

# OData Producer Library for PHP

## INDEX

<a href="#"><u>Overview</u></a>	03
<a href="#"><u>How OData Works: Technology Basics</u></a>	03
<a href="#"><u>The OData Data-Model</u></a>	04
<a href="#"><u>Protocol Basics</u></a>	04
<a href="#"><u>OData Producer Library Library for PHP</u></a>	05
<a href="#"><u>System Requirements</u></a>	06
<a href="#"><u>For Windows</u></a>	06
<a href="#"><u>For Linux</u></a>	06
<a href="#"><u>Directory Structure</u></a>	06
<a href="#"><u>OData Service Implementation</u></a>	06
<a href="#"><u>Implementation of Interfaces</u></a>	07
<a href="#"><u>IDataServiceMetadata Provider</u></a>	08
<a href="#"><u>Member Methods</u></a>	09
<a href="#"><u>Implementing IDataServiceMetadaProvider</u></a>	09
<a href="#"><u>Creating Metadata</u></a>	10
<a href="#"><u>Adding Stream Data/Property</u></a>	13
<a href="#"><u>Exposing Metadata via IDataServiceMetadataProvider</u></a>	14

<a href="#"><u>Creating Meta-Data for NorthWind Service</u></a>	16
<a href="#"><u>Adding Complex Type</u></a>	17
<a href="#"><u>Relationships</u></a>	20
<a href="#"><u>Building relationship through metadata</u></a>	22
<a href="#"><u>IDataServiceMetadataProvider Changes</u></a>	24
<a href="#"><u>IDataServiceQuery Provider</u></a>	25
<a href="#"><u>Member Methods</u></a>	25
<a href="#"><u>Implementing IDataServiceQueryProvider</u></a>	26
<a href="#"><u>IDataServiceStreamProvider</u></a>	34
<a href="#"><u>Member Methods</u></a>	34
<a href="#"><u>Implementing IDataServiceStreamProvider</u></a>	35
<a href="#"><u>IDataServiceStreamProvider2</u></a>	37
<a href="#"><u>Member Methods</u></a>	37
<a href="#"><u>Implementing IDataServiceStreamProvider2</u></a>	38
<a href="#"><u>IServiceProvider</u></a>	40
<a href="#"><u>Member Methods</u></a>	40
<a href="#"><u>Implementing IServiceProvider</u></a>	41
<a href="#"><u>Changes in the service.config file</u></a>	41
<a href="#"><u>Configuring the OData Service Parameter</u></a>	42
<a href="#"><u>Member Methods</u></a>	43
<a href="#"><u>How to set Configuration Parameter – An Example</u></a>	45
<a href="#"><u>Entity Set Rights</u></a>	45
<a href="#"><u>Configuring the OData Services</u></a>	49
<a href="#"><u>Configuring PHP</u></a>	49
<a href="#"><u>Configuring Web-Server</u></a>	49
<a href="#"><u>IIS</u></a>	49
<a href="#"><u>Apache2</u></a>	49
<a href="#"><u>Database : Installation and Configuration</u></a>	50
<a href="#"><u>SQL-Server</u></a>	50

<a href="#"><u>MySQL</u></a>	50
<a href="#"><u>Installing the NorthWind DB</u></a>	50
<a href="#"><u>Configuring the library for NorthWind Services</u></a>	50
<a href="#"><u>Directory Structure</u></a>	51
<a href="#"><u>Configuring Query Provider to connect to the underlying DB</u></a>	51
<a href="#"><u>IIS</u></a>	51
<a href="#"><u>MySQL</u></a>	52
<a href="#"><u>Creating and configuring the Web-Site(Only for IIS)</u></a>	52
<a href="#"><u>Making-Sure that Everything is working Fine</u></a>	53
<a href="#"><u>Accessing the service</u></a>	54

## 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.

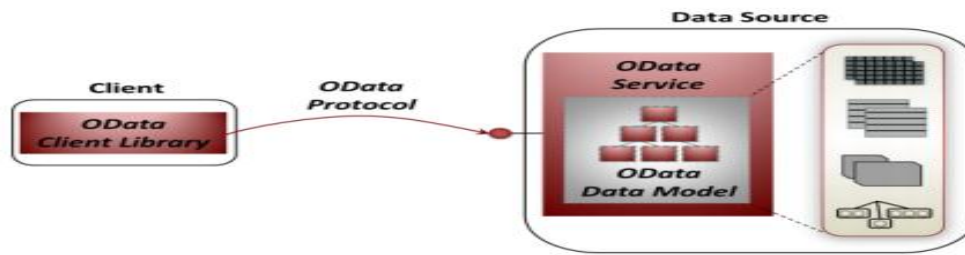
There may be many possible sources of data and many possible clients for one service like Web browsers, apps on mobile devices, business intelligence (BI) tools, and more. How can this varied set of clients access these diverse data sources?

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.

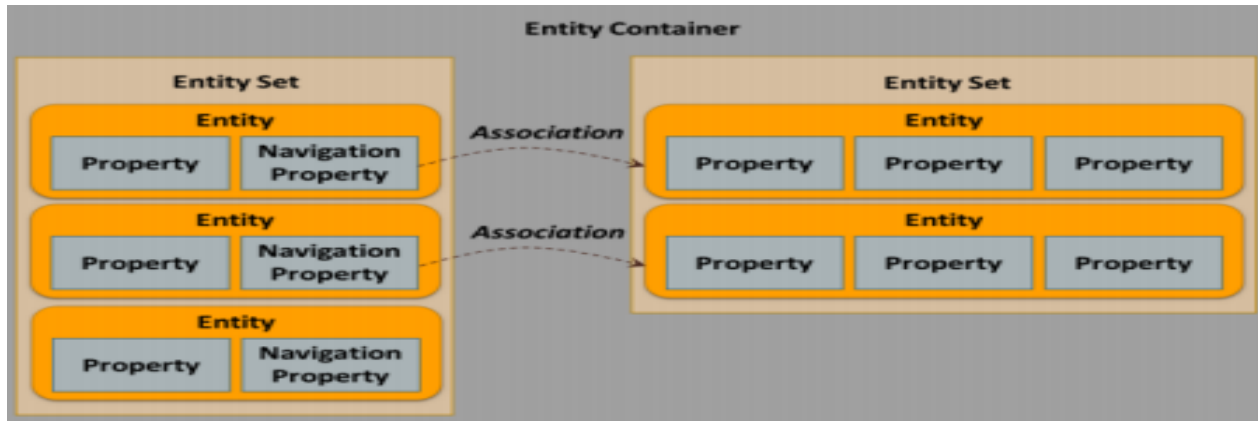
### The OData Data-Model

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 LIBRARY LIBRARY FOR PHP

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

Main components of the OData Producer Library Library 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 [IDataService](#).
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 the URL, Validating the URL and execution of the Query.

#### 6. Response Writer

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

OData Producer Library Library supports all 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

---

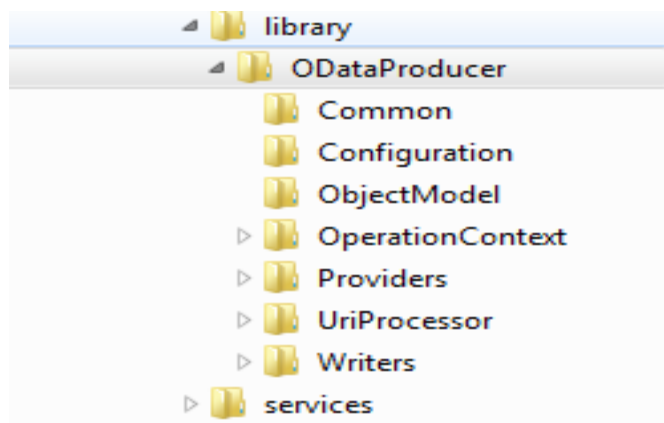
- 1) IIS, URL Rewrite(<http://www.iis.net/download/URLRewrite>) / Apache2 with URL rewrite mode
- 2) PHP-5.3

### For Linux

---

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

## Directory Structure



## ODATA SERVICE IMPLEMENTATION

With using of OData Producer Library Library , 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 Library 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 Library:

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.
IDataServiceStreamProvider ( <b>Optional</b> 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.

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

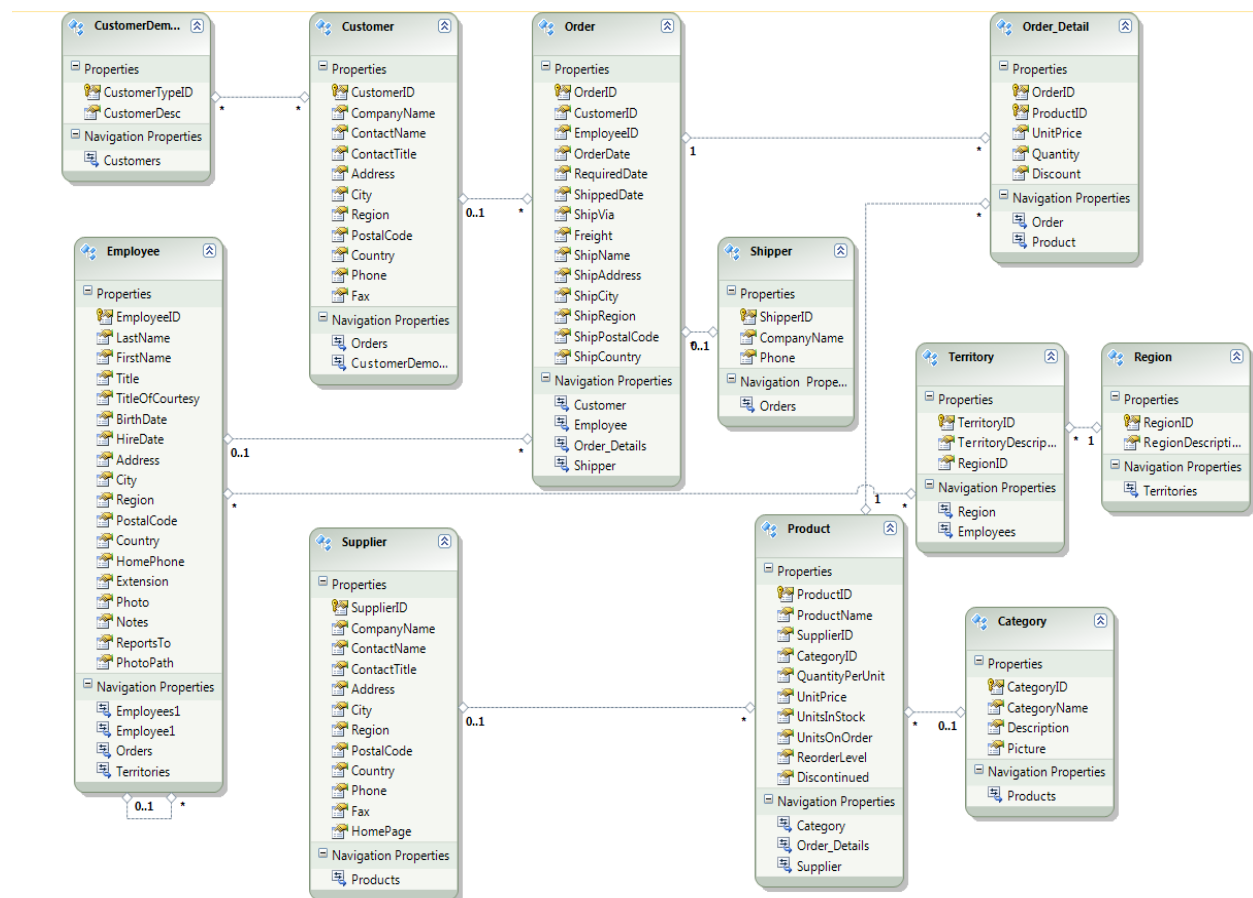
1. Defining a [class DataServiceCustomMetadata that implements interface IDataServiceMetadataProvider](#). This class also defines some helper functions that will be used for defining the metadata.
2. Defining [proxy classes representing the entities exposed by the data service](#).

3. [Defining a function CreateNorthWindMetaData](#) that uses the helper methods of `DataServiceCustomMetadata` class to define the shape of the service (container and namespace name, entities and its properties, entity sets and association between entities)
4. Implementing the [mechanism to get the resource from the underlying DB](#).
5. Defining [a class which represents the data service](#) (by deriving from framework's `DataService` class) and implements `IServiceProvider` interface.
6. [Modifications in the Service.Config.xml file](#) to register the new service with the library.
7. [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 Library for PHP.

We will use the NorthWind DB to show the process to create a new OData feed using OData Producer Library 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.

## 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

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

```
class NorthWindMetadata implements IDataServiceMetadataProvider
{
```

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

Please click here to check the [implementation of IServiceProvider::getService\(\)](#).

## 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.

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
}

```

Now you want to create a ResourceType for Order. To create a ResourceType for Order, this is how you would do it:

```

$OrderEntityType = new ResourceType(
    new ReflectionClass('Order'), // Entity class
    ResourceTypeKind::ENTITY, // Entity, ComplexType etc
    "Order", // Name of the resource
    "Namespace", // Namespace of the resource
    null, // Base type of the resource, if exists
    false // Whether resource is abstract
);

$OrderID = new ResourceProperty(
    "OrderID",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE |
    ResourcePropertyKind::KEY, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::INT32)
);

$customerID = new ResourceProperty(
    "CustomerID",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$EmployeeID = new ResourceProperty(
    "EmployeeID ",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::INT32)
);

//In the Order entity OrderDate is primitive ETag
$OrderDate = new ResourceProperty(
    "OrderDate",
    null, // Mime type of the property

```

```

ResourcePropertyKind::PRIMITIVE |
ResourcePropertyKind::ETAG, // The kind of property
ResourceType::getPrimitiveResourceType(EdmPrimitiveType::DATETIME)
);

$requiredDate = new ResourceProperty(
    "RequiredDate",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::DATETIME)
);

$shippedDate = new ResourceProperty(
    "ShippedDate",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::DATETIME)
);

$shipVia = new ResourceProperty(
    "ShipVia",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::INT32)
);

$freight = new ResourceProperty(
    "Freight",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::DECIMAL)
);

$shipName = new ResourceProperty(
    "ShipName", 0
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

/**
 * Note: Register the remaining properties in Order entity as above.
 * To register an etag property, you have to pass the kind of property as
either ResourcePropertyKind::PRIMITIVE or ResourcePropertyKind::ETAG.
 * Note : For all the remaining properties, Resource Type would be STRING as
specified for "ShipName"
**/

```

The OrderID is the entity key, so we passed ResourcePropertyKind::PRIMITIVE or ResourcePropertyKind::KEY as the kind of property while creating the property.

In order to add the properties belongs to the order resource type instance

```

$orderEntityType->addProperty($orderId);

$orderEntityType->addProperty($customerId);

```

```
$orderEntityType->addProperty($orderDate);
```

**\*\*Note: Add the remaining properties as above.**

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

Next to create a ResourceSet called Orders to expose our ResourceType, you simply do this:

```
$ordersResourceSet = new ResourceSet('Orders', $orderEntityType);
```

### Adding Stream Data/Property

The ResourceProperty constructor allows specifying the mime type of property. This will help if a property contains binary data. Consider the following example:

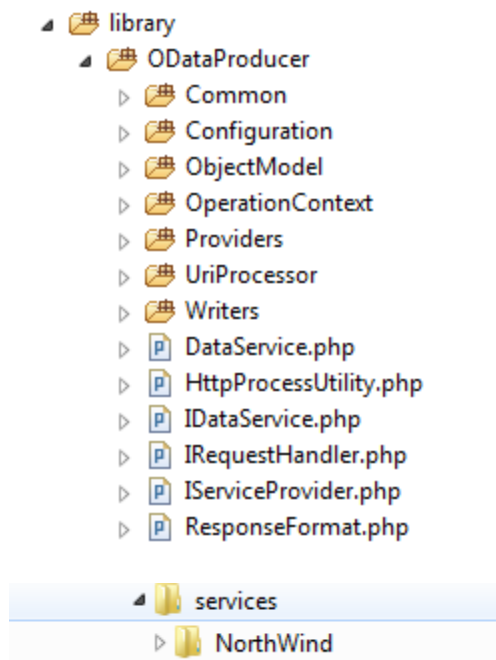
```
$photo = new ResourceProperty(  
    "Photo",  
    "image/jpeg", // Mime type of the property  
    ResourcePropertyKind::PRIMITIVE, // The kind of property  
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::BINARY)  
);
```

```
$employeeEntityType->addProperty($photo);
```

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. The 'MimeType' attribute helps the client to understand the kind of encoded data.

```
- <EntityType Name="Employee" m:HasStream="true">  
  - <Key>  
    <PropertyRef Name="EmployeeID"/>  
  </Key>  
  <Property Name="EmployeeID" Type="Edm.Int32" Nullable="false"/>  
  <Property Name="Photo" Type="Edm.Binary" Nullable="true" m:MimeType="image/jpeg"/>  
  <Property Name="LastName" Type="Edm.String" Nullable="true"/>  
  <Property Name="Title" Type="Edm.String" Nullable="true"/>
```

After implementing the providers /services folder should look like :



And services\NorthWind folder will contain following files:

	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 Library for PHP”.

### Exposing Metadata via IDataServiceMetadataProvider

Once you’ve built your metadata you just need to expose it via your implementation.

Here is what your implementation might look like:

```
class NorthWindMetadata implements IDataServiceMetadataProvider
{
    protected $_resourceSets = array();
    protected $_resourceTypes = array();
    protected $_containerName;
    protected $_namespaceName;

    public function NorthWindMetadata($containerName, $namespaceName)
    {
        $this->_containerName = $containerName;
        $this->_namespaceName = $namespaceName;
        $this->createNorthWindMetadata();
    }
}
```

```

    public function addResourceType(ResourceType $resourceType)
    {
        if (array_key_exists($resourceType->getName(), $this->_resourceTypes))
        {
            throw new InvalidOperationException('Type with same name already added');
        }
        $this->_resourceTypes[$resourceType->getName()] = $resourceType;
    }

    public function addResourceSet(ResourceSet $resourceSet)
    {
        if (array_key_exists($resourceSet->getName(), $this->_resourceSets))
        {
            throw new InvalidOperationException('Resource Set already added');
        }
        $this->_resourceSets[$resourceSet->getName()] = $resourceSet;
        $resourceSet->getResourceType()->setCustomState($this->_resourceSets[$name]);
    }

    public function getContainerName()
    {
        return $this->_containerName;
    }

    public function getContainerNamespace()
    {
        return $this->_namespaceName;
    }

    public function getResourceSets()
    {
        return array_values($this->_resourceSets);
    }

    public function getTypes()
    {
        return array_values($this->_resourceTypes);
    }

    public function resolveResourceSet($name)
    {
        if (array_key_exists($name, $this->_resourceSets))
        {
            return $this->_resourceSets[$name];
        }
        return NULL;
    }

    public function resolveResourceType($name)
    {
        if (array_key_exists($name, $this->_resourceTypes))
        {
            return $this->_resourceTypes[$name];
        }
    }

```

```

    }
    return NULL;
}

public function getDerivedTypes(ResourceType $resourceType)
{
    return null;
}

public function hasDerivedTypes(ResourceType $resourceType)
{
    return false;
}

public function getResourceAssociationSet(ResourceSet $sourceResourceSet,
ResourceType $sourceResourceType, ResourceProperty $targetResourceProperty)
{
    throw new InvalidOperationException("No relationships.");
}

private function createNorthWindMetadata()
{
}
}

```

## Creating Meta-Data for NorthWind Service

Inside the function createNorthWindMetadata you do something like this:

```

private function createNorthWindMetadata()
{
    $orderEntityType = new ResourceType(
        new ReflectionClass('Order'), // Entity class
        ResourceTypeKind::ENTITY, // Entity, ComplexType etc
        "Order", // Name of the resource
        "Namespace", // Namespace of the resource
        null, // Base type of the resource, if exists
        false // Whether resource is abstract
    );

    $orderId = new ResourceProperty(
        "OrderID",
        null, // Mime type of the property
        ResourcePropertyKind::PRIMITIVE |
        ResourcePropertyKind::KEY, // The kind of property
        ResourceType::getPrimitiveResourceType(EdmPrimitiveType::INT32)
    );

    $orderEntityType->addProperty($orderId);
    $customerId = new ResourceProperty(
        "CustomerID",
        null, // Mime type of the property
        ResourcePropertyKind::PRIMITIVE, // The kind of property
        ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
    );
}

```



```

/**
 *Note: we can add other property same as above.
 */
//Now we can add the Resource Set
$orderEntityType->addProperty($shipCountry);
$this->addResourceType($orderEntityType);
$this->addResourceSet(
    new ResourceSet("Orders", $orderEntityType)
);
}

```

## 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;
    //array(Address)
    public $OtherAddresses;
}

```

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

```

$addressEntityType = new ResourceType(
    new ReflectionClass('Address'), // Entity class
    ResourceTypeKind::COMPLEX, // Entity, ComplexType etc
    "Address", // Name of the resource
    "Namespace", // Namespace of the resource
    null, // Base type of the resource, if exists
    false // Whether resource is abstract
);

$streetName = new ResourceProperty(
    "StreetName",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$city = new ResourceProperty(
    "City",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$region = new ResourceProperty(
    "Region",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$postalCode = new ResourceProperty(
    "PostalCode",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$country = new ResourceProperty(
    "Country",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$altAddress = new ResourceProperty(
    "AltAddress",
    null, // Mime type of the property
    ResourcePropertyKind::COMPLEX_TYPE, // The kind of property
    $addressEntityType
);

```

In the Address complex type the property AltAddress is again a complex property.

Add the properties belongs to the address resource type instance

```
$addressEntityType->addProperty($streetName);
$addressEntityType->addProperty($city);
$addressEntityType->addProperty($region);
$addressEntityType->addProperty($postalCode);
$addressEntityType->addProperty($country);
$addressEntityType->addProperty($altAddress);
```

Next register the entity type Customer

```
$customerEntityType = new ResourceType(
    new ReflectionClass('Customer'), // Entity class
    ResourceTypeKind::ENTITY, // Entity, ComplexType etc
    "Customer", // Name of the resource
    "Namespace", // Namespace of the resource
    null, // Base type of the resource, if exists
    false // Whether resource is abstract
);

$customerID = new ResourceProperty(
    "CustomerID",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE |
    ResourcePropertyKind::KEY, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$companyName = new ResourceProperty(
    "CompanyName",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$contactName = new ResourceProperty(
    "ContactName",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$phone = new ResourceProperty(
    "Phone",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$fax = new ResourceProperty(
    "Fax",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)
);

$address = new ResourceProperty(
    "Address",
```

```

    null, // Mime type of the property
    ResourcePropertyKind::COMPLEX_TYPE, // The kind of property
    $addressEntityType
);

$otherAddresses = new ResourceProperty(
    "OtherAddresses",
    null, // Mime type of the property
    ResourcePropertyKind::COMPLEX_TYPE |
    ResourcePropertyKind::BAG, // The kind of property
    $addressEntityType
);

```

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

Add the properties belongs to the customer resource type instance

```

$customerEntityType->addProperty($customerID);
$customerEntityType->addProperty($companyName);
$customerEntityType->addProperty($contactName);
$customerEntityType->addProperty($phone);
$customerEntityType->addProperty($fax);
$customerEntityType->addProperty($address);
$customerEntityType->addProperty($otherAddresses);

```

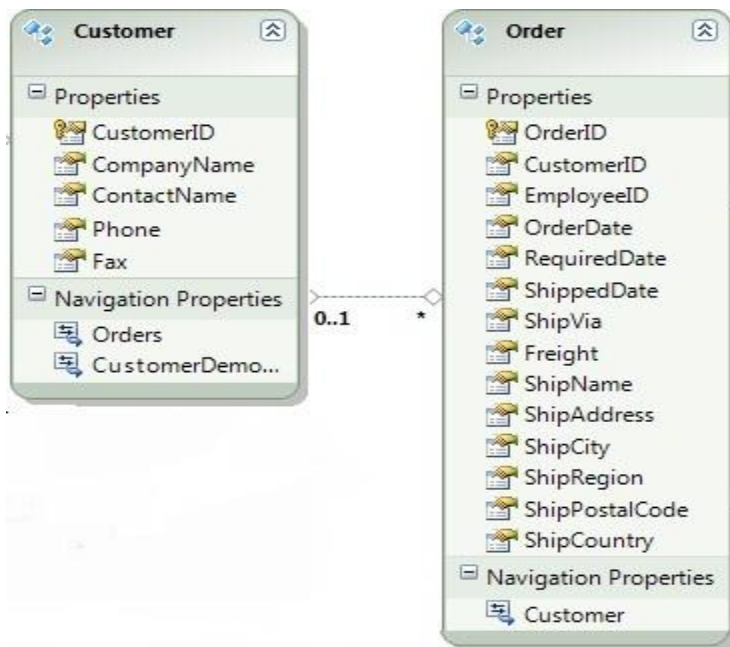
## Relationships

---

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 $Phone;
    //Edm.String
    public $Fax;
    //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;
}

```

## Building relationships through metadata

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

Register Order and its primitive properties as explained above:

```

$orderEntityType = new ResourceType(
    new ReflectionClass('Order'), // Entity class
    ResourceTypeKind::ENTITY, // Entity, ComplexType etc
    "Order", // Name of the resource
    "Namespace", // Namespace of the resource
    null, // Base type of the resource, if exists
    false // Whether resource is abstract
);

$orderID = new ResourceProperty(
    "OrderID",
    null, // Mime type of the property
    // Refer 1) http://www.ietf.org/rfc/rfc2045.txt
    //          2) http://www.ietf.org/rfc/rfc2046.txt
    ResourcePropertyKind::PRIMITIVE |
    ResourcePropertyKind::KEY, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::INT32)
);

$orderDate = new ResourceProperty(
    "OrderDate",
    null, // Mime type of the property
    ResourcePropertyKind::PRIMITIVE |
    ResourcePropertyKind::ETAG, // The kind of property
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::DATETIME)
); //Note: We can add resource property same as above.

```

Add the properties belongs to the Order resource type instance

```

$orderEntityType->addProperty($orderID);
$orderEntityType->addProperty($orderDate);
//Note: We can add property same as above.

```

Now register Customer and its primitive properties:

```
$customerEntityType = new ResourceType(  
    new ReflectionClass('Customer'), // Entity class  
    ResourceTypeKind::ENTITY, // Entity, ComplexType etc  
    "Customer", // Name of the resource  
    "Namespace", // Namespace of the resource  
    null, // Base type of the resource, if exists  
    false // Whether resource is abstract  
);  
  
$customerID = new ResourceProperty(  
    "CustomerID",  
    null, // Mime type of the property  
    ResourcePropertyKind::PRIMITIVE |  
    ResourcePropertyKind::KEY, // The kind of property  
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)  
);  
  
$companyName = new ResourceProperty(  
    "CompanyName",  
    null, // Mime type of the property  
    ResourcePropertyKind::PRIMITIVE, // The kind of property  
    ResourceType::getPrimitiveResourceType(EdmPrimitiveType::STRING)  
);  
//Note: we can add resource property same as above.
```

Add the properties belong to the Customer resource type instance:

```
$customerEntityType->addProperty($customerID);  
$customerEntityType->addProperty($companyName);  
//Note: We can add property same as above.
```

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

```
$ordCustomer = new ResourceProperty(  
    "Customer",  
    null,  
    ResourcePropertyKind::RESOURCE_REFERENCE,  
    $customerEntityType  
);  
$orderEntityType->addProperty($ordCustomer);  
  
$cusOrders = new ResourceProperty(  
    "Orders",  
    null,  
    ResourcePropertyKind::RESOURCESET_REFERENCE,  
    $orderEntityType  
);  
  
$customerEntityType->addProperty($cusOrders);
```

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

Next we need a `ResourceSet` for both `Customer` and `Order`:

```
$ordersSet = new ResourceSet("Orders", $orderEntityType)
$customersSet = new ResourceSet("Customers", $customerEntityType);
```

Once you done that we need to create an `AssociationSet` that link those two navigation properties and resource sets together:

```
$orderCustomerAssociationSet = new ResourceAssociationSet(
    "orderCustomer",
    new ResourceAssociationSetEnd(
        $ordersSet,
        $orderEntityType,
        $ordCustomer
    ),
    new ResourceAssociationSetEnd(
        $customersSet,
        $customerEntityType,
        $cusOrders
    )
);
```

Finally we need to tell the `NorthWindMetadata`, about both `ResourceTypes`, both `ResourceSets` and the `ResourceAssociationSet`:

Add the following code to the function `createNorthWindMetadata` under `NorthWindMetadata` class

```
$this->addResourceType($orderEntityType);
$this->addResourceSet($ordersSet);
$this->addResourceType($customerEntityType);
$this->addResourceSet($customersSet);
$this->addAssociationSet($orderCustomerAssociationSet);
```

## [IDataServiceMetadataProvider Changes](#)

---

For this to work we need to make some changes to our `NorthWindMetadata` to actually support Relationships:

First we need a place to hold `AssociationSets`:

```
protected $_associationSets = array();
```

Then we need a way to register new `AssociationSets`:



```

public function addAssociationSet(ResourceAssociationSet $associationSet)
{
    $this->_associationSets[$assoicationSetKey] = $associationSet;
}

```

And finally we need to update our implementation of method `IDataServiceMetadataProvider::getResourceAssociationSet()` like this:

```

public function getResourceAssociationSet(
    ResourceSet $sourceResourceSet,
    ResourceType $sourceResourceType,
    ResourceProperty $targetResourceProperty
)
{
    $targetResourceSet = $targetResourceProperty->getResourceType()-
>getCustomState();
    if (is_null($targetResourceSet)) {
        throw new InvalidOperationException('Failed to retrieve the custom
state from ' . $targetResourceProperty->getResourceType()->getName());
    }
    //Customer Orders Orders, Order Customer Customers
    $key = $sourceResourceType->getName() . '_' . $targetResourceProperty-
>getName() . '_';
    if (array_key_exists($key, $this->_associationSets)) {
        return $this->_associationSets[$key];
    }

    return null;
}

```

## 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
<code>getResourceSet ()</code>	<code>array(Object)/array()</code>	Get the collection of entities belongs to an entity set.
<code>getResourceFromResourceSet ()</code>	<code>Object/NULL</code>	Gets an entity instance from an entity set identified by a key.
<code>getResourceFromRelatedResourceSet ()</code>	<code>Object/NULL</code>	Gets a related entity instance from an 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 IDataServiceQueryProvider

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

Please click here to check the [implementation of IServiceProvider::getService\(\)](#).

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', 'MySQLDB');

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

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

/** MySQL hostname */
define('DB_HOST', 'localhost');
class MySQLDBQueryProvider 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\('ALFKI'\)](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](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\(10643\)](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 property 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 property 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](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 property with name: '
. $navigationPropertyName);
    }
}
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)  \$operationContext (Type : Reference of the current context, 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 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	String	This method invoked by the data services	NIL

Tag()	of entity, MUST)  \$operationContext (Type : Reference of the currentContext, MUST)		framework to obtain the ETag of the stream associated with the entity specified.	
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, MUST)  \$checkETagForEquality (Type : Boolean, MUST)  \$operationContext (Type : Reference of the current context, MUST)	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 <code>DataServiceException</code> . 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.
getStreamContentType	\$entity (Type : Object	String	This method invoked by the data services framework to obtain the	NIL

2()	of entity, MUST)  \$resourceStreamInfo (Type :ResourceStreamInfo ,MUST)  \$operationContext ( Type : Reference of the currentContext, MUST)		content type (media type) of the named stream associated with the specified entity.	
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 ( 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 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 = new NorthWindMetadata();
            }
            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;
    }

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

## Changes in the service.config FILE

We have to modify the “Services\service.config.xml” file to register the service with the OData Producer Library 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) :

- 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,	Void	Sets the access rights on the specified resource set.

	MUST)		
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.
<u>getAcceptCountRequests()</u>	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.
<u>setAcceptProjectionRequests()</u>	\$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.
<u>getMaxDataServiceVersionObject()</u>	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 maxDataServiceVersion 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 following URL: [http://localhost:8086/NorthWind.svc/Customers\('ALFKI'\)/Orders](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 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/.htaccess 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 Library 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 Library for PHP".

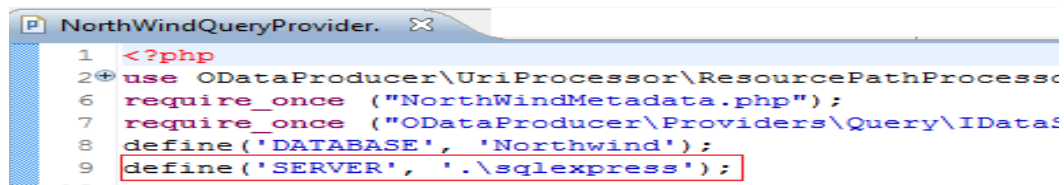
### Configuring QueryProvider to connect to underlying DB

---

## IIS

---

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



```
1 <?php
2 use ODataProducer\UriProcessor\ResourcePathProcess
6 require_once ("NorthWindMetadata.php");
7 require_once ("ODataProducer\Providers\Query\IData
8 define('DATABASE', 'Northwind');
9 define('SERVER', 'localhost');
```

## 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');

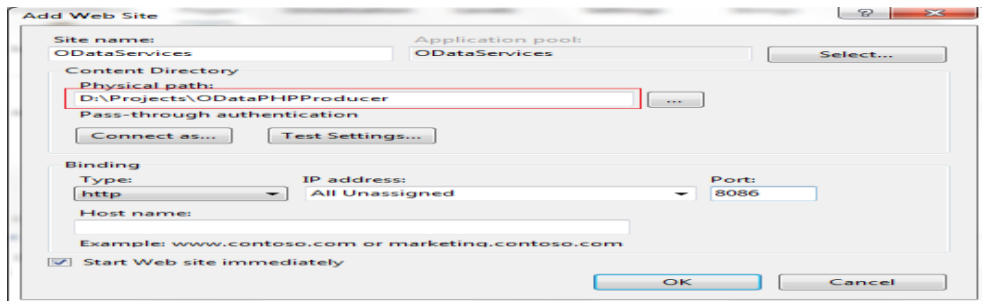
/**
 * WordPressQueryProvider implementation of IDataServiceQueryProvider.
 *
 * @category Service
 * @package WordPress
 * @author Bibin Kurian <odataphpproducer_alias@microsoft.com>
 * @copyright 2011 Microsoft Corp. (http://www.microsoft.com)
 * @license Apache License, Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)
 * @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.



Now restart the Web-Site.

## Making-Sure that Everything is working fine

---

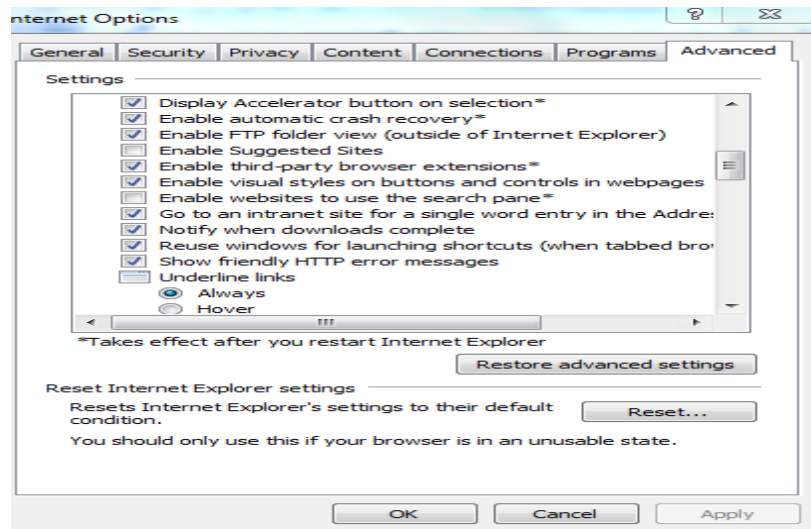
We can't test the things until and unless we implement all the required providers and configure the Web-Server.

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](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"



## Accessing the Service

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

