ODATA PRODUCER LIBRARY FOR PHP

INDEX

OVERVIEW

Open Data Protocol is an open protocol for sharing data. It is built upon AtomPub (RFC 5023) and JSON. OData is a REST (Representational State Transfer) protocol, therefore a simple web browser can view the data exposed through an OData service.

The basic idea behind OData is to use a well-known data format (Atom feed) to expose a list of entities. AtomPub extends the basic Atom Protocol by allowing not only read but the whole set of CRUD operations. OData extends AtomPub by enabling simple queries over feeds.

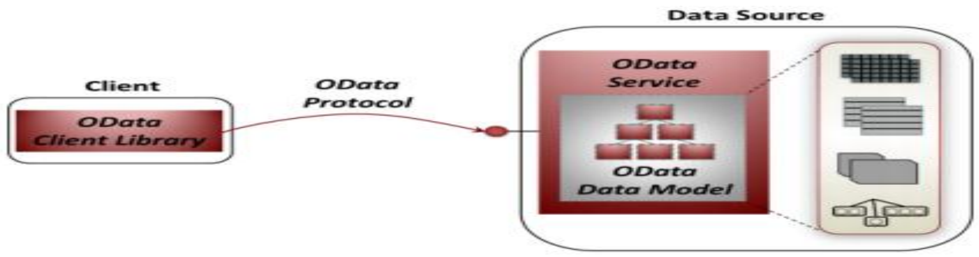
There may be many possible sources of data and many possible clients for one service like Web browsers, apps on mobile devices, business intelligence (BI) tools, and more. How can this varied set of clients access these diverse data sources?

One solution is for every data source to define its own approach to exposing data. It requires every client to contain unique code for each data source it will access, a burden for the people who write those clients. It requires the creators of each data source to specify and implement their own approach to getting at their data, making each one reinvent this wheel.

Defining a common approach makes much more sense. it would make sense to build this technology with existing Web standards as much as possible. OData allows mixing and matching clients and data sources. While it's possible to access an OData data source from an ordinary browser -- the protocol is based on HTTP.

The fundamental idea is that any OData client can access any OData data source. Rather than creating unique ways to expose and access data, data sources and their clients can instead rely on the single solution that OData provides.

How OData Works: Technology Basics



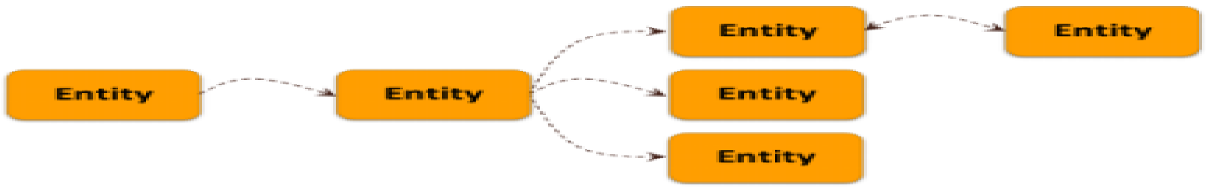
An OData service exposes data via the OData data model, which clients access with an OData client library and the OData

The OData technology has four main parts:

- The *OData data model*, which provides a generic way to organize and describe data.
- The *OData protocol*, which lets a client make requests to and get responses from an OData service. Data sent by an OData service can be represented on the wire today either in the XML-based format defined by Atom/AtomPub or in JavaScript Object Notation (JSON).
- *OData client libraries*, OData clients are applications making OData requests and getting results via the OData protocol.
- An *OData service*, which exposes an endpoint that allows access to data. This service implements the OData protocol, and it also uses the abstractions of the OData data model to translate data between its underlying forms.

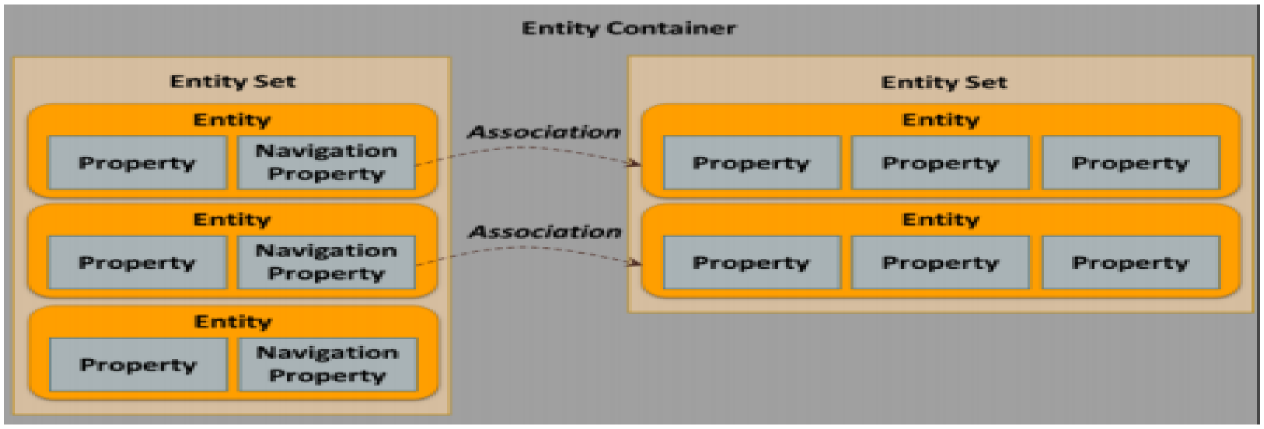
The OData Data-Model

OData uses the Entity Data Model (EDM).The EDM models data as entities and associations among those entities. Thus OData work with pretty much any kind of data.



The Entity Data Model describes data as entities connected by associations.

Associations between entities can be one-to-one, many-to-one, unidirectional or bi-directional. EDM describes only the logical structure of data.



In the EDM, an entity container holds entity sets, while each entity has one or more properties.

Protocol Basics

An OData client accesses data provided by an OData service using standard HTTP. The OData protocol largely follows the conventions defined by REST, which define how HTTP verbs are used. The most important of these verbs are:

- GET : Reads data from one or more entities.
- PUT : Updates an existing entity, replacing all of its properties.
- MERGE : Updates an existing entity, but replaces only specified properties.
- POST : Creates a new entity.
- DELETE : Removes an entity.

Each HTTP request is sent to a specific URI, identifying some resource in the target OData service's data model.

ODATA PRODUCER FOR PHP

ODataProducer for PHP is a server library which is requires to exposes the data source by using of OData Protocol.

Main components of the ODataProducer are:

1. Dispatcher

   Dispatcher is responsible for resolving the service and delegating the request handling process to the resolved service.

2. Data Service

   The DataService class is the base class for all service specific classes. This class implements two interfaces IRequestHandler and IDataServcie.

3. OperationContext

   The Operation Context classes provide access to HTTP Context of the current request and response.

4. Object Model Serializer
   a. ATOM
   b. JSON
   c. Metadata

   This module is responsible building OData Object Model from Domain Object Model. Domain Object Model is the result of execution of any Query.
5. Query Processor
   a. URI Parser
   b. Query Executor

   The Query Processor module is responsible for parsing and validation of the URL and execution of the Query.

6. Response Writer

   Response Writer is responsible to set the headers in the Outgoing Response.

OData Producer supports only Read-Only operation specified in Protocol version 2.0:

- It provides two formats for representing resources, the XML-based Atom format and the JSON format.
- Servers expose a metadata document that describes the structure of the service and its resources.
- Clients can retrieve a feed, Entry or service document by issuing an HTTP GET request against its URI.
- Servers support retrieval of individual properties within Entries.
- It supports pagination, query validation and system query options like $format, $top, $linecount, $filter, $select, $expand, $orderby, $skip .
- User can access the binary stream data.

System Requirements

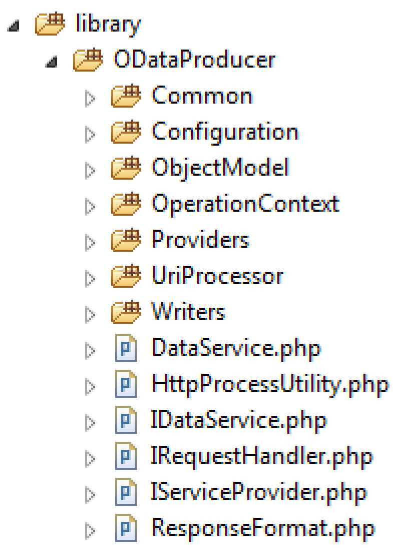
For Windows

1) IIS,URL Rewrite(http://www.iis.net/download/URLRewrite) / Apache2 with URL rewrite mode
2) PHP-5.3

For Linux

1) Apache2 with URL rewrite mode
2) PHP-5.4

Directory Structure



```
library
  ODataProducer
    Common
    Configuration
    ObjectModel
    OperationContext
    Providers
    UriProcessor
    Writers
    DataService.php
    HttpProcessUtility.php
    IDataService.php
    IRequestHandler.php
    IServiceProvider.php
    ResponseFormat.php
```

ODATA SERVICE IMPLEMENTATION

With using of OData PHP Producer, you can create data services that expose OData feeds. Data in these feeds can come from a variety of data sources.

One of the coolest things about Data Services is its provider model. Any data-source can be exposed as an OData Data Service simply by implementing a few interfaces. Developer can use data providers to expose this data as an OData feed.

The PHP OData Producer framework ships with some internal providers, and makes it possible for developer to create custom providers. The data service can use custom providers for interacting with the underlying Data Source means any data-source can be exposed as an OData Data Service. The provider implementation defines the data model for the service.

We have to implement set of interfaces to expose data by using of OData Producer:

Provider Interface	**Description**
IDataServiceMetadataProvider	Framework uses the class which implements this interface to get information about available ResourceTypes, Properties, Keys, NavigationProperties, and ResourceSets.
IDataServiceQueryProvider	Framework uses the class which implements this interface to fulfill all HTTP GET requests.
IDataServiceQueryProvider2 (**Optional :** If we need some optimization like library should not concern about filtering then only its implementation is require)	It is same as IDataServiceQueryProvider except one difference that with IDataServiceQueryProvider library does the filtering and if result-set is huge then performance of the library will not up to the marks but with IDataServiceQueryProvider2,library appends the filtering criteria in the SQL query hence library don't need to scrub out the innecessory records to get the final subset of data.
IDataServiceStreamProvider (**Optional :** if entity contains any binary data only)	Framework uses the class which implements this interface to manipulate an underlying stream for Media Link Entries.

IServiceProvider	Informs the framework that the class represents a service with Data Service Providers implementation.
IExpressionProvider (**Optional :** External Implementation of IExpressionProvider is require only with IDataServiceQueryProvider2 else library has one internal implementation and end-developer don't need to implement this interface.)	Framework uses this class to convert filtering criteria into one expression corresponding to the underlying DB.

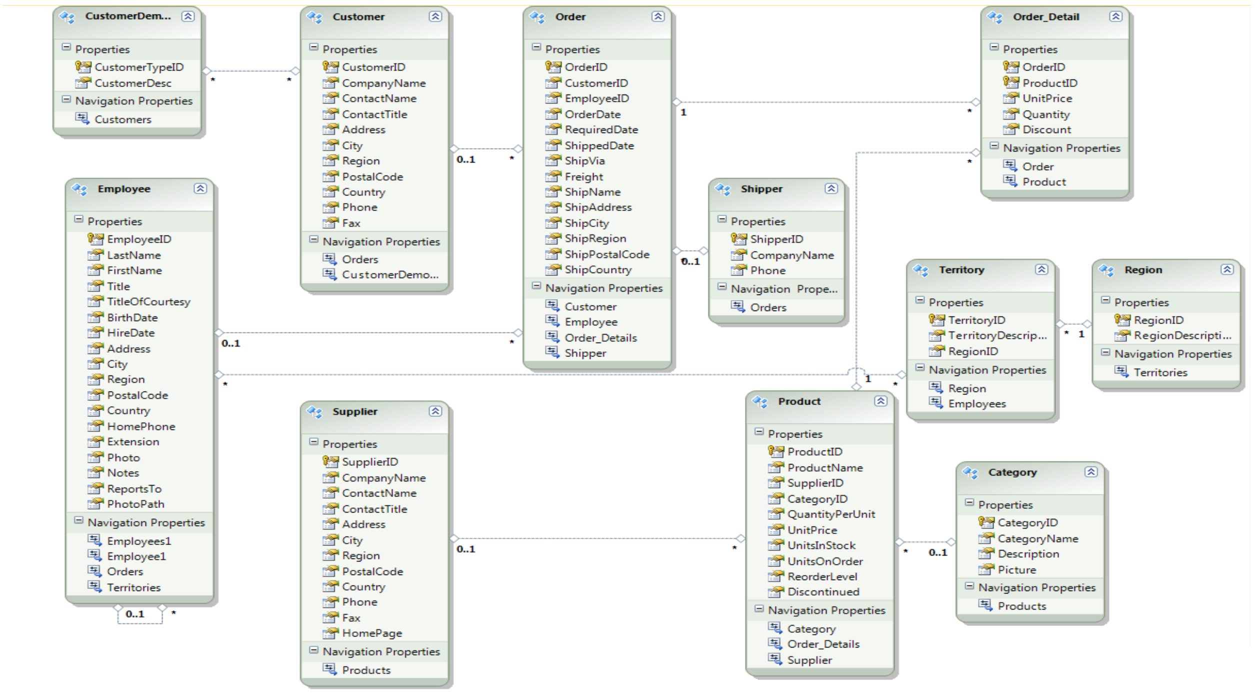
So the task to write a data service with custom metadata provider support (CMPS) can be divided into the following sub-tasks:

- Defining proxy classes representing the entities exposed by the data service.
- Defining a class `CreateNorthWindMetaData` and method CreateNorthWindMetaData::Create() that uses the methods of ServiceBaseMetadata class to define the shape of the service (container and namespace name, entities and its properties, entity sets and association between entities).
- Implement a mechanism to get the the underlying DB means we have to implement either IDataServiceQueryProvider or IDataServiceQueryProvider2 interface.
- Defining a class which represents the data service (by deriving from framework's DataService class) and this class also implements `IServiceProvider` interface.
- If end-developer implemented IDataServiceQueryProvider2 then we have to implement IExpressionProvider also so that library can convert the filtering criteria into an expression which would be compatible to underlying data source.
- If we implemented IDataServiceQueryProvider2 and property names defined in the Metadata are not matching with the names defined in the underlying data source then we have to define one more method CreateNorthWindMetaData::mappingInitialize().
    a. CreateNorthWindMetaData::mappingInitialize() is not required in few scenarios:
        i. If we are not implementing IDataServiceQueryProvider2.
        ii. If we are implementing IDataServiceQueryProvider2 but the property and entity names defined in the metadata are same as defined in the data-source and our queries are not using any name alias also for table names.
- Modifications in the Service.Config.xml file to register the new service with the library.
- Configuring the URL Rewrite module with IIS/Apache.
    a. Click here to check that how can configure the URL Rewrite mode for IIS/Apache.

IMPLEMENTATION OF INTERFACES

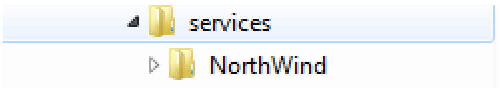
This section will explain the implementation of all the interfaces to build a data service using OData Producer for PHP.

We will use the NorthWind DB to show the process to create a new OData feed using OData Producer for PHP. Following diagram is the EDM representation of NorthWind DB Which is showing all the entities, their properties and association between entities.



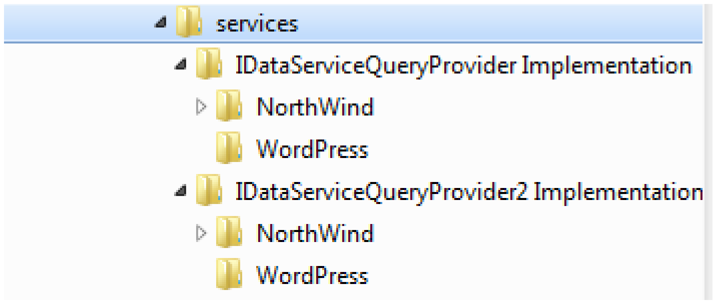
Our sample implementation requires only 3 entities: Customer, Order and Order_Detail.

First we have to create a service\<Service-Name> folder to keep the implementation of customized providers for one specific service.



For instance we have to create D:\Projects\ODataPHPProducer\service\NorthWind folder for "NorthWind" service and the newly created folder will contain files for these customized provider.

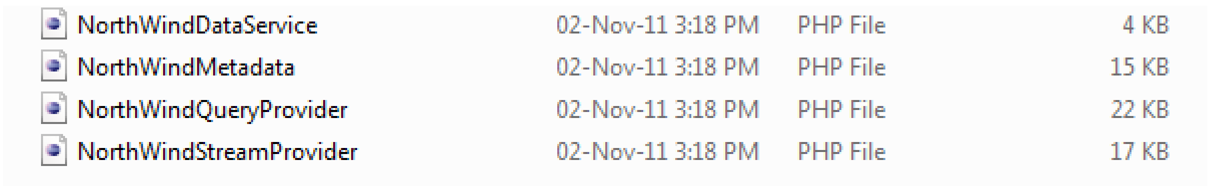
Library has few sample service implementation under "service" folder and its looks like:



services
IDataServiceQueryProvider Implementation
NorthWind
WordPress
IDataServiceQueryProvider2 Implementation
NorthWind
WordPress

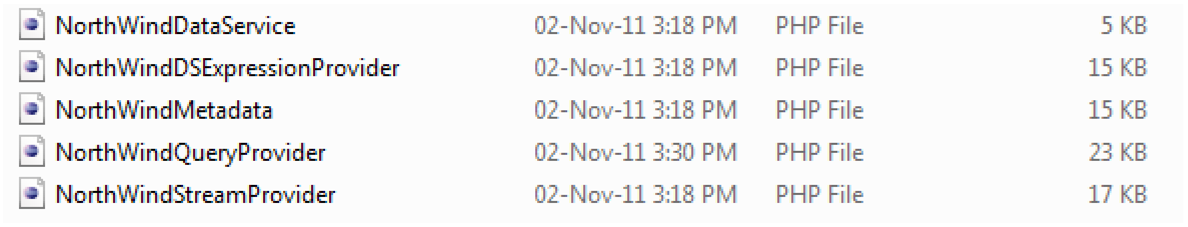
Here folder “service\IDataServiceQueryProvider Implementation\NorthWind” contains the files which are required when user implements IDataServiceQueryProvider interface and folder “service\IDataServiceQueryProvider2 Implementation\NorthWind” contains the files which are required when user implements IDataServiceQueryProvider2 interface.

Here folder “service\IDataServiceQueryProvider Implementation\NorthWind” contain following files:



NorthWindDataService	02-Nov-11 3:18 PM	PHP File	4 KB
NorthWindMetadata	02-Nov-11 3:18 PM	PHP File	15 KB
NorthWindQueryProvider	02-Nov-11 3:18 PM	PHP File	22 KB
NorthWindStreamProvider	02-Nov-11 3:18 PM	PHP File	17 KB

And folder “service\IDataServiceQueryProvider2 Implementation\NorthWind” contains following files:



NorthWindDataService	02-Nov-11 3:18 PM	PHP File	5 KB
NorthWindDSExpressionProvider	02-Nov-11 3:18 PM	PHP File	15 KB
NorthWindMetadata	02-Nov-11 3:18 PM	PHP File	15 KB
NorthWindQueryProvider	02-Nov-11 3:30 PM	PHP File	23 KB
NorthWindStreamProvider	02-Nov-11 3:18 PM	PHP File	17 KB

IDataServiceMetaDataProvider

IDataServiceMetadataProvider is responsible for describing the shape of the Resources, ResourceSets and ResourceAssociationSet available from your Data Source.

MEMBER METHODS

Name	Return Type	Description
getContainerName()	String	Get the container name for the data source.
getContainerNamespace()	String	Get the container namespace for the data source.
getResourceSets()	array(ResourceSet)	Get all entity set information.
getTypes()	array(ResourceType)	Get all resource types in the data source.

resolveResourceSet ()	ResourceSet/NULL	Get a resource set based on the name specified
resolveResourceType ()	ResourceType/NULL	Get a resource type based on the name specified
getDerivedTypes ()	array(ResourceType)/NULL	Attempts to return all types that derive from the specified resource type.
hasDerivedTypes()	Boolean	Determines whether a resource type has derived types.
getResourceAssociationSet()	ResourceAssociationSet	Gets the ResourceAssociationSet instance when given the source association end.

Implementing IDataServiceMetadataProvider

We don't need to implement IdataServiceMetada directly because library has one file "Providers\Metadata\ServiceBaseMetadata.php" which actually implements the interface IdataServiceMetadataProvider. We have to use object of ServiceBaseMetadata to define the metadata.

When we asked for the service type IDataServiceMetadataProvider in the implementation of IServiceProvider::getService() it returns instance of ServiceBaseMetadata which implements IDataServiceMetadataProvider.

Creating Metadata

Before proceeding first we try to understand few terminologies:

Resource Type (Entity Type) is the fundamental building block for describing the structure of data with the Entity Data Model (EDM).

A complex type is a template for defining rich, structured properties on entity types or on other complex types.

An entity key is a property or a set of properties of an entity type that are used to determine identity. The properties that make up an entity key are chosen at design time. The values of entity key properties must uniquely identify an entity type instance within an entity set at run time. The properties that make up an entity key should be chosen to guarantee uniqueness of instances in an entity set.

Resource properties are the fundamental building blocks of entity types and complex types. This defines the shape and characteristics of data that an entity type instance or complex type instance will contain.

An entity set is a logical container for instances of an entity type and instances of any type derived from that entity type.

First we have to define the class for each entity which is going to be exposed by the service.

Imagine if you have this entity class:

```
//Order entity type
class Order
{
    //Key Edm.Int32
    public $OrderID;
    //Edm.String
    public $CustomerID;
    //Edm.Int32
    public $EmployeeID;
    //Edm.DateTime
    public $OrderDate;
    //Edm.DateTime
    public $RequiredDate;
    //Edm.DateTime
    public $ShippedDate;
    //Edm.Int32
    public $ShipVia;
    //Edm.Decimal
    public $Freight;
    //Edm.String
    public $ShipName;
    //Edm.String
    public $ShipAddress;
    //Edm.String
    public $ShipCity;
    //Edm.String
    public $ShipRegion;
    //Edm.String
    public $ShipPostalCode;
    public $ShipCountry; //Edm.String
}
```

We have to define one class for OData service like CreateNorthWindMetadata and has to define method CreateNorthWindMetadata::create().This function will define the metadata structure of that OData service using of methods defined in ServiceBaseMetadata class:

```
class CreateNorthWindMetadata
{
    /**
     * create metadata
     *
     * @throws InvalidOperationException
     *
     * @return NorthWindMetadata
     */
    public static function create()
    {
        $metadata = new ServiceBaseMetadata('NorthWindEntities', 
'NorthWind');
```

Now we have to define Entity/ResourceType and its properties for Order in method CreateNorthWindMetadata::create() by following way:

```
        //Register the entity (resource) type 'Order'
        $orderEntityType = $metadata->addEntityType(new 
ReflectionClass('Order'), 'Order', 'NorthWind');
        $metadata->addKeyProperty($orderEntityType, 'OrderID', 
EdmPrimitiveType::INT32);
        $metadata->addPrimitiveProperty($orderEntityType, 'CustomerID', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'EmployeeID', 
EdmPrimitiveType::INT32);
        //Adding an etag property
        $metadata->addETagProperty($orderEntityType, 'OrderDate', 
EdmPrimitiveType::DATETIME);
        $metadata->addPrimitiveProperty($orderEntityType, 'RequiredDate', 
EdmPrimitiveType::DATETIME);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShippedDate', 
EdmPrimitiveType::DATETIME);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipVia', 
EdmPrimitiveType::INT32);
        $metadata->addPrimitiveProperty($orderEntityType, 'Freight', 
EdmPrimitiveType::DECIMAL);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipName', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipAddress', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipCity', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipRegion', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipPostalCode', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($orderEntityType, 'ShipCountry', 
EdmPrimitiveType::STRING);
```

The OrderID is the entity key, so we called ServiceBaseMetadata::addKeyProperty() else we had to call ServiceBaseMetadata::addPrimitiveProperty() or ServiceBaseMetadata::addEtagProperty().

Now we have to add ‘Orders’ as a resource-set:

```
        $ordersResourceSet = $metadata->addResourceSet('Orders', 
$orderEntityType);
```

If you don’t want to expose a property there is no need to add the resource property to the resource type.

Adding Complex types

In the Order entity type all the properties are primitive types, but there are some cases where you want to add complex types. Imagine if you have this entity class:

```
class Customer
{
    //Key Edm.String
    public $CustomerID;
    //Edm.String
    public $CompanyName;
    //Edm.String
    public $ContactName;
    //Edm.String
    public $Phone;
    //Edm.String
    public $Fax;
    //NorthWind.Address
    public $Address;
    public $EmailAddresses;
    //array(Address)
    public $OtherAddresses;
    //Navigation Property Orders (ResourceSetReference)
    public $Orders;
}
```

In this Customer entity class the property Address is complex property. The complex class for Address is:

```
class Address
{
    //Edm.String
    public $StreetName;
    //Edm.String
    public $City;
    //Edm.String
    public $Region;
    //Edm.String
    public $PostalCode;
    //Edm.String
    public $Country;
    //NorthWind.Address
    public $AltAddress;

}
```

To create the resource type for Customer first you need to register the complex type Address

```
        //Register the complex type 'Address' having a property of same type.
        $addressComplexType = $metadata->addComplexType(new
ReflectionClass('Address'), 'Address', 'NorthWind', null);
        $metadata->addPrimitiveProperty($addressComplexType, 'StreetName',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($addressComplexType, 'City',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($addressComplexType, 'Region',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($addressComplexType, 'PostalCode',
EdmPrimitiveType::STRING);
```

```
        $metadata->addPrimitiveProperty($addressComplexType, 'Country',
EdmPrimitiveType::STRING);
        //A complex sub property to hold alternate address
        $metadata->addComplexProperty($addressComplexType, 'AltAddress',
$addressComplexType);
```

Please note that In the Address complex type the property AltAddress is again a complex property.

Next register the entity type Customer

```
        //Register the entity (resource) type 'Customer'
        $customersEntityType = $metadata->addEntityType(new
ReflectionClass('Customer'), 'Customer', 'NorthWind');
        $metadata->addKeyProperty($customersEntityType, 'CustomerID',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($customersEntityType, 'CompanyName',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($customersEntityType, 'ContactName',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($customersEntityType, 'ContactTitle',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($customersEntityType, 'Phone',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($customersEntityType, 'Fax',
EdmPrimitiveType::STRING);
        $metadata->addComplexProperty($customersEntityType, 'Address',
$addressComplexType);
        //Add a bag property (bag of complex type) to hold array of other
addresses
        $metadata->addComplexProperty($customersEntityType, 'OtherAddresses',
$addressComplexType, true);
        //Add a bag property (bag of primitve type) to hold array of email
addresses
        $metadata->addPrimitiveProperty($customersEntityType,
'EmailAddresses', EdmPrimitiveType::STRING, true);
```

Note the property OtherAddresses is a bag property that hold an array of other addresses.

Now we have to add 'Customers' as a resource-set:

```
         $customersResourceSet = $metadata->addResourceSet('Customers',
$customersEntityType);
```

Adding Stream Data/Property

In the Order entity type all the properties are primitive types and for Customer all the entity types are either Primitive or Complex, but there are some cases where you want to add stream data. Imagine if you have this entity class:

```
class Employee
{
    //Key Edm.Int32
     public $EmployeeID;
     //Edm.String
     public $FirstName;
     //Edm.String
     public $LastName;
     //Edm.String
     public $Title;
     //Edm.String
     public $TitleOfCourtesy;
     //Edm.DateTime
     public $BirthDate;
     //Edm.DateTime
     public $HireDate;
     //Edm.String
     public $Address;
     //Edm.String
     public $City;
     //Edm.String
     public $Region;
     //Edm.String
     public $PostalCode;
     //Edm.String
     public $Country;
     //Edm.String
     public $HomePhone;
     //Edm.String
     public $Extension;
     //Edm.String
     public $Notes;
     //Bag of strings
     public $Emails;
     //Edm.Int32
     public $ReportsTo;
     //Edm.Binary
     public $Photo;
     //Edm.String
     public $PhotoPath;
}
```

Here the entity type ‘Employee’ has a property named ‘Photo’ which is a binary data (BLOB type in DB for example) , instance of this field will hold base 64 encoded string. So you have to add Employee entity as follows:

```
        //Register the entity (resource) type 'Employee'
        $employeeEntityType = $metadata->addEntityType(new 
ReflectionClass('Employee'), 'Employee', 'NorthWind');
        $metadata->addKeyProperty($employeeEntityType, 'EmployeeID', 
EdmPrimitiveType::INT32);
        $metadata->addPrimitiveProperty($employeeEntityType, 'FirstName', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'LastName', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Title', 
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 
'TitleOfCourtesy', EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'BirthDate', 
EdmPrimitiveType::DATETIME);
        $metadata->addPrimitiveProperty($employeeEntityType, 'HireDate', 
EdmPrimitiveType::DATETIME);
```

```
        $metadata->addPrimitiveProperty($employeeEntityType, 'Address',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'City',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Region',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'PostalCode',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Country',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'HomePhone',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Extension',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Notes',
EdmPrimitiveType::STRING);
        $metadata->addPrimitiveProperty($employeeEntityType, 'ReportsTo',
EdmPrimitiveType::INT32);
        //$metadata->addPrimitiveProperty($employeeEntityType, 'Photo',
EdmPrimitiveType::BINARY);
        $metadata->addPrimitiveProperty($employeeEntityType, 'Emails',
EdmPrimitiveType::STRING, true);
        $metadata->addPrimitiveProperty($employeeEntityType, 'PhotoPath',
EdmPrimitiveType::STRING);
        //Set Employee entity type as MLE thus the url
http://host/NorthWind.svc/Employee(1875)/$value will give the stream
associated with employee with id 1875
        $employeeEntityType->setMediaLinkEntry(true);
```

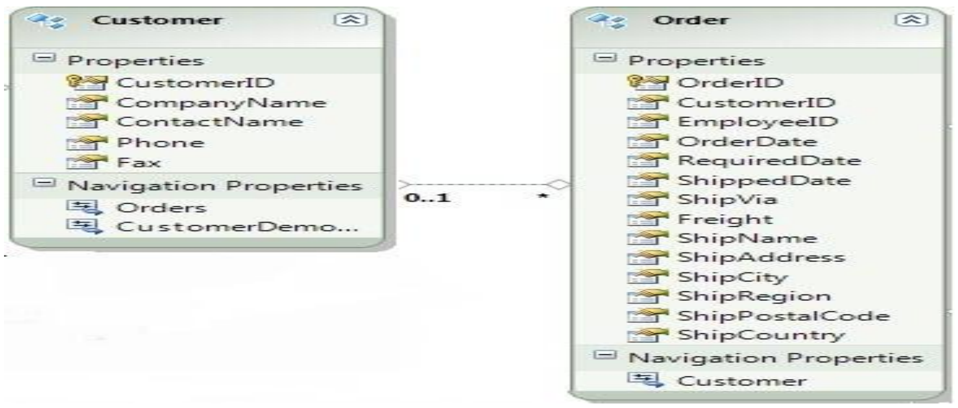
Now we have to add ‘Employees’ as a resource-set:

```
         $customersResourceSet = $metadata->addResourceSet('Customers',
$customersEntityType);
```

Relationships

Imagine you have an Order class and a Customer class.

Suppose relationship between these two is one to many i.e. from Order to Customer cardinality is one and from Customer to Order cardinality is many.



In this case Order class can contain a property that represents the associated Customer resource (called Resource reference) and Customer can contain a property that represents associated Order resources (called Resource Set reference).

```
class Customer
{
    //Key Edm.String
    public $CustomerID;
    //Edm.String
    public $CompanyName;
    //Edm.String
    public $ContactName;
    //Edm.String
    public $ContactTitle;
    //Edm.String
    public $Phone;
    //Edm.String
    public $Fax;
    //NorthWind.Address
    public $Address;
    //array(string)
    public $EmailAddresses;
    //array(Address)
    public $OtherAddresses;     //Navigation Property Orders
(ResourceSetReference)
    public $Orders;
}


class Order
{
    //Key Edm.Int32
    public $OrderID;
    //Edm.String
    public $CustomerID;
    //Edm.Int32
    public $EmployeeID;
    //Edm.DateTime
    public $OrderDate;
    //Edm.DateTime
    public $RequiredDate;
    //Edm.DateTime
    public $ShippedDate;
    //Edm.Int32
    public $ShipVia;
    //Edm.Decimal
    public $Freight;
    //Edm.String
    public $ShipName;
    //Edm.String
    public $ShipAddress;
    //Edm.String
    public $ShipCity;
    //Edm.String
    public $ShipRegion;
    //Edm.String
    public $ShipPostalCode;
    //Edm.String
    public $ShipCountry;
    //Navigation Property Customer (ResourceReference)
    public $Customer;
    //Navigation Property Order_Details (ResourceSetReference)
    public $Order_Details;
}
```

Building relationships through metadata

We have to create the entities first and then we have to build a relationship between them.

First we have to register Order and its primitive properties as explained above and then we have to do it for Customer entity also,as explained above.

The next step is to tell Data Services about order.Customer and customer.Orders.

```
        //Register the assoications (navigations)
        //Customers (1) <==> Orders (0-*)
        $metadata->addResourceSetReferenceProperty($customersEntityType,
'Orders', $ordersResourceSet);
        $metadata->addResourceReferenceProperty($orderEntityType, 'Customer',
$customersResourceSet);
```

Notice that **order.Customer is a ResourceReference, because there is just one customer, whereas customer.Orders is a ResourceSetReference, because it is a collection**.

IDataServiceQueryProvider

The class which implements this interface is used by the framework to perform the query operations on underlying data source.

Member Methods

Name	Return Type	Description
getResourceSet ()	array(Object)/array()	Get the collection of entities belongs to an entity set.
getResourceFromResourceSet ()	Object/NULL	Gets an entity instance from an entity set identified by a key.
getResourceFromRelatedResour	Object/NULL	Gets a related entity instance from an

ceSet ()		entity set identified by a key.
getRelatedResourceSet ()	array(Object)/array()	Get related resource set for a resource.
getRelatedResourceReference ()	Object/NULL	Get related resource for a resource.

Implementing IDataSrviceQueryProvider

When we asked for the service type IDataServiceQueryProvider in the implementation of IServiceProvider::getService () it returns instance of type which implements IDataServiceQueryProvider.

The first step in implementing the IDataServiceQueryProvider is to create a class that implements the interface.

```
class NorthWindQueryProvider implements IDataServiceQueryProvider
{
```

Inside the constructor of this class connect to the database as follows:

For SQL-Server:

```
class NorthWindQueryProvider implements IDataServiceQueryProvider
{
    /**
     * Handle to connection to Database
     */
    private $_connectionHandle = null;

    /**
     * Constructs a new instance of NorthWindQueryProvider
     */
    public function __construct()
    {
        /*
        Connect to the local server using Windows Authentication and specify
        the NorthWind database as the database in use. To connect using
        SQL Server Authentication, set values for the "UID" and "PWD"
        Attributes in the connection info parameter. For example:
        array("UID" => $uid, "PWD" => $pwd, "Database"=>"NorthWind");
        */
        $this->_connectionHandle = sqlsrv_connect(SERVER, array("Database"=>
DATABASE));
        if( $this->_connectionHandle ) {
        } else {
             die( print_r( sqlsrv_errors(), true));
        }
    }

}
```

For MySql:

```
define('DB_NAME', 'northwind');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'root');

/** MySQL hostname */
define('DB_HOST', 'localhost');
class NorthWindQueryProvider implements IDataServiceQueryProvider
{
    /**
     * Handle to connection to Database
     */
    private $_connectionHandle = null;

    /**
     * Constructs a new instance of WordPressQueryProvider
     */
    public function __construct()
    {
        //In case of MySQL we need host, username  and password to connect to
DB
        $this->_connectionHandle = @mysql_connect( DB_HOST, DB_USER,
DB_PASSWORD, true );
        if( $this->_connectionHandle ) {
        } else {
             die( print_r( mysql_error(), true));
        }
        mysql_select_db(DB_NAME, $this->_connectionHandle);
    }
}
```

Next is to start implementing the methods. There are five methods in IDataServiceQueryProvider that you need to implement. The following methods are only called when you actually query one of the ResourceSets:

getResourceSet(ResourceSet $resourceSet)

This method implementation should behave in such a way that for the given resource set (entity set); method should return all entities belonging to that resource set.

Example : The query http://localhost/NorthWind.svc/Customers will display all the customers in the NorthWind DB. In this case we retrieve all the customers in the NorthWind DB and serialize the sql/mysql result array in to array list of Customer object as below:

```
    public function getResourceSet(ResourceSet $resourceSet)
    {

        $resourceSetName =  $resourceSet->getName();
        if ($resourceSetName !== 'Customers'
            && $resourceSetName !== 'Orders'
            && $resourceSetName !== 'Order_Details'
            && $resourceSetName !== 'Employees'
        ) {
            die('(NorthWindQueryProvider) Unknown resource set ' .
$resourceSetName);
        }

        if ($resourceSetName === 'Order_Details') {
            $resourceSetName = 'Order Details';
        }

        $query = "SELECT * FROM [$resourceSetName]";
        $stmt = sqlsrv_query($this->_connectionHandle, $query);
        if ($stmt === false) {
             die(print_r(sqlsrv_errors(), true));
        }

        $returnResult = array();
        switch ($resourceSetName) {
        case 'Customers':
            $returnResult = $this->_serializeCustomers($stmt);
            break;
        case 'Orders':
            $returnResult = $this->_serializeOrders($stmt);
            break;
        case 'Order Details':
            $returnResult = $this->_serializeOrderDetails($stmt);
            break;
        case 'Employees':
            $returnResult = $this->_serializeEmployees($stmt);
            break;
        }

        sqlsrv_free_stmt($stmt);
        return $returnResult;
    }

    /**
     * Serialize the sql result array into Customer objects
     *
     * @param array(array) $result result of the sql query
     *
     * @return array(Object)
     */
    private function _serializeCustomers($result)
    {
        $customers = array();
        while ($record = sqlsrv_fetch_array($result, SQLSRV_FETCH_ASSOC)) {
             $customers[] = $this->_serializeCustomer($record);
        }

        return $customers;
    }
    /**
     * Serialize the sql row into Customer object
     * @param array $record
     * @return Object
     */
    private function _serializeCustomer($record)
    {
        $customer = new Customer();
        $customer->CustomerID = $record['CustomerID'];
        $customer->CompanyName = $record['CompanyName'];
        $customer->ContactName = $record['ContactName'];
        $customer->Phone = $record['Phone'];
        $customer->Fax = $record['Fax'];
        $customer->Address = new Address();
        $customer->Address->StreetName = $record['Address'];
        $customer->Address->City = $record['City'];
        $customer->Address->Region = $record['Region'];
        $customer->Address->PostalCode = $record['PostalCode'];
        $customer->Address->Country = $record['Country'];
        //Set alternate address
        $customer->Address->AltAddress = new Address();
        $customer->Address->AltAddress->StreetName = $record['Alt_Address'];
        $customer->Address->AltAddress->City = $record['Alt_City'];
        $customer->Address->AltAddress->Region = $record['Alt_Region'];
        $customer->Address->AltAddress->PostalCode =
$record['Alt_PostalCode'];
        $customer->Address->AltAddress->Country = $record['Alt_Country'];
        return $customer;
    }
//Note: Similarly we can serialize other resource sets from the database.
```

If there are no customers you should return an empty array.

getResourceFromResourceSet (ResourceSet $resourceSet, KeyDescriptor $keyDescriptor)

This method implementation should behave in such a way that for the given resource set (entity) and value of key(s), method should return the specific entity in the entity set with the specified key. KeyDescriptor is a type used to represent Key (identifier) for an entity (resource). Entity's identifier is a collection of value for key properties. These values can be named or positional, depending on how they were specified in the URI.
Eg:

- Named values: Customers(CustomerID = 'ALFKI'), Order_Details(OrderID=10248,ProductID=11)
- Positional values: Customers('ALFKI'), Order_Details(10248, 11)

The query http://localhost/NorthWind.svc/Customers('ALFKI') will display the customer with customer id as ALFKI. For this we retrieve only one customer with the CustomerID as ALFKI inside the getResourceFromResourceSet method, and you should return null if such a customer doesn't exist.

```
public function getResourceFromResourceSet(ResourceSet $resourceSet,
KeyDescriptor $keyDescriptor)
{
    $resourceSetName =  $resourceSet->getName();
    if ($resourceSetName !== 'Customers'
        && $resourceSetName !== 'Orders'
        && $resourceSetName !== 'Order_Details'
        && $resourceSetName !== 'Products'
        && $resourceSetName !== 'Employees'
     ) {
        die('(NorthWindQueryProvider) Unknown resource set ' .
$resourceSetName);
     }

     if ($resourceSetName === 'Order_Details') {
         $resourceSetName = 'Order Details';
     }

     $namedKeyValues = $keyDescriptor->getValidatedNamedValues();
     $condition = null;
     foreach ($namedKeyValues as $key => $value) {
        $condition .= $key . ' = ' . $value[0] . ' and ';
     }

     $len = strlen($condition);
     $condition = substr($condition, 0, $len - 5);
     $query = "SELECT * FROM [$resourceSetName] WHERE $condition";
     $stmt = sqlsrv_query($this->_connectionHandle, $query);
     if ($stmt === false) {
         die(print_r(sqlsrv_errors(), true));
     }

     //If resource not found return null to the library
     if (!sqlsrv_has_rows($stmt)) {
        return null;
     }

     $result = null;
     while ( $record = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
        switch ($resourceSetName) {
            case 'Customers':
                $result = $this->_serializeCustomer($record);
                break;
            case 'Orders':
                $result = $this->_serializeOrder($record);
                break;
            case 'Order Details':
                $result = $this->_serializeOrderDetail($record);
                break;
            case 'Employees':
                $result = $this->_serializeEmployee($record);
                break;
        }
     }

     sqlsrv_free_stmt($stmt);
     return $result;
}

**Note: See the serialize function above.
```

getRelatedResourceSet (ResourceSet $sourceResourceSet, $sourceEntityInstance, ResourceSet $targetResourceSet, ResourceProperty $targetProperty)

The implementation of this method should behave such a way that given the source resource (entity) set, source entity instance, target resource set which is having association with source resource set and name of the property in source resource representing target resource set, method should return the subset target entity set identified by the parameters.

Example: The query http://localhost/NorthWind.svc/Customers('ALFKI')/Orders will display the orders associated with the customer ALFKI. Then in the implementation of getRelatedResourceSet you do something like this:

```
public function  getRelatedResourceSet(ResourceSet
$sourceResourceSet,$sourceEntityInstance,
        ResourceSet $targetResourceSet,
        ResourceProperty $targetProperty
    ) {
    $result = array();
    $srcClass = get_class($sourceEntityInstance);
    $navigationPropName = $targetProperty->getName();
    if ($srcClass === 'Customer') {
       if ($navigationPropName === 'Orders') {
           $query = "SELECT * FROM Orders WHERE CustomerID =
'$sourceEntityInstance->CustomerID'";
           $stmt = sqlsrv_query($this->_connectionHandle, $query);
           if ($stmt === false) {
               die(print_r(sqlsrv_errors(), true));
           }
           $result = $this->_serializeOrders($stmt);
       } else {
           die('Customer does not have navigation porperty with name: ' .
$navigationPropName);
       }
    } else if ($srcClass === 'Order') {
       if ($navigationPropName === 'Order_Details') {
           $query = "SELECT * FROM [Order Details] WHERE OrderID =
$sourceEntityInstance->OrderID";
           $stmt = sqlsrv_query($this->_connectionHandle, $query);
           if ($stmt === false) {
               die(print_r(sqlsrv_errors(), true));
           }
           $result = $this->_serializeOrderDetails($stmt);
       } else {
           die('Order does not have navigation porperty with name: ' .
$navigationPropName);
       }
    }
    return $result;
}
```

getResourceFromRelatedResourceSet (ResourceSet $sourceResourceSet, $sourceEntityInstance, ResourceSet $targetResourceSet, ResourceProperty $targetProperty, KeyDescriptor $keyDescriptor)

This method should return a related entity instance from an entity set identified by a key. If there is no entity instance is found return null.

Example: The query http://localhost/NorthWind.svc/Customers('ALFKI')/Orders(10643) will display the order identified by the OrderID 10643 of the customer identified by CustomerID ALFKI. The implementation of getResourceFromRelatedResourceSet will be as follows:

```
    public function  getResourceFromRelatedResourceSet(ResourceSet
$sourceResourceSet,
        $sourceEntityInstance,
        ResourceSet $targetResourceSet,
        ResourceProperty $targetProperty,
        KeyDescriptor $keyDescriptor
    ) {
        $result = array();
        $srcClass = get_class($sourceEntityInstance);
        $navigationPropName = $targetProperty->getName();
        $key = null;
        foreach ($keyDescriptor->getValidatedNamedValues() as $keyName =>
$valueDescription) {
            $key = $key . $keyName . '=' . $valueDescription[0] . ' and ';
        }

        $key = rtrim($key, ' and ');
        if ($srcClass === 'Customer') {
            if ($navigationPropName === 'Orders') {
                $query = "SELECT * FROM Orders WHERE CustomerID =
'$sourceEntityInstance->CustomerID' and $key";
                $stmt = sqlsrv_query($this->_connectionHandle, $query);
                if ($stmt === false) {
                     die(print_r(sqlsrv_errors(), true));
                }
                $result = $this->_serializeOrders($stmt);
            } else {
                die('Customer does not have navigation porperty with name: '
. $navigationPropName);
            }
        } else if ($srcClass === 'Order') {
            if ($navigationPropName === 'Order_Details') {
                $query = "SELECT * FROM [Order Details] WHERE OrderID =
$sourceEntityInstance->OrderID";
                $stmt = sqlsrv_query($this->_connectionHandle, $query);
                if ($stmt === false) {
                     die(print_r(sqlsrv_errors(), true));
                }
                $result = $this->_serializeOrderDetails($stmt);
            } else {
                die('Order does not have navigation porperty with name: ' .
$navigationPropName);
            }
        }
        return empty($result) ? null : $result[0];
    }
```

getRelatedResourceReference (ResourceSet $sourceResourceSet, $sourceEntityInstance, ResourceSet $targetResourceSet, ResourceProperty $targetProperty)

This method returns related resource for a resource. Return null if there is no resource found.

Example: The query http://localhost/NorthWind.svc/Orders(10643)/Customer will display the customer who placed the order with OrderID 10643. In the implementation of getRelatedResourceReference you do something like this:

```
public function getRelatedResourceReference(ResourceSet $sourceResourceSet,
     $sourceEntityInstance,
     ResourceSet $targetResourceSet,
     ResourceProperty $targetProperty
) {
     $result = null;
     $srcClass = get_class($sourceEntityInstance);
     $navigationPropName = $targetProperty->getName();
     if ($srcClass === 'Order') {
          if ($navigationPropName === 'Customer') {
               if (empty($sourceEntityInstance->CustomerID)) {
                    $result = null;
               } else {
                    $query = "SELECT * FROM Customers WHERE CustomerID =
'$sourceEntityInstance->CustomerID'";
                    $stmt = sqlsrv_query($this->_connectionHandle, $query);
                    if ($stmt === false) {
                         die(print_r(sqlsrv_errors(), true));
                    }
                    if (!sqlsrv_has_rows($stmt)) {
                         $result =  null;
                    }
                    $result = $this-
>_serializeCustomer(sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC));
                   }
             } else {
                  die('Customer does not have navigation porperty with name: '
. $navigationPropName);
             }
        } else if ($srcClass === 'Order_Details') {
             if ($navigationPropName === 'Order') {
                  if (empty($sourceEntityInstance->OrderID)) {
                       $result = null;
                  } else {
                       $query = "SELECT * FROM Orders WHERE OrderID =
$sourceEntityInstance->OrderID";
                       $stmt = sqlsrv_query($this->_connectionHandle, $query);
                       if ($stmt === false) {
                            die(print_r(sqlsrv_errors(), true));
                    }
                    if (!sqlsrv_has_rows($stmt)) {
                        $result =  null;
                    }
                    $result = $this-
>_serializeOrder(sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC));
                }
            } else {
                die('Order_Details does not have navigation porperty with name: '
. $navigationPropName);
            }
        }
        return $result;
}
```

IDataServiceQueryProvider2

Implementation of IDataServiceQueryProvider2 interface is optional and it is required only when we are not implementing IDataServiceQueryProvider.

IDataServiceQueryProvider is a basic query provider and library does the filtering to get the required subset of data whereas when we implement IDataServiceQueryProvider2,library use the filtering criteria in the query itself hence library don’t need to do any extra filtering to get the subset of data.

MEMBER METHODS

Name	Parameters	Return Type	Description
getResourceSet ()	$resourceSet(Type: Object of ResourceSet ,MUST) $filterOption(Type : String, OPTIONAL) $orderby(Type : String, OPTIONAL) $select(Type : String, OPTIONAL) $top(Type : String, OPTIONAL) $skip(Type : String, OPTIONAL)	array(Object)/ array()	Get the collection of entities belongs to an entity set. If we need optimized way of query execution then we have to pass filterOptions as string. Parameters orderBy, select, top, skip are kept for future purpose, **no need to pass these parameters as of now.**

getResourceFromResourceSet ()	`$resourceSet`(Type: Object of ResourceSet ,MUST) `$keyDescriptor`(Type: Object of `KeyDescriptor` ,MUST)	Object/NULL	Gets an entity instance from an entity set identified by a key.
getResourceFromRelatedResourceSet ()	`$sourceResourceSet`(Type: Object of `ResourceSet`,MUST) `$sourceEntityInstance`(Type: Object of `EntityInstance` ,MUST) `$targetResourceSet`(Type: Object of `ResourceSet` ,MUST) `$targetProperty`(Type : Object of `ResourceProperty` ,MUST) `$keyDescriptor`(Type: Object of `KeyDescriptor` ,MUST)	Object/NULL	Gets a related entity instance from an entity set identified by a key.
getRelatedResourceSet ()	$filterOption(Type : String, OPTIONAL) $orderby(Type : String, OPTIONAL) $select(Type : String, OPTIONAL) $top(Type : String, OPTIONAL) $skip(Type : String, OPTIONAL)	array(Object)/ array()	Get related resource set for a resource. If we need optimized way of query execution then we have to pass filterOptions as string. Parameters orderBy, select, top, skip are kept for future purpose, **no need to pass these parameters as of now.**

getRelatedResourceReference ()	$sourceResourceSet(Type: Object of ResourceSet,MUST) $sourceEntityInstance(Type: Object of EntityInstance ,MUST) $targetResourceSet(Type: Object of ResourceSet ,MUST) $targetProperty(Type: Object of ResourceProperty ,MUST)	Object/NULL	Get related resource for a resource.

Implementing IDataSrviceQueryProvider2

When we asked for the service type IDataServiceQueryProvider2 in the implementation of IServiceProvider::getService() it returns instance of type which implements IDataServiceQueryProvider2.

With IDataServiceQueryProvider2 implementation we have to do minor changes in the getService().Please click here to check the implementation of getService() with IDataServiceQueryProvider2.

The first step in implementing the IDataServiceQueryProvider2 is to create a class that implements the interface.

```
class NorthWindQueryProvider implements IDataServiceQueryProvider2
{
```

Inside the constructor of this class connect to the database as follows:

For SQL-Server:

```
class NorthWindQueryProvider implements IDataServiceQueryProvider2
{
    /**
     * Handle to connection to Database
     */
    private $_connectionHandle = null;

    /**
     * Constructs a new instance of NorthWindQueryProvider
     */
    public function __construct()
    {
        /*
        Connect to the local server using Windows Authentication and specify
        the NorthWind database as the database in use. To connect using
        SQL Server Authentication, set values for the "UID" and "PWD"
        Attributes in the connection info parameter. For example:
        array("UID" => $uid, "PWD" => $pwd, "Database"=>"NorthWind");
        */
        $this->_connectionHandle = sqlsrv_connect(SERVER, array("Database"=>
DATABASE));
        if( $this->_connectionHandle ) {
        } else {
             die( print_r( sqlsrv_errors(), true));
        }
    }

}
```

For MySql:

```
define('DB_NAME', 'northwind');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'root');

/** MySQL hostname */
define('DB_HOST', 'localhost');
class NorthWindQueryProvider implements IDataServiceQueryProvider
{
    /**
     * Handle to connection to Database
     */
    private $_connectionHandle = null;

    /**
     * Constructs a new instance of WordPressQueryProvider
     */
    public function __construct()
    {
        //In case of MySQL we need host, username  and password to connect to
DB
        $this->_connectionHandle = @mysql_connect( DB_HOST, DB_USER,
DB_PASSWORD, true );
        if( $this->_connectionHandle ) {
        } else {
             die( print_r( mysql_error(), true));
        }
        mysql_select_db(DB_NAME, $this->_connectionHandle);
    }
}
```

Next is to start implementing the methods. There are five methods in IDataServiceQueryProvider2 which we need to implement.

All methods of IDataServiceQueryProvider2 are same as defined in IDataServiceQueryProvider except following two methods : IDSQP2::getResourceSet() and IDSQP2::getRelatedResourceSet().

In this section we will check the implementation of IDSQP2::getResourceSet() and IDSQP2::getRelatedResourceSet().

THESE TWO METHODS CONTAINS FEW OTHER ARGUMENTS SO THAT LIBRARY CAN FIRE A MORE OPTIMIZED QUERY AND HENCE CAN AVOID EXTRA PROCESSING DONE BY LIBRARY.AS OF NOW ONLY FILTEROPTION ARGUMENT IS BEING USE IN BOTH OF THESE METHODS AND OTHER ARGUMENTS(SELECT, ORDERBY, TOP, SKIP) ARE KEPT FOR FUTURE PURPOSE **SO NO NEED TO PASS** (SELECT, ORDERBY, TOP, SKIP) **PARAMETERS AS OF NOW**.getResourceSet(ResourceSet $resourceSet, $FILTEROPTION = NULL, $SELECT=NULL, $ORDERBY=NULL, $TOP=NULL, $SKIP=NULL)

This method implementation should behave in such a way that for the given resource set (entity set); method should not return all entities belonging to that resource set and it apply the filtering criteria in the query and it should return only required set of data only.

Example : The query http://localhost/NorthWind.svc/Customers will display all the customers in the NorthWind DB. In this case we retrieve all the customers in the NorthWind DB and serialize the sql/mysql result array in to array list of Customer object as below:

```
    public function getResourceSet(ResourceSet $resourceSet, $filterOption = 
null,$select=null, $orderby=null, $top=null, $skip=null)
    {
        $resourceSetName =  $resourceSet->getName();
        if ($resourceSetName !== 'Customers'
            && $resourceSetName !== 'Orders'
            && $resourceSetName !== 'Order_Details'
            && $resourceSetName !== 'Employees'
        ) {
            die('(NorthWindQueryProvider) Unknown resource set ' . 
$resourceSetName);
        }
        if ($resourceSetName === 'Order_Details') {
            $resourceSetName = 'Order Details';
        }
        $query = "SELECT * FROM [$resourceSetName]";
        if ($filterOption != null) {
            $query .= ' WHERE ' . $filterOption;
        }
        $stmt = sqlsrv_query($this->_connectionHandle, $query);

        if ($stmt === false) {
             die(print_r(sqlsrv_errors(), true));
        }

        $returnResult = array();
        switch ($resourceSetName) {
        case 'Customers':
            $returnResult = $this->_serializeCustomers($stmt);
            break;
        case 'Orders':
            $returnResult = $this->_serializeOrders($stmt);
            break;
        case 'Order Details':
            $returnResult = $this->_serializeOrderDetails($stmt);
            break;
        case 'Employees':
            $returnResult = $this->_serializeEmployees($stmt);
            break;
        }

        sqlsrv_free_stmt($stmt);
        return $returnResult;
    }

    /**
     * Serialize the sql result array into Customer objects
     *
     * @param array(array) $result result of the sql query
     *
     * @return array(Object)
     */
    private function _serializeCustomers($result)
    {
        $customers = array();
        while ($record = sqlsrv_fetch_array($result, SQLSRV_FETCH_ASSOC)) {
             $customers[] = $this->_serializeCustomer($record);
        }

        return $customers;
    }
    /**
     * Serialize the sql row into Customer object
     * @param array $record
     * @return Object
     */
    private function _serializeCustomer($record)
    {
        $customer = new Customer();
        $customer->CustomerID = $record['CustomerID'];
        $customer->CompanyName = $record['CompanyName'];
        $customer->ContactName = $record['ContactName'];
        $customer->ContactTitle = $record['ContactTitle'];
        $customer->Phone = $record['Phone'];
        $customer->Fax = $record['Fax'];
        $customer->Address = new Address();
        $customer->Address->StreetName = ($record['Address']);
        $customer->Address->City = $record['City'];
        $customer->Address->Region = $record['Region'];
        $customer->Address->PostalCode = $record['PostalCode'];
        $customer->Address->Country = $record['Country'];
        //Set alternate address
        $customer->Address->AltAddress = new Address();
        $customer->Address->AltAddress->StreetName = 'ALT_' . $customer-
>Address->StreetName;
        $customer->Address->AltAddress->City = 'ALT_' . $customer->Address-
```

```
>City;
        $customer->Address->AltAddress->Region = 'ALT_' . $customer->Address-
>Region;
        $customer->Address->AltAddress->PostalCode = 'ALT_' . $customer-
>Address->PostalCode;
        $customer->Address->AltAddress->Country = 'ALT_' . $customer-
>Address->Country;
        $customer->EmailAddresses = array();
        for ($i = 1; $i < 4; $i++) {
            $customer->EmailAddresses[] = $customer->CustomerID . $i .
'@live.com';
        }

        $customer->OtherAddresses = array();
        for ($i = 0; $i < 2; $i++) {
            $customer->OtherAddresses[$i] = new Address();
            $this->_copyAddress($customer->Address, $customer-
>OtherAddresses[$i], $i + 1);
        }

        return $customer;
    }
//Note: Similarly we can serialize other resource sets from the database.
```

IF THERE ARE NO CUSTOMERS YOU SHOULD RETURN AN EMPTY ARRAY.getRelatedResourceSet (ResourceSet $sourceResourceSet, $sourceEntityInstance, ResourceSet $targetResourceSet, ResourceProperty $targetProperty,$FILTEROPTION = NULL, $SELECT=NULL, $ORDERBY=NULL, $TOP=NULL, $SKIP=NULL)

The implementation of this method should behave such a way that given the source resource (entity) set, source entity instance, target resource set which is having association with source resource set and name of the property in source resource representing target resource set, method should return the subset target entity set identified by the parameters.

Example: The query http://localhost/NorthWind.svc/Customers('ALFKI')/Orders will display the orders associated with the customer ALFKI. Then in the implementation of getRelatedResourceSet you do something like this:

```
public function  getRelatedResourceSet(ResourceSet
$sourceResourceSet,$sourceEntityInstance,
        ResourceSet $targetResourceSet,
        ResourceProperty $targetProperty,
        $filterOption = null, $select=null, $orderby=null, $top=null,
$skip=null
    ) {
        $result = array();
        $srcClass = get_class($sourceEntityInstance);
        $navigationPropName = $targetProperty->getName();
        if ($srcClass === 'Customer') {
            if ($navigationPropName === 'Orders') {
                $query = "SELECT * FROM Orders WHERE CustomerID =
'$sourceEntityInstance->CustomerID'";
                if ($filterOption != null) {
                    $query .= ' AND ' . $filterOption;
                }
                $stmt = sqlsrv_query($this->_connectionHandle, $query);
                if ($stmt === false) {
                     die(print_r(sqlsrv_errors(), true));
                }

                $result = $this->_serializeOrders($stmt);
            } else {
                die('Customer does not have navigation porperty with name: '
. $navigationPropName);
            }
        } else if ($srcClass === 'Order') {
            if ($navigationPropName === 'Order_Details') {
                $query = "SELECT * FROM [Order Details] WHERE OrderID =
$sourceEntityInstance->OrderID";
                if ($filterOption != null) {
                    $query .= ' AND ' . $filterOption;
                }
                $stmt = sqlsrv_query($this->_connectionHandle, $query);
                if ($stmt === false) {
                     die(print_r(sqlsrv_errors(), true));
                }

                $result = $this->_serializeOrderDetails($stmt);
            } else {
                die('Order does not have navigation porperty with name: ' .
$navigationPropName);
            }
        }
        return $result;
}
```

IDataServiceStreamProvider

If some stream data is associated to the entity and we want to expose this stream data to the end-user by OData then developer has to implement this interface.

Member Methods

Name	Parameters	Return Type	Description	Comments
getReadStream()	$entity (Type: Object of entity,MUST) $eTag (Type : String, MUST) $checkETagForEquality (Type : Boolean, MUST)	Stream	This method is invoked by the data services framework to retrieve the default stream associated with the entity instance specified by the entity parameter.	An implementer of this method MUST perform concurrency checks. If the concurrency check passes, this method should return the requested stream

	$operationContext (Type : Reference of the current context, MUST)			else should throw a DataServiceException. Null should never be returned from this method.
getStreamContentType()	$entity (Type : Object of entity, MUST) $operationContext (Type : Reference of the currentContext, MUST)	String	This method invoked by the data services framework to obtain the content type (media type) of the stream associated with the specified entity.	NIL
getStreamETag()	$entity (Type : Object of entity, MUST) $operationContext (Type : Reference of the currentContext, MUST)	String	This method invoked by the data services framework to obtain the ETag of the stream associated with the entity specified.	NIL
getReadStreamUri()	$entity (Type : Object of entity, MUST) $operationContext (Type : Reference of the currentContext, MUST)	array(ResourceType)	This method is invoked by the data services framework to obtain the URI clients should use when making retrieve (ie. GET) requests to the stream(ie. Media Resource).	NIL

Implementing IDataServiceStreamProvider

The first step in implementing the IDataServiceStreamProvider is to create a class that implements the interface and we have to define the image path for the services in this class.

Then we have to define the functions to get the eTag, Content-Type and stream associated to the entities

Then we have to define the methods declared in the interface:IDataServiceStreamProvider:

```
class NorthWindStreamProvider implements IDataServiceStreamProvider
{
    // NOTE: update this path as per your configuration
    const IMAGE_PATH_ROOT =
'D:\\Projects\\ODataPHPProducer\\services\\NorthWind\\images\\';

   /**
    * $eTag :  The etag value sent by the  client (as the value of an  If[-
None-]Match header) as part of the HTTP request,  This parameter will be
```

```
null if no If[-None-]Match  header was present.
    * $checkETagForEquality :  True if an value of the etag  parameter was
sent  to the server as the value  of an If-Match HTTP  request header,  False
if an value of the etag  parameter was sent to the  server as the the value
of an If-None-Match HTTP  request header null if  the HTTP request for the
stream was not a  conditional request.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getReadStream($entity, $eTag,
        $checkETagForEquality, $operationContext)
    {
        // NOTE: In this impementation we are not checking the eTag equality
        // We will return the stream irrespective of the whether the eTag
match of not
        if (!($entity instanceof Employee)) {
            throw new ODataException('Internal Server Error.', 500);
        }
        $filePath = self::IMAGE_PATH_ROOT . 'Employee_' . $entity->EmployeeID
            . '.jpg';
        if (file_exists($filePath)) {
            $handle = fopen($filePath, 'r');
            $stream = fread($handle, filesize($filePath));
            fclose($handle);
            return $stream;
        } else {
            throw new ODataException('The image file could not be
found',500);
        }
    }

   /**
    * $entity :  The entity instance associated with the stream for which the
content type is to be obtained.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getStreamContentType($entity, $operationContext)
    {
        if (!($entity instanceof Employee)) {
            throw new ODataException('Internal Server Error.', 500);
        }
        return 'image/jpeg';
    }

   /**
    * $entity :  The entity instance associated with the stream for which the
content type is to be obtained.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getStreamETag($entity, $operationContext)
    {
        if (!($entity instanceof Employee)) {
            throw new ODataException('Internal Server Error.', 500);
        }
        $lastModifiedTime = null;
        $filePath = self::IMAGE_PATH_ROOT . 'Employee_' . $entity->EmployeeID
. '.jpg';
        if (file_exists($filePath)) {
            $lastModifiedTime = date("\"m-d-Y H:i:s\"",filemtime($filePath));
        } else {
            return null;
        }
        return $lastModifiedTime;
    }

   /**
    * $entity :  The entity instance associated with the stream for which the
content type is to be obtained.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getReadStreamUri($entity, $operationContext)
    {
        //let library creates default media url.
        return null;
   }
}
```

IDataServiceStreamProvider2

Implementation of this interface is required for named stream or if multiple streams are associated to one entity and we want to get collection of streams based on the eTag.

Both of the stream provider interface IDataServiceStreamProvider or IDataServiceStreamProvider2 are mutually exclusive and we can implement only one interface at one time.

This interface extends the IDataServiceStreamProvider interface so if developer wants to implement this interface then he has to define those functions also which are declared in the IDataServiceStreamProvider interface.

Member methods

Name	Parameters	Return Type	Description	Comments
getReadStream2 ()	$entity (Type: Object of entity,MUST) $resourceStreamInfo (Type :ResourceStreamInfo ,MUST) $eTag (Type : String,	Stream	This method is invoked by the data services framework to retrieve the named stream associated with the entity instance specified by the entity parameter.	An implementer of this method MUST perform concurrency checks. If the concurrency check passes, this method should return the requested stream else should throw a DataServiceExceptio

	MUST) $checkETagForEquality (Type : Boolean, MUST) $operationContext (Type : Reference of the current context, MUST)			n. Null should never be returned from this method. If an error occurs while reading the stream, then the data services framework will generate an in-stream error.
getStreamContentType2()	$entity (Type : Object of entity, MUST) $resourceStreamInfo (Type :ResourceStreamInfo ,MUST) $operationContext (Type : Reference of the currentContext, MUST)	String	This method invoked by the data services framework to obtain the content type (media type) of the named stream associated with the specified entity.	NIL
getStreamETag2()	$entity (Type : Object of entity, MUST) $resourceStreamInfo (Type :ResourceStreamInfo ,MUST) $operationContext (Type : Reference of the currentContext, MUST)	String	This method invoked by the data services framework to obtain the ETag of the named stream associated with the entity specified.	NIL
getReadStreamUri2()	$entity (Type : Object of entity, MUST) $resourceStreamInfo (Type :ResourceStreamInfo ,MUST) $operationContext (	array(ResourceType)	This method is invoked by the data services framework to obtain the URI clients should use when making retrieve (ie. GET) requests to the stream(ie. Media Resource).	NIL

	Type : Reference of the currentContext, MUST)			

Implementing IDataServiceStreamProvider2

The first step in implementing the IDataServiceStreamProvider is to create a class that implements the interface. Then we have to define the functions declared in the interface IDataServiceStreamProvider and IDataServiceStreamProvider2.

```
class NorthWindStreamProvider implements IDataServiceStreamProvider2
{
   /**
    * $eTag :  The etag value sent by the  client (as the value of an  If[-
None-]Match header) as part of the HTTP request,  This parameter will be
null if no If[-None-]Match  header was present.
    * $resourceStreamInfo : The ResourceStreamInfo instance that describes
the named stream.
    * $checkETagForEquality :  True if an value of the etag  parameter was
sent  to the server as the value  of an If-Match HTTP  request header,  False
if an value of the etag  parameter was sent to the  server as the the value
of an If-None-Match HTTP  request header null if  the HTTP request for the
stream was not a  conditional request.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getReadStream2($entity, ResourceStreamInfo
$resourceStreamInfo, $eTag, $checkETagForEquality, $operationContext)
    {
        if (!($entity instanceof Employee)) {
            throw new ODataException('Internal Server Error.', 500);
        }
        $filePath = self::IMAGE_PATH_ROOT . 'Employee_' . $entity->EmployeeID
              . '_' . $resourceStreamInfo->getName() . '.png';
       if (file_exists($filePath)) {
           $handle = fopen($filePath, 'r');
           $stream = fread($handle, filesize($filePath));
           fclose($handle);
           return $stream;
       } else {
           throw new ODataException('The image file could not be found',
500);
       }
    }

   /**
    * $resourceStreamInfo : The ResourceStreamInfo instance that describes
the named stream.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getStreamContentType2($entity, ResourceStreamInfo
$resourceStreamInfo,$operationContext)
    {
        if (!($entity instanceof Employee)) {
            throw new ODataException('Internal Server Error.', 500);
        }
    return 'image/png';
    }

   /**
    * $resourceStreamInfo : The ResourceStreamInfo instance that describes
the named stream.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getStreamETag2($entity, ResourceStreamInfo
$resourceStreamInfo, $operationContext)
    {
        return null;
    }

   /**
    * $resourceStreamInfo : The ResourceStreamInfo instance that describes
the named stream.
    * $operationContext :  A reference to the context  for the current
operation.
    */
    public function getReadStreamUri2($entity, ResourceStreamInfo
$resourceStreamInfo,$operationContext)
    {
        return null;
    }
}
```

IDataServiceStreamProvider2 extends the IDataServiceStreamProvider so we have to define methods of the IDataServiceStreamProvider interface also as explained in the section:

IServiceProvider

IServiceProvider is the mechanism for retrieving a service object. End-Developer has to write a new class for each different data service and this class should implement this interface.

Member Methods

Name	Parameters	Return Type	Description
getService()	$serviceType(Type:String, MUST)	Reference of MetadataProvider, QueryProvider,StreamProvider or NULL if parameter is invalid	It provides the instance of MetadataProvider, QueryProvider or StreamProviders.

Implementing IServiceProvider

The first step in implementing the IServiceProvider is to create a class that implements the interface.

Note : This class should extend the DataService class also.

Then we have to define getService method of the IserviceProvider interface in class NorthWindDataService which should return the instance of MetadataProvider, QueryProvider or StreamProvider and we can also define initializeService function in class NorthWindDataService to set the service level configuration.

```
class NorthWindDataService extends DataService implements IServiceProvider

{
    public function getService($serviceType)
    {
        if ($serviceType === 'IDataServiceMetadataProvider') {
            if (is_null($this->_northWindMetadata)) {
                $this->_northWindMetadata =
CreateNorthWindMetadata::create();
            }
            return $this->_northWindMetadata;
        } else if ($serviceType === 'IDataServiceQueryProvider') {
            if (is_null($this->_northWindQueryProvider)) {
                $this->_northWindQueryProvider = new
NorthWindQueryProvider();
            }
            return $this->_northWindQueryProvider;
        } else if ($serviceType === 'IDataServiceStreamProvider') {
            return new NorthWindStreamProvider();
        }
        return null;
    }
}
```

If we implemented IDataServiceQueryProvider2 then few changes are required in the above implementation:

```
    /**
     * Get the service like IDataServiceMetadataProvider,
IDataServiceQueryProvider,
     * IDataServiceStreamProvider
     *
     * @param String $serviceType Type of service
IDataServiceMetadataProvider,
     *                            IDataServiceQueryProvider,
     *                            IDataServiceStreamProvider
     *
     * @see
library/ODataProducer/ODataProducer.IServiceProvider::getService()
     * @return object
     */
    public function getService($serviceType)
    {
      if(($serviceType === 'IDataServiceMetadataProvider') ||
            ($serviceType === 'IDataServiceQueryProvider2') ||
            ($serviceType === 'IDataServiceStreamProvider')) {
            if (is_null($this->_northWindMetadata )){
```

```
                        $this->_northWindExpressionProvider = new
NorthWindDSExpressionProvider();
            }
      }
        if ($serviceType === 'IDataServiceMetadataProvider') {
            if (is_null($this->_northWindMetadata)) {
                $this->_northWindMetadata =
CreateNorthWindMetadata::create();
                $this->_northWindMetadata->mappedDetails =
CreateNorthWindMetadata::mappingInitialize();
            }
            return $this->_northWindMetadata;
        } else if ($serviceType === 'IDataServiceQueryProvider2') {
            if (is_null($this->_northWindQueryProvider)) {
                $this->_northWindQueryProvider = new
WordPressQueryProvider();
            }
            return $this->_northWindQueryProvider;
        } else if ($serviceType === 'IDataServiceStreamProvider') {
            return new NorthWindStreamProvider();
        }
        return null;
    }
```

IExpressionProvider

Implementation of IExpressionProvider is require only for IDataServiceQueryProvider2 because library needs one expression for the filtering criteria so that library can use that expression in the SQL-Query for the underlying DB.

Implementation of this interface depends on the underlying DB and its implementation may vary from one DB to another DB.

Actually library parses the complete URL and saves in different kind of data-structures and one is Expression-Tree. This interface implementation is require to traversal that expression –tree so that library can generate an expression which should be compatible to the underlying data source so that library can use that newly generated expression in a SQL query to populate an optimized query so that library don’t need to handle the filtering from itself.

Member methods

Name	Parameters	Return Type	Description	Comments
getIteratorName ()	Void	String	Get the name of the iterator.	Infect this method is not useful for explicit implementation. Library has one default implementation of

				IExpressionProvider and that method is require only for that default implementation. Hence user has to declare the function prototype only and no need to define any step inside this function.
onLogicalExpression()	$expressionType (Type : String, MUST) $left (Type :String ,MUST) $right (Type : String, MUST)	String	Call back for logical expression Generate binary expression for logical operators	Implementation of all these function would be almost same for each data-source. These functions basically replaces OData operator by corresponding operator available in data-source so we just need to make sure that the constants which we defined in **Step-2** are proper for underlying DB then function definition would be same for each data-source.
onArithmaticExpression()	$expressionType (Type : String, MUST) $left (Type :String ,MUST) $right (Type : String, MUST)	String	Call back for arithmetic expression Generate binary expression for arithmetic operators	
onRelationalExpression()	$expressionType (Type : String, MUST) $left (Type :String ,MUST) $right (Type : String, MUST)	String	Call back for relational expression Generate binary expression for relational operators	
onUnaryExpression()	$expressionType (Type : String, MUST) $child (Type :String ,MUST)	String	Call back for unary expression Generate expression for unary operators	
onConstantExpression()	$type (Type: IType object,MUST)	String	Call back for constant expression	This function definition also would be same for each data source.

	$value (Type: String,MUST)		Returns the constants value	
onPropertyAccessExpression	$expression(Type: String,MUST)	String	Call back for property access expression	NIL
onFunctionCallExpression()	$functioanDescription (Type : Object of Function-Description class,MUST) $params(Type: array of operands,MUST)	String	Call back for function call expression.	Implementation of this function will vary for each different data source. This function will return corresponding function-call string (available in underlying DB) for each OData function call.

Implementing IExpressionProvider

The first step in implementing the IExpressionProvider is to create a class that implements the interface.

```
class NorthWindDSExpressionProvider implements IExpressionProvider
{
```

Second step is to define few constants in this class,one constant for each kind of operator which is available in the underlying DB:

```
    const ADD                   = '+';
    const CLOSE_BRACKET         = ')';
    const COMMA                 = ',';
    const DIVIDE                = '/';
    const SUBTRACT              = '-';
    const EQUAL                 = '=';
    const GREATERTHAN           = '>';
    const GREATERTHAN_OR_EQUAL  = '>=';
    const LESSTHAN              = '<';
    const LESSTHAN_OR_EQUAL     = '<=';
    const LOGICAL_AND           = 'AND';
    const LOGICAL_NOT           = '!';
    const LOGICAL_OR            = 'OR';
    const MEMBERACCESS          = '';
    const MODULO                = '%';
    const MULTIPLY              = '*';
    const NEGATE                = '-';
    const NOTEQUAL              = '!=';
    const OPEN_BRAKET           = '(';
```

Then we have to define the methods declared in the interface:IExpressionProvider:

```
    /**
     * Constructs new instance of MySQLExpressionProvider
     *
     * @param string $iterName The name of the iterator
     */
    public function __construct()
    {
        //$this->_iterName = $iterName;
    }

    /**
     * Get the name of the iterator
     *
     * @return string
     */
    public function getIteratorName()
    {
        //return $this->_iterName;
    }

    /**
     * Call-back for constant expression
     *
     * @param IType  $type  The type of constant
     * @param objetc $value The value of the constant
     *
     * @return string
     */
    public function onConstantExpression(IType $type, $value)
    {
        return $value;
    }
```

onLogicalExpression($expressionType, $left, $right)

```
    /**
     * Call-back for logical expression
     *
     * @param ExpressionType $expressionType The type of logical expression.
     * @param string         $left           The left expression.
     * @param string         $right          The left expression.
     *
     * @return string
     */
    public function onLogicalExpression($expressionType, $left, $right)
    {
        switch($expressionType) {
        case ExpressionType::AND_LOGICAL:
            return $this->_prepareBinaryExpression(self::LOGICAL_AND, $left,
$right);
            break;
        case ExpressionType::OR_LOGICAL:
            return $this->_prepareBinaryExpression(self::LOGICAL_OR, $left,
$right);
            break;
        default:
            throw new \InvalidArgumentException('onLogicalExpression');
        }
    }
```

This function replaces all the logical operators (used in the filtering criteria) by corresponding logical operators available in the underlying data-source.

onArithmeticExpression($expressionType, $left, $right)

This function replaces all the arithmetic operators(used in the filtering criteria) by corresponding arithmetic operators available in the underlying data-source.

```
    /**
     * Call-back for arithmetic expression
     *
     * @param ExpressionType $expressionType The type of arithmetic
expression.
     * @param string         $left           The left expression.
     * @param string         $right          The left expression.
     *
     * @return string
     */
    public function onArithmeticExpression($expressionType, $left, $right)
    {
        switch($expressionType) {
        case ExpressionType::MULTIPLY:
            return $this->_prepareBinaryExpression(self::MULTIPLY, $left,
$right);
            break;
        case ExpressionType::DIVIDE:
            return $this->_prepareBinaryExpression(self::DIVIDE, $left,
$right);
            break;
        case ExpressionType::MODULO:
            return $this->_prepareBinaryExpression(self::MODULO, $left,
$right);
            break;
        case ExpressionType::ADD:
            return $this->_prepareBinaryExpression(self::ADD, $left, $right);
            break;
        case ExpressionType::SUBTRACT:
            return $this->_prepareBinaryExpression(self::SUBTRACT, $left,
$right);
            break;
        default:
            throw new \InvalidArgumentException('onArithmeticExpression');
        }
    }
```

onRelationalExpression($expressionType, $left, $right)

This function replaces all the relational operators (used in the filtering criteria) by corresponding relational operators available in the underlying data-source.

```
    /**
     * Call-back for relational expression
     *
     * @param ExpressionType $expressionType The type of relation expression
     * @param string         $left           The left expression
     * @param string         $right          The left expression
     *
     * @return string
     */
    public function onRelationalExpression($expressionType, $left, $right)
    {
        switch($expressionType) {
        case ExpressionType::GREATERTHAN:
            return $this->_prepareBinaryExpression(self::GREATERTHAN, $left,
$right);
            break;
        case ExpressionType::GREATERTHAN_OR_EQUAL:
            return $this->_prepareBinaryExpression(
                self::GREATERTHAN_OR_EQUAL, $left, $right
            );
            break;
        case ExpressionType::LESSTHAN:
            return $this->_prepareBinaryExpression(self::LESSTHAN, $left,
$right);
            break;
        case ExpressionType::LESSTHAN_OR_EQUAL:
            return $this->_prepareBinaryExpression(
                self::LESSTHAN_OR_EQUAL, $left, $right
            );
            break;
        case ExpressionType::EQUAL:
            return $this->_prepareBinaryExpression(self::EQUAL, $left,
$right);
            break;
        case ExpressionType::NOTEQUAL:
            return $this->_prepareBinaryExpression(self::NOTEQUAL, $left,
$right);
            break;
        default:
            throw new \InvalidArgumentException('onArithmeticExpression');
        }
    }
```

onUnaryExpression(iType $type, $child)

```
    /**
     * Call-back for unary expression
     *
     * @param ExpressionType $expressionType The type of unary expression
     * @param string         $child          The child expression
     *
     * @return string
     */
    public function onUnaryExpression($expressionType, $child)
    {
        switch($expressionType) {
        case ExpressionType::NEGATE:
            return $this->_prepareUnaryExpression(self::NEGATE, $child);
            break;
        case ExpressionType::NOT_LOGICAL:
            return $this->_prepareUnaryExpression(self::LOGICAL_NOT, $child);
            break;
        default:
            throw new \InvalidArgumentException('onUnaryExpression');
        }
    }
```

This function replaces all the unary operators (used in the filtering criteria) by corresponding unary operators available in the underlying data-source.

onPropertyAccessExpression($expression)

```
    /**
     * Call-back for property access expression
     *
     * @param PropertyAccessExpression $expression The property access
expression
     *
     * @return string
     */
    public function onPropertyAccessExpression($expression)
    {
        $parent = $expression;
        $variable = null;

        do {
            $variable
                = $parent->getResourceProperty()->getName()
                . self::MEMBERACCESS
                . $variable;
            $parent = $parent->getParent();
        } while ($parent != null);

        $variable = rtrim($variable, self::MEMBERACCESS);
        $variable = $this->getIteratorName() . self::MEMBERACCESS .
$variable;
        return $variable;
    }
```

onFunctionCallExpression($expression)

This function replaces all the OData function calls in filtering criteria by their corresponding functions available in the underlying data-source.

This function depends on the underlying DB so its implementation for one data source may be different than the implementation for other data source.

Implementation of onFunctionCallExpression for MySQL

```
    /**
     * Call-back for function call expression
     *
     * @param FunctionDescription $functionDescription Description of the
function.
     * @param array<string>       $params              Paameters to the
function.
     *
     * @return string
     */
    public function onFunctionCallExpression($functionDescription, $params)
    {
        switch($functionDescription->functionName) {
        case ODataConstants::STRFUN_COMPARE:
            return "STRCMP($params[0], $params[1])";
            break;
        case ODataConstants::STRFUN_ENDSWITH:
            return "(STRCMP($params[1],RIGHT($params[0],LENGTH($params[1])))
= 0)";
            break;
        case ODataConstants::STRFUN_INDEXOF:
            return "INSTR($params[0], $params[1])";
            break;
        case ODataConstants::STRFUN_REPLACE:
            return "REPLACE($params[0],$params[1],$params[2])";
            break;
        case ODataConstants::STRFUN_STARTSWITH:
            return "(STRCMP($params[1],LEFT($params[0],LENGTH($params[1]))) =
0)";
            break;
        case ODataConstants::STRFUN_TOLOWER:
            return "LOWER($params[0])";
            break;
        case ODataConstants::STRFUN_TOUPPER:
            return "UPPER($params[0])";
            break;
        case ODataConstants::STRFUN_TRIM:
            return "TRIM($params[0])";
            break;
        case ODataConstants::STRFUN_SUBSTRING:
            return count($params) == 3 ?
                "SUBSTRING($params[0], $params[1], $params[2])" :
                "SUBSTRING($params[0], $params[1])";
            break;
        case ODataConstants::STRFUN_SUBSTRINGOF:
            return "(LOCATE($params[0], $params[1]) > 0)";
            break;
        case ODataConstants::STRFUN_CONCAT:
            return CONCAT($params[0],$params[1]);
            break;
        case ODataConstants::STRFUN_LENGTH:
            return "LENGTH($params[0])";
            break;
        case ODataConstants::GUIDFUN_EQUAL:
            return self::TYPE_NAMESPACE . "Guid::guidEqual($params[0],
$params[1])";
```

```
            break;
        case ODataConstants::DATETIME_COMPARE:
            return
                self::TYPE_NAMESPACE
                    . "DateTime::dateTimeCmp($params[0], $params[1])";
            break;
        case ODataConstants::DATETIME_YEAR:
            return "EXTRACT(YEAR from ".$params[0].")";
            break;
        case ODataConstants::DATETIME_MONTH:
            return "EXTRACT(MONTH from ".$params[0].")";
            break;
        case ODataConstants::DATETIME_DAY:
            return "EXTRACT(DAY from ".$params[0].")";
            break;
        case ODataConstants::DATETIME_HOUR:
            return "EXTRACT(HOUR from ".$params[0].")";
            break;
        case ODataConstants::DATETIME_MINUTE:
            return "EXTRACT(MINUTE from ".$params[0].")";
            break;
        case ODataConstants::DATETIME_SECOND:
            return "EXTRACT(SECOND from ".$params[0].")";
            break;
        case ODataConstants::MATHFUN_ROUND:
            return "ROUND($params[0])";
            break;
        case ODataConstants::MATHFUN_CEILING:
            return "CEIL($params[0])";
            break;
        case ODataConstants::MATHFUN_FLOOR:
            return "FLOOR($params[0])";
            break;
        case ODataConstants::BINFUL_EQUAL:
            return
                self::TYPE_NAMESPACE
                    . "Binary::binaryEqual($params[0], $params[1])";
            break;
        case 'is_null':
            return "is_null($params[0])";
            break;

        default:
            throw new \InvalidArgumentException('onFunctionCallExpression');
        }
    }
```

At the end, we have to define following two methods to generate an expression with using of operands and operator.

```
    /**
     * To format binary expression
     *
     * @param string $operator The binary operator.
     * @param string $left     The left operand.
     * @param string $right    The right operand.
     *
     * @return string
     */
    private function _prepareBinaryExpression($operator, $left, $right)
    {
        return self::OPEN_BRAKET . $left . ' ' . $operator . ' ' . $right .
self::CLOSE_BRACKET;
    }

    /**
     * To format unary expression
     *
     * @param string $operator The unary operator.
     * @param string $child    The operand.
     *
     * @return string
     */
    private function _prepareUnaryExpression($operator, $child)
    {
        return $operator . self::OPEN_BRAKET . $child . self::CLOSE_BRACKET;
    }
```

Implementation of onFunctionCallExpression for MS-SQL server

```
    /**
     * Call-back for function call expression
     *
     * @param FunctionDescription $functionDescription Description of the
function.
     * @param array<string>       $params              Paameters to the
function.
     *
     * @return string
     */
    public function onFunctionCallExpression($functionDescription, $params)
    {
        switch($functionDescription->functionName) {
        case ODataConstants::STRFUN_COMPARE:
            return "STRCMP($params[0]; $params[1])";
            break;
        case ODataConstants::STRFUN_ENDSWITH:
            return "STRCMP($params[1]; RIGHT($params[0], LEN($params[1])))";
            break;
        case ODataConstants::STRFUN_INDEXOF:
            return "CHARINDEX($params[1], $params[0])";
            break;
        case ODataConstants::STRFUN_REPLACE:
            return "REPLACE($params[0],$params[1],$params[2])";
            break;
        case ODataConstants::STRFUN_STARTSWITH:
            return "STRCMP($params[1]; LEFT($params[0],LEN($params[1])))";
            break;
        case ODataConstants::STRFUN_TOLOWER:
            return "LOWER($params[0])";
            break;
        case ODataConstants::STRFUN_TOUPPER:
            return "UPPER($params[0])";
            break;
        case ODataConstants::STRFUN_TRIM:
            return "TRIM($params[0])";
            break;
        case ODataConstants::STRFUN_SUBSTRING:
            if (count($params) == 3) {
                return "SUBSTRING($params[0], $params[1]+1, $params[2])";
            } else {
                throw new
\InvalidArgumentException("onFunctionCallExpression_Substring_needs_3_Paramet
ers");
            }
            break;
        case ODataConstants::STRFUN_SUBSTRINGOF:
            return "PATINDEX('%'+$params[0]+'%', $params[1])";
            break;
        case ODataConstants::STRFUN_CONCAT:
            return "$params[0] + $params[1]";
            break;
        case ODataConstants::STRFUN_LENGTH:
            return "LEN($params[0])";
            break;
        case ODataConstants::GUIDFUN_EQUAL:
            return self::TYPE_NAMESPACE . "Guid::guidEqual($params[0],
$params[1])";
            break;
        case ODataConstants::DATETIME_COMPARE:
            return
                self::TYPE_NAMESPACE
                    . "DateTime::dateTimeCmp($params[0], $params[1])";
            break;
        case ODataConstants::DATETIME_YEAR:
            return "YEAR($params[0])";
            break;
        case ODataConstants::DATETIME_MONTH:
            return "MONTH($params[0])";
            break;
        case ODataConstants::DATETIME_DAY:
            return "DAY( $params[0] )";
            break;
        case ODataConstants::DATETIME_HOUR:
            return "DATENAME(HOUR, $params[0])";
            break;
        case ODataConstants::DATETIME_MINUTE:
            return "DATENAME(MINUTE, $params[0])";
            break;
        case ODataConstants::DATETIME_SECOND:
            return "DATENAME(SECOND, $params[0])";
            break;
        case ODataConstants::MATHFUN_ROUND:
            return "ROUND($params[0])";
            break;
        case ODataConstants::MATHFUN_CEILING:
            return "CEIL($params[0])";
            break;
        case ODataConstants::MATHFUN_FLOOR:
            return "FLOOR($params[0])";
            break;
        case ODataConstants::BINFUL_EQUAL:
            return
                self::TYPE_NAMESPACE
                    . "Binary::binaryEqual($params[0], $params[1])";
            break;
        case 'is_null':
            return "is_null($params[0])";
            break;
        default:
            throw new \InvalidArgumentException('onFunctionCallExpression');
        }
}
```

OData library expects that all kind of string comparisons should be returned in form of STRCMP(PARAM1,PARAM2) and if string comparison is possible by another way like by using '=' operator then also we have to return in the same fashion and we have to manipulate _prepareBinayExpression() to handle the string comparison as per the underlying data source.

Above implementation for MSSQL-Server shows it as MS-SQL server supports string comparison directly even though we are returning STRCMP function call.

```
        switch($functionDescription->functionName) {
        case ODataConstants::STRFUN_COMPARE:
            return "STRCMP($params[0]; $params[1])";
            break;
        case ODataConstants::STRFUN_ENDSWITH:
            return "STRCMP($params[1]; RIGHT($params[0], LEN($params[1])))";
            break;
        case ODataConstants::STRFUN_STARTSWITH:
            return "STRCMP($params[1]; LEFT($params[0],LEN($params[1])))";
            break;
```

So as I explained that we have to handle string comparison separately for MSSQL-Server hence we have to define one more methods in NorthWindDSExpressionProvider.php.

```
    /**
     * To format binary expression
     *
     * @param string $operator The binary operator.
     * @param string $left     The left operand.
     * @param string $right    The right operand.
     *
     * @return string
     */
    private function _prepareBinaryExpression($operator, $left, $right)
    {
      if (!substr_compare($left, "STRCMP", 0, 6)) {
            $str = explode(';', $left, 2);
            $str[0] = str_replace('STRCMP', '', $str[0]);
            if ($right == 'false' and $right != '0') {
                  if (!substr_compare($operator, '!', 0, 1)) {
                      $operator = str_replace('!', '', $operator);
                  } else if ($operator == '>=') {
                    $operator = '<';
                  } else if ($operator == '<=') {
                    $operator = '>';
                  } else {
                    $operator = "!".$operator;
                  }
                  return self::OPEN_BRAKET
                    . $str[0] . ' ' . $operator
                    . ' ' . $str[1] . self::CLOSE_BRACKET;
            } else {
                return self::OPEN_BRAKET
                    . $str[0] . ' ' . $operator
                    . ' ' . $str[1] . self::CLOSE_BRACKET;
            }
        }
        //In MSSQL Server Index starts from 1 instead of 0.
        if (!substr_compare($left, "CHARINDEX", 0, 9)) {
            $right++;
        }
        if (!substr_compare($left, "PATINDEX", 0, 8)) {
            if ($right == 'true') {
                if (!substr_compare($operator, '!', 0, 1)) {
                    $operator = str_replace('!', '', $operator);
                  } else if ($operator == '>=') {
                    $operator = '<';
                  } else if ($operator == '<=') {
                    $operator = '>';
                  } else {
                    $operator = "!".$operator;
                  }
            }
            $right = '0';
        }
        return
            self::OPEN_BRAKET . $left . ' ' . $operator . ' ' . $right .
self::CLOSE_BRACKET;
    }

    /**
     * To format unary expression
     *
     * @param string $operator The unary operator.
     * @param string $child    The operand.
     *
     * @return string
     */
    private function _prepareUnaryExpression($operator, $child)
    {
        return $operator . self::OPEN_BRAKET . $child . self::CLOSE_BRACKET;
    }
```

Metadata Mapping

If we implemented IDataServiceQueryProvider2 and property names defined in the Metadata are not matching with the names defined in the underlying data source then we have to define one more method CreateNorthWindMetaData::mappingInitialize() because library has to generate an expression for filtering criteria and this expression will contain few conditions and library has to replace all the property-names appearing in the expression by their actual name which are existing in the DB.

CreateNorthWindMetaData::mappingInitialize() is not required in multiple senses:

- If we are not implementing IDataServiceQueryProvider2.
- If we are implementing IDataServiceQueryProvider2 but the property and entity names defined in the metadata are same as defined in the data-source and we are not using any name alias in our queries.

So first end-developer has to define the mapping first and then library uses the defined mapping info for generating the proper expression and append it to the SQL query before firing the query.

We have to use object of MetadataMapping class to define the mapping between properties names defined in the metadata and column names which are already existing in the DB.

Member methods

This class has few functions but only 2 functions are required for defining the metadata mapping. Other functions are required for library to extract the mapping info defined by the end-developer.

Name	Parameters	Return Type	Description	Comments
mapEntity()	$entityName(resource-set name defined in the metadata definition.Type :String,MUST) $mappedEntityName(Table-name defined in the DB schema for that resource-set,Type:String,MUST)	String	This function maps the resource set name which is defined in the metadata to the table name which is physically existing in the DB.	NIL
mapProperty()	$entityName(resource-set name defined in the metadata definition.Type :String,MUST) $metaPropertyName(Entity name defined in the metadata,Type:String,MUST) $dsPropertyName(Corresponding	String	This function maps the property name which is defined in the metadata for one particular resource to the fields name which is physically existing in the DB.	NIL

	field name defined in the DB schema,Type:String,MUST)			

We have one sample implementation of MetadataMapping for WordPress service because its implementation is not required for NorthWind DB because names defined in the Metada for northwind service are same as defined in the database schema.

```
    public static function mappingInitialize() {
      $mappedDetails = new MetadataMapping();
      $mappedDetails->mapEntity('Posts', 'wp_posts');
      $mappedDetails->mapProperty('Posts', 'PostID', 'ID');
      $mappedDetails->mapProperty('Posts', 'Author', 'post_author');
      $mappedDetails->mapProperty('Posts', 'Date', 'post_date');
      $mappedDetails->mapProperty('Posts', 'DateGmt', 'post_date_gmt');
      $mappedDetails->mapProperty('Posts', 'Content', 'post_content');
      $mappedDetails->mapProperty('Posts', 'Title', 'post_title');
      $mappedDetails->mapProperty('Posts', 'Excerpt', 'post_excerpt');
      $mappedDetails->mapProperty('Posts', 'Status', 'post_status');
      $mappedDetails->mapProperty('Posts', 'CommentStatus',
'comment_status');
      $mappedDetails->mapProperty('Posts', 'PingStatus', 'ping_status');
      $mappedDetails->mapProperty('Posts', 'Password', 'post_password');
      $mappedDetails->mapProperty('Posts', 'Name', 'post_name');
      $mappedDetails->mapProperty('Posts', 'ToPing', 'to_ping');
      $mappedDetails->mapProperty('Posts', 'Pinged', 'pinged');
        $mappedDetails->mapProperty('Posts', 'ModifiedGmt',
'post_modified_gmt');
        $mappedDetails->mapProperty('Posts', 'ContentFiltered',
'post_content_filtered');
        $mappedDetails->mapProperty('Posts', 'ParentID', 'post_parent');
        $mappedDetails->mapProperty('Posts', 'Guid', 'guid');
        $mappedDetails->mapProperty('Posts', 'MenuOrder', 'menu_order');
        $mappedDetails->mapProperty('Posts', 'Type', 'post_type');
        $mappedDetails->mapProperty('Posts', 'MimeType', 'post_mime_type');
        $mappedDetails->mapProperty('Posts', 'CommentCount',
'comment_count');

      $mappedDetails->mapEntity('Tags', 'wp_terms');
      $mappedDetails->mapProperty('Tags', 'TagID', 't.term_id');
      $mappedDetails->mapProperty('Tags', 'Name', 't.name');
      $mappedDetails->mapProperty('Tags', 'Slug', 't.slug');
      $mappedDetails->mapProperty('Tags', 'Description', 'tt.description');

      $mappedDetails->mapEntity('Categories', 'wp_terms');
      $mappedDetails->mapProperty('Categories', 'CategoryID', 't.term_id');
      $mappedDetails->mapProperty('Categories', 'Name', 't.name');
      $mappedDetails->mapProperty('Categories', 'Slug', 't.slug');
      $mappedDetails->mapProperty('Categories', 'Description',
'tt.description');

      $mappedDetails->mapEntity('Comments', 'wp_comments');
      $mappedDetails->mapProperty('Comments', 'CommentID', 'comment_id');
      $mappedDetails->mapProperty('Comments', 'PostID', 'comment_post_id');
      $mappedDetails->mapProperty('Comments', 'Author', 'comment_author');
      $mappedDetails->mapProperty('Comments', 'AuthorEmail',
'comment_author');
      $mappedDetails->mapProperty('Comments', 'AuthorUrl',
'comment_author_url');
      $mappedDetails->mapProperty('Comments', 'AuthorIp',
'comment_author_email');
      $mappedDetails->mapProperty('Comments', 'DateGmt', 'comment_date');
      $mappedDetails->mapProperty('Comments', 'Content', 'comment_content');
      $mappedDetails->mapProperty('Comments', 'Karma', 'comment_karma');
      $mappedDetails->mapProperty('Comments', 'Approved',
'comment_approved');
      $mappedDetails->mapProperty('Comments', 'Agent', 'comment_agent');
      $mappedDetails->mapProperty('Comments', 'Type', 'comment_type');
      $mappedDetails->mapProperty('Comments', 'ParentID', 'comment_parent');
      $mappedDetails->mapProperty('Comments', 'UserID', 'user_id');


      $mappedDetails->mapEntity('Users', 'wp_users');
      $mappedDetails->mapProperty('Users', 'UserID', 'ID');
      $mappedDetails->mapProperty('Users', 'Login', 'user_login');
      $mappedDetails->mapProperty('Users', 'Nicename', 'user_nicename');
      $mappedDetails->mapProperty('Users', 'Email', 'user_email');
      $mappedDetails->mapProperty('Users', 'Url', 'user_url');
      $mappedDetails->mapProperty('Users', 'Registered', 'user_registered');
      $mappedDetails->mapProperty('Users', 'Status', 'user_status');
      $mappedDetails->mapProperty('Users', 'DisplayName', 'display_name');
      return $mappedDetails;
    }
```

Above example shows that if we are using Join for any resource-set or using table-name alias in our query then also we can define this by metadata mapping .

```
      $mappedDetails->mapEntity('Tags', 'wp_terms');
      $mappedDetails->mapProperty('Tags', 'TagID', 't.term_id');
      $mappedDetails->mapProperty('Tags', 'Name', 't.name');
      $mappedDetails->mapProperty('Tags', 'Slug', 't.slug');
      $mappedDetails->mapProperty('Tags', 'Description', 'tt.description');
```

We have to call this method after creating the instance of metadata provider.We have implementation of IServiceProvider i.e. NorthWindDataService.php and getService() will call this method.

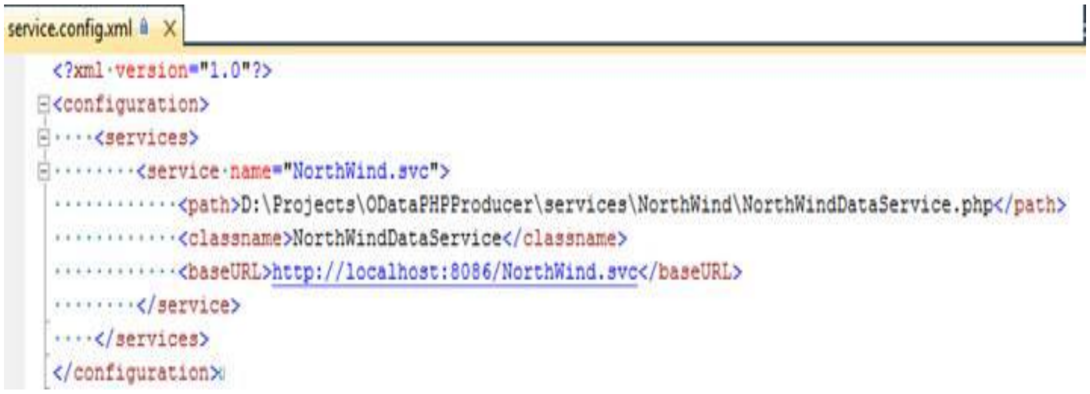
```
    public function getService($serviceType)
    {
      if(($serviceType === 'IDataServiceMetadataProvider') ||
            ($serviceType === 'IDataServiceQueryProvider2') ||
            ($serviceType === 'IDataServiceStreamProvider')) {
            if (is_null($this->_wordPressExpressionProvider)) {
                  $this->_wordPressExpressionProvider = new
WordPressDSExpressionProvider();
            }
      }
        if ($serviceType === 'IDataServiceMetadataProvider') {
            if (is_null($this->_wordPressMetadata)) {
                $this->_wordPressMetadata =
CreateWordPressMetadata::create();
```

```
                $this->_wordPressMetadata->mappedDetails =
CreateWordPressMetadata::mappingInitialize();
            }
            return $this->_wordPressMetadata;
        }
```

Changes in the service.config FILE

We have to modify the “Services\service.config.xml” file to register the service with the OData Producer for PHP so that our library can generate the OData feeds for the specified service.

Make sure that <path> and <baseURL> tag contains proper value in the configuration file (D:\Projects\ODataPHPProducer\Services\service.config.xml) :



```
<?xml version="1.0"?>
<configuration>
    <services>
        <service name="NorthWind.svc">
            <path>D:\Projects\ODataPHPProducer\services\NorthWind\NorthWindDataService.php</path>
            <classname>NorthWindDataService</classname>
            <baseURL>http://localhost:8086/NorthWind.svc</baseURL>
        </service>
    </services>
</configuration>
```

i. **<path>** : The absolute/relative path of the ‘NorthWindDataService.php’

ii. **<baseURL>** : The base url to the web site you configured in step 3 followed by ‘NorthWind.svc.

Make sure that the providers which we are creating for our service

We can specify the relative path also in the configuration file.

```
<?xml version="1.0"?>
<configuration>
    <services>
        <service name="NorthWind.svc">
            <path>\services\NorthWind\NorthWindDataService.php</path>
            <classname>NorthWindDataService</classname>
            <baseURL>http://localhost:8086/NorthWind.svc</baseURL>
        </service>
        <service name="WordPress.svc">
          <path>\services\WordPress\WordPressDataService.php</path>
          <classname>WordPressDataService</classname>
          <baseURL>http://localhost:8086/WordPress.svc</baseURL>
      </service>
    </services>
</configuration>
```

Configuring the OData Service Parameter

Developer can define his service configuration within the initializeService method of data service class. Implementation of IserviceProvider will contain definition of InitializeService method.

This method will be invoked after the dispatcher creates an instance of data service class. This method also accept one parameter of type DataServiceConfiguration, developer can use this parameter to configure the service.

This section will explain about the DataServiceConfiguration class which is implementing IDataServiceConfiguration interface.

Member Methods

Name	Parameters	Return Type	Description
getMaxExpandCount()	Void	Integer	Gets maximum number of segments to be expanded allowed in a request.
setMaxExpandCount()	$maxExpandCount (Type : Integer, MUST)	Void	Sets maximum number of segments to be expanded allowed in a request.
getMaxExpandDepth()	Void	Integer	Gets the maximum number of segments in a single $expand path.
setMaxExpandDepth()	$maxExpandDepth (Type : Integer, MUST)	Void	Sets the maximum number of segments in a single $expand path.
getMaxResultsPerCollection()	Void	Integer	Gets maximum number of elements in each returned collection.
setMaxresultsPerCollection()	$maxResultPerCollection(Type : Integer,MUST)	Void	Sets maximum number of elements in each returned collection .
getUseVerboseErrors()	Void	Boolean	Gets whether verbose errors should be used by default.
setUseVerboseErrors()	$useVerboseError(Type: Boolean,MUST)	Void	Sets whether verbose errors should be used by default.

getEntitySetAccessRule()	$resourceSet(Type: , MUST)	EntitySetRights	Gets the access rights on the specified resource set.
setEntitySetAccessRule()	$name(Type:String, MUST) $rights(Type: EntitySetRights, MUST)	Void	Sets the access rights on the specified resource set.
getEntitySetPageSize()	$resourceSet(Type: Object, MUST)	Integer	Gets the maximum page size for an entity set resource.
setEntitySetPageSize()	$name(Type:String, MUST) $pageSize(Type: Integer,MUST)	Void	Sets the maximum page size for an entity set resource.
getAcceptCountRequests()	Void	Boolean	Gets whether requests with the $count path segment or the $inlinecount query options are accepted.
setAcceptCountRequests()	$acceptCountRequest(Type: Boolean,MUST)	Void	Sets whether requests with the $count path segment or the $inlinecount query options are accepted.
getAcceptProjectionRequests()	Void	Boolean	Gets whether projection requests ($select) should be accepted.
setAcceptProjectionRequests()	$acceptProjectionRequest(Type: Boolean,MUST)	Void	Sets whether projection requests ($select) should be accepted.
getMaxDataServiceVersion()	Void	Integer	Gets maximum version of the response sent by server.

getMaxDataServiceVersionObject()	Void	Object of Version class	Gets Maximum version of the response sent by server.
setMaxDataServiceVersion()	Integer	Void	Sets maximum version of the response sent by server

How to set configuration parameters - An Example

This section shows examples that how we can set configuration parameters for a service.

```
//This method is called only once to initialize service-wide policies
public static function initializeService(DataServiceConfiguration &$config)
{
    $config->setEntitySetPageSize('*', 5); //Configure the page size
    $config->setEntitySetAccessRule('*', EntitySetRights::ALL);
    $config->setAcceptCountRequests(true);
    $config->setAcceptProjectionRequests(true);
    //Configure the maxDataServiceVesion header value which should be take
care to generate the response
    $config->setMaxDataServiceVersion(DataServiceProtocolVersion::V3);
}
```

This function we need to define in the implementation of the IserviceProvider interface.

Entity set Rights

An enumeration used to define access rights to data that is deployed by Data Services.This enumeration bitwise combination of its member values.

The EntitySetRight enumeration is used to specify access rights for entity set resources that are available on the data service.

This enumeration is used as second argument of DataServiceConfiguration::setEntitySetAccessRule API.

DataServiceConfiguration::SetEntitySetAccessRule(name, rights)

- name: Name of the entity set for which to set access rights, an asterisk (*) value can be supplied for the name parameter to set access for all remaining entity sets to the same level.
- rights: One of EntitySetRights enum value specifies the access rights to be granted to this entity set

Member name	Description
None	Denies all rights to access data.

ReadSingle	Authorization to read single data items.
ReadMultiple	Authorization to read sets of data.
WriteAppend	Authorization to create new data items in data sets.
WriteReplace	Authorization to replace data.
WriteDelete	Authorization to delete data items from data sets.
WriteMerge	Authorization to merge data.
AllRead	Authorization to read data.
AllWrite	Authorization to write data.
All	Authorization to create, read, update, and delete data.

Usually we use “ALL” from this enumeration as right now we are providing only read-only functions so other values are for further enhancement purpose.

Click here to see the use of “EntitySetRights”.

```
// Enum for entity set access rights

class EntitySetRightsEnum
{
    public static $None = 0x000000;

    /**
     * Specifies the right to read one resource belonging to this entity set per
request.
     */
    public static $ReadSingle = 0x000001;

      /**
     * Specifies the right to read multiple resources belonging to this entity
set per request.
     */
    public static $ReadMultiple = 0x000010;

    /**
     * Specifies the right to add (POST) new resources to the entity set.
     */
    public static $WriteAppend = 0x000100;
    /**
     * Specifies the right to replace (PUT) existing resource in the entity set
     */
    public static $WriteReplace = 0x001000;

    /**
     * Specifies the right to delete existing resource in the entity set
     */
    public static $WriteDelete = 0x010000;

      /**
       * Specifies the right to update (MERGE) existing resource in the container
       */
    public static $WriteMerge = 0x100000;

    /**
     * Specifies the right to read single or multiple resources in a single
request
     */
    public static $AllRead = 0x000001 | 0x000010;

    /**
     * Specifies right to perform CUD
     */
    public static $AllWrite = 0x000100 | 0x010000 | 0x001000 | 0x100000;

    /**
     * specifies rights to perform all operations
     */
    public static $All = 0x000100 | 0x010000 | 0x001000 | 0x100000 | 0x000001 |
0x000010;

}
```

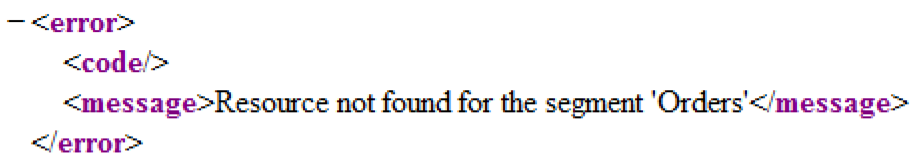
By using of Entity Set Rights we can change the visibility of one specific entity or all entities.

For example if we define this function in following-file:**Services/NorthWind/NorthWindDataService.php**

```
//This method is called only once to initialize service-wide policies
public static function initializeService(DataServiceConfiguration &$config)
{
    $config->setEntitySetPageSize('*', 5);
    $config->setEntitySetAccessRule('Orders', EntitySetRights::NONE);
    $config->setEntitySetAccessRule('*', EntitySetRights::ALL);
    $config->setAcceptCountRequests(true);
    $config->setAcceptProjectionRequests(true);
    $config->setMaxDataServiceVersion(DataServiceProtocolVersion::V3);
}
```

Here even though entity set 'Orders' is registered in metadata, it will not appear in edmx as its visibility is NONE and If client tried to access 'Orders' Entity set server will through resource not found error.

Now if user try to access "http://localhost:8086/NorthWind.svc/Customers('ALFKI')/Orders" then user will get following o/p:



```
– <error>
    <code/>
    <message>Resource not found for the segment 'Orders'</message>
  </error>
```

Configuring the OData Services

First assume that developer has the OData library in following path: D:\Projects\ODataPHPProducer

Configuring PHP

Modify the PHP configuration-file:C:\PHP5\php.ini to include the path of the OData Producer library and restart the IIS server:

include_path = “.;D:\Projects\ODataPHPProducer\library”;

Configuring Web-Server

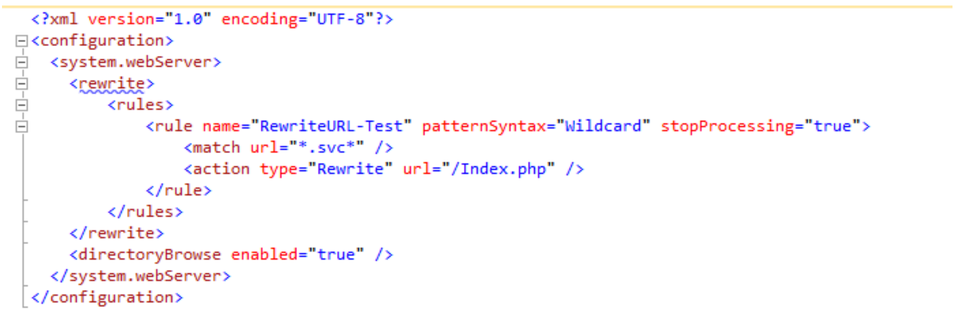
IIS

This step is optional and required if we are using IIS as web server:

We have to install URL rewrite module for IIS from following path:

i. http://www.iis.net/download/URLRewrite

Please make sure that C:\Projects\ODataPHPProducer\web.config file contains following entry in



```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <rewrite>
        <rules>
            <rule name="RewriteURL-Test" patternSyntax="Wildcard" stopProcessing="true">
                <match url="*.svc*" />
                <action type="Rewrite" url="/Index.php" />
            </rule>
        </rules>
    </rewrite>
    <directoryBrowse enabled="true" />
  </system.webServer>
</configuration>
```

Apache2

If developer wants to use this library with apache then we have to modify “.htaccess” file for url-rewrite by following way:

1) Make sure that /var/www/.htacccess file contains following entry:

```
<IfModule mod_rewrite.c>

        RewriteEngine on

        RewriteRule    (\.svc.*) OData/Index.php

</IfModule>
```

Here /var/www/OData folder contains the complete library code.

2) Make sure to add the following entries in the /etc/apache2/httpd.conf file:

```
php_value include_path "./:/var/www/OData/library"

php_value date.timezone "America/Los_Angeles"
```

3) Make sure that /etc/apach2/apache2.conf file has following entries:

```
Include /etc/apache2/httpd.conf
```

Database: Installation and Configuration

SQL-Server

This step is required only if we are going to use SQL-Server as data-source.

We have to install SQLEXPRESS (With instance name ;SQLEXPRESS)

MySQL

This step is required only if we are going to use SQL-Server as data-source.

We can download and install the MySQL-Server from the following URL: http://dev.mysql.com/downloads/mysql/

Installing the NorthWind DB(For testing purpose)

This step is required only for testing the sample NorthWind service.

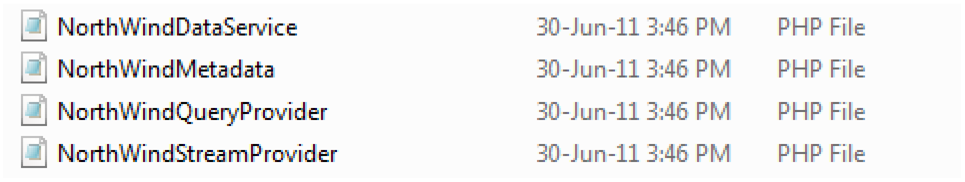
We have to install NorthWind database from the following URL: (http://www.microsoft.com/downloads/en/details.aspx?FamilyID=06616212-0356-46a0-8da2-eebc53a68034)

Configuring the library for NorthWind Services (For testing purpose)

We will take a NorthWind DB example to understand that how to configure the service with the OData Producer for PHP. For instance we are configuring the sample NorthWind services with IIS that listen to the port:"**8086**".

Directory Structure

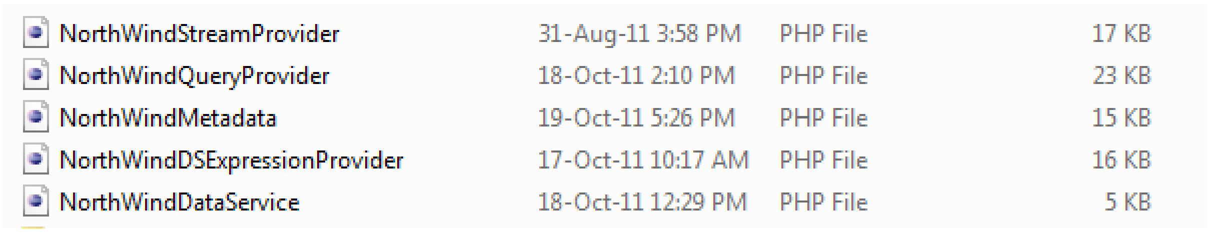
Please make sure that required files for services are available in the service folder as we have files for the sample service:”NorthWind” in /services/NorthWindDB



NorthWindDataService	30-Jun-11 3:46 PM	PHP File
NorthWindMetadata	30-Jun-11 3:46 PM	PHP File
NorthWindQueryProvider	30-Jun-11 3:46 PM	PHP File
NorthWindStreamProvider	30-Jun-11 3:46 PM	PHP File

NOTE : Here last one file “NorthWindStreamProvider.php” is an optional file and we need to create this file only if we want to expose some stream data by using of “OData Producer for PHP”.

Here if we are implementing IDataServiceQueryProvider2 then we have to implement IExpressionProvider also and service folder services\NorthWind will contain following files:

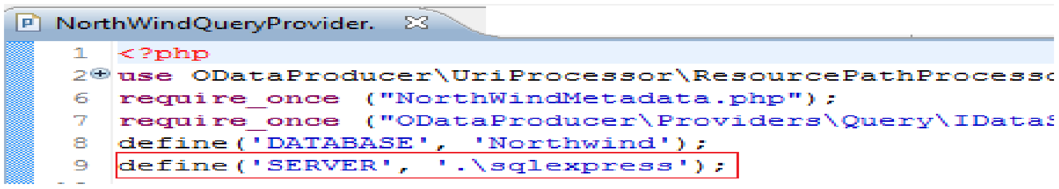


NorthWindStreamProvider	31-Aug-11 3:58 PM	PHP File	17 KB
NorthWindQueryProvider	18-Oct-11 2:10 PM	PHP File	23 KB
NorthWindMetadata	19-Oct-11 5:26 PM	PHP File	15 KB
NorthWindDSExpressionProvider	17-Oct-11 10:17 AM	PHP File	16 KB
NorthWindDataService	18-Oct-11 12:29 PM	PHP File	5 KB

Configuring QueryProvider to connect to underlying DB

IIS

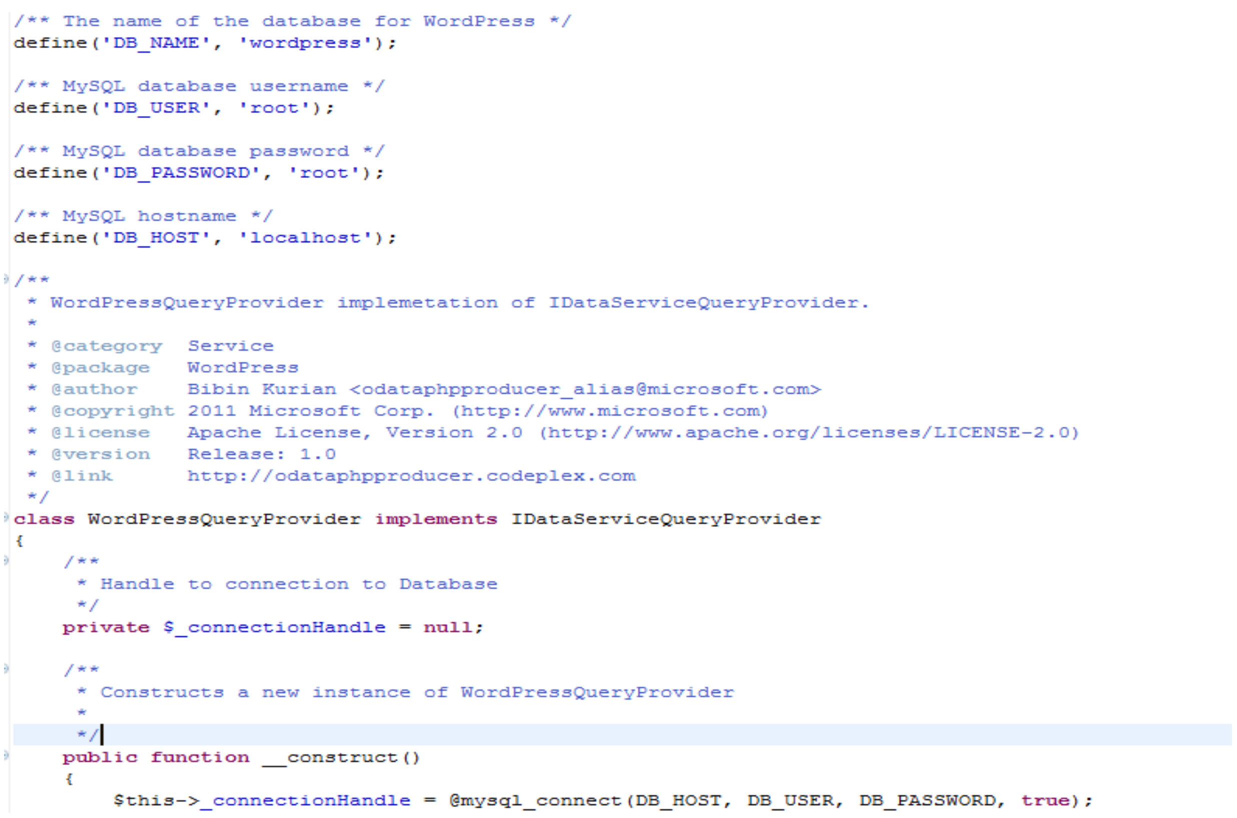
Open ‘services\NorthWind\NorthWindQueryProvider.php’ and update ‘SERVER’ constant with your SQL server instance name



```
<?php
use ODataProducer\UriProcessor\ResourcePathProcesso
require_once ("NorthWindMetadata.php");
require_once ("ODataProducer\Providers\Query\IDataS
define('DATABASE', 'Northwind');
define('SERVER', '.\sqlexpress');
```

MySQL

We can modify this QueryProvider.php file For MySQL we can modify this file like following way

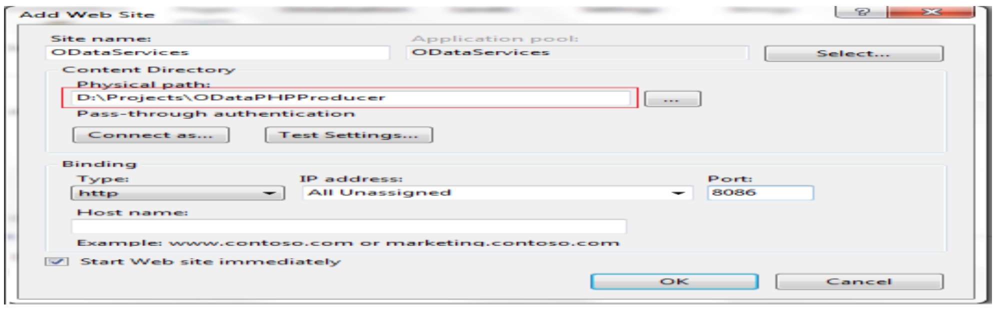


```
/** The name of the database for WordPress */
define('DB_NAME', 'wordpress');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'root');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/**
 * WordPressQueryProvider implemetation of IDataServiceQueryProvider.
 *
 * @category  Service
 * @package   WordPress
 * @author    Bibin Kurian <odataphpproducer_alias@microsoft.com>
 * @copyright 2011 Microsoft Corp. (http://www.microsoft.com)
 * @license   Apache License, Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)
 * @version   Release: 1.0
 * @link      http://odataphpproducer.codeplex.com
 */
class WordPressQueryProvider implements IDataServiceQueryProvider
{
    /**
     * Handle to connection to Database
     */
    private $_connectionHandle = null;

    /**
     * Constructs a new instance of WordPressQueryProvider
     *
     */
    public function __construct()
    {
        $this->_connectionHandle = @mysql_connect(DB_HOST, DB_USER, DB_PASSWORD, true);
```

Creating and Configuring the Web-Site(Required only for IIS)

This step is required only for IIS.

Create a Web-Site in IIS with physical path-points to the framework folder.



Add Web Site

Site name: ODataServices
Application pool: ODataServices
Content Directory
Physical path: D:\Projects\ODataPHPProducer
Pass-through authentication
Binding
Type: http
IP address: All Unassigned
Port: 8086
Host name:
Example: www.contoso.com or marketing.contoso.com
[✓] Start Web site immediately

Now restart the Web-Site.

Making-Sure that Everything is working fine

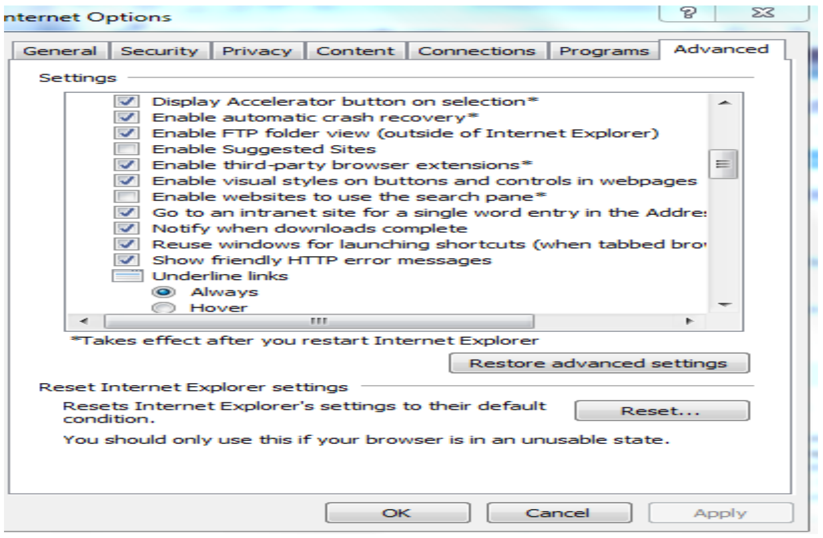
We can’t test the things until and unless we implement all the required providers and configure the Web-Server.
After implementing the provider and configuring the service and also after configuring the Web-Server we can start testing to make sure that whether we implemented the provider in the right way or not.

- First we have to set-up one web-site if you are using IIS.

- Then we have to configure the Web-Server for URL-Rewrite mode so that if user mentioned .svc in the URL then request should reached to our Index.php file.
- For instance if user setup the NorthWind.svc and if Web-Site is configured for 8086 port on localhost
  - If user request for http://localhost:8086/NorthWind.svc then library should return the service document.
  - If user request for http://localhost:8086/NorthWind.svc/$metadata then user should get metadata document which will explain the entities and properties of each entity and their relationships.
  - If user request for http://localhost:8086/NorthWind.svc/Customers then he should get list of 5 customers if page-size is configured as 5 in the implementation of IserviceProvider.

If user does something wrong in the implementation of provider then library will throw serialized error everytime.

- If User is using IE then everytime he will get generalized error message from IE and he can check the exact error message in Mozilla/Firefox.
  - For checking the exact error message in IE user has to do following changes in the IE configuration:
    - Access IE->InternetOptions->Advanced
    - Uncheck “Show Friendly HTTP Error-Message”



Settings

- [✓] Display Accelerator button on selection*
- [✓] Enable automatic crash recovery*
- [✓] Enable FTP folder view (outside of Internet Explorer)
- [ ] Enable Suggested Sites
- [✓] Enable third-party browser extensions*
- [✓] Enable visual styles on buttons and controls in webpages
- [ ] Enable websites to use the search pane*
- [✓] Go to an intranet site for a single word entry in the Addre:
- [✓] Notify when downloads complete
- [✓] Reuse windows for launching shortcuts (when tabbed bro
- [✓] Show friendly HTTP error messages
- Underline links
  - ● Always
  - ○ Hover

*Takes effect after you restart Internet Explorer

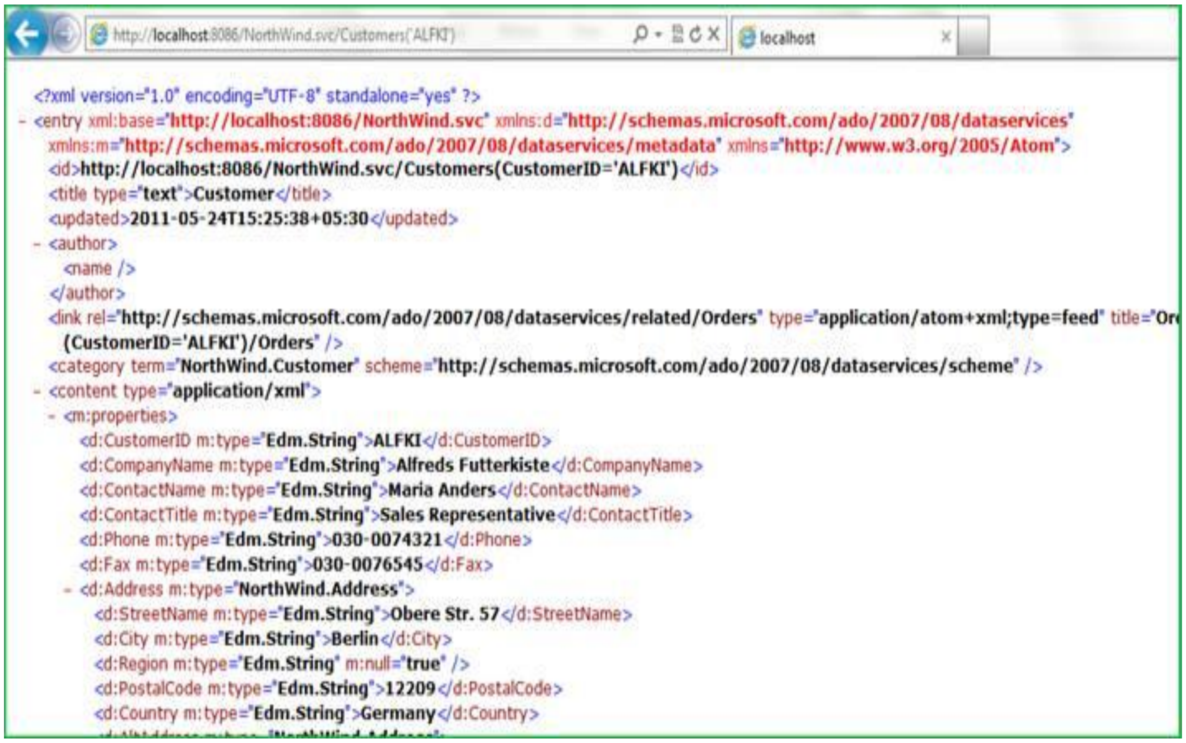
Reset Internet Explorer settings

Resets Internet Explorer's settings to their default condition.

You should only use this if your browser is in an unusable state.

Accessing the Service

The service has been configured and now you can browse the service:



```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <entry xml:base="http://localhost:8086/NorthWind.svc" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
    <id>http://localhost:8086/NorthWind.svc/Customers(CustomerID='ALFKI')</id>
    <title type="text">Customer</title>
    <updated>2011-05-24T15:25:38+05:30</updated>
  - <author>
      <name />
    </author>
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders" type="application/atom+xml;type=feed" title="Or
      (CustomerID='ALFKI')/Orders" />
    <category term="NorthWind.Customer" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  - <content type="application/xml">
    - <m:properties>
        <d:CustomerID m:type="Edm.String">ALFKI</d:CustomerID>
        <d:CompanyName m:type="Edm.String">Alfreds Futterkiste</d:CompanyName>
        <d:ContactName m:type="Edm.String">Maria Anders</d:ContactName>
        <d:ContactTitle m:type="Edm.String">Sales Representative</d:ContactTitle>
        <d:Phone m:type="Edm.String">030-0074321</d:Phone>
        <d:Fax m:type="Edm.String">030-0076545</d:Fax>
      - <d:Address m:type="NorthWind.Address">
          <d:StreetName m:type="Edm.String">Obere Str. 57</d:StreetName>
          <d:City m:type="Edm.String">Berlin</d:City>
          <d:Region m:type="Edm.String" m:null="true" />
          <d:PostalCode m:type="Edm.String">12209</d:PostalCode>
          <d:Country m:type="Edm.String">Germany</d:Country>
```