services C# client communicates with the eXtreme Scale grid through the REST data service using the .NET framework. The WCF Data Services client can be used to both update and query the grid.

In general, the following steps need to be completed in order to use the sample:

1. Configure and start the eXtreme Scale grid (see section 3.3)
2. Configure and start the REST data service in a web server (see section 3.3).
3. Run a client to interact with the REST data service. Two options are available:
 - a. Run the sample Java client to populate the grid with data using the EntityManager API and query the data in the grid using a web browser and the eXtreme Scale REST data service. See sections 3.4 and 3.4.2).
 - b. Run the sample WCF Data Services C# client, see section 3.6.

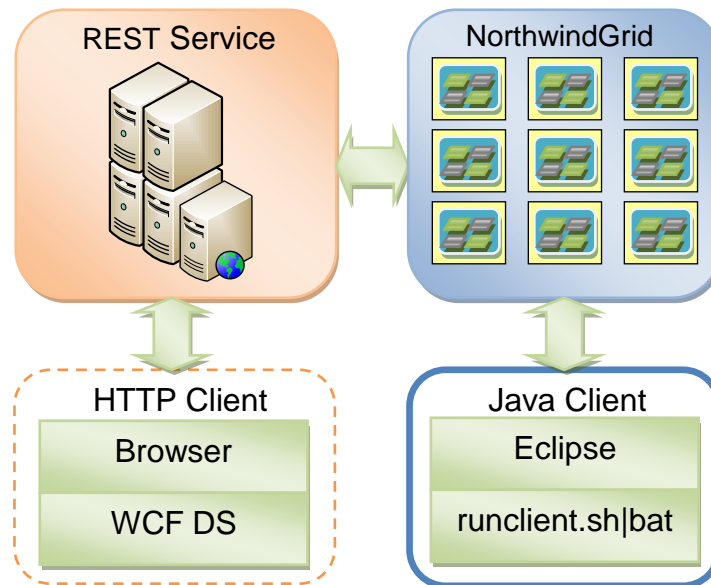


Figure 3: Getting started sample topology

3.1 Getting started sample overview

The getting started sample uses entities similar to the tables included with the [Microsoft SQL Server Northwind database sample](#). Each database table is modeled in eXtreme Scale using EntityManager API entities and maps. The eXtreme Scale REST data service then utilizes the eXtreme Scale entity metadata to represent each entity as an EntitySet.

3.1.1 Creating a scalable data model in eXtreme Scale

The Microsoft Northwind sample uses the Order Detail table to establish a many-to-many association between Orders and Products. Object to relational mapping specifications (ORMs) such as the ADO.NET Entity Framework and Java Persistence API (JPA) can map the tables and relationships using entities. However, this architecture is does not scale. Everything must be located on the same machine, or an expensive cluster of machines to perform well.



Figure 4: The Microsoft SQL Server Northwind sample schema diagram

In order to create a scalable version of the sample, the entities must be modeled such that each entity or group of related entities can be partitioned based off a single key. By doing this, requests can be spread out among multiple, independent servers. To achieve this, the entities have been divided into two trees: The Customer/Order tree (see Figure 5) and the Product/Category tree (see Figure 6). In this model, each tree can be partitioned independently and therefore can grow at different rates and scale.

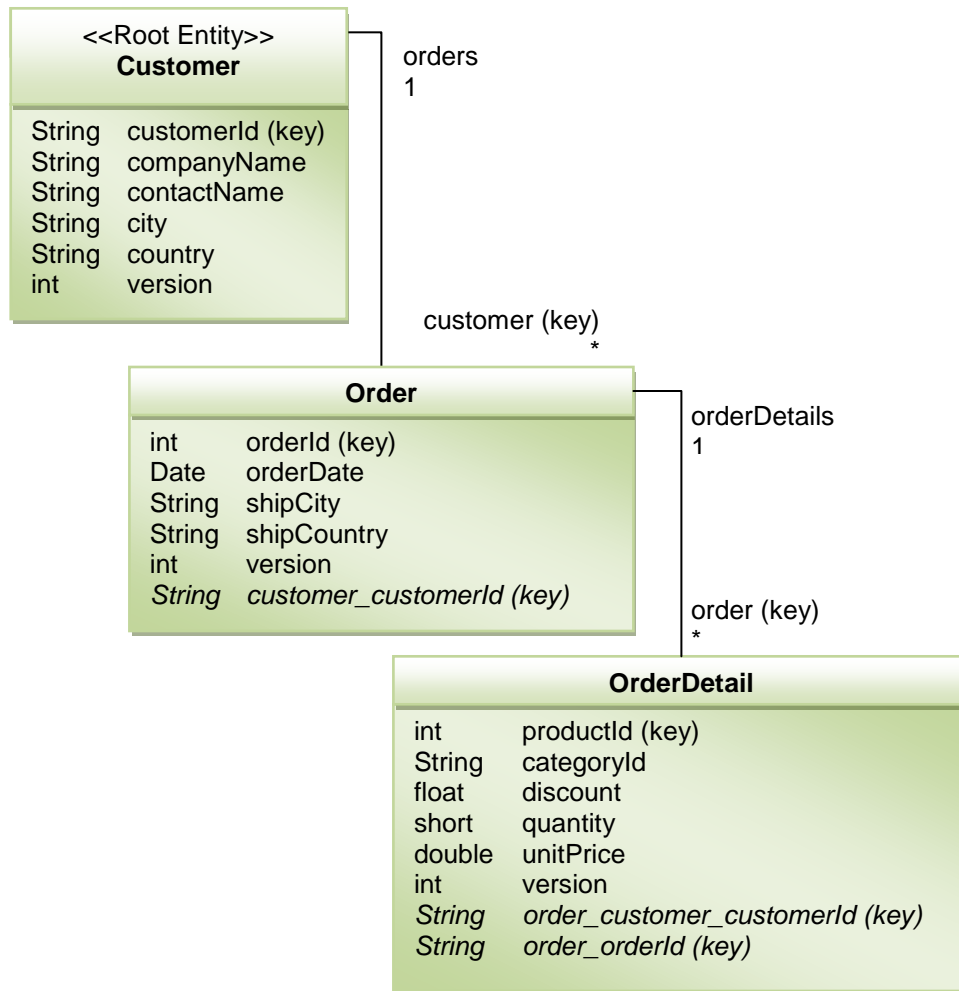


Figure 5: The Customer and Order entity schema diagram

For example, both Order and Product have unique, separate integers as keys. In fact, the Order and Product tables are really independent of each other. For example, consider the effect of the size of a catalog (number of products you sell) with the total number of orders. Intuitively, it may seem that having many products implies also having many orders, but this is not necessarily the case. If this were true, you could easily increase sales by just adding more products to your catalog. Orders and products have their own independent tables. You can further extend this concept so that orders and products each have their own separate, eXtreme Scale grids. With independent grids, you can control the number of partitions and servers, in addition to the size of each grid separately so that your application can scale. If you double the size of your catalog, you must double the products grid, but the ogrid may be unchanged. The converse is true for an order surge, or expected order surge.

In the schema, a Customer has zero or more Orders, and an Order has line items (OrderDetail), each with one specific product. A Product is identified by id (the Product key) in each OrderDetail. A single grid stores Customers, Orders and, with Customer as the root entity of the grid. You can retrieve Customers by Id, but you have to get Orders

starting with the Customer id. So customer id is added to Order as part of its key. Likewise, the customer id and order id are part of the OrderDetail id.

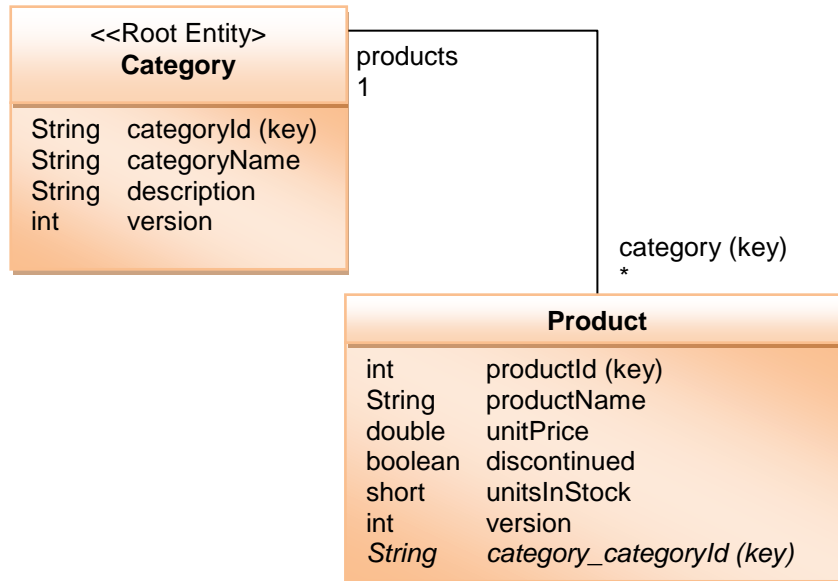


Figure 6: The Category and Product entity schema diagram

In the Category and Product schema, the Category is the schema root. This allows customers to query products by category. See the following section 3.1.2 for additional details on key associations and their importance.

3.1.2 Retrieving and updating data

The OData protocol requires that all entities can be addressed by their canonical form. This means that each entity must include the key of the partitioned, root entity (the schema root).

For example, in eXtreme Scale, you can use the association from the root entity to address the child: `/Customer('ACME')/order(100)`

In WCF Data Services, the child entity must be directly addressable, meaning that the key in the schema root must be a part of the key of the child:

`/Order(customer_customerId='ACME', orderId=100).`

This is achieved by creating an association to the root entity where the one-to-one or many-to-one association to the root entity is also identified as a key. When entities are included as part of the key, the attributes of the parent entity are exposed as key properties.

In Figure 5, the Customer/Order entity schema diagram illustrates how each entity is partitioned using the Customer. The Order entity includes the Customer as part of its key and is therefore directly accessible. The REST data service exposes all key associations as individual properties: Order has `customer_customerId` and OrderDetail has `order_customer_customerId` and `order_orderId`.

Using the EntityManager API, you can find the Order using the Customer and order id:

```
transaction.begin();
// Look-up the Order using the Customer. We only include the Id
// in the Customer class when building the OrderId key instance.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

When using the REST data service, the Order can be retrieved with either of the URLs:

- /Order(orderId=100, customer_customerId='ACME')
- /Customer('ACME')/orders?\$filter=orderId eq 100

The customer key is addressed using the attribute name of the Customer entity, an underscore character and the attribute name of the customer id: customer_customerId.

An entity can also include a non-root entity as part of its key if all of the ancestors to the non-root entity have key associations to the root. In this example, OrderDetail has a key-association to Order and Order has a key-association to the root Customer entity. Using the EntityManager API:

```
transaction.begin();
// Construct an OrderDetailId key instance. It includes
// The Order and Customer with only the keys set.
Customer customerACME = new Customer("ACME");
Order order100 = new Order(100, customerACME)
OrderDetailId orderDetailKey =
    new OrderDetailId(order100, "COMP");
OrderDetail orderDetail = (OrderDetail)
    em.find(OrderDetail.class, orderDetailKey);
...
```

The REST data service allows addressing the OrderDetail directly:

- /OrderDetail(productId=500, order_customer_customerId='ACME', order_orderId =100)

The association from the OrderDetail entity to the Product entity has been broken to allow partitioning the Orders and Product inventory independently. The OrderDetail entity stores the category and product id instead of a hard relationship. By decoupling the two entity schemas, only one partition is accessed at a time.

The Category and Product schema, illustrated in Figure 6, shows that the root entity is Category and each Product has an association to a Category entity. The Category entity is included in the Product identity. The REST data service exposes a key property: category_categoryId which allows directly addressing the Product.

Since Category is the root entity, in a partitioned environment, the Category must be known in order to find the Product. Using the EntityManager API, the transaction must be pinned to the Category entity prior to finding the Product. Using the EntityManager API:

```
transaction.begin();
// Create the Category root entity with only the key. This
// allows us to construct a ProductId without needing to find
// The Category first. The transaction is now pinned to the
```

```
// partition where Category "COMP" is stored.
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

The REST data service allows addressing the Product directly:

- `/Product(productId=500, category_categoryId='COMP')`

3.2 Starting the sample eXtreme Scale grid

Before starting the REST data service, start the eXtreme Scale grid. The following steps will start a single eXtreme Scale catalog service process and two container server processes.

WebSphere eXtreme Scale can be installed using three different methods:

- Trial install - See section 3.2.1.
- Stand-alone deployment - See section 3.2.1.
- WebSphere Application Server integrated deployment – See section 3.2.2.

3.2.1 Starting the sample grid for a stand-alone deployment

Follow these steps to start the WebSphere eXtreme Scale REST service sample grid for a stand-alone eXtreme Scale deployment.

Before you begin

Install the WebSphere eXtreme Scale Trial or full product:

- [Install](#) the stand-alone version of the WebSphere eXtreme Scale 7.0 product and apply WebSphere eXtreme Scale 7.0.0.0 cumulative fix 2.
- [Download](#) and extract the WebSphere eXtreme Scale Version 7.0 trial which includes the WebSphere eXtreme Scale REST data service.

About this task

Start the WebSphere eXtreme Scale sample grid:

A. Start the catalog service process:

1. Open a command-line or terminal window and set the JAVA_HOME environment variable:

Windows

```
set JAVA_HOME=<java_home>
```

>

UNIX

```
export JAVA_HOME=<java_home>
```

>

2. `cd restservice_home/gettingstarted`

-
3. Start the catalog service process:

```
Windows  
runcat.bat  
UNIX  
./runcat.sh
```

- B. Start two container server processes:

1. Open another command-line or terminal window and set the JAVA_HOME environment variable:

```
Windows  
set JAVA_HOME=<java_home>  
>  
UNIX  
export JAVA_HOME=<java_home>  
>
```

2. `cd restservice_home/gettingstarted`

3. Start a container server process:

```
Windows  
runcontainer.bat container0  
UNIX  
./runcontainer.sh container0
```

4. Open another command-line or terminal window and set the JAVA_HOME environment variable:

```
Windows  
set JAVA_HOME=<java_home>  
>  
UNIX  
export JAVA_HOME=<java_home>  
>
```

5. `cd restservice_home/gettingstarted`

6. Start a second container server process:

```
Windows  
runcontainer.bat container1  
UNIX  
./runcontainer.sh container1
```

7. Wait until the eXtreme Scale containers are ready before proceeding with the next steps. The container servers are ready when the following message is displayed in the terminal window:

```
CWOBJ1001I: ObjectGrid Server <container name> is ready to  
process requests.
```

Where <container name> is the name of the container that was started in the previous step.

3.2.2 Starting the sample grid for a WebSphere Application Server integrated deployment

Follow these steps to start a stand-alone WebSphere eXtreme Scale REST service sample grid for a WebSphere eXtreme Scale deployment that is integrated with WebSphere Application Server. Although eXtreme Scale is integrated with WebSphere Application

Server, these steps will start a stand-alone eXtreme Scale catalog service process and container

Before you begin

[Install](#) the WebSphere eXtreme Scale 7.0 product into a WebSphere Application Server Version 7.0.0.5 or later installation directory (with security disabled), augment at least one Application Server profile and apply WebSphere eXtreme Scale 7.0.0.0 cumulative fix 2.

About this task

Start the WebSphere eXtreme Scale sample grid:

A. Start the catalog service process:

1. Open a command-line or terminal window.
2. `cd restservice_home/gettingstarted`
3. Start the catalog service process:

Windows
<code>runcat.bat</code>
UNIX
<code>./runcat.sh</code>

B. Start two container server processes:

1. Open another command-line or terminal window.
2. `cd restservice_home/gettingstarted`
3. Start a container server process:

Windows
<code>runcontainer.bat container0</code>
UNIX
<code>./runcontainer.sh container0</code>
4. Open another command-line or terminal window.
5. `cd restservice_home/gettingstarted`
6. Start a second container server process:

Windows
<code>runcontainer.bat container1</code>
UNIX
<code>./runcontainer.sh container1</code>
7. Wait until the eXtreme Scale containers are ready before proceeding with the next steps. The container servers are ready when the following message is displayed in the terminal window:

```
CWOBJ1001I: ObjectGrid Server <container name> is ready to process requests.
```

Where <container name> is the name of the container that was started in the previous step.

3.3 Configuring and starting your web server

The eXtreme Scale REST data service is hosted by a JEE web server. The following web servers are supported (see section 4.1 for detailed software requirements):

- WebSphere Application Server (see section 3.3.1),
- WebSphere Application Server integrated with WebSphere eXtreme Scale (see section 3.3.2),
- WebSphere Application Server Community Edition (see section 3.3.3) and
- Apache Tomcat (see section 3.3.4)

The following sections describe how to install the eXtreme Scale REST data service into the web server.

Note: These instructions assume you will use the trial version of WebSphere eXtreme Scale. To run the REST data service with your installed version of WebSphere eXtreme Scale version 7, see section 3.3.2.

3.3.1 Getting started with WebSphere Application Server version 7.0

This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere Application Server version 7.0.

If the WebSphere Application Server installation is integrated with WebSphere eXtreme Scale, see section 3.3.2.

Before you begin

Verify that the sample eXtreme Scale grid is started. See section 3.2.1 for details on how to start the grid.

About this task

To get started with the WebSphere eXtreme Scale REST data service using WebSphere Application Server, follow these steps:

- A. [Download](#) and install WebSphere Application Server Version 7.0 for Developers.

Note: Do not enable security.

- B. [Download](#) and install WebSphere Application Server Version 7.0 Fix Pack 5 or later.

- C. Add the WebSphere eXtreme Scale client runtime JAR, `wsogclient.jar`, and the REST data service configuration JAR or directory to the application server classpath:

1. Open the WebSphere Administration Console
2. Navigate to **Environment -> Shared libraries**
3. Click **New**
4. Add the following entries into the fields:
 - a. Name: `extremescale_client_v7`

-
- b. Classpath:
`wxs_home/lib/wsogclient.jar`
 5. Click **OK**
 6. Click **New**
 7. Add the following entries into the appropriate fields:
 - a. Name: `extremescale_gettingstarted_config`
 - b. Classpath:
`restservice_home/gettingstarted/restclient/bin`
`restservice_home/gettingstarted/common/bin`

Note: Add each path on a separate line.
 8. Click **OK**
 9. Save the changes to the master configuration
- D. Install the REST data service web module, `wxsrestservice.war`, to the WebSphere Application Server using the WebSphere administration console:
1. Open the WebSphere administration console
 2. Navigate to **Applications -> New Application**
 3. Browse to the `restservice_home/lib/wxsrestservice.war`, select the file and click **Next**.
 4. Choose the detailed installation options, and click **Next**.
 5. On the application security warnings screen, click **Continue**.
 6. Choose the default installation options, and click **Next**.
 7. Choose a server to map the application to, and click **Next**.
 8. On the JSP reloading page, use the defaults, and click **Next**.
 9. On the shared libraries page, map the "wxsrestservice_war" module to the following shared libraries that were defined during step C:
 - `extremescale_client_v7`
 - `extremescale_gettingstarted_config`
 10. On the map shared library relationship page, use the defaults, and click **Next**.
 11. On the map virtual hosts page, use the defaults, and click **Next**
 12. On the map context roots page, set the context root to: `/wxsrestservice`
 13. On the Summary screen, click **Finish** to complete the installation.
 13. Save the changes to the master configuration
- E. Start the application server and the "wxsrestservice_war" eXtreme Scale REST data service application.
1. After the application is started review the SystemOut.log for the application server and verify that the following message appears:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- F. Verify that the REST data service is working:

-
1. Open a browser and navigate to:
<http://localhost:9080/wxsrestservice/restservice/NorthwindGrid>
The service document for the NorthwindGrid is displayed.
 2. Navigate to:
[http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/\\$metadata](http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata)
The Entity Model Data Extensions (EDMX) document is displayed.

G. To stop the grid processes, use CTRL+C in the respective command window.

3.3.2 Getting started with WebSphere eXtreme Scale integrated with WebSphere Application Server version 7.0

This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere Application Server version 7.0 that has been integrated and augmented with WebSphere eXtreme Scale. If using WebSphere Application Server version 7.0 that has not been integrated with WebSphere eXtreme Scale, see section 3.3.1.

Before you begin

Verify that the sample stand-alone eXtreme Scale grid is started. See section 3.2.2 for details on how to start a stand-alone grid in an environment that has WebSphere Application Server integrated with WebSphere eXtreme Scale

About this task

To get started with the WebSphere eXtreme Scale REST data service using WebSphere Application Server, follow these steps:

- A. Add the WebSphere eXtreme Scale REST data service sample configuration JAR to the classpath:
 1. Open the WebSphere Administration Console
 2. Navigate to **Environment -> Shared libraries**
 3. Click **New**
 4. Add the following entries into the appropriate fields:
 - a. Name: extremescale_gettingstarted_config
 - b. Classpath:
`restservice_home/gettingstarted/restclient/bin`
`restservice_home/gettingstarted/common/bin`
Note: Each path must appear on a different line.
 5. Click **OK**
 6. Save the changes to the master configuration
- B. Install the REST data service web module, wxsrestservice.war, to the WebSphere Application Server using the WebSphere administration console:
 1. Open the WebSphere administration console
 2. Navigate to **Applications -> New Application**
 3. Browse to `restservice_home/lib/wxsrestservice.war` file on the file system. Select the file and click **Next**.
 4. Choose the detailed installation options, and click **Next**.

-
5. On the application security warnings screen, click **Continue**.
 6. Choose the default installation options, and click **Next**.
 7. Choose a server to map the application to, and click **Next**.
 8. On the JSP reloading page, use the defaults, and click **Next**.
 9. On the shared libraries page, map the "wxsrestservice_war" module to the following shared libraries that were defined during step A:
 - extremescale_gettingstarted_config
 10. On the map shared library relationship page, use the defaults, and click **Next**.
 11. On the map virtual hosts page, use the defaults, and click **Next**.
 12. On the map context roots page, set the context root to: /wxsrestservice
 13. On the Summary screen, click **Finish** to complete the installation.
 14. Save the changes to the master configuration
- C. Start the application server and the "wxsrestservice_war" eXtreme Scale REST data service application.
1. After the application is started review the SystemOut.log for the application server and verify that the following message appears:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- D. Verify that the REST data service is working:
1. Open a browser and navigate to:
<http://localhost:9080/wxsrestservice/restservice/NorthwindGrid>
The service document for the NorthwindGrid is displayed.
 2. Navigate to:
[http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/\\$metadata](http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata)
The Entity Model Data Extensions (EDMX) document is displayed.
- E. To stop the grid processes, use CTRL+C in the respective command window to stop the process.

3.3.3 Getting started with WebSphere Application Server Community Edition

This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere Application Server Community Edition.

Before you begin

Verify that the sample eXtreme Scale grid is started. See section 3.2 for details on how to start the grid.

About this task

To get started with the WebSphere eXtreme Scale REST data service using WebSphere Application Server Community Edition (CE), follow these steps:

- A. [Download](#) and install WebSphere Application Server CE 2.1.1.3 or later to *wasce_root*.
For example: /opt/IBM/wasce

-
- B. Start the WebSphere Application Server CE server by running the following command:

Windows
`wasce_root/bin/startup.bat`

UNIX
`wasce_root/bin/startup.sh`

- C. Install the eXtreme Scale REST data service and the provided sample into the WebSphere Application Server CE server:

1. Add the ObjectGrid client runtime JAR to the WebSphere Application Server CE repository:

- a. Open the WebSphere Application Server CE administration console and log in.

Note: The default URL is: <http://localhost:8080/console> and the default userid is "system" and password is "manager".

- b. Click the "Repository" link on the left side of the console window, in the Services folder.

- c. In the "Add Archive to Repository" section, fill in the following into the input text boxes:

File: `wxs_home/lib/ogclient.jar`
Group: `com.ibm.websphere.xs`
Artifact: `ogclient`
Version: `7.0`
Type: `jar`

- d. Click the Install button.

Note: See the following tech note for details on different methods of configuration class and library dependencies:
<http://www.ibm.com/support/docview.wss?uid=swg21266061>

2. Deploy the REST data service module: `wxsrestservice.war` to the WebSphere Application Server CE server.

- a. Edit the sample `restservice_home/gettingstarted/wasce/geronimo-web.xml` deployment XML file to include path dependencies to the getting started sample classpath directories:

- Change the "classesDirs" for the two getting started client GBeans:

The "classesDirs" path for the GettingStarted_Client_SharedLib GBean should be set to:
`restservice_home/gettingstarted/restclient/bin`

The "classesDirs" path for the GettingStarted_Common_SharedLib GBean should be set to:
`restservice_home/gettingstarted/common/bin`

- b. Open the WebSphere Application Server CE administration console and log in.

Note: The default URL is: <http://localhost:8080/console> and the default userid is "system" and password is "manager".

- c. Click on the "Deploy New" link on the left side of the console window.

-
- d. On the "Install New Applications" page, enter the following values into the text boxes:
 - Archive: `restservice_home/lib/wxsrestservice.war`
 - Plan: `restservice_home/gettingstarted/wasce/geronimo-web.xml`
 - e. Click on the Install button.
The console page should indicate that the application was successfully installed and started.
 - f. Examine the WebSphere Application Server CE system output log or console to verify that the REST data service has started successfully by verify that the following message is present:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- D. Verify that the REST data service is working:
1. Open a browser and navigate to:
<http://localhost:8080/wxsrestservice/restservice/NorthwindGrid>
The service document for the NorthwindGrid grid is displayed.
 2. Navigate to:
[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/\\$metadata](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata)
The Entity Model Data Extensions (EDMX) document is displayed.
- E. To stop the grid processes, use CTRL+C in the respective command window to kill the process.
- F. To stop WebSphere Application Server CE, use the following command:
- Windows**
`wasce_root\bin\shutdown.bat`
- UNIX**
`wasce_root/bin/shutdown.sh`
- Note:* The default userid is "system" and password is "manager". If using a custom port use the `-port` option.

3.3.4 Getting started with Tomcat

This topic describes how to configure and start the eXtreme Scale REST data service using Apache Tomcat, version 5.5 or later.

Before you begin

Verify that the sample eXtreme Scale grid is started. See section 3.2 for details on how to start the grid.

About this task

To get started with the WebSphere eXtreme Scale REST data service using Apache Tomcat, follow these steps:

- A. [Download](#) and install Apache Tomcat Version 5.5 or later to `tomcat_root`.
For example: `/opt/tomcat`
- B. Install the eXtreme Scale REST data service and the provided sample into the Tomcat server
 1. If using a Sun JRE or JDK, you must install the IBM ORB into Tomcat:

If using Tomcat version 5.5:

- a. Copy all of the JAR files from:
wxs_home/lib/endorsed
to:
tomcat_root/common/endorsed

If using Tomcat version 6.0:

- a. Create an "endorsed" directory:

Windows
`md tomcat_root/endorsed`

UNIX
`mkdir tomcat_root/endorsed`

- b. Copy all of the JAR files from:
wxs_home/lib/endorsed
to:
tomcat_root/endorsed

2. Deploy the REST data service module: *wxsrestservice.war* to the Tomcat server.
 - a. Copy the *wxsrestservice.war* file from:
restservice_home/lib
to:
tomcat_root/webapps
 3. Add the ObjectGrid client runtime JAR and the application JAR to the shared classpath in Tomcat:
 - a. Edit the *tomcat_root/conf/catalina.properties* file
 - b. Append the following path names to the end of the `shared.loader` property, separating each with a comma:
 - *wxs_home/lib/ogclient.jar*
 - *restservice_home/gettingstarted/restclient/bin*
 - *restservice_home/gettingstarted/common/bin*

Note: The path separator must be a forward slash.
- C. Start the Tomcat server with the REST data service:


If using Tomcat 5.5 on UNIX or Windows, or Tomcat 6.0 on UNIX:

1. `cd tomcat_root/bin`
2. Start the server:

Windows
`catalina.bat run`

UNIX
`./catalina.sh run`
3. The Apache Tomcat logs are displayed to the console. When the REST data service has started successfully, the following message is displayed in the administration console:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.



If using Tomcat 6.0 on Windows:

-
1. `cd tomcat_root/bin`
 2. Start the Apache Tomcat 6 configuration tool:
 `tomcat6w.exe`
 3. Click on the Start button on the Apache Tomcat 6 properties window to start the Tomcat server.
 4. Review the following logs to verify that the Tomcat server has started successfully:
 - `tomcat_root/bin/catalina.log`
Displays the status of the Tomcat server engine
 - `tomcat_root/bin/stdout.log`
Displays the system output log.
 5. When the REST data service has started successfully, the following message is displayed in the system output log:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- D. Verify that the REST data service is working:
1. Open a browser and navigate to:
<http://localhost:8080/wxsrestservice/restservice/NorthwindGrid>
The service document for the NorthwindGrid is displayed.
 2. Navigate to:
[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/\\$metadata](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata)
The Entity Model Data Extensions (EDMX) document is displayed.
- E. To stop the grid processes, use CTRL+C in the respective command window.
- F. To stop Tomcat, use CTRL +C in the window in which you started it.

3.4 Adding data with the sample Java client application

The previous sections described how to create an eXtreme Scale grid and configure and start the eXtreme Scale REST data service. The Java client application uses the eXtreme Scale EntityManager API to insert data into the grid. It does not demonstrate how to use the REST interfaces. The purpose of this client is to demonstrate how the EntityManager API is used to interact with the eXtreme Scale grid, and allow modifying data in the grid. To view data in the grid using the REST data service, use a web browser or use the Visual Studio 2008 client application described in section 3.4.2.

To quickly add content to the eXtreme Scale grid, run the following command:

1. Open a command-line or terminal window and set the JAVA_HOME environment variable:
 `set JAVA_HOME=<java_home`
>
 `export JAVA_HOME=<java_home`
>
2. `cd restservice_home/gettingstarted`

-
3. Insert some data into the grid. The data that is inserted will be retrieved later using a Web browser and the REST data service.

Windows

```
runclient.bat load default
```

UNIX

```
./runclient.sh load default
```

3.4.1 Java client command syntax

Windows

```
runclient.bat <command>
```

UNIX

```
runclient.sh <command>
```

The following commands are available:

- `load default`
Loads a predefined set of Customer, Category and Product entities into the grid and creates a random set of Orders for each customer.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`
Creates a product Category and a fixed number of Product entities in the grid. The `firstProductId` parameter identifies the id number of the the first product and each subsequent product is assigned the next id until the specified number of products is created.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`
Loads a new Customer into the grid and creates a fixed set of Order entities for any random product currently loaded in the grid. The number of Orders is determined by setting the `<numOrders>` parameter. Each Order will have a random number of OrderDetail entities up to `<maxItems>`
- `display customer <companyCode>`
Display a Customer entity and the associated Order and OrderDetail entities.
- `display category <categoryId>`
Display a product Category entity and the associated Product entities.

Examples:

- `runclient.bat load default`
- `runclient.bat load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `runclient.bat load category 5 "Household Items" 100 5`
- `runclient.bat display customer IBM`
- `runclient.bat display category 5`

3.4.2 Running and building the sample grid and Java client with Eclipse

The REST data service getting started sample can be updated and enhanced using Eclipse. For details on how to setup your Eclipse environment see the text document: `restservice_home/gettingstarted/ECLIPSE_README.txt`.

Once the WXSRestGettingStarted project is imported into Eclipse and is building successfully, the sample will automatically re-compile and the script files used to start the container server and client will automatically pick up the class files and XML files. The REST data service will also automatically detect any changes since the Web server is configured to read the Eclipse build directories automatically.

Note: When changing source or configuration files, both the eXtreme Scale container server and the REST data service application must be restarted. The eXtreme Scale container server must be started before the REST data service Web application.

3.5 Using a web browser to view sample data

The eXtreme Scale REST data service creates ATOM feeds by default when using a web browser. The ATOM feed format may not be compatible with older browsers or may be interpreted such that the data cannot be viewed as XML. For older browsers, you will be prompted to save the files to disk. Once the files are downloaded, use your favorite XML reader to look at the files. The generated XML is not formatted to be displayed, so everything will be printed on one line. Most XML reading programs, such as Eclipse, support reformatting the XML into a readable format.

For modern browsers, such as Microsoft Internet Explorer Version 8 and Firefox Version 3, the ATOM XML files can be displayed natively in the browser. The following topics provide details on how to configure Internet Explorer Version 8 and Firefox Version 3 to display the ATOM feeds and XML within the browser.

Once the browser is configured, see section 3.5.3 for example URLs.

3.5.1 Configuring Internet Explorer Version 8

To enable Internet Explorer to read the ATOM feeds that the REST data service generates use the following steps, as shown in **Error! Reference source not found.**:

1. Click **Tools -> Internet Options**
2. Click the **Content** tab
3. Click the **Settings** button in the Feeds and Web Slices section.
4. Uncheck the box: **Turn on feed reading view.**
5. Click **OK** to return to the browser and restart Internet Explorer.

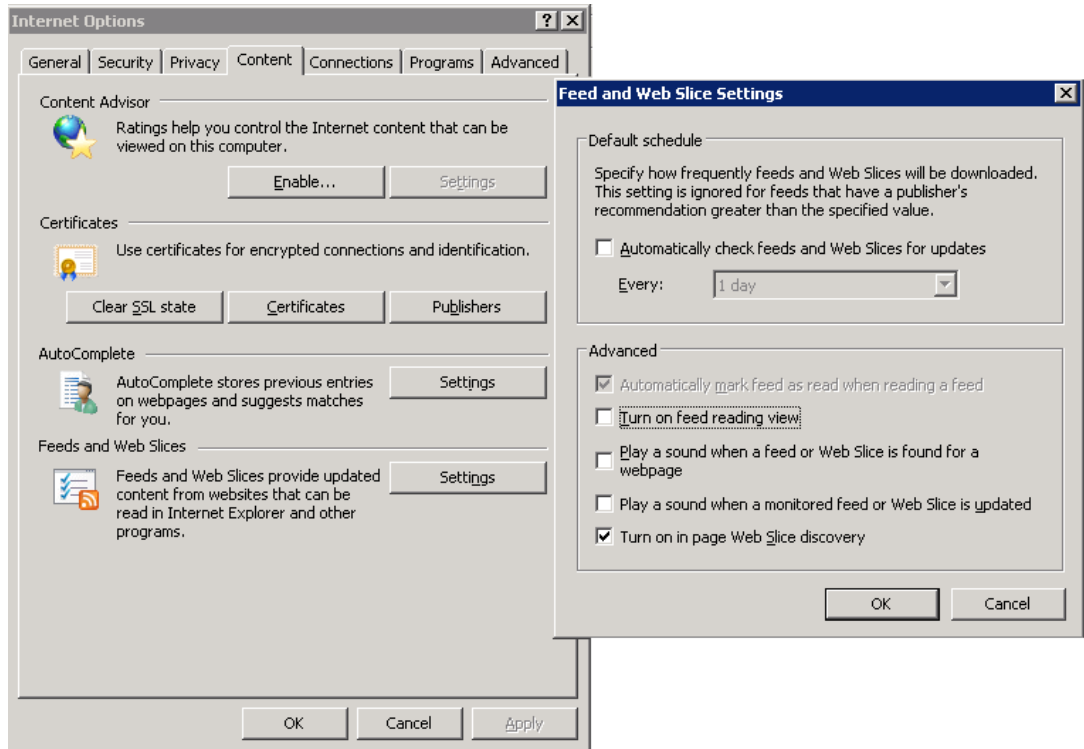


Figure 7: Configuring Internet Explorer 8 to display ATOM feeds as XML

3.5.2 Configuring Firefox Version 3

Firefox will not automatically display pages with content type: application/atom+xml. The first time a page is displayed, it will prompt to save the file. To display the page, open the file itself with Firefox:

1. From the application chooser dialog box (See Select the Open with radio button and click the Browse button.

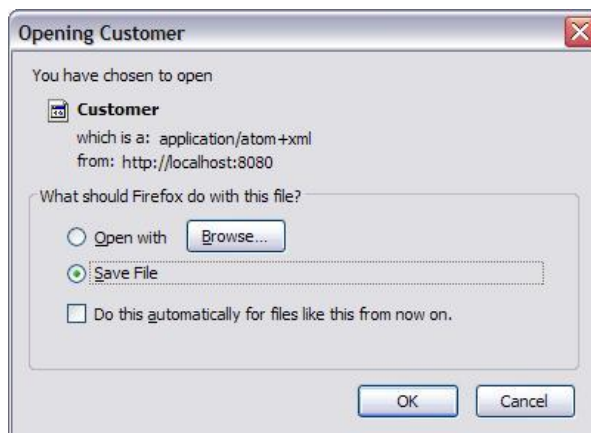


Figure 8: Firefox Application Chooser Window

2. Navigate to your Firefox installation directory. For example: C:\Program Files\Mozilla Firefox
3. Select firefox.exe and hit the **OK** button.

-
4. Check the “**Do this automatically for files like this...**” check box.
 5. Click the **OK** button.
 6. The ATOM XML page is now displayed in a new Firefox browser window or tab.

Firefox automatically renders ATOM feeds in readable format. However, the feeds that the REST data service creates include XML. Firefox cannot display the XML unless you disable the feed renderer. Unlike Internet Explorer, in Firefox, the ATOM feed rendering plug-in must be explicitly edited. To configure Firefox to read ATOM feeds as XML files, follow these steps:

1. Open the following file in a text editor:
`<firefoxInstallRoot>\components\FeedConverter.js`
Where `<firefoxInstallRoot>` is the root directory where Firefox is installed.

For Windows operating systems, the default directory is: C:\Program Files\Mozilla Firefox.

2. Search for the line that looks as follows:

```
// show the feed page if it wasn't sniffed and we have a document,  
// or we have a document, title, and link or id  
if (result.doc && (!this._sniffed ||  
    (result.doc.title && (result.doc.link || result.doc.id)))) {
```

3. Comment out the two lines that begin with *if* and *result* by placing two // forward slashes in front of them.
4. Add the following *if* statement below:

```
if(0) {
```

5. The resulting text should look as follows:

```
// show the feed page if it wasn't sniffed and we have a document,  
// or we have a document, title, and link or id  
//if (result.doc && (!this._sniffed ||  
//    (result.doc.title && (result.doc.link || result.doc.id)))) {  
if(0) {
```

6. Save the file.
7. Restart Firefox.
8. All feeds should now be automatically displayed in the browser.

3.5.3 Example URLs

This section describes some URLs that can be used to view the data that was added by the getting started sample provided with the eXtreme Scale REST data service. Before using the following URLs, add the default data set to the eXtreme Scale sample grid using either the sample Java client (see section 3.3.4) or the sample Visual Studio WCF Data Services client (see section 3.6).

The following examples assume the port is 8080 which can vary. See section 3.3 for details on how to configure the REST data service on different application servers.

View a single customer with the id of "ACME":

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('ACME'\)](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME'))

View all of the orders for customer "ACME":

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('ACME'\)/orders](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders)

View the customer "ACME" and the orders:

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('ACME'\)?\\$expand=orders](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders)

View order 1000 for customer "ACME":

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order\(orderId=1000,customer_customerId='ACME'\)](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME'))

View order 1000 for customer "ACME" and its associated Customer:

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order\(orderId=1000,customer_customerId='ACME'\)?\\$expand=customer](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')?$expand=customer)

View order 1000 for customer "ACME" and its associated Customer and OrderDetails:

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order\(orderId=1000,customer_customerId='ACME'\)?\\$expand=customer,orderDetails](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderDetails)

View all orders for customer "ACME" for the month of October, 2009 (GMT):

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\(customerId='ACME'\)/orders?\\$filter=orderDate ge datetime'2009-10-01T00:00:00' and orderDate lt datetime'2009-11-01T00:00:00'](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer(customerId='ACME')/orders?$filter=orderDate ge datetime'2009-10-01T00:00:00' and orderDate lt datetime'2009-11-01T00:00:00')

View all the first 3 orders and orderDetails for customer "ACME" for the month of October, 2009 (GMT):

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\(customerId='ACME'\)/orders?\\$filter=orderDate ge datetime'2009-10-01T00:00:00' and orderDate lt datetime'2009-11-01T00:00:00'&\\$orderby=orderDate&\\$top=3&\\$expand=orderDetails](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer(customerId='ACME')/orders?$filter=orderDate ge datetime'2009-10-01T00:00:00' and orderDate lt datetime'2009-11-01T00:00:00'&$orderby=orderDate&$top=3&$expand=orderDetails)

3.6 Using the sample Visual Studio 2008 WCF Data Services client application

The eXtreme Scale REST data service getting started sample includes a WCF Data Services client that can interact with the eXtreme Scale REST data service. The sample is written as a command-line application in C#.

3.6.1 Software requirements

The WCF Data Services C# sample client requires the following:

- One of the following operating systems:
 - Microsoft Windows XP
 - Microsoft Windows Server 2003
 - Microsoft Windows Server 2008
 - Microsoft Windows Vista

-
- Microsoft Visual Studio 2008 with Service Pack 1
<http://www.microsoft.com/downloads/details.aspx?FamilyID=fbee1648-7106-44a7-9649-6d9f6d58056>

Note: See the previous link for additional hardware and software requirements.

- Microsoft .NET Framework 3.5 Service Pack 1
<http://www.microsoft.com/downloads/details.aspx?FamilyID=AB99342F-5D1A-413D-8319-81DA479AB0D7>
- Microsoft .NET Framework 3.5 Service Pack 1 Update
<http://support.microsoft.com/kb/959209>

3.6.2 Building and running the getting started client

The WCF Data Services sample client includes a Visual Studio 2008 project and solution and the source code for running the sample. The sample must be loaded into Visual Studio 2008 and compiled into a Windows runnable program before it can be run.

To build and run the sample, see the text document:

`restservice_home/gettingstarted/VS2008_README.txt`.

3.6.3 WCF Data Services C# client command syntax

Windows

```
WXSRESTGettingStarted.exe <service URL> <command>
```

The <service URL> is the URL of the eXtreme Scale REST data service configured in section 3.3.

The following commands are available:

- `load default`
Loads a predefined set of Customer, Category and Product entities into the grid and creates a random set of Orders for each customer.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`
Creates a product Category and a fixed number of Product entities in the grid. The `firstProductId` parameter identifies the id number of the the first product and each subsequent product is assigned the next id until the specified number of products is created.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`
Loads a new Customer into the grid and creates a fixed set of Order entities for any random product currently loaded in the grid. The number of Orders is determined by setting the `<numOrders>` parameter. Each Order will have a random number of OrderDetail entities up to `<maxItems>`
- `display customer <companyCode>`
Display a Customer entity and the associated Order and OrderDetail entities.
- `display category <categoryId>`
Display a product Category entity and the associated Product entities.

-
- unload
Remove all entities that were loaded using the "default load" command.

Examples:

- WXSRestGettingStarted.exe
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load default
- WXSRestGettingStarted.exe
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05
- WXSRestGettingStarted.exe
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load category 5 "Household Items" 100 5
- WXSRestGettingStarted.exe
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display customer IBM
- WXSRestGettingStarted.exe
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display category 5

4 Installing the REST data service

This topic describes how to install the eXtreme Scale REST data service into a Web server.

4.1 Software requirements

The WebSphere eXtreme Scale REST data service is a Java Web application that can be deployed to any application server that supports Java servlet specification, Version 2.3 and a Java runtime environment, Version 5 or later.

The following software is required:

- Java Standard Edition 5 or later¹
- Web servlet container, Version 2.3 or later, which includes one of the following:
 - WebSphere Application Server Version 6.1.0.25 or later
 - WebSphere Application Server Version 7.0.0.5 or later
 - WebSphere Community Edition Version 2.1.1.3 or later
 - Apache Tomcat Version 5.5 or later

¹ Note that while WebSphere eXtreme Scale supports Java Standard Edition 1.4 or later, the REST data service requires Java Standard Edition 5 or later.

-
- WebSphere eXtreme Scale, Version 7.0.0.0 cumulative fix 2 or later (including the trial)

4.2 Packaging overview

The WebSphere eXtreme Scale REST data service includes a single WAR file `wxsrestservice.war`. The `wxsrestservice.war` includes a single servlet that acts as a gateway between your WCF Data Services client applications or any other HTTP REST client and an eXtreme Scale grid.

The REST data service includes a sample that allows you to quickly create an eXtreme Scale grid and interact with it using an eXtreme Scale client or the REST data service. See the Getting started topic, section 3 for details on using the sample.

When WebSphere eXtreme Scale 7.0.0.0 with cumulative fix 2 is installed or the WebSphere eXtreme Scale Version 7.0 trial is extracted, the following directories and files are included:

restservice_home/lib

The `lib` directory contains the `wxsrestservice.war` web application. The `wxsrestservice.war` is tightly coupled with the WebSphere eXtreme Scale runtime. If eXtreme Scale is upgraded to a new version or a fix pack applied, the `wxsrestservice.war` will need to be manually upgraded to the version installed in this directory.

restservice_home/gettingstarted

The `gettingstarted` directory contains a simple sample that demonstrates how to use the eXtreme Scale REST data service with an eXtreme Scale grid.

4.3 Packaging and deploying the REST data service

The REST data service is designed as a self-contained WAR module. To configure the REST data service, you must first package the REST data service configuration and optional eXtreme Scale configuration files into a JAR file or directory. This application packaging is then referenced by the web container server runtime. Figure 9 illustrates files used by the eXtreme Scale REST data service.

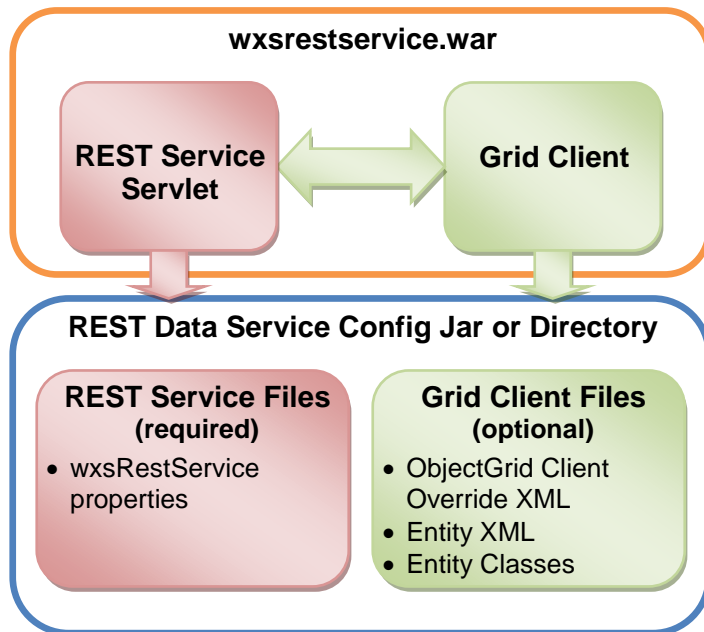


Figure 9: WebSphere eXtreme Scale REST Data Service Files

The REST service configuration JAR or directory must contain the following file:

- **wxsRestService.properties** – The `wxsRestService.properties` file includes the configuration options for the REST data service. This includes the catalog service endpoints, ObjectGrid names to expose, trace options and more. See section 5.1: Configuring the REST data service properties file for details.

The following ObjectGrid client files are optional:

- **META-INF/objectGridClient.xml** – The ObjectGrid client override XML file is used to connect to the remote eXtreme Scale grid. By default this file is not required. If not present, the REST service will use the server configuration, disabling the near cache.

The name of the file can be overridden using the [objectGridClientXML](#) REST data service configuration property. If provided, this XML file should include:

1. Include any ObjectGrids that you want to expose to the REST data service.
2. Include a reference to the entity descriptor xml file associated with each ObjectGrid configuration.

- **META-INF/<entity descriptor xml files>** - One or more entity descriptor XML files are required only if the client needs to override the entity definition of the client. The entity descriptor XML file must be used in conjunction with the ObjectGrid client override XML descriptor file. For details on the eXtreme Scale configuration files, see the [eXtreme Scale administration guide](#).

-
- **Entity classes** – Annotated entity classes or an entity descriptor XML file can be used to describe the entity metadata. The REST service only requires entity classes in the classpath if the eXtreme Scale servers are configured with entity metadata classes and a client override entity XML descriptor is not used.

An example with the minimum required configuration file, where the entities are defined in XML on the servers:

```
restserviceconfig.jar:
  wxsRestService.properties
    The property file contains:
      catalogServiceEndPoints=localhost:2809
      objectGridNames=NorthwindGrid
```

An example with one entity, override XML files and entity classes:

```
restserviceconfig.jar:
  wxsRestService.properties
    The property file contains:
      catalogServiceEndPoints=localhost:2809
      objectGridNames=NorthwindGrid

  com/acme/entities/Customer.class
  META-INF/objectGridClient.xml
    The client ObjectGrid descriptor XML file contains:
      <objectGrid name="CustomerGrid"
        entityMetadataXMLFile="emd.xml"/>
  META-INF/emd.xml
    The entity metadata descriptor XML file contains:
      <entity class-name="com.acme.entities.Customer"
        name="Customer"/>
```

For details on the EntityManager API and configuring an eXtreme Scale client and server, see the [WebSphere eXtreme Scale version 7 administration guide](#).

4.4 Deploying on WebSphere Application Server

This topic describes how to configure the eXtreme Scale REST data service on WebSphere Application Server or WebSphere Network Deployment Version 6.1.0.25 or later. These instructions also apply to deployments where WebSphere eXtreme Scale is integrated with the WebSphere Application Server deployment.

Prerequisites:

One of the following:

1. WebSphere Application Server with the stand-alone eXtreme Scale client:
 - The WebSphere eXtreme Scale Trial Version 7.0 with the REST data service is downloaded and extracted or the WebSphere eXtreme Scale 7.0.0.0 with cumulative fix 2 product is installed into a stand-alone directory.

-
- WebSphere Application Server Version 6.1.0.25 or 7.0.0.5 or later is installed and running with security disabled.
 - 2. WebSphere Application Server integrated with WebSphere eXtreme Scale:
 - WebSphere eXtreme Scale Version 7.0.0.0 with cumulative fix 2 is installed on top of WebSphere Application Server Version 6.1.0.25 or 7.0 (or later) with security disabled.

The eXtreme Scale REST data service only requires that the eXtreme Scale Client option be installed. The profile does not need to be augmented.

Procedure:

- A. Configure and start an eXtreme Scale grid.
 - 1. For details on configuring an eXtreme Scale grid for use with the REST data service, see the Configuring WebSphere eXtreme Scale topic.
 - 2. Verify that an eXtreme Scale client can connect to and access entities in the grid. For an example, see the [Getting Started](#) section of this document.
- B. Build the eXtreme Scale REST service configuration JAR or directory. See the Packaging and deploying the REST data service topic for details.
- C. Add the REST data service configuration JAR or directory to the application server classpath:
 - 1. Open the WebSphere administration console
 - 2. Navigate to **Environment -> Shared libraries**
 - 3. Click **New**
 - 4. Add the following entries into the appropriate fields:
 - a. Name: `extremescale_rest_configuration`
 - b. Classpath: `<REST service configuration jar or directory>`
 - 5. Click **OK**
 - 6. Save the changes to the master configuration
- D. If eXtreme Scale is integrated with the WebSphere Application Server installation, skip this step and proceed to step E. Otherwise, continue:

Add the WebSphere eXtreme Scale client runtime JAR, `wsogclient.jar`, and the REST data service configuration JAR or directory to the application server classpath:

- 1. Open the WebSphere administration console
- 2. Navigate to **Environment -> Shared libraries**
- 3. Click **New**
- 4. Add the following entries into the fields:
 - a. Name: `extremescale_client_v7`
 - b. Classpath: `wxs_home/lib/wsogclient.jar`
- 5. Click **OK**
- 6. Save the changes to the master configuration

-
- E. Install the REST data service Web module, `wxsrestservice.war`, to the WebSphere Application Server using the WebSphere administration console:
1. Open the WebSphere administration console
 2. Navigate to **Applications -> New Application**
 3. Browse to `restservice_home/lib/wxsrestservice.war` file on the file system and select it and click **Next**.
 - a. If using WebSphere Application Server version 7.0, click **Next**.
 - b. If using WebSphere Application Server version 6.1, enter a Context Root value with the name: `/wxsrestservice` and continue to the next step.
 4. Choose the detailed installation option, and click **Next**.
 5. On the application security warnings screen, click **Continue**.
 6. Choose the default installation options, and click **Next**.
 7. Choose a server to map the application to, and click **Next**.
 8. On the JSP reloading page, use the defaults, and click **Next**.
 9. On the shared libraries page, map the "wxsrestservice_war" module to the following shared libraries defined in steps C and D:
 - `extremescale_rest_configuration`
 - `extremescale_client_v7`
Note: This shared library is required only if eXtreme Scale is not integrated with WebSphere Application Server)
 10. On the map shared library relationship page, use the defaults, and click **Next**.
 11. On the map virtual hosts page, use the defaults, and click **Next**.
 12. On the map context roots page, set the context root to: `/wxsrestservice`
 13. On the Summary screen, click **Finish** to complete the installation.
 13. Save the changes to the master configuration.
- F. Start the "wxsrestservice_war" eXtreme Scale REST data service application:
1. If using WebSphere Application Server version 7.0:
In the administration console, click on **Applications -> Application Types -> WebSphere Applications**.

If using WebSphere Application Server version 6.1:
In the administration console, click on **Applications -> Enterprise Applications**
 2. Check the check box next to the "wxsrestservice_war" application, and click **Start**.
 3. Review the SystemOut.log for the application server profile. When the REST data service has started successfully, the following message is displayed in the SystemOut.log for the server profile:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- G. Verify the REST data service is working:
1. Open a Web browser and navigate to the following URL:
`http://<host>:<port>/wxsrestservice/restservice/<Grid Name>`
-

The port number can be found in the SystemOut.log within the application server profile logs directory by looking at the first port displayed for message identifier: SRVE0250I. The default port is 9080.

For example:

<http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/>

Result: The AtomPub service document is displayed.

4.5 Deploying on WebSphere Application Server Community Edition

This topic describes how to configure the eXtreme Scale REST data service on WebSphere Application Server Community Edition Version 2.1.1.3 or later.

Prerequisites:

- An IBM (recommended) or Sun JRE or JDK, Version 5 or later is installed and the JAVA_HOME environment variable is set.
- [Download](#) and install WebSphere Application Server CE 2.1.1.3 or later to *wasce_root*. See [this link](#) for 2.1.1 installation instructions. Use [this link](#) for other versions.
- The eXtreme Scale Trial Version 7.0 with the REST data service is downloaded and extracted or the WebSphere eXtreme Scale 7.0.0.0 with cumulative fix 2 product is installed into a stand-alone directory.

Procedure:

- A. Configure and start an eXtreme Scale grid.
 1. For details on configuring an eXtreme Scale grid for use with the REST data service, see the Configuring WebSphere eXtreme Scale topic.
 2. Verify that an eXtreme Scale client can connect to and access entities in the grid. For an example, see the [Getting Started](#) section of this document.
- B. Build the eXtreme Scale REST service configuration JAR or directory. See the Packaging and deploying the REST data service topic for details.
- C. Add the ObjectGrid client runtime JAR to the WebSphere Application Server CE repository:
 1. Open the WebSphere Application Server CE administration console and log in. The default URL is: <http://localhost:8080/console> and the default userid is "system" and password is "manager".
 2. Click the "Repository" link on the left side of the console window, in the "Services" folder.
 3. In the "Add Archive to Repository" section, fill in the following into the input text boxes:

File:	<code>wxs_home/lib/ogclient.jar</code>
Group:	<code>com.ibm.websphere.xs</code>
Artifact:	<code>ogclient</code>
Version:	<code>7.0</code>
Type:	<code>jar</code>

-
- d. Click the Install button.

Note: See the following tech note for details on different ways class and library dependencies can be configured:
<http://www.ibm.com/support/docview.wss?uid=swg21266061>

- D. Deploy and start the REST data service module: `wxsrestservice.war` to the WebSphere Application Server CE server.

1. Copy and edit the sample deployment plan XML file: `restservice_home/gettingstarted/wasce/geronimo-web.xml` to include path dependencies to your REST data service configuration JAR or directory. See section 3.3.3 for an example on setting the classpath to include your `wxsRestService.properties` file and other configuration files and metadata classes.
2. Open the WebSphere Application Server CE administration console and log in.

Note: The default URL is: <http://localhost:8080/console> and the default userid is "system" and password is "manager".

3. Click on the "Deploy New" link on the left side of the console window.
4. On the "Install New Applications" page, enter the following values into the text boxes:
 - Archive: `restservice_home/lib/wxsrestservice.war`
 - Plan: `restservice_home/gettingstarted/wasce/geronimo-web.xml`

Note: Use the path to the `geronimo-web.xml` that you copied and edited in step C.1.

5. Click on the Install button.
The console page should indicate that the application was successfully installed and started.
6. Examine the WebSphere Application Server CE system output log or console to verify that the REST data service has started successfully by verify that the following message is present:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.

- E. Start the WebSphere Application Server CE server by running the following command:

```
Windows  
wasce_root/bin/startup.bat  
UNIX  
wasce_root/bin/startup.sh
```

- F. Install the eXtreme Scale REST data service and the provided sample into the WebSphere Application Server CE server:

1. Add the ObjectGrid client runtime JAR to the WebSphere Application Server CE repository:
 - a. Open the WebSphere Application Server CE administration console and log in. (The default settings are `http://localhost:8080/console/` with userid of system and password of manager.)

-
- b. Click the "Repository" link on the left side of the console window, in the Services folder.
 - c. In the "Add Archive to Repository" section, fill in the following into the input text boxes:
 - File: `wxs_home/lib/ogclient.jar`
 - Group: `com.ibm.websphere.xs`
 - Artifact: `ogclient`
 - Version: `7.0`
 - Type: `jar`
 - d. Click the Install button.

Note: See the following tech note for details on different ways class and library dependencies can be configured:
<http://www.ibm.com/support/docview.wss?uid=swg21266061>

2. Deploy the REST data service module: `wxsrestservice.war` to the WebSphere Application Server CE server.
 - a. Edit the sample `restservice_home/gettingstarted/wasce/geronimo-web.xml` deployment XML file to include path dependencies to the getting started sample classpath directories:
 - Change the "classesDirs" for the two getting started client GBeans:

The "classesDirs" path for the GettingStarted_Client_SharedLib GBean should be set to:
`restservice_home/gettingstarted/restclient/bin`

The "classesDirs" path for the GettingStarted_Common_SharedLib GBean should be set to:
`restservice_home/gettingstarted/common/bin`
 - b. Open the WebSphere Application Server CE administration console and log in.
 - c. Click on the "Deploy New" link on the left side of the console window.
 - d. On the "Install New Applications" page, enter the following values into the text boxes:
 - Archive: `restservice_home/lib/wxsrestservice.war`
 - Plan: `restservice_home/gettingstarted/wasce/geronimo-web.xml`
 - e. Click on the Install button.
The console page should indicate that the application was successfully installed and started.
 - f. Examine the WebSphere Application Server CE system output log to verify that the REST data service has started successfully by verifying that the following message is present:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.

G. Verify that the REST data service is working:

1. Open a Web browser and navigate to the following URL:
`http://<host>:<port>/<context root>/restservice/<Grid Name>`

The default port for WebSphere Application Server CE is 8080 and is defined

using the "HTTPPort" property in the `wasce_root/var/config/config-substitutions.properties` file.

For example:

<http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/>

Result: The AtomPub service document is displayed.

4.6 Deploying on Apache Tomcat

This topic describes how to configure the eXtreme Scale REST data service on Apache Tomcat Version 5.5 or later.

Prerequisites:

- An IBM or Sun JRE or JDK, Version 5 or later is installed and the JAVA_HOME environment variable is set.
- Apache Tomcat Version 5.5 or later is installed. See <http://tomcat.apache.org> for details on how to install Tomcat.
- The eXtreme Scale Trial Version 7.0 with the REST data service is downloaded and extracted or the WebSphere eXtreme Scale 7.0.0.0 with cumulative fix 2 product is installed into a stand-alone directory.

Procedure:

- A. If using a Sun JRE or JDK, install the IBM ORB into Tomcat:

If using Tomcat version 5.5:

1. Copy all of the JAR files from:
`wxs_home/lib/endorsed`
to:
`tomcat_root/common/endorsed`

If using Tomcat version 6.0:

1. Create an "endorsed" directory:

```
Windows  
md tomcat_root/endorsed  
UNIX  
mkdir tomcat_root/endorsed
```
2. Copy all of the JAR files from:
`wxs_home/lib/endorsed`
to:
`tomcat_root/endorsed`

- B. Configure and start an eXtreme Scale grid.

1. For details on configuring an eXtreme Scale grid for use with the REST data service, see the Configuring WebSphere eXtreme Scale topic.
2. Verify that an eXtreme Scale client can connect to and access entities in the grid. For an example, see the [Getting Started](#) section of this document.

-
- C. Build the eXtreme Scale REST service configuration JAR or directory. See the Packaging and deploying the REST data service topic for details.
 - D. Deploy the REST data service module: `wxsrestservice.war` to the Tomcat server.
 1. Copy the `wxsrestservice.war` file from:
`restservice_home/lib`
to:
`tomcat_root/webapps`
 - E. Add the ObjectGrid client runtime JAR and the application JAR to the shared classpath in Tomcat:
 1. Edit the `tomcat_root/conf/catalina.properties` file
 2. Append the following path names to the end of the `shared.loader` property separating each with a comma:
 - `wxs_home/lib/ogclient.jar`
 - `restservice_home/gettingstarted/restclient/bin`
 - `restservice_home/gettingstarted/common/bin`
 - F. Start the Tomcat server:

If using Tomcat 5.5 on UNIX or Windows, or the Tomcat 6.0 ZIP distribution:

1. `cd tomcat_root/bin`
2. Start the server:
 - Windows**
`catalina.bat run`
 - UNIX**
`./catalina.sh run`
3. The Apache Tomcat logs are displayed to the console. When the REST data service has started successfully, the following message is displayed in the administration console:
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.

If using Tomcat 6.0 on Windows using the Windows installer distribution:

1. `cd tomcat_root/bin`
2. Start the Apache Tomcat 6 configuration tool:
 - Windows**
`tomcat6w.exe`
3. Click on the Start button on the Apache Tomcat 6 properties window to start the Tomcat server.
4. Review the following logs to verify that the Tomcat server has started successfully:
 - `tomcat_root/bin/catalina.log`
Displays the status of the Tomcat server engine
 - `tomcat_root/bin/stdout.log`
Displays the system output log.
5. When the REST data service has started successfully, the following message is displayed in the system output log:

CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.

G. Verify the REST data service is working:

1. Open a Web browser and navigate to the following URL:
`http://<host>:<port>/<context root>/restservice/<Grid Name>`

The default port for Tomcat is 8080 and is configured in the `tomcat_root/conf/server.xml` file in the `<Connector>` element.

For example:

<http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/>

Result: The AtomPub service document is displayed.

5 Configuring the REST data service

The REST data service requires two configuration steps:

- Step 1) Configuring the REST data service properties file
- Step 2) Configuring WebSphere eXtreme Scale

5.1 Configuring the REST data service properties file

The REST data service properties file is the main configuration file for the eXtreme Scale REST data service. This file is a typical [Java property file](#) with key and value pairs. By default, the REST data service runtime will look for a well-named `wxsRestService.properties` file in the classpath. The file can also be explicitly defined by using the system property: `wxs.restservice.props`.

Example `-Dwxs.restservice.props=/usr/configs/dataservice.properties`

When the REST data service is loaded, the property file used is displayed in the log files:
CWOBJ4004I: The eXtreme Scale REST data service properties files were loaded:
[/usr/configs/RestService.properties]

The REST data service properties file supports the following properties:

Property	Description
catalogServiceEndpoints	The required comma-delimited list of hosts and ports of a catalog service grid in the format: <code><host:port></code> . This is optional if using WebSphere Application Server integrated with eXtreme Scale to host the REST data service. See the WebSphere eXtreme Scale product documentation for details on how to configure and start a catalog service . <u>Example</u> <code>catalogServiceEndpoints=server1:2809,server2:2809</code>
objectGridNames	The required names of the ObjectGrids to expose to the REST service. At least one ObjectGrid name is required. Separate multiple ObjectGrid names using a comma:

	<u>Example</u> ECommerceGrid,InventoryGrid
objectGridClientXML	The optional name of the ObjectGrid client override XML file. The name specified here will be loaded from the classpath. The default is: /META-INF/objectGridClient.xml. See the WebSphere eXtreme Scale product documentation for details on how to configure an eXtreme Scale client .
traceFile	The optional name of the file to redirect the trace output to. The default is logs/trace.log.
traceSpec	The optional trace specification that the eXtreme Scale runtime server should initially use. The default is *=all=disabled. To trace the entire REST data service, use: ObjectGridRest*=all=enabled
verboseOutput	If set to true, REST data service clients will receive additional diagnostic information when failures occur. The default is false. This optional value should be set to false for production environments as sensitive information may be revealed.

5.2 Configuring WebSphere eXtreme Scale

The eXtreme Scale REST data service interacts with eXtreme Scale using the EntityManager API. An entity schema is defined for an eXtreme Scale grid and the meta-data for the entities is automatically consumed by the REST data service. For details on how to configure an entity schema, see the [eXtreme Scale EntityManager documentation](#).

For example, you can define an entity representing a Person in an eXtreme Scale grid as follows:

```
@Entity
public class Person {
    @Id String taxId;
    String firstName;
    String lastName;
}
```

Note: The annotations used here are in the com.ibm.websphere.projector.annotations package.

The REST service will automatically create an ADO.NET Entity Data Model for Data Services (EDMX) document, which is available using the \$metadata URI:

Example [http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/\\$metadata](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata)

Once the eXtreme Scale grid is configured and running, an eXtreme Scale client should be configured and packaged. For details on configuring the eXtreme Scale REST data service client package, see topic: Packaging and deploying the REST data service.

The following topics describe how eXtreme Scale is exposed as a REST service.

5.2.1 Creating an entity model

WebSphere eXtreme Scale entities are modeled using the entity annotations or an entity metadata descriptor file. See the topic on [defining an entity schema](#) in the *Programming*

Guide of the WebSphere eXtreme Scale V7.0 documentation for details on how to configure an eXtreme Scale entity schema. The eXtreme Scale REST service uses the entity metadata to automatically create an EDMX model for the data service.

This version of the WebSphere eXtreme Scale REST data service has the following schema restrictions:

- When defining entities in a partitioned grid, all entities must have a direct or indirect single valued association to the root entity (a key association). The WCF data service client runtime must be able to access every entity directly through its canonical address. Therefore, the key of the root entity that is used for partition routing (the schema root) must be part of the key in the child entity.

For example:

```
@Entity(schemaRoot=true)
public class Person {
    @Id String taxId;
    String firstName;
    String lastName;
    @OneToMany (mappedBy="person")
    List<Address> addresses;
}

@Entity
public class Address {
    @Id int addrId;
    @Id @ManyToOne Person person;
    String street;
}
```

- Bi-directional and uni-directional associations are supported. However, uni-directional associations may not always work from a Microsoft WCF Data Services client since they can only be navigated in one direction and the Microsoft specification requires all associations to be bi-directional.
- Referential constraints are not supported. The eXtreme Scale runtime does not validate keys between entities. Associations between entities must be managed by the client.
- Complex types are not supported. The eXtreme Scale EntityManager API does not support embeddable attributes. All attributes are expected to be simple type attributes (see the simple attribute types listed below). Non-simple type attributes are treated as a binary object from the perspective of the client.
- Entity inheritance is not supported. The eXtreme Scale EntityManager API does not support inheritance.
- Media Resources and Media Links are not supported. The HasStream attribute of the EntityType in the Conceptual Schema Definition Language Document for Data Services is never used.

Mapping between EDM data types and Java data types

The OData protocol defines the following list of Entity Data Model (EDM) types in its abstract type system. The following topics describe how the eXtreme Scale REST adapter chooses the EDM type based on the basic type defined in the entity. For details on EDM types, see: [http://msdn.microsoft.com/en-us/library/dd541295\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541295(PROT.10).aspx).

The following EDM types are available in WCF Data Services:

- Edm.Binary
- Edm.Boolean
- Edm.Byte
- Edm.DateTime
- Edm.Time
- Edm.Decimal
- Edm.Double
- Edm.Single
- Edm.Float
- Edm.Guid *
- Edm.Int16
- Edm.Int32
- Edm.Int64
- Edm.SByte
- Edm.String

* The EDM type: Edm.Guid is not supported by the eXtreme Scale REST data service.

Mapping Java Types to EDM Types

The eXtreme Scale REST data service will automatically convert basic entity types into EDM types. The type mapping can be seen by displaying the Entity Data Model Extensions (EDMX) metadata document using the \$metadata URI. The EDM type is what is used by clients to read and write data to the REST data service.

Table 1 shows the mapping from the Java type defined for an entity to the EDM data type. When retrieving data using a query, the data will be represented with these types:

Table 1 Java Type to EDM Type Mapping for Retrieve Requests

Java Type	EDM Type
boolean	Edm.Boolean
java.lang.Boolean	

byte java.lang.Byte	Edm.SByte
short java.lang.Short	Edm.Int16
int java.lang.Integer	Edm.Int32
long java.lang.Long	Edm.Int64
float java.lang.Float	Edm.Single
double java.lang.Double	Edm.Double
java.math.BigDecimal java.math.BigInteger	Edm.Decimal
java.lang.String char java.lang.Character Char[] java.lang.Character[]	Edm.String
java.util.Calendar java.util.Date java.sql.Date java.sql.Timestamp java.sql.Time	Edm.DateTime
Other types	Edm.Binary

Mapping from EDM types to Java Types

For Update requests and Insert requests, the payload specifies the data to be updated or inserted into the eXtreme Scale REST data service. The service can automatically convert compatible data types to the data types defined in the EDMX document. The REST data service converts the XML encoded string representations of the value into the correct type using the following two-step process:

- Step 1) a type check is performed to make sure the EDM type is compatible with the Java type. An EDM type is compatible with a Java type if the data supported by the EDM type is a subset of the data supported by the Java type. For example, Edm.int32 type is compatible with a Java long type, but Edm.int32 type is not compatible with a Java short type.
- Step 2) a target Java type object will be created which represents the string value in the payload.

Table 2 describes which EDM types and their compatibility with Java types.

Table 2 Compatible EDM type to Java type

EDM type	Java type
Edm.Boolean	boolean java.lang.Boolean
Edm.SByte	byte java.lang.Byte short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character
Edm.Byte Edm.Int16	short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character
Edm.Int32	int java.lang.Integer long java.lang.Long

	float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Int64	long java.lang.Long double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Double	double java.lang.Double java.math.BigDecimal
Edm.Decimal	double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Single	float java.lang.Float double java.lang.Double java.math.BigDecimal
Edm.String	java.lang.String char java.lang.Character Char[] java.lang.Character[] java.math.BigDecimal java.math.BigInteger
Edm.DateTime	java.util.Calendar java.util.Date java.sql.Date java.sql.Time java.sql.Timestamp
Edm.Time	java.sql.Time java.sql.Timestamp

Mapping temporal types

Java includes five temporal types for storing date, time or both: `java.util.Date`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp` and `java.util.Calendar`. All of these types are expressed in the entity data model as `Edm.DateTime`. The eXtreme Scale REST service automatically converts and normalizes the data depending on the Java type. This topic describes several issues that developers must be aware of when using any temporal type.

Time zone differences

In WCF Data Services, the descriptions of time values in the `Edm.DateTime` type are always expressed using the Coordinated Universal Time (UTC) standard, which is the internationally recognized name for Greenwich Mean Time (GMT). Coordinated Universal Time is the time as measured at zero degrees longitude, the UTC origin point. Daylight saving time is not applicable to UTC.

Converting between entity and EDM types

When a client sends a request to the REST data service, the date and time is represented as a GMT time zone time.

Example "2000-02-29T21:30:30.654123456"

The REST data service will then construct the appropriate Java temporal type instance and insert it into the entity in the grid.

When a client requests a property which is a Java temporal type from the eXtreme Scale REST data service, the value is always normalized as a GMT time zone value. For example, if an entity `java.util.Date` is constructed as follows:

```
Calendar c = Calendar.getInstance();
c.clear();
c.set(2000, 1, 29, 21, 30, 30);
Date d = c.getTime();
```

The date and time are represented using the default time zone of the Java process because `Calendar.getInstance()` will create a `Calendar` object with local time zone. If the local time zone is CST, then the date, when retrieved from the REST data service will be the GMT representation of the time:

```
"2000-03-01T03:30:30"
```

java.sql.Date normalization

An eXtreme Scale entity can define an attribute with Java type `java.sql.Date`. This data type does not include the time and is normalized by the REST data service. This means that the eXtreme Scale runtime does not store any hours, minutes, seconds, or milliseconds information in the `java.sql.Date` attribute. Regardless of the time zone offset, the date is always represented as a local date.

For example, if the client updates a `java.sql.Date` property with the value "2009-01-01T03:00:00", the REST data service, which is in the CST time zone (-06:00), will simply create a `java.sql.Date` instance of which the time is set to "2009-01-01T00:00:00" of the local CST time. There is no time zone conversion done to create the `java.sql.Date` value. When the REST service client retrieves the value of this attribute, it will be displayed as

"2009-01-01T00:00:00Z". If a time zone conversion were done, the value would be displayed as having the date of "2008-12-31", which would be incorrect.

java.sql.Time normalization

Similar to `java.sql.Date`, the `java.sql.Time` values are normalized and do not include date information. This means that the eXtreme Scale run time does not store the year, month or day. The time is stored using the GMT time from the epoch January 1, 1970, which is consistent with the `java.sql.Time` implementation.

For example, if the client updates a `java.sql.Time` property with the value "2009-01-01T03:00:00", the REST data service, will create a `java.sql.Time` instance with the milliseconds set to $3*60*60*1000$, which is equal to 3 hours. When the rest service retrieves the value, it will be displayed as "1970-01-01:03:00:00Z".

Associations

Associations define the relationship between two peer entities. The eXtreme Scale REST service reflects the associations modeled with entities defined with eXtreme Scale annotated entities or entities defined using an entity descriptor XML file.

Association maintenance

The eXtreme Scale REST data service does not support referential integrity constraints. The client should ensure that references are updated when entities are removed or added. If a target entity of an association is removed from the grid, but the link between the source and target entity is not removed, then the link is broken. The eXtreme Scale REST data service and EntityManager API tolerates broken links and will log them as CWPRJ1022W warnings. Broken associations will simply be removed from the request payload.

Use a batch request to group association updates in a single transaction to avoid broken links. See section 6.2.6 for details on batch requests.

The ADO.NET Entity Data Model `ReferentialConstraint` element is not used by the eXtreme Scale REST data service.

Association multiplicity

Entities can have multi-valued associations or single-valued associations. Multi-valued associations, or collections, are one-to-many or many-to-many associations. Single-valued associations are one-to-one or many-to-one associations.

In a partitioned grid, all entities should have a single-valued key-association path to a root entity. Because the root entity is used to partition the entity, many-to-many associations are not allowed for partitioned grids. For an example on how to model a relational entity schema for a partitioned grid, see section 3.1.1. See the following section: Key for details on how to define a key association.

The following example describes how the EntityManager API association types, modeled using annotated Java classes map to the ADO.NET Entity Data Model:

```
@Entity
public class Customer {
    @Id String customerId;
```

```

    @OneToOne TaxInfo taxInfo;
    @ManyToOne Address homeAddress;
    @OneToMany Collection<Order> orders;
    @ManyToMany Collection<SalesPerson> salespersons;
}

<Association Name="Customer_TaxInfo">
  <End Type="Modell.Customer" Role="Customer" Multiplicity="1" />
  <End Type="Modell.TaxInfo" Role="TaxInfo" Multiplicity="1" />
</Association>

<Association Name="Customer_Address">
  <End Type="Modell.Customer" Role="Customer" Multiplicity="1" />
  <End Type="Modell.Address" Role="TaxInfo" Multiplicity="*" />
</Association>

<Association Name="Customer_Order">
  <End Type="Modell.Customer" Role="Customer" Multiplicity="*" />
  <End Type="Modell.Order" Role="TaxInfo" Multiplicity="1" />
</Association>

<Association Name="Customer_SalesPerson">
  <End Type="Modell.Customer" Role="Customer" Multiplicity="*" />
  <End Type="Modell.SalesPerson" Role="TaxInfo" Multiplicity="*" />
/>
</Association>

```

Bi-directional and uni-directional associations

Entities associations can be uni-directional or bi-directional. By specifying the "mappedBy" attribute on the @OneToOne, @OneToMany or @ManyToMany annotation or the "mapped-by" attribute on the one-to-one, one-to-many or many-to-many XML attribute tag, the entity becomes bi-directional. The OData protocol currently requires all entities to be bi-directional, allowing clients to generate navigation paths in both directions. The eXtreme Scale EntityManager API allows modeling uni-directional associations which can save memory and simplify maintenance of the associations. If a uni-directional association is used, the REST data services client must only navigate through the association using the defined association.

For example: If a uni-directional many-to-one association is defined between Address and Country, the following URI is not allowed:

```
/restservice/CustomerGrid/Country('USA')/addresses
```

Key associations

Single-valued associations (one-to-one and many-to-one) can also be included as all or part of the entities key. This is known as a key-association.

Key associations are required when using a partitioned grid. The key association must be defined for all child entities in a partitioned entity schema. The OData protocol requires that all entities are directly addressable. This means that the key in the child entity must include the key used for partitioning.

In the following example, Customer has a one-to-many association to Order. The Customer entity is the root entity and the customerId attribute is used to partition the entity. Order has included the Customer as part of its identity:

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer") Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

When the REST data service generates the EDMX document for this model, the Customer key fields are automatically included as part of the Order entity:

```
<EntityType Name="Order">
  <Key>
    <PropertyRef Name="orderId"/>
    <PropertyRef Name="customer_customerId"/>
  </Key>

  <Property Name="orderId" Type="Edm.Int64" Nullable="false"/>
  <Property Name="customer_customerId" Type="Edm.String"
    Nullable="false"/>
  <Property Name="orderDate" Type="Edm.DateTime" Nullable="true"/>

  <NavigationProperty Name="customer"
    Relationship="NorthwindGridModel.Customer_orders"
    FromRole="Order" ToRole="Customer"/>

  <NavigationProperty Name="orderDetails"
    Relationship="NorthwindGridModel.Order_orderDetails"
    FromRole="Order" ToRole="OrderDetail"/>
</EntityType>
```

When an entity is created, the key must never change. This means if the key association between a child entity and its parent must change, the child entity must be removed and re-created with a different parent. In a partitioned grid, this will require two different batch change sets since the move will likely involve more than one partition.

Cascading operations

The EntityManager API allows a flexible cascade policy. Associations can be marked to cascade a persist, remove, invalidate or merge operation. Such cascade operations can happen on one or both sides of a bi-directional association.

The OData protocol only allows cascade delete operations on the single-side of the association. The CascadeType.REMOVE annotation or cascade-remove XML attribute cannot be defined on both sides of a one-to-one bi-directional association or on the many-

side of a one-to-many association. The following example illustrates a valid `Cascade.REMOVE` bi-directional association:

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer", cascade=CascadeType.REMOVE)
    Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

The resulting EDMX association looks as follows:

```
<Association Name="Customer_orders">
  <End Type="NorthwindGridModel.Customer" Role="Customer"
    Multiplicity="1">
    <OnDelete Action="Cascade"/>
  </End>
  <End Type="NorthwindGridModel.Order" Role="Order"
    Multiplicity="*" />
</Association>
```

6 Using the REST data service

After you start the eXtreme Scale REST data service, you can use any HTTP client to interact with it. A Web browser, PHP client, Java client or WCF Data Services client can be used to issue any of the supported request operations.

The REST service implements a subset of the [Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions](#) specification, Version 1.0 which is part of [OData protocol](#). This chapter describes which of the features of the specification are supported and how they are mapped to eXtreme Scale.

6.1 Service root URI

Microsoft WCF Data Services typically defines a service per data source or entity model. The eXtreme Scale REST data service defines a service per defined ObjectGrid. Each ObjectGrid that is defined in the eXtreme Scale ObjectGrid client override XML file is automatically exposed as a separate REST service root.

The URI for the service root is:

```
http://<host>:<port>/<contextroot>/restservice/<gridname>
```

Where *contextroot* is defined when the REST data service application is deployed, and is dependent on the application server, and *gridname* is the name of the ObjectGrid.

6.2 Request types

The following list describes the Microsoft WCF Data Services request types which the eXtreme Scale REST data service supports.

For details on each request type that WCF Data Services supports, see:
<http://msdn.microsoft.com/en-us/library/dd541602%28PROT.10%29.aspx>

6.2.1 Insert request types

Clients can insert resources using the POST HTTP verb with the following limitations:

- InsertEntity Request: Supported.
- InsertLink request: Supported.
- InsertMediaResource request: Not supported due to media resource support restriction.

For additional information, see:
[http://msdn.microsoft.com/en-us/library/dd541376\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541376(PROT.10).aspx)

6.2.2 Update request types

Clients can update resources using the PUT and MERGE HTTP verbs with the following limitations:

- UpdateEntity Request: Supported.
- UpdateComplexType Request: Not Supported due to complex type restriction.
- UpdatePrimitiveProperty Request: Supported.
- UpdateValue Request: Supported.
- UpdateLink Request: Supported.
- UpdateMediaResource Request: Not supported due to media resource support restriction.

For additional information, see:
[http://msdn.microsoft.com/en-us/library/dd541376\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541376(PROT.10).aspx)

6.2.3 Delete request types

Clients can delete resources using the DELETE HTTP verb with the following limitations:

- DeleteEntity Request: Supported.
- DeleteLink Request: Supported.
- DeleteValue request: Supported.

For additional information, see:
[http://msdn.microsoft.com/en-us/library/dd541534\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541534(PROT.10).aspx)

6.2.4 Retrieve request types

Clients can retrieve resources using the GET HTTP verb with the following limitations:

- RetrieveEntitySet Request: Supported.
- RetrieveEntity Request: Supported.
- RetrieveComplexType Request: Not supported due to complex type restriction.
- RetrievePrimitiveProperty Request: Supported.
- RetrieveValue Request: Supported.
- RetrieveServiceMetadata Request: Supported.
- RetrieveServiceDocument Request: Supported.
- RetrieveLink Request: Supported.
- Retrieve Request Containing a Customizable Feed Mapping: Not supported
- RetrieveMediaResource: Not supported due to media resource restriction.

For additional information, see:

[http://msdn.microsoft.com/en-us/library/dd541450\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541450(PROT.10).aspx)

System query options

Queries are supported which allow clients to identify a collection of entities or a single entity. System query options are specified in a data service URI and are supported with the following limitations:

- \$expand: Supported
- \$filter: Supported.
- \$orderby: Supported.
- \$format: Not supported. The acceptable format is identified in the HTTP Accept request header.
- \$skip: Supported
- \$top: Supported

For additional information, see:

[http://msdn.microsoft.com/en-us/library/dd541320\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541320(PROT.10).aspx)

Partition routing

Partition routing is based on the root entity. A request URI infers a root entity if its resource path starts with a root entity or with an entity that has a direct or indirect association to the entity.

In a partitioned environment, any request that cannot infer a root entity will be rejected. Any request that infers a root entity will be routed to the correct partition.

For additional information on defining a schema with associations and root entities, see section 3.1.1 in this document and the [partitioning topic](#) in the *Product Overview* of the WebSphere eXtreme Scale documentation.

6.2.5 Invoke request

Invoke requests are not supported.

For additional information, see:

[http://msdn.microsoft.com/en-us/library/dd541482\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541482(PROT.10).aspx)

6.2.6 Batch request

Clients can batch multiple Change Sets or Query Operations within a single request. This can reduce the number of round trips to the server and allows multiple requests to participate in a single transaction.

For additional information, see:

[http://msdn.microsoft.com/en-us/library/dd541539\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541539(PROT.10).aspx)

6.2.7 Tunneled requests

Tunneled requests are not supported.

For additional information, see:

[http://msdn.microsoft.com/en-us/library/dd541243\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541243(PROT.10).aspx)

6.3 Request Protocols and Examples

This topic describes the request protocols when interacting with the REST service. In general, the protocols are the same as those described in the WCF Data Services AtomPub protocol directly. However, we do provide additional details from eXtreme Scale Entity Model perspective. Users are expected to be familiar with the WCF Data Services protocols before reading this section. Alternatively, users can read this section with the WCF Data Services protocol section.

Examples are provided to illustrate the request and response. These examples apply to both the eXtreme Scale REST data service and WCF Data Services.

Since Web browsers can only retrieve data, the CUD (create, update and delete) operations must be performed by another client such as Java, JavaScript, RUBY or PHP.

6.3.1 Retrieve requests

6.3.1.1 Retrieving an entity

A RetrieveEntity Request is used by a client to retrieve an eXtreme Scale entity. The response payload contains the entity data in AtomPub or JSON format.

Also, the system operator \$expand can be used to expand the relations. The relations are represented in line within the data service's response as an Atom Feed Document (to-many relation) or an Atom Entry Document (to-one relation).

For more details on the RetrieveEntity protocol defined in WCF Data Services, refer to [http://msdn.microsoft.com/en-us/library/dd541268\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541268(PROT.10).aspx)

The following RetrieveEntity example retrieves a Customer entity with key.

AtomPub

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

Request Header:

Accept: application/atom+xml

Request Payload:

None

Response Header:

Content-Type: application/atom+xml

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base =
"http://localhost:8080/wxsrestservice/restservice" 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">
  <category term = "NorthwindGridModel.Customer" scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('ACME')</id>
  <title type = "text"/>
  <updated>2009-12-16T19:52:10.593Z</updated>
  <author>
    <name/>
  </author>
  <link rel = "edit" title = "Customer" href =
"Customer('ACME')"/>
  <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/orde
rs" type = "application/atom+xml;type=feed" title = "orders" href =
"Customer('ACME')/orders"/>
  <content type = "application/xml">
    <m:properties>
      <d:customerId>ACME</d:customerId>
```

```
        <d:city m:null = "true"/>
        <d:companyName>RoaderRunner</d:companyName>
        <d:contactName>ACME</d:contactName>
        <d:country m:null = "true"/>
        <d:version m:type = "Edm.Int32">3</d:version>
    </m:properties>
</content>
</entry>
```

Response Code:

200 OK

JSON

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{ "d": { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')",
  "type": "NorthwindGridModel.Customer",
  "customerId": "ACME",
  "city": null,
  "companyName": "RoaderRunner",
  "contactName": "ACME",
  "country": null,
  "version": 3,
  "orders": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders" } } } }
```

Response Code:

200 OK

6.3.1.1.1 Queries

A query can also be used with a RetrieveEntitySet or RetrieveEntity request. A query is specified by the system \$filter operator.

For details on the \$filter operator, refer to:
[http://msdn.microsoft.com/en-us/library/dd541344\(prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541344(prot.10).aspx)

The OData protocol supports several common expressions. The eXtreme Scale REST data service supports a subset of the expressions defined in the specification:

- Boolean expressions:
 - eq, ne, lt, le, gt, ge
 - negate
 - not
 - parenthesis
 - and, or
- Arithmetic expressions
 - add, sub, mul, div
- Primitive literals
 - String, date-time, decimal, single, double, int16, int32, int64, binary, null and byte.

The following expressions are NOT available:

- Boolean expressions:
 - isof
 - cast
- Method call expressions
- Arithmetic expressions
 - mod
- Primitive literals:
 - Guid
- Member expressions

For a complete list and description of the expressions that are available in Microsoft WCF Data Services, see section 2.2.3.6.1.1:
<http://msdn.microsoft.com/en-us/library/dd541448%28prot.10%29.aspx>

The following example demonstrates a RetrieveEntity request with a query. In this example, all customers whose contact name is "RoadRunner" are retrieved. The only customer which matches this filter is Customer('ACME') as shown in the response payload.

Note: This query will only work for non-partitioned entities. If Customer is partitioned, then the customer's key is required.

AtomPub

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?\$filter=contactName eq 'RoadRunner'

Request Header:

Accept: application/atom+xml

Input Payload:

None

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
  xml:base="http://localhost:8080/wxsrestservice/restservice"
  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">Customer</title>
  <id>
    http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
  </id>
  <updated>2009-09-16T04:59:28.656Z</updated>
  <link rel="self" title="Customer" href="Customer" />
  <entry>
    <category term="NorthwindGridModel.Customer"
      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
    />
    <id>
      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
    </id>
    <title type="text" />
    <updated>2009-09-16T04:59:28.656Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Customer" href="Customer('ACME')" />
    <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/orders"
    />
  </entry>
</feed>
```

```
    type="application/atom+xml;type=feed" title="orders"
    href="Customer('ACME')/orders" />
<content type="application/xml">
  <m:properties>
    <d:customerId>ACME</d:customerId>
    <d:city m:null = "true"/>
    <d:companyName>RoaderRunner</d:companyName>
    <d:contactName>ACME</d:contactName>
    <d:country m:null = "true"/>
    <d:version m:type = "Edm.Int32">3</d:version>
  </m:properties>
</content>
</entry>
</feed>
```

Response Code:

200 OK

JSON

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?\$filter=contactName eq 'RoadRunner'

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{ "d": [ { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')",
  "type": "NorthwindGridModel.Customer",
  "customerId": "ACME",
  "city": null,
  "companyName": "RoaderRunner",
  "contactName": "ACME",
  "country": null,
  "version": 3,
  "orders": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders" } } } ] }
```

Response Code:

200 OK

6.3.1.1.2 System operator \$expand

The system operator \$expand can be used to expand associations. The associations are represented in line in the data service response. Multi-valued (to-many) associations are represented as an Atom Feed Document or JSON array. Single-valued (to-one) associations, are represented as n Atom Entry Document or JSON object.

For more details on the \$expand system operator, refer to:
[http://msdn.microsoft.com/en-us/library/dd541606\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541606(Prot.10).aspx)

Here is an example of using the \$expand system operator. In this example, we retrieve the entity Customer('IBM') which has an Orders 5000, 5001 and others associated with it. The \$expand clause is set to "orders", so the order collection is expand as inline in the response payload. Only orders 5000 and 5001 are displayed here.

AtomPub

Method:

GET

Request URI:

`http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders`

Request Header:

Accept: application/atom+xml

Request Payload:

None

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base =
"http://localhost:8080/wxsrestservice/restservice" 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">
  <category term = "NorthwindGridModel.Customer" scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

  <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:50:18.156Z</updated>
```

```

    <author>
      <name/>
    </author>
    <link rel = "edit" title = "Customer" href =
"Customer('IBM')"/>
    <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/order
s" type = "application/atom+xml;type=feed" title = "orders" href =
"Customer('IBM')/orders">
      <m:inline>
        <feed>
          <title type = "text">orders</title>

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('IBM')/orders</id>
      <updated>2009-12-16T22:50:18.156Z</updated>
      <link rel = "self" title = "orders" href =
"Customer('IBM')/orders"/>
    <entry>
      <category term = "NorthwindGridModel.Order"
scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Order(orderId=5000,customer_customerId='IBM')</id>
      <title type = "text"/>
      <updated>2009-12-16T22:50:18.156Z</updated>
      <author>
        <name/>
      </author>
      <link rel = "edit" title = "Order" href =
"Order(orderId=5000,customer_customerId='IBM')"/>
      <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/cust
omer" type = "application/atom+xml;type=entry" title = "customer"
href = "Order(orderId=5000,customer_customerId='IBM')/customer"/>
      <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/order
Details" type = "application/atom+xml;type=feed" title =
"orderDetails" href =
"Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
      <content type = "application/xml">
        <m:properties>
          <d:orderId m:type =
"Edm.Int32">5000</d:orderId>

<d:customer_customerId>IBM</d:customer_customerId>
          <d:orderDate m:type =
"Edm.DateTime">2009-12-16T19:46:29.562</d:orderDate>
          <d:shipCity>Rochester</d:shipCity>
          <d:shipCountry m:null = "true"/>
          <d:version m:type =
"Edm.Int32">0</d:version>
        </m:properties>
      </content>
    </entry>
  </entry>

```

```

        <category term = "NorthwindGridModel.Order"
scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Order(orderId=5001,customer_customerId='IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:50:18.156Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Order" href =
"Order(orderId=5001,customer_customerId='IBM')"/>
    <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/cust
omer" type = "application/atom+xml;type=entry" title = "customer"
href = "Order(orderId=5001,customer_customerId='IBM')/customer"/>
    <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/orde
rDetails" type = "application/atom+xml;type=feed" title =
"orderDetails" href =
"Order(orderId=5001,customer_customerId='IBM')/orderDetails"/>
    <content type = "application/xml">
        <m:properties>
            <d:orderId m:type =
"Edm.Int32">5001</d:orderId>
<d:customer_customerId>IBM</d:customer_customerId>
            <d:orderDate m:type =
"Edm.DateTime">2009-12-16T19:50:11.125</d:orderDate>
            <d:shipCity>Rochester</d:shipCity>
            <d:shipCountry m:null = "true"/>
            <d:version m:type =
"Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
</feed>
</m:inline>
</link>
<content type = "application/xml">
    <m:properties>
        <d:customerId>IBM</d:customerId>
        <d:city m:null = "true"/>
        <d:companyName>IBM Corporation</d:companyName>
        <d:contactName>John Doe</d:contactName>
        <d:country m:null = "true"/>
        <d:version m:type = "Edm.Int32">4</d:version>
    </m:properties>
</content>
</entry>

```

Response Code:

200 OK

JSON

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?\$expand=orders

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{
  "d": {
    "__metadata": {
      "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')",
      "type": "NorthwindGridModel.Customer",
      "customerId": "IBM",
      "city": null,
      "companyName": "IBM Corporation",
      "contactName": "John Doe",
      "country": null,
      "version": 4,
      "orders": [
        {
          "__metadata": {
            "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')",
            "type": "NorthwindGridModel.Order",
            "orderId": 5000,
            "customer_customerId": "IBM",
            "orderDate": "\\Date(1260992789562)\\",
            "shipCity": "Rochester",
            "shipCountry": null,
            "version": 0,
            "customer": {
              "__deferred": {
                "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/customer"
              }
            },
            "orderDetails": {
              "__deferred": {
                "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/orderDetails"
              }
            }
          }
        },
        {
          "__metadata": {
            "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')",
            "type": "NorthwindGridModel.Order",
            "orderId": 5001,
            "customer_customerId": "IBM",
            "orderDate": "\\Date(1260993011125)\\",
            "shipCity": "Rochester",
            "shipCountry": null,

```

```
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/orderDetails"}}}]}}
```

Response Code:

200 OK

6.3.1.2 Retrieving an entity collection

A RetrieveEntitySet Request can be used by a client to retrieve a set of eXtreme Scale entities. The entities are represented as an Atom Feed Document or JSON array in the response payload.

For more details on the RetrieveEntitySet protocol defined in WCF Data Services, refer to [http://msdn.microsoft.com/en-us/library/dd541423\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541423(PROT.10).aspx)

The following RetrieveEntitySet request example retrieves all the Order entities associated with the Customer('IBM') entity. Only orders 5000 and 5001 are displayed here.

AtomPub

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders

Request Header:

Accept: application/atom+xml

Request Payload:

None

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
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">Order</title>
```

```

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Order</id>
  <updated>2009-12-16T22:53:09.062Z</updated>
  <link rel = "self" title = "Order" href = "Order"/>
  <entry>
    <category term = "NorthwindGridModel.Order" scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Order(orderId=5000,customer_customerId='IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:53:09.062Z</updated>
  <author>
    <name/>
  </author>
  <link rel = "edit" title = "Order" href =
"Order(orderId=5000,customer_customerId='IBM')"/>
  <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/cust
omer" type = "application/atom+xml;type=entry" title = "customer"
href = "Order(orderId=5000,customer_customerId='IBM')/customer"/>
  <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/order
Details" type = "application/atom+xml;type=feed" title =
"orderDetails" href =
"Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
  <content type = "application/xml">
    <m:properties>
      <d:orderId m:type = "Edm.Int32">5000</d:orderId>
      <d:customer_customerId>IBM</d:customer_customerId>
      <d:orderDate m:type = "Edm.DateTime">2009-12-
16T19:46:29.562</d:orderDate>
      <d:shipCity>Rochester</d:shipCity>
      <d:shipCountry m:null = "true"/>
      <d:version m:type = "Edm.Int32">0</d:version>
    </m:properties>
  </content>
</entry>
<entry>
  <category term = "NorthwindGridModel.Order" scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Order(orderId=5001,customer_customerId='IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:53:09.062Z</updated>
  <author>
    <name/>
  </author>
  <link rel = "edit" title = "Order" href =
"Order(orderId=5001,customer_customerId='IBM')"/>
  <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/cust
omer" type = "application/atom+xml;type=entry" title = "customer"
href = "Order(orderId=5001,customer_customerId='IBM')/customer"/>

```

```
    <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/order
Details" type = "application/atom+xml;type=feed" title =
"orderDetails" href =
"Order(orderId=5001,customer_customerId='IBM')/orderDetails"/>
    <content type = "application/xml">
      <m:properties>
        <d:orderId m:type = "Edm.Int32">5001</d:orderId>
        <d:customer_customerId>IBM</d:customer_customerId>
        <d:orderDate m:type = "Edm.DateTime">2009-12-
16T19:50:11.125</d:orderDate>
        <d:shipCity>Rochester</d:shipCity>
        <d:shipCountry m:null = "true"/>
        <d:version m:type = "Edm.Int32">0</d:version>
      </m:properties>
    </content>
  </entry>
</feed>
```

Response Code:

200 OK

JSON

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/re
stservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM
')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\\/Date(1260992789562)\\/","
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
```

```
"customer":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/orderDetails"}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')"},
"type":"NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\\/Date(1260993011125)\\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/orderDetails"}}}
```

Response Code:

200 OK

6.3.1.3 Retrieve a Property

A RetrievePrimitiveProperty request can be used to get the value of a property of an eXtreme Scale entity instance. The property value is represented as XML format for AtomPub requests and a JSON object for JSON requests in the response payload.

For more details on RetrievePrimitiveProperty request, refer to [http://msdn.microsoft.com/en-us/library/dd541245\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541245(Prot.10).aspx)

The following RetrievePrimitiveProperty request example retrieves the contactName property of the Customer('IBM') entity.

AtomPub

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName

Request Header:

Accept: application/xml

Request Payload:

None

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<contactName
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  John Doe
</contactName>
```

Response Code:

200 OK

JSON**Method:**

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{"d":{"contactName":"John Doe"}}
```

Response Code:

200 OK

6.3.1.4 Retrieve a Property Value

A RetrieveValue request can be used to get the raw value of a property on an eXtreme Scale entity instance. The property value is represented as a raw value in the response payload.

If the entity type is one of the following, then the media type of the response is “text/plain”. Otherwise the response’ media type is “application/octet-stream”. These types are:

- Java primitive types and its wrappers
- java.lang.String
- byte[],
- Byte[],
- char[],
- Character[],
- enums;
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

For more details on the RetrieveValue request, refer to [http://msdn.microsoft.com/en-us/library/dd541523\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541523(PROT.10).aspx)

The following RetrieveValue request example retrieves the raw value of the contactName property of the Customer('IBM') entity.

Request Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/\$value

Request Header:

Accept: text/plain

Request Payload:

None

Response Header:

Content-Type: text/plain

Response Payload:

John Doe

Response Code:

200 OK

6.3.1.5 Retrieve a Link

A RetrieveLink Request can be used to get the link(s) representing a to-one association or to-many association. For the to-one association, the link is from one eXtreme Scale Entity instance to another, and the link is represented in the response payload. For the to-many association, the links are from one eXtreme Scale Entity instance to all others in a specified eXtreme Scale entity collection, and the response is represented as a set of links in the response payload.

For more details on RetrieveValue request, refer to:

[http://msdn.microsoft.com/en-us/library/dd541339\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541339(PROT.10).aspx)

Here is a RetrieveLink request example. In this example, we retrieve the association between entity Order(orderId=5000,customer_customerId='IBM') and its customer. The response shows the Customer entity URI.

AtomPub**Method:**

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/\$links/customer

Request Header:

Accept: application/xml

Request Payload:

None

Response Header:

Content-Type: application/xml

Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('IBM')</uri>
```

Response Code:

200 OK

JSON

Method:

GET

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/\$links/customer

Request Header:

Accept: application/json

Request Payload:

None

Response Header:

Content-Type: application/json

Response Payload:

```
{ "d": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM') " } }
```

6.3.1.6 Retrieve Service Metadata

A RetrieveServiceMetadata Request can be used to get the conceptual schema definition language (CSDL) document, which describes the data model associated with the eXtreme Scale REST data service.

For more details on RetrieveServiceMetadata request, refer to [http://msdn.microsoft.com/en-us/library/dd541530\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541530(Prot.10).aspx)

6.3.1.7 Retrieve Service Document

A RetrieveServiceDocument Request can be used to retrieve the Service Document describing the collection of resources exposed by the eXtreme Scale REST data service,

For more details on RetrieveServiceMetadata request, refer to [http://msdn.microsoft.com/en-us/library/dd541594\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541594(Prot.10).aspx)

6.3.2 Insert Request

6.3.2.1 Insert Entity Request

An InsertEntity Request can be used to insert a new eXtreme Scale entity instance, potentially with new related entities, into the eXtreme Scale REST data service.

When inserting an entity, the client may specify if the resource or entity should be automatically linked to other existing entities in the data service. The client must include

the required binding information in the representation of the associated relation in the request payload.

In addition to supporting the insertion of a new Entity Type instance (E1), the InsertEntity request also allows inserting new entities related to E1 (described by an entity relation) in a single Request. For example, when inserting a Customer('IBM'), we can insert all the orders with Customer('IBM'). This form of an InsertEntity Request is also known as a "deep insert". In the "deep insert" case, the related entities must be represented using the inline representation of the relation associated with E1 that identifies the link to the (to-be-inserted) related entities.

The properties of the entity to be inserted are specified in the request payload. The properties are parsed by the REST data service and then set to the correspondent property on the entity instance. For the AtomPub format, the property is specified as a <d:PROPERTY_NAME> XML element. For JSON, the property is specified as a property of a JSON object.

If a property is missing in the request payload, then the REST data service sets the entity property value to the java default value. However, the database backend might reject such a default value, for example, if the column is not nullable in the database. Then a 500 response code will be returned to indicate an Internal Server error.

If there are duplicate properties specified in the payload, the last property will be used. All the previous values for the same property name are ignored by the REST data service.

If the payload contains a non-existent property, then the REST data service returns a 400 (Bad Request) response code to indicate the request sent by the client was syntactically incorrect.

If the key properties are missing, then the REST data service returns a response code of 400 (Bad Request) to indicate a missing key property.

If the payload contains a link to a related entity with a non-existent key, then the REST data service returns a 404 (Not Found) response code to indicate the linked entity cannot be found.

If the payload contains a link to a related entity with an incorrect association name, then the REST data service returns a 400 (Bad Request) response code to indicate the link cannot be found.

If the payload contains more than one link to a to-one relation, the last link will be used. All the previous links for the same association are ignored.

For more details on InsertEntity request, refer to:
[http://msdn.microsoft.com/en-us/library/dd541128\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541128(PROT.10).aspx)

An InsertEntity request inserts a Customer entity with key 'IBM'.

AtomPub

Method:

POST

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')

Request Header:

Accept: application/atom+xml
Content-Type: application/atom+xml

Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/meta
data"
  xmlns="http://www.w3.org/2005/Atom">
  <category term="NorthwindGridModel.Customer"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/schem
e" />
<content type="application/xml">
  <m:properties>
    <d:customerId>Rational</d:customerId>
    <d:city>Rochester</d:city>
    <d:companyName>Rational</d:companyName>
    <d:contactName>John Doe</d:contactName>
    <d:country>USA</d:country>
  </m:properties>
</content>
</entry>
```

Response Header:

Content-Type: application/atom+xml

Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base =
"http://localhost:8080/wxsrestservice/restservice" 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">
  <category term = "NorthwindGridModel.Customer" scheme =
"http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>

<id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('Rational')</id>
  <title type = "text"/>
  <updated>2009-12-16T23:25:50.875Z</updated>
  <author>
    <name/>
  </author>
  <link rel = "edit" title = "Customer" href =
"Customer('Rational')"/>
  <link rel =
"http://schemas.microsoft.com/ado/2007/08/dataservices/related/orde
```

```
rs" type = "application/atom+xml;type=feed" title = "orders" href =
"Customer('Rational')/orders"/>
  <content type = "application/xml">
    <m:properties>
      <d:customerId>Rational</d:customerId>
      <d:city>Rochester</d:city>
      <d:companyName>Rational</d:companyName>
      <d:contactName>John Doe</d:contactName>
      <d:country>USA</d:country>
      <d:version m:type = "Edm.Int32">0</d:version>
    </m:properties>
  </content>
</entry>
```

Response Code:

201 Created

JSON

Method:

POST

Request URI:

<http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer>

Request Header:

Accept: application/json

Content-Type: application/json

Request Payload:

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

Response Header:

Content-Type: application/json

Response Payload:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/res
tservice/NorthwindGrid/Customer('Rational')"},
"type":"NorthwindGridModel.Customer"},
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
```

```
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('Rational')/orders"}}}}
```

Response Code:

201 Created

6.3.2.2 Insert Link Request

An InsertLink Request can be used to create a new [Link](#) between two eXtreme Scale entity instances. The URI of the request must resolve to an eXtreme Scale to-many association. The payload of the request contains a single link which points to the to-many association target entity.

If the URI of the InsertLink request represents a to-one association, the REST data service returns a 400 (Bad request) response.

If the URI of the InsertLink request points to an association which does not exist, the REST data service returns a 404 (Not Found) response to indicate the link cannot be found.

If the payload contains a link with a key which does not exist, the REST data service returns a 404 (Not Found) response to indicate the linked entity cannot be found.

If the payload contains more than one link, the eXtreme Scale Rest Data Service will parse the first link. The remaining links are ignored.

For more details on InsertLink request, refer to:

[http://msdn.microsoft.com/en-us/library/dd541360\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541360(PROT.10).aspx)

The following InsertLink request example creates a link from Customer('IBM') to Order(orderId=5000,customer_customerId='IBM').

AtomPub

Method:

POST

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/\$link/orders

Request Header:

Content-Type: application/xml

Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')</uri>
```

Response Payload:

None

Response Code:

204 No Content

JSON

Method:

POST

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/\$links/orders

Request Header:

Content-Type: application/json

Request Payload:

```
{ "uri":  
  "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM') " }
```

Response Payload:

None

Response Code:

204 No Content

6.3.3 Update Requests

6.3.3.1 Update an entity

An UpdateEntity Request can be used to update an existing eXtreme Scale entity. The client can use an HTTP PUT method to replace an existing eXtreme Scale entity, or use an HTTP MERGE method to merge the changes into an existing eXtreme Scale entity.

When updating the entity, the client may specify if the entity (in addition to being updated) should be automatically linked to other existing entities in the data service that are related through single valued (to-one) associations.

The property of the entity to be updated is in the request payload. The property is parsed by the REST data service and then set to the correspondent property on the entity. For the AtomPub format, the property is specified as a <d:PROPERTY_NAME> XML element. For JSON, the property is specified as a property of a JSON object.

If a property is missing in the request payload, the REST data service sets the entity property value to the java default value for HTTP PUT method. However, the database

backend might reject such a default value if, for example, the column is not nullable in the database. Then a 500 (Internal Server Error) response code will be returned to indicate an Internal Server Error. If a property is missing in the HTTP MERGE request payload, the REST data service will not change the existing property value.

If there are duplicate properties specified in the payload, the last property will be used. All the previous values with the same property name are ignored by the REST data service.

If the payload contains a non-existent property, the REST data service returns a 400 (Bad Request) response code to indicate the request sent by the client was syntactically incorrect.

As part of the serialization of a resource, if the payload of an Update request contains any of the key properties for the entity, the REST data service ignores those key values since entity keys are immutable.

For details on UpdateEntity request, refer to [http://msdn.microsoft.com/en-us/library/dd541157\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541157(PROT.10).aspx)

An UpdateEntity request updates the city name of Customer('IBM') to 'Raleigh'.

AtomPub

Method:

PUT

Request URI:

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('IBM'\)](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM'))

Request Header:

Content-Type: application/atom+xml

Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/meta
data"
xmlns="http://www.w3.org/2005/Atom">
  <category term="NorthwindGridModel.Customer"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/schem
e" />
  <title />
  <updated>2009-07-28T21:17:50.609Z</updated>
  <author>
    <name />
  </author>
  <id />
  <content type="application/xml">
    <m:properties>
```

```
<d:customerId>IBM</d:customerId>
<d:city>Raleigh</d:city>
<d:companyName>IBM Corporation</d:companyName>
<d:contactName>Big Blue</d:contactName>
<d:country>USA</d:country>
</m:properties>
</content>
</entry>
```

Response Payload:

None

Response Code:

204 No Content

JSON

Method:

PUT

Request URI:

[http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('IBM'\)](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM'))

Request Header:

Content-Type: application/json

Request Payload:

```
{ "customerId": "IBM",
  "city": "Raleigh",
  "companyName": "IBM Corporation",
  "contactName": "Big Blue",
  "country": "USA", }
```

Response Payload:

None

Response Code:

204 No Content

6.3.3.2 Update an Entity Primitive Property

The UpdatePrimitiveProperty Request can update a property value of an eXtreme Scale entity. The property and value to be updated are in the request payload. The property cannot be a key property since eXtreme Scale does not allow clients to change entity keys.

For more details on the UpdatePrimitiveProperty request, refer to:
[http://msdn.microsoft.com/en-us/library/dd541206\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541206(Prot.10).aspx)

Here is an UpdatePrimitiveProperty request example. In this example, we update the city name of Customer('IBM') to 'Raleigh'.

AtomPub

Method:

PUT

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city

Request Header:

Content-Type: application/xml

Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  Raleigh
</city>
```

Response Payload:

None

Response Code:

204 No Content

JSON

Method:

PUT

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city

Request Header:

Content-Type: application/json

Request Payload:

```
{"city": "Raleigh"}
```

Response Payload:

None

Response Code:

204 No Content

6.3.3.3 Update an Entity Primitive Property value

The UpdateValue Request can update a raw property value of an eXtreme Scale entity. The value to be updated is represented as a raw value in the request payload. The property cannot be a key property since eXtreme Scale does not allow clients to change entity keys.

The content type of the request can be “text/plain” or “application/octet-stream” depending on the property type. Refer to section 6.3.1.4 for more details.

For more details on the UpdateValue request, refer to [http://msdn.microsoft.com/en-us/library/dd541483\(Prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541483(Prot.10).aspx)

Here is an UpdateValue request example. In this example we update the city name of Customer('IBM') to 'Raleigh'.

Method:

PUT

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/\$value

Request Header:

Content-Type: text/plain

Request Payload:

Raleigh

Response Payload:

None

Response Code:

204 No Content

6.3.3.4 Update a Link

The UpdateLink request can be used to establish an association between two eXtreme Scale entity instances. The association can be a single valued (to-one) relation or a multi valued (to-many) relation.

Updating a link between two eXtreme Scale entity instances can not only establish associations, but also remove associations. For example, if the client establishes a to-one association between an Order(orderId=5000,customer_customerId='IBM') entity and entity

and Customer('ALFKI') instance, it has to dissociate the Order(orderId=5000,customer_customerId='IBM') entity and entity from its currently associated Customer instance.

If either of the entity instances specified in the UpdateLink request cannot be found, the REST data service returns a 404 (Not Found) response.

If the URI of the UpdateLink request specifies a non-existent association, the REST data service returns a 404 (Not Found) response to indicate the link cannot be found.

If the URI specified in the UpdateLink request payload does not resolve to the same entity or the same key as specified in the URI, if exists, then the eXtreme Scale Rest Data Service returns a 400 (Bad Request) response.

If the UpdateLink request payload contains multiple links, then the REST data service will only parse the first link. The rest of the links are ignored.

For more details on the UpdateLink request, refer to:
[http://msdn.microsoft.com/en-us/library/dd541580\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541580(PROT.10).aspx)

Here is an UpdateLink request example. In this example, we update the customer relation of Order(orderId=5000,customer_customerId='IBM') entity and from Customer('IBM') to Customer('IBM').

Note: This example is for illustration only. Because all associations are typically key-associations for a partitioned grid, the link cannot be changed.

AtomPub

Method:

PUT

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/\$links/customer

Request Header:

Content-Type: application/xml

Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
  http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

Response payload:

None

Response Code:

204 No Content

JSON

Method:

PUT

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/\$links/customer

Request Header:

Content-Type: application/xml

Request Payload:

```
{"uri":  
"http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer  
( 'IBM' ) "}
```

Response Payload:

None

Response Code:

204 No Content

6.3.4 Delete Requests

6.3.4.1 Delete an entity

The DeleteEntity Request can delete an eXtreme Scale entity from the REST data service.

If any relation to the to-be-deleted entity has cascade-delete set, then the eXtreme Scale Rest data service will delete the related entity or entities.

For more details on the DeleteEntity request, refer to [http://msdn.microsoft.com/en-us/library/dd541417\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541417(PROT.10).aspx)

The following DeleteEntity request deletes the customer with key 'IBM'.

Method:

DELETE

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')

Request Payload:

None

Response Payload:

None

Response Code:

204 No Content

6.3.4.2 Delete a Property Value

The DeleteValue Request sets an eXtreme Scale entity property to null.

Any property of an eXtreme Scale entity can be set to null with a DeleteValue request. To set a property to null, ensure all of the following:

- For any primitive number type and its wrapper, BigInteger, or BigDecimal, the property value is set to 0.
- For Boolean or boolean type, the property value is set to false.
- For char or Character type, the property value is set to character #X1 (NIL).
- For enum type, the property value is set to the enum value with ordinal 0.
- For all other types, the property value is set to null.

However, such a delete request could be rejected by the database backend if, for example, the property is not nullable in the database. In this case, the REST data service returns a 500 (Internal Server Error) response.

For more details on the DeleteValue request, refer to:

[http://msdn.microsoft.com/en-us/library/dd541270\(PROT.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541270(PROT.10).aspx)

Here is a DeleteValue request example. In this example, we set the contact name of Customer('IBM') to null.

Method:

DELETE

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName

Request Payload:

None

Response Payload:

None

Response Code:

204 No Content

6.3.4.3 Delete a Link

The DeleteLink request can remove an association between two eXtreme Scale entity instances. The association can be a to-one relation or a to-many relation.

However, such a delete request could be rejected by the database backend if, for example, the foreign key constraint is set. In this case, the REST data service returns a 500 (Internal Server Error) response.

For more details on the DeleteLink request, refer to:

[http://msdn.microsoft.com/en-us/library/dd541543\(prot.10\).aspx](http://msdn.microsoft.com/en-us/library/dd541543(prot.10).aspx)

The following DeleteLink request removes the association between Order(101) and its associated Customer.

Method:

DELETE

Request URI:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/\$links/customer

Request Payload:

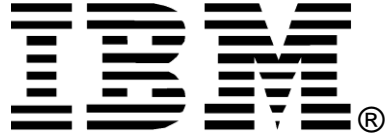
None

Response Payload:

None

Response Code:

204 No Content



© Copyright IBM Corporation 2009

IBM United States of America

Produced in the United States of America

All Rights Reserved

The e-business logo, the eServer logo, IBM, the IBM logo, OS/390, zSeries, SecureWay, S/390, Tivoli, DB2, Lotus and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries or both.

Lotus, Lotus Discovery Server, Lotus QuickPlace, Lotus Notes, Domino, and Sametime are trademarks of Lotus Development Corporation and/or IBM Corporation.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PAPER "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Information in this paper as to the availability of products (including portlets) was believed accurate as of the time of publication. IBM cannot guarantee that identified products (including portlets) will continue to be made available by their suppliers.

This information could include technical inaccuracies or typographical errors. Changes may be made periodically to the information herein; these changes may be incorporated in subsequent versions of the paper. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this paper at any time without notice.

Any references in this document to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
4205 South Miami Boulevard
Research Triangle Park, NC 27709 U.S.A.