

## **XML based database conversion routines**

---

[www.newnhams.com](http://www.newnhams.com)

**xPollinate**

**Version 1.0**

**27<sup>th</sup> November 2004**

### ***Product Overview***

XPollinate is a simple group of PHP routines that can be run against any supported database to produce a series of files containing database independent XML schema and data files. These files can be used to produce a database that is a mirror of the source database, but in a different DBMS. The following databases have been tested as both source and target databases.

- MySQL
- SQLite
- Oracle
- IBM DB2
- Microsoft SQL server

Other databases may be supported. For a list of database that may work, check the documentation at <http://adodb.sourceforge.net>, which is the home page for the database abstraction layer that is used to create the new databases.

The routines provide a method of:

- Extracting the data schema from the source database, and importing it into the target
- Extracting the data from the source database, and importing it into the target.

### ***System Requirements***

#### **Databases**

You must have the appropriate database management software installed

#### **PHP**

Download and install PHP from <http://www.php.net>. These routines only use the PHP CLI interface, so a web server is not required. However, this document doesn't help you establish a PHP connection to your chosen database. There are some resources available to help.

- Sample dbconnect.inc files in the connect\_samples directory of this distribution.
- The adodb documentation and support forum at [adodb.sourceforge.net](http://adodb.sourceforge.net)
- The PHP documentation and mailing lists
- Google
- If all else fails, email the author and I'll try to help.

### ***Licensing***

xPollinate is issued under the terms of the BSD license. This means you can incorporate it into your own software, commercial or otherwise.

### **Testing the connections**

1. In the **source** database, ensure that there is a user with appropriate privileges to read both the data and schema files.
2. In the **target** database, create an empty database with an appropriate name, and ensure that there is a user with appropriate privileges to issue both **CREATE TABLE** and **INSERT** statements in that database.
3. Modify the `source_connect.inc` file to reflect your required connection to the source database
4. Modify the `target_connect.inc` file to reflect your required connection to the target database
5. In the test directory, run the `source.php` file using the following syntax:
  - `<path_to_php_cli> source.php`
6. The program should return a list of all the defined tables in the source database. If it does not, you should fix the problem before proceeding.
7. In the test directory, run the `target.php` file using the following syntax:
  - `<path_to_php_cli> target.php`
8. The program will attempt to create a table called **test** in the target database. If it does not, you should fix the problem before continuing. If it has created the table, you should **DROP** the table before continuing.

If you have completed the above section, you can continue on to the schema transfer.

### **Transferring the schema.**

#### **Dumping the schema from the target database**

run `<path_to_php_cli> dumpschema.php <schema file>`. The schema will be dumped into an XML format file specified by the parameter `<schema file>`. You can review the file in any XML editor or Browser.

#### **Loading the schema into the target database.**

run `<path_to_php_cli> loadschema.php <schema file>`. The schema will be loaded from the XML format file specified by the parameter `<schema file>`, created earlier with `loadschema.php`.

Once the procedure has terminated, check the target database to ensure that the schema has been successfully recreated. No further work is required.

#### **Field and File Name conversion**

The system makes no check if the field/file name to be transferred is a reserved word in the target database. Attempts to convert these will result in an error when the schema is being transferred.

#### **Debugging**

Change the line `DEFINE('XMLS_DEBUG',false)` in `dumpschema.php` or `loadschema.php` to `true`. This will produce copious debugging output from both the schema processor and database handler.

## Transferring the data

## Chapter

# 3

### **Dumping the data from the target database**

run `<path_to_php_cli> dumpdata <dump directory> [table]`. The data will be dumped into XML format files in the directory specified by the parameter `<dump directory>`. You can optionally specify a table name from the source directory to only dump that data.

Once the procedure has terminated, check the target database to ensure that the schema has been successfully recreated. No further work is required.

### **OK – So why did it dump all these XML datafiles?**

There is a limitation in the XML handler, where it loads all of the data into memory first before processing it. Files bigger than about 500k (approx) cause the load routine to blow up. Don't worry, however, the load program knows which order to load them.

### **Loading the schema into the target database.**

run `<path_to_php_cli> loaddata.php <dump directory> [table]`. If you do not specify a table, the program will attempt to load data from the dump directory for all tables that exist in the target database.

### **OK – So why is it loading so slow?**

The load procedure isn't optimized in any way, It just uses a straight SQL 'INSERT' statement for every record. That means it will work across the maximum number of databases.

## ***The Rest***

That's it ! enjoy, report bugs and successful conversions as well as suggested improvements to [mark@newnhams.com](mailto:mark@newnhams.com). Thanks to the ADODB, and ADODB-XMLSCHEMA projects for such great work