

ODATA CONNECTOR FOR MYSQL(MYSQL PRODUCER)

INDEX

<u>Overview</u>	02
How OData works : technology basics	02
Required PHP classes for implementing OData service	02
<u>System Requirements</u>	03
Windows	03
Linux	03
<u>Directory Structure</u>	04
<u>Configuration</u>	04
<u>Execution</u>	06
EDMX Generation	07
Generation of all the providers	07
<u>Post Execution</u>	08
<u>Configuring the OData connector for MySQL</u>	08
Doctrine ORM : Configuration and Installation	08
PEAR	08
Manual Instalation	09
Configuring PHP.ini	09

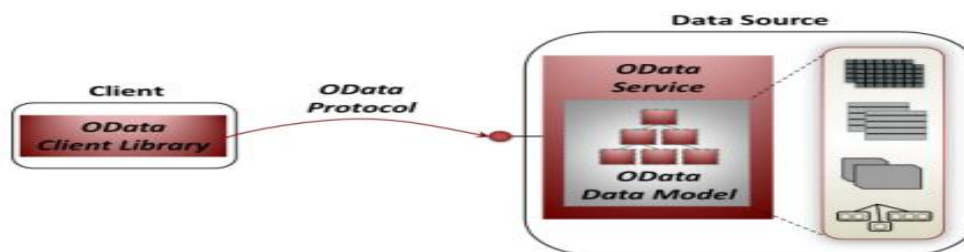
OVERVIEW

MySQL Producer is a command line tool to use with ODataProducer if underlying data source is MySQL. It generates few PHP classes which are required to use the OData Producer to expose your database using OData protocol.

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

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.

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.

Required PHP classes for implementing OData Service

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 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.

User has to implement following set of interfaces to expose their data source by using of OData Producer:

Provider Interfaces	Description
IServiceProvider	Informs the framework that the class represents a service with Data Service Providers implementation.
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 filtering should be taken care by DB rather than library 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 big 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 concern about filtering.
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.
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.

Here MySQL Producer comes in the picture and if underlying data source is MySQL then user can use this tool to generate all the required providers (Except IDataServiceStreamProvider).

SYSTEM REQUIREMENTS

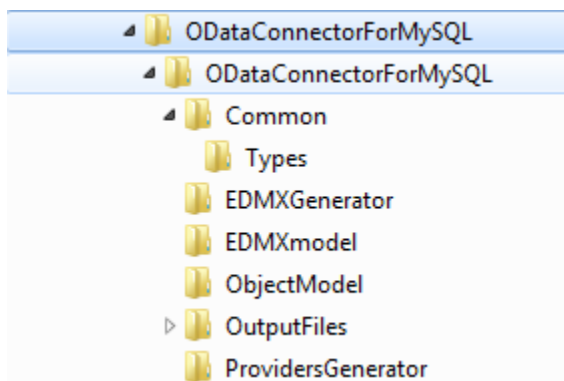
Windows

- 1) PHP-5.3
- 2) Doctrine DBAL & Doctrine Common
- 3) MySQL

Linux

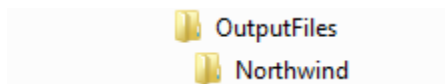
- 1) PHP-5.4
- 2) Doctrine DBAL & Doctrine Common
- 3) MySQL

Directory Structure



Here “ProvidersGenerator” folder contain .xsl files which are required to generate the PHP classes and “OutputFiles” folder contains the generated output files(PHP classes and EDMX files). First MySQL Producer will create one folder in “OuputFiles” for the service and that service specific folder contain all the output files.

“OutputFiles” would look like:









Here “Northwind” folder created by MySQL-Producer because we executed this tool to generate the required PHP classes for exposing northwind DB. Here Northwind folder contains following files:

	NorthwindDataService	30-Nov-11 6:15 PM	PHP File	4 KB
	NorthwindDSExpressionProvider	30-Nov-11 6:15 PM	PHP File	12 KB
	NorthwindEDMX	30-Nov-11 6:14 PM	XML Document	25 KB
	NorthwindMetadata	30-Nov-11 6:15 PM	PHP File	23 KB
	NorthwindQueryProvider	30-Nov-11 6:15 PM	PHP File	36 KB
	service.config	30-Nov-11 6:15 PM	XML Document	1 KB

Here tool will generate NorthWindDSEExpressionProvider if we are implementing interface IDataServiceQueryProvider2.

CONFIGURATION

MySQL Producer has one “system.config.xml” file to configure few of the options before execution. This file is shown in the following file structure:

 Common	11/29/2011 5:01 PM	File folder
 EDMXGenerator	11/29/2011 5:01 PM	File folder
 EDMXmodel	11/29/2011 5:01 PM	File folder
 ObjectModel	11/29/2011 5:01 PM	File folder
 OutputFiles	11/29/2011 5:04 PM	File folder
 ProvidersGenerator	11/29/2011 5:01 PM	File folder
 service.config	12/1/2011 4:49 PM	XML Document

Here one default service.config file is given as shown in figure.

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <rules>
    <!-- this flag is used for processing many to many relationship if exists -->
    <viewManyToManyRelationship>true</viewManyToManyRelationship>
    <!-- defines the query provider version -->
    <queryProviderVersion>2</queryProviderVersion>
    <!-- Accessible entity set names(* means all) for defined page size -->
    <nameEntitySetPageSize>*</nameEntitySetPageSize>
    <!-- Page size of the response -->
    <pageSizeEntitySetPageSize>10</pageSizeEntitySetPageSize>
    <!-- Accessible entity set names(* means all) for defined access rule -->
    <nameEntitySetAccessRule>*</nameEntitySetAccessRule>
    <!-- Access rule for requested Data -->
    <rightsEntitySetAccessRule>ALL</rightsEntitySetAccessRule>
    <!-- flag for accept count request -->
    <acceptCountRequest>true</acceptCountRequest>
    <!-- flag for accept projection request -->
    <acceptProjectionRequest>true</acceptProjectionRequest>
    <!-- Maximum Data service version supported -->
    <maxDataServiceVersion>V3</maxDataServiceVersion>
    <!-- maximum expand count per request -->
    <maxExpandCount>null</maxExpandCount>
    <!-- maximum expand depth per request -->
    <maxExpandDepth>null</maxExpandDepth>
    <!-- maximum results per collection -->
    <maxResultsPerCollection>null</maxResultsPerCollection>
    <!-- used for get verbose errors -->
    <useVerboseErrors>null</useVerboseErrors>
  </rules>
</configuration>

```

Specification of all these options is given below:

- <viewManyToManyRelationship>**: This tag is used for handling many to many relationships in MetadataProvider.php and QueryProvider.php files.
 Allowable values for this tag are **true** and **false** only. If value of this tag is true then EDMX file will generates many to many relationship in below scenario:
 There is one table B which contains only two fields one field is foreign key of one table A and other field is foreign key of another table B. And the relation between A to B is Many(A) to One(B). And relation between B to C is One(B) to Many(C). Then EDMX file will show relation between A and C is Many(A) to Many(C).
- <queryProviderVersion>** : This tag is used for defines version of query provider. Allowed values for this tag are **1** and **2** only. If value is 2 then tool will generate implementation of IDataServiceQueryProvider2 as well as IDataServiceExpressionProvider interfaces.
- <nameEntitySetPageSize>** : This tag is used to set allowable entity sets for the given page size. Allowed values for this tag are ***** and **comma(,) separated entity sets** values

- **<pagesizeEntitySetPageSize>** : This tag will set page size for given entity sets. Allowed values for this tag are integer value **>= 1**.
- **<nameEntitySetAccessRule>** : Allowed values for this tag are ***** and **comma(,) separated entity sets** values.
- **<pagesizeEntitySetAccessRule>** : This tag will set access rule for given entity sets. Allowed values for this tag are **NONE | READ_SINGLE | READ_MULTIPLE | WRITE_APPEND | WRITE_REPLACE | WRITE_DELETE | WRITE_MERGE | READ_ALL | WRITE_ALL | ALL**.
- **<acceptCountRequest>** : This tag is used to allow or disallow count request. Allowable values for this tag are **true** and **false** only.
- **<acceptProjectionRequest>** : This tag is used to allow or disallow Projection request. Allowable values for this tag are **true** and **false** only.
- **<maxDataServiceVersion>** : This tag will set the maximum allowable data service version for the requests. Allowed values for this tag are **V1, V2** and **V3** only.
- **<maxExpandCount>** : This tag will set the maximum allowable expand counts for the requests. Allowed values for this tag are integer value **>=1** or **NULL**.
- **<maxExpandDepth>** : This tag will set the maximum allowable expand depth for the requests. Allowed values for this tag are integer value **>=1** or **NULL**.
- **<maxResultsPerCollection>** : This tag will set the maximum allowable results per collection for the requests. Allowed values for this tag are integer value **>=1** or **NULL**.
- **<useVerboseErrors>** : This tag is used to display verbose error or not. Allowable values for this tag are **true, false** or **NULL** only.

EXECUTION

Functionality of MySQL-Producer is divided in 2 different stages:

- EDMX Generation
 - This stage parse the underlying DB schemas and creates EDMX in xml format. This file contains details about each entity, properties and navigations
- Generations of all the providers
 - This stage uses the EDMX as input and generates PHP classes which we can use with OData Producer to implement an Odata service.

Tool provides flexibility to the end user that they can modify the EDMX file also if they don't want to expose few of the entities and navigations and then user can run the second stage of MySQL connector which would use modified EDMX file as input source and will generate the required PHP classes which user want to expose using Odata.

EDMX Generation

EDMX generation uses doctrine to generate the EDMX file. In this stage tool reads all the defined objects

in the underlying DB and then generates a EDMX file in XML format.

We have to execute following command for this stage:

```
php MySQLConnector.php /db=<DB Name> /srv=<Service Name> /u=<MySQL Username>  
/pw=<MySQL Password> [/p=<port of MySQL Server>] /h=<host name of MySQL server
```

Here we have to pass all the connection parameters which are required to make a connection to underlying DB so that tool can capture the schema definitions which are defined in the underlying DB and can use those definitions to generate the EDMX file which would contain details about all the entities defined in the DB and their navigations, primary keys, secondary/foreign keys and properties.

- We can specify all the command line options in any order.
- Specifying portnumber(/p=<port number>) is optional and it is required only if mysql server is not running on default port.
- Here we don't need to specify angular brackets(<>) in the command.

After generation of EDMX file, tool gives an option to the user so that they can break the execution, modify the EDMX file and then can start the second stage execution, if user doesn't want to expose few of the entities and their navigations else it would continue the execution for the second stage and will use the generated EDMX file for generating the required PHP classes.

Generations of all the providers

This stage use the EDMX file as input file(generated in the previous stage) and generate the PHP classes(providers) based on that EDMX file.

We have to execute the following command for this stage:

```
php MySQLConnector.php /srv=<Service Name>
```

Here we need to pass only service-name and that must be same as mentioned in the first stage.

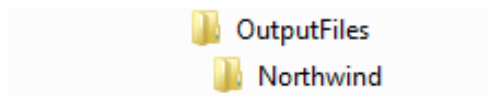
Please make sure that above mentioned command to execute this stage is required only if we had terminated the processing of first stage (After generation of EDMX file) to modifying the EDMX file else our Stage-1 command will continue the processing for this stage and will generate all the required PHP classes for ODataProducer.

POST EXECUTION

After successful execution of Stage-2 we will have all the required PHP classes in "OutputFiles/<Service-Name" folder which we can use to implement odata based service. All the generated files will contain value of service-name as prefix.

We would have one specific sub-folder of that service inside "OutputFiles" folder like we have one folder

for Northwind service(As displayed in following diagram)



- Now please make sure that we have all the required PHP classes in this sub-folder and this folder must contain one service.config file also which is required to configure an odata service.
- Copy all these files PHP files into the OData service folder.
- Now open the service.config file which we found in above mentioned stage and append those xml contents in the Odata config file to configure the service is an odata enabled service PHP so that OData-Producer can generate the OData feeds for the specified service.
 - We would have following contents for northwind service

```
<Service Name="Northwind.svc">
  <path>Services\Northwind\NorthwindDataService.php</path>
  <classname>NorthwindDataService</classname>
  <baseURL>/Northwind.svc</baseURL>
</Service>
```

Now you are ready to expose your data by using of OData-Producer.

Configuring the OData Connector For MySQL

We are assuming that end-developer already has downloaded & configured the Odata-Producer.

Doctrine : Configuration & installation

Doctrine ORM is the package which contains Doctrine-Comman and Doctrine-DBAL which are pre-requisite of ODataConnectorForMySQL (MySQL-Producer)

PEAR

Download & install the Doctrine ORM using Pear utility. Execute the following commands to install Doctrine:

```
pear channel-discover pear.doctrine-project.org
pear install doctrine/DoctrineORM
```

For more information refer (<http://www.doctrine-project.org/projects/orm/download>).

MANUAL INSTALLATION

If above mentioned pear commad doesn't works then we can use following approach to install the doctrine manually:

- 1) Download the Doctrine ORM from <http://www.doctrine-project.org/projects/orm/download> .

- 2) Include the path of Doctrine ORM in PHP.ini.

```
include_path = "./path/to/doctrine_orm";
```

- 3) If application is still not including the doctrine ORM path after setting in the step 2, add following path in OdataConnctorForMySQL/MysqlConnector.php at top of the file.

```
set_include_path('/path/to/doctrine_orm');
```

OR

```
ini_set('include_path', '/path/to/doctrine_orm');
```

Configuring PHP.ini

Include the following include_path entry in the PHP.ini file:

```
include_path = "./Path/to/ODataConnectorForMySQL/ODataConnectorForMySQL";
```

Database: Installation and Configuration

MySQL

This step is required only if we are going to use MySQL 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. The sample MySQL database for Northwind is in "ODataConnectorForMySQL\Tests".

We have to just make database named "northwind" in MySQL and import the file "northwind.sql".