FRONTEX™

Xcelerix PHP Connector

User Guide

CONTENTS

Introduction

Xcelerix PHP connector was designed to allow access to Xcelerix database from PHP scripts. To enable PHP connector, you must use **php_xlxSQLphp** module. Read the information below to see how to use this module in php scripts.

System Requirements

- PHP4 must be installed, configured and compiled with external modules support.
- Xcelerix database v.8.0.5 or later.
- The current version of *Xcelerix PHP Connector* supports only Linux OS.

Note

The $ATAROOT/bin directory path should be in the PATH environment variable and $ATAROOT/lib directory path should be in the LD_LIBRARY_PATH for Linux system and in the DYLD_LIBRARY_PATH for OSX platform.

PHP Connector Usage

- To use **xlxSQLphp** module object, load the module manually by:

```
if (!extension_loaded("php_xlxSQLphp")) {
    if (!dl("php_xlxSQLphp.so")) return;
}
```

commands.

Note

To successfully load this module, make sure that **php_xlxSQLphp** file is lying under the **PHP4** extension directory. You can find a detailed description of PHP extension modules in the relevant PHP4 documentation. Besides, **libxlxsql** library must be placed in one of the directories specified by **LD_LIBRARY_PATH** environment variable.

- To create a database access object, use the following command:

```
$dbobj = new xlxSQL;
```

- To connect to a database, set up the database environment using the following method:

```
$dbobj->SetDbEnvironment(ataroot,atabin,ataenv);
```

Params: all parameters are strings.

Return value: true - in case of success, false - otherwise.

- To get a database name, use the following method:

```
$db = $dbobj->GetDbName();
```

Return value: A string with database name or with "NULL"-string in case of an error.

- To open a connection, use the following method:

```
$dbobj->Open();
```

Return value: true - in case of success, false - otherwise.

Note

Calling this method is optional, as connection will automatically be established on first query execution. But if you wish to handle database opening errors, you should still call it.

- To check a database connection status, use the following method:

```
$dbobj->IsOpened();
```

Return value: true - if connection is opened, false - otherwise.

- To close a connection, use the following method:

```
$dbobj->Close();
```

Return value: void.

- To run a single SQL query, use the following method:

```
$dbobj->Query(query);
```

Params: <*query*> is a string, contains a completed SQL query.

Return value: true - in case of success, false - otherwise.

- To prepare SQL query for future execution (dynamic SQL query), use the following method:

  ```
  $dbobj->PrepareQuery(query);
  ```

  Params: <*query*> is a string, contains an SQL query, which optionally can contain wildcards as “?” mark. Refer to Xcelerix documentation for details.

  Return value: true - in a case of success, false - otherwise.

- To initialize a Usage array (data storage for dynamic SQL query in case of wildcards usage) and add the first data element to it, use the following method:

  ```
  $dbobj->FirstDynamicDataElement(datatype,data);
  ```

  Params:

  `<datatype> is a number, representing a Xcelerix data type. The possible values are:`

  ```
  xlxCharType_get();
  xlxShortType_get();
  xlxLongType_get();
  xlxRealType_get();
  xlxDoubleType_get();
  xlxEventType_get();   // not implemented yet.
  xlxTextType_get();    // not implemented yet.
  ```

  `<data>` is any PHP data, which can be converted to the appropriate Xcelerix data type. For example, PHP data type LONG can be converted into Xcelerix Short, Long, Real and Double data type. `<data>` can also be a PHP array, representing the data set for Xcelerix non-character fields with Count greater then 1. Note that all PHP array elements should be of appropriate type.

  Return value: true - in case of success, false - otherwise.

- To add the data element to Usage array, use the following method:

  ```
  $dbobj->AddDynamicDataElement(datatype,data);
  ```

  Params:

  <*datatype*> (see FirstDynamicDataElement method)

  <*data*> (see FirstDynamicDataElement method)

Return value: true - in case of success, false - otherwise.

- To run a prepared SQL query (if Using array is present, it will be used in a query), use the following method:

```
$dbobj->RunQuery();
```

Return value: true - in case of success, false - otherwise.

- To get an SQL execution error as a string, use the following method:

```
$dbobj->Error2String();
```

Return value: A string with error description or with "NULL"-string in case of an error.

- To check an SQL query result type, use the following method:

```
$dbobj->IsQueryResultAsTable();
```

Return value: true in case of “like table” result, false - otherwise.

- To get the number of rows in a query result, use the following method:

```
$dbobj->GetRowsCount();
```

Return value: A count of rows in a “like table” query result.

- To get the number of columns in a query result, use the following method:

```
$dbobj->GetColumnsCount();
```

Return value: A count of columns in a “like table” query result.

- To get all column names in a query result, use the following method:

```
$dbobj->GetHeader();
```

Return value: A PHP array with column names as elements.

- To fetch the first row in a query result, use the following method:

  ```
  $dbobj->FetchFirstRow();
  ```

  Return value: true - in case of success, false - otherwise.

- To fetch the next row in a query result, use the following method:

  ```
  $dbobj->FetchNextRow();
  ```

  Return value: true - in case of success, false - otherwise.

- To fetch rows by number in a query result, use the following method:

  ```
  $dbobj->FetchRowByNumber(row);
  ```

  Params: <row> is number of the required row.

  Return value: true - in a case of success, false - otherwise.

- To get all column values represented as strings in a fetched row, use the following method (it may be faster then other ones):

  ```
  $dbobj->GetRowValues_s();
  ```

  Return value: a PHP array of column values. If column value is null, the “VNULL” string will be returned as a value.

- To get all column values in a fetched row, use the following method (it may be faster then others):

  ```
  $dbobj->GetRowValues();
  ```

  Return value: a PHP array of column values. If column value is null, the “VNULL” string will be returned as a column value. If column value contains more then 1 non-character element, an array will be returned as a column value.

- To get a first column value represented as strings in a fetched row, use the following method:

  ```
  $dbobj->GetFirstColumnValue_s();
  ```

Return value: a string with column value. In case of an error, "NULL" string will be returned. If column value is null, the "VNULL" string will be returned.

- To get a first column value in a fetched row, use the following method:

```
$dbobj->GetFirstColumnValue();
```

Return value: a PHP typed column value. In case of an error, a PHP NULL will be returned. If column value is null, the "VNULL" string will be returned. If column value contains more then 1 non-character element, an array will be returned as column value.

- To get a next column value represented as strings in a fetched row, use the following method:

```
$dbobj->GetNextColumnValue_s();
```

Return value: a string with column value. In case of an error, "NULL" string will be returned. If column value is null, the "VNULL" string will be returned.

- To get a next column value in a fetched row, use the following method:

```
$dbobj->GetNextColumnValue();
```

Return value: a PHP typed column value. In case of an error, a PHP NULL will be returned. If column value is null, the "VNULL" string will be returned. If column value contains more then 1 non-character element, an array will be returned as column value.

- To get a column value by number, represented as strings in a fetched row, use the following method:

```
$dbobj->GetColumnValueByNumber_s(col);
```

Params: <*col*> is the required column number.

Return value: a string with column value. In case of an error, "NULL" string will be returned. If column value is null, the "VNULL" string will be returned.

- To get a column value by number in a fetched row, use the following method:

```
$dbobj->GetColumnValueByNumber(col);
```

Params: <*col*> is number of the required column.

Return value: a PHP typed column value. In case of an error, a PHP NULL will be returned. If column value is null, the “VNULL” string will be returned. If column value contains more then 1 non-character element, an array will be returned as column value.

- To get a column value by name in a fetched row and represented as strings, use the following method:

```
$dbobj->GetColumnValueByName_s(colname);
```

Params: <colname> is the required column name. See UseAlias() method.

Return value: a string with column values. In case of an error, “NULL” string will be returned. If column value is null, the “VNULL” string will be returned.

- To get a column value by name in a fetched row, use the following method:

```
$dbobj->GetColumnValueByName(colname);
```

Params: <colname> is the required column name. See UseAlias() method.

Return value: a PHP typed column value. In case of an error, a PHP NULL will be returned. If column value is null, the “VNULL” string will be returned. If column value contains more then 1 non-character element, an array will be returned as column value.

We can get or access a query result column names either by SQL column name or by alias, which represents a database field name. This feature applies to GetHeader() and GetColumnByName() methods. By default, SQL column names are used.

- To set a column name type (SQL column name or alias), use the following method:

```
$dbobj->UseAlias(uaflag);
```

Params: set <uaflag> to true, to use filed aliases, false to use SQL column names. Default is false.

Return value: void.

For examples of this object’s usage see xlx-php/example*.php files.

Note

The Xcelerix “Event” and “Text” data types are not supported yet.