dbx

Introduction

The dbx module is a database abstraction layer (db 'X', where 'X' is a supported database). The dbx functions allow you to access all supported databases using a single calling convention. The dbx-functions themselves do not interface directly to the databases, but interface to the modules that are used to support these databases.

Note
This extension has been moved to the » PECL repository and is no longer bundled with PHP as of PHP 5.1.0.

Installing/Configuring

Requirements

To be able to use a database with the dbx-module, the module must be either linked or loaded into PHP, and the database module must be supported by the dbx-module. Currently, the following databases are supported, but others will follow:

- FrontBase (available from PHP 4.1.0).
- Microsoft SQL Server
- MySQL
- ODBC
- PostgreSQL
- Sybase-CT (available from PHP 4.2.0).
- Oracle (oci8) (available from PHP 4.3.0).
- SQLite (PHP 5).

Documentation for adding additional database support to dbx can be found at » http://www.guidance.nl/php/dbx/doc/.

Installation

In order to have these functions available, you must compile PHP with dbx support by using the *--enable-dbx* option and all options for the databases that will be used, e.g. for MySQL you must also specify *--with-mysql=[DIR]*. To get other supported databases to work with the dbx-module refer to their specific documentation.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

DBX Configuration Options

Name	Default	Changeable	Changelog
dbx.colnames_case	"unchanged"	PHP_INI_SYSTEM	Available since PHP 4.3.0. Removed in PHP 5.1.0.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

dbx.colnames_case string
: Columns names can be returned "unchanged" or converted to "uppercase" or "lowercase". This directive can be overridden with a flag to dbx_query().

Resource Types

There are two resource types used in the dbx module. The first one is the link- object for a database connection, the second a result- object which holds the result of a query.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

DBX_MYSQL (integer)

DBX_ODBC (integer)

DBX_PGSQL (integer)

DBX_MSSQL (integer)

DBX_FBSQL (integer)

DBX_OCI8 (integer) (available from PHP 4.3.0)

DBX_SYBASECT (integer)

DBX_SQLITE (integer) (PHP 5)

DBX_PERSISTENT (integer)

DBX_RESULT_INFO (integer)

DBX_RESULT_INDEX (integer)

DBX_RESULT_ASSOC (integer)

DBX_RESULT_UNBUFFERED (integer) (PHP 5)

DBX_COLNAMES_UNCHANGED (integer) (available from PHP 4.3.0)

DBX_COLNAMES_UPPERCASE (integer) (available from PHP 4.3.0)

DBX_COLNAMES_LOWERCASE (integer) (available from PHP 4.3.0)

DBX_CMP_NATIVE (integer)

DBX_CMP_TEXT (integer)

DBX_CMP_NUMBER (integer)

DBX_CMP_ASC (integer)

DBX_CMP_DESC (integer)

dbx Functions

dbx_close

dbx_close -- Close an open connection/database

Description

int **dbx_close** (object `$link_identifier`)

Parameters

`link_identifier`
The DBX link object to close.

Return Values

Returns 1 on success and 0 on errors.

Examples

Example #1 - dbx_close() example

```
<?php
$link = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password")
   or die("Could not connect");

echo "Connected successfully";
dbx_close($link);
?>
```

Notes

Note

Always refer to the module-specific documentation as well.

See Also

- dbx_connect()

dbx_compare

dbx_compare -- Compare two rows for sorting purposes

Description

int **dbx_compare** (array `$row_a`, array `$row_b`, string `$column_key` [, int `$flags`])

dbx_compare() is a helper function for dbx_sort() to ease the make and use of the custom sorting function.

Parameters

`row_a`
: First row

`row_b`
: Second row

`column_key`
: The compared column

`flags`
: The *`flags`* can be set to specify comparison direction:

  - **DBX_CMP_ASC** - ascending order
  - **DBX_CMP_DESC** - descending order

  and the preferred comparison type:

  - **DBX_CMP_NATIVE** - no type conversion
  - **DBX_CMP_TEXT** - compare items as strings
  - **DBX_CMP_NUMBER** - compare items numerically

  One of the direction and one of the type constant can be combined with bitwise OR operator (|). The default value for the *`flags`* parameter is **DBX_CMP_ASC** | **DBX_CMP_NATIVE**.

Return Values

Returns *0* if the *row_a[$column_key]* is equal to *row_b[$column_key]*, and *1* or *-1* if the former is greater or is smaller than the latter one, respectively, or vice versa if the *`flag`* is set to **DBX_CMP_DESC**.

Examples

Example #2 - dbx_compare() example

```
<?php
function user_re_order($a, $b)
{
   $rv = dbx_compare($a, $b, "parentid", DBX_CMP_DESC);
   if (!$rv) {
       $rv = dbx_compare($a, $b, "id", DBX_CMP_NUMBER);
   }
   return $rv;
}

$link   = dbx_connect(DBX_ODBC, "", "db", "username", "password")
   or die("Could not connect");

$result = dbx_query($link, "SELECT id, parentid, description FROM table
ORDER BY id");
   // data in $result is now ordered by id

dbx_sort($result, "user_re_order");
   // date in $result is now ordered by parentid (descending), then by id

dbx_close($link);
?>
```

See Also

- dbx_sort()

dbx_connect

dbx_connect -- Open a connection/database

Description

object **dbx_connect** (mixed `$module`, string `$host`, string `$database`, string `$username`, string `$password` [, int `$persistent`])

Opens a connection to a database.

Parameters

`module`
The *`module`* parameter can be either a string or a constant, though the latter form is preferred. The possible values are given below, but keep in mind that they only work if the module is actually loaded.

- **DBX_MYSQL** or "mysql"
- **DBX_ODBC** or "odbc"
- **DBX_PGSQL** or "pgsql"
- **DBX_MSSQL** or "mssql"
- **DBX_FBSQL** or "fbsql" (available from PHP 4.1.0)
- **DBX_SYBASECT** or "sybase_ct" (available from PHP 4.2.0)
- **DBX_OCI8** or "oci8" (available from PHP 4.3.0)
- **DBX_SQLITE** or "sqlite" (PHP 5)

`host`
The SQL server host

`database`
The database name

`username`
The username

`password`
The password

`persistent`
The *`persistent`* parameter can be set to **DBX_PERSISTENT**, if so, a persistent connection will be created.

The *host*, *database*, *username* and *password* parameters are expected, but not always used depending on the connect functions for the abstracted module.

Return Values

Returns an object on success, **FALSE** on error. If a connection has been made but the database could not be selected, the connection is closed and **FALSE** is returned.

The returned *object* has three properties:

database
: It is the name of the currently selected database.

handle
: It is a valid handle for the connected database, and as such it can be used in module-specific functions (if required).

```
<?php
$link = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password");
mysql_close($link->handle); // dbx_close($link) would be better here
?>
```

module
: It is used internally by dbx only, and is actually the module number mentioned above.

Examples

Example #3 - dbx_connect() example

```
<?php
$link = dbx_connect(DBX_ODBC, "", "db", "username", "password",
DBX_PERSISTENT)
   or die("Could not connect");

echo "Connected successfully";
dbx_close($link);
?>
```

Notes

Note

Always refer to the module-specific documentation as well.

See Also

- dbx_close()

dbx_error

dbx_error -- Report the error message of the latest function call in the module

Description

string **dbx_error** (object `$link_identifier`)

dbx_error() returns the last error message.

Parameters

`link_identifier`
The DBX link object returned by dbx_connect()

Return Values

Returns a string containing the error message from the last function call of the abstracted module (e.g. mysql module). If there are multiple connections in the same module, just the last error is given. If there are connections on different modules, the latest error is returned for the module specified by the *`link_identifier`* parameter.

Examples

Example #4 - dbx_error() example

```
<?php
$link   = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password")
   or die("Could not connect");

$result = dbx_query($link, "select id from non_existing_table");
if ($result == 0) {
   echo dbx_error($link);
}
dbx_close($link);
?>
```

Notes

Note

Always refer to the module-specific documentation as well.

The error message for Microsoft SQL Server is actually the result of the mssql_get_last_message() function.

The error message for Oracle (oci8) is not implemented yet.

dbx_escape_string

dbx_escape_string -- Escape a string so it can safely be used in an sql-statement

Description

string **dbx_escape_string** (object `$link_identifier`, string `$text`)

Escape the given string so that it can safely be used in an sql-statement.

Parameters

`link_identifier`
The DBX link object returned by dbx_connect()

`text`
The string to escape.

Return Values

Returns the text, escaped where necessary (such as quotes, backslashes etc). On error, **NULL** is returned.

Examples

Example #5 - dbx_escape_string() example

```
<?php
$link   = dbx_connect(DBX_MYSQL, "localhost", "db", "username", "password")
   or die("Could not connect");

$text = dbx_escape_string($link, "It\'s quoted and backslashed (\\).");
$result = dbx_query($link, "insert into tbl (txt) values ('" . $text .
"')");
if ($result == 0) {
   echo dbx_error($link);
}
dbx_close($link);
?>
```

See Also

- dbx_query()

dbx_fetch_row

dbx_fetch_row -- Fetches rows from a query-result that had the **DBX_RESULT_UNBUFFERED** flag set

Description

mixed **dbx_fetch_row** (object `$result_identifier`)

dbx_fetch_row() fetches rows from a result identifier that had the **DBX_RESULT_UNBUFFERED** flag set.

When the **DBX_RESULT_UNBUFFERED** is not set in the query, dbx_fetch_row() will fail as all rows have already been fetched into the results data property.

As a side effect, the rows property of the query-result object is incremented for each successful call to dbx_fetch_row().

Parameters

`result_identifier`
A result set returned by dbx_query().

Return Values

Returns an object on success that contains the same information as any row would have in the dbx_query() result data property, including columns accessible by index or fieldname when the flags for dbx_query() were set that way.

Upon failure, returns *0* (e.g. when no more rows are available).

Examples

Example #6 - How to handle the returned value

```
<?php
$result = dbx_query($link, 'SELECT id, parentid, description FROM table',
DBX_RESULT_UNBUFFERED);

echo "<table>\n";
while ($row = dbx_fetch_row($result)) {
   echo "<tr>\n";
   foreach ($row as $field) {
       echo "<td>$field</td>";
   }
   echo "</tr>\n";
```

```
}
echo "</table>\n";
?>
```

See Also

- dbx_query()

dbx_query

dbx_query -- Send a query and fetch all results (if any)

Description

mixed **dbx_query** (object `$link_identifier`, string `$sql_statement` [, int `$flags`])

Sends a query and fetch all results.

Parameters

`link_identifier`
The DBX link object returned by dbx_connect()

`sql_statement`
SQL statement.

`flags`
The *`flags`* parameter is used to control the amount of information that is returned. It may be any combination of the following constants with the bitwise OR operator (|). The DBX_COLNAMES_* flags override the dbx.colnames_case setting from *php.ini*.

DBX_RESULT_INDEX
It is *always* set, that is, the returned object has a data property which is a 2 dimensional array indexed numerically. For example, in the expression *data[2][3] 2* stands for the row (or record) number and *3* stands for the column (or field) number. The first row and column are indexed at 0. If **DBX_RESULT_ASSOC** is also specified, the returning object contains the information related to **DBX_RESULT_INFO** too, even if it was not specified.

DBX_RESULT_INFO
It provides info about columns, such as field names and field types.

DBX_RESULT_ASSOC
It effects that the field values can be accessed with the respective column names used as keys to the returned object's data property. Associated results are actually references to the numerically indexed data, so modifying *data[0][0]* causes that *data[0]['field_name_for_first_column']* is modified as well.

DBX_RESULT_UNBUFFERED (PHP 5)
This flag will not create the data property, and the rows property will initially be 0. Use this flag for large datasets, and use dbx_fetch_row() to retrieve the results row by row. The dbx_fetch_row() function will return rows that are conformant to the flags set with this query. Incidentally, it will also update the rows each time it is called.

DBX_COLNAMES_UNCHANGED (available from PHP 4.3.0)
The case of the returned column names will not be changed.

DBX_COLNAMES_UPPERCASE (available from PHP 4.3.0)
The case of the returned column names will be changed to uppercase.

DBX_COLNAMES_LOWERCASE (available from PHP 4.3.0)
The case of the returned column names will be changed to lowercase.

Note that **DBX_RESULT_INDEX** is always used, regardless of the actual value of *flags* parameter. This means that only the following combinations are effective:

- **DBX_RESULT_INDEX**
- **DBX_RESULT_INDEX** | **DBX_RESULT_INFO**
- **DBX_RESULT_INDEX** | **DBX_RESULT_INFO** | **DBX_RESULT_ASSOC** - this is the default, if *flags* is not specified.

Return Values

dbx_query() returns an object or *1* on success, and *0* on failure. The result object is returned only if the query given in *sql_statement* produces a result set (i.e. a SELECT query, even if the result set is empty).

The returned *object* has four or five properties depending on *flags*:

handle
It is a valid handle for the connected database, and as such it can be used in module specific functions (if required).

```
<?php
$result = dbx_query($link, "SELECT id FROM table");
mysql_field_len($result->handle, 0);
?>
```

cols and rows
These contain the number of columns (or fields) and rows (or records) respectively.

```
<?php
$result = dbx_query($link, 'SELECT id FROM table');
echo $result->rows; // number of records
echo $result->cols; // number of fields
?>
```

info (optional)
It is returned only if either **DBX_RESULT_INFO** or **DBX_RESULT_ASSOC** is specified in the *flags* parameter. It is a 2 dimensional array, that has two named rows (*name* and *type*) to retrieve column information.

Example #7 - lists each field's name and type

```
<?php
```

```
$result = dbx_query($link, 'SELECT id FROM table',
                    DBX_RESULT_INDEX | DBX_RESULT_INFO);

for ($i = 0; $i < $result->cols; $i++ ) {
   echo $result->info['name'][$i] . "\n";
   echo $result->info['type'][$i] . "\n";
}
?>
```

data

This property contains the actual resulting data, possibly associated with column names as well depending on *`flags`*. If **DBX_RESULT_ASSOC** is set, it is possible to use *$result->data[2]["field_name"]*.

Example #8 - outputs the content of data property into HTML table

```
<?php
$result = dbx_query($link, 'SELECT id, parentid, description FROM
table');

echo "<table>\n";
foreach ($result->data as $row) {
   echo "<tr>\n";
   foreach ($row as $field) {
       echo "<td>$field</td>";
   }
   echo "</tr>\n";
}
echo "</table>\n";
?>
```

Example #9 - How to handle UNBUFFERED queries

```
<?php

$result = dbx_query ($link, 'SELECT id, parentid, description FROM
table', DBX_RESULT_UNBUFFERED);

echo "<table>\n";
while ($row = dbx_fetch_row($result)) {
   echo "<tr>\n";
   foreach ($row as $field) {
       echo "<td>$field</td>";
   }
   echo "</tr>\n";
}
echo "</table>\n";

?>
```

Examples

Example #10 - How to handle the returned value

```
<?php
$link   = dbx_connect(DBX_ODBC, "", "db", "username", "password")
   or die("Could not connect");

$result = dbx_query($link, 'SELECT id, parentid, description FROM table');

if (is_object($result) ) {
   // ... do some stuff here, see detailed examples below ...
   // first, print out field names and types
   // then, draw a table filled with the returned field values
} else {
   exit("Query failed");
}

dbx_close($link);
?>
```

Notes

Note

Always refer to the module-specific documentation as well.

Column names for queries on an Oracle database are returned in lowercase.

See Also

- dbx_escape_string()
- dbx_fetch_row()
- dbx_connect()

dbx_sort

dbx_sort -- Sort a result from a dbx_query by a custom sort function

Description

bool **dbx_sort** (object `$result`, string `$user_compare_function`)

Sort a result from a dbx_query() call with a custom sort function.

Parameters

`result`
A result set returned by dbx_query().

`user_compare_function`
The user-defined comparison function. It must accept two arguments and return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #11 - dbx_sort() example

```
<?php
function user_re_order($a, $b)
{
   $rv = dbx_compare($a, $b, "parentid", DBX_CMP_DESC);
   if (!$rv) {
       $rv = dbx_compare($a, $b, "id", DBX_CMP_NUMBER);
   }
   return $rv;
}

$link   = dbx_connect(DBX_ODBC, "", "db", "username", "password")
   or die("Could not connect");

$result = dbx_query($link, "SELECT id, parentid, description FROM tbl ORDER
BY id");
   // data in $result is now ordered by id

dbx_sort($result, "user_re_order");
   // data in $result is now ordered by parentid (descending), then by id
```

```
dbx_close($link);
?>
```

Notes

Note
It is always better to use *ORDER BY SQL* clause instead of dbx_sort() if possible.

See Also

- dbx_compare()