Bs_Db:
select bells, whistles from database

Andrej Arn
Sam Blume

BlueShoes

THIS IS A STEP-BY-STEP GUIDE THAT EXPLAINS HOW TO USE BS_DB AND THE SUBCLASSES LIKE BS_MYSQL ETC. IN THIS EXAMPLE WE USE MYSQL, BUT YOU COULD ALSO USE THE OTHERS (ORACLE AND MSSQL). THE MYSQL IMPLEMENTATION IS THE ONE WITH THE MOST FEATURES BUILT IN.

Bells and whistles vs. supporting all RDBMS vendors:
In English: limited number of supported db servers vs. limited set of features.

In theory it sounds amazing to be able to switch the database system in your existing software, without a single code change. A db layer like ADOdb [1] helps you. The big drawback is that, if you want to be able to do that, you need to use the smallest common multiple of all features. And that is not much. Some missing db features can be simulated by code, but such hacks won't make it faster.

In addition, there are so many different syntaxes for the same features. This would mean to drop these features as well, or to fork in your code based on RDBMS used.

We recommend choosing a db vendor, and coding for it. If you once have to import data from another system, or access another system, use a general abstraction layer like ADOdb [1].

What you need:
- An existing installation of PHP/MySQL/Apache (or IIS) and ***BlueShoes***.
- A website running ***BlueShoes***, with an existing MySQL database.
- Some knowledge about PHP, understanding of how to use existing classes. Some knowledge of how to use SQL/MySQL.

You can find the full source here: core/db/examples/mysql.php
The file may be helpful if you get stuck or to copy/paste.

At first you need an instance of Bs_Db. That gets done in your global.conf.php, but maybe you deactivated it cause you don't need it for everything. Here it is again with comments:

```
1.  <?php
2.  require_once($_SERVER["DOCUMENT_ROOT"]    . "../global.conf.php");
3.  include_once($APP['path']['core/Db']      . 'Bs_MySql.class.php');
4.
5.  $dbSettings['name']         = "yourdatabasename";
6.  $dbSettings['host']         = 'localhost';
7.  $dbSettings['port']         = '3306';
8.  $dbSettings['user']         = 'username';
9.  $dbSettings['pass']         = 'password';
10. $dbSettings['socket']       = '';
11. $dbSettings['syntax']       = 'mysql';
12. $dbSettings['type']         = 'mysql';
13. $dbSettings['transactions'] = FALSE;
14.
15. $isOk = FALSE;
16. do {
17. $bsDb =& new Bs_MySql();
18. $connId = $bsDb->connect($dbSettings);
19. if ($bsDb->isException($connId)) {
20. //crap. connection failed.
21. $connId->stackTrace('', __FILE__, __LINE__, 'fatal');
22. $connId->stackDump('alert');
23. break;
24. }
25.
26. //oki, we have a valid db connection now. our instance is $bsDb.
27. $isOk = TRUE;
28. } while (FALSE);
29.
30. if ($isOk) {
31. echo 'everything successful.';
32. }
33. ?>
```

At first we define an array ($dbSettings) with the connection information. Then we create a new instance, connect, and check for errors.

The do { ... } while (FALSE) together with the status var $isOk is a java-like try-catch structure. It's not absolutely necessary to check for exceptions after each method call. It blows up your code and makes it a little bit slower. But if you want secure code, do it. The db server could die, whatever, there are always things that can go wrong.

Now let's create a table. Don't worry, it will be removed later. There won't stay any junk. Add these lines between line 26 and 27 (right before the $isOk = TRUE;)

```
$sql = "
  CREATE TABLE IF NOT EXISTS test.BsMysqlTest(
    ID          INT NOT NULL DEFAULT 0 AUTO_INCREMENT,
    firstField   VARCHAR(30) NOT NULL DEFAULT '',
    secondField  VARCHAR(30) NOT NULL DEFAULT '',
    PRIMARY KEY ID (ID)
  )
";
$status = $bsDb->write($sql);
```

Go and run that script. You should see the message on your screen: "everything successful." If you get an exception dump, check the connection settings in $dbSettings.

Now insert the following code (again right before that $isOk line):

```
$sql = "INSERT INTO test.BsMysqlTest (firstField, secondField) VALUES ('first',
'second')";
$newId = $bsDb->idWrite($sql);
```

```
if ($bsDb->isException($newId)) {
  $newId->stackTrace('', __FILE__, __LINE__, 'fatal');
  $newId->stackDump('alert');
  break;
} else {
  echo "Inserted record with new ID: {$newId}<br><br>";
}
```

So write() just sends the given sql code to the database and returns a status, while idWrite() is used to insert records and automatically get the new id back. If an error occures, we get an exception back in both cases.

Run the code again.
Note: each time you run the code there is 1 record more in the database.

Now go on with this one:

```
$sql = "UPDATE test.BsMysqlTest SET firstField='FIRST'";
$numRecs = $bsDb->countWrite($sql);
if ($bsDb->isException($numRecs)) {
  $numRecs->stackTrace('', __FILE__, __LINE__, 'fatal');
  $numRecs->stackDump('alert');
  break;
} else {
  echo "Updated {$numRecs} record(s)<br><br>";
}
```

The method countWrite() is useful to see how many records were touched (affected). It can also be used with a DELETE or REPLACE query.

Now we go on with countRead() which tells us how many records were selected by a SELECT query.

```
$sql = "SELECT * FROM  test.BsMysqlTest";
$numRecs = $bsDb->countRead($sql);
if ($bsDb->isException($numRecs)) {
  $numRecs->stackTrace('', __FILE__, __LINE__, 'fatal');
  $numRecs->stackDump('alert');
  break;
} else {
  echo "Selected {$numRecs} record(s)<br><br>";
}
```

Our table now looks like this (maybe you have another number of records, that doesn't matter):

```
mysql> select * from BsMysqlTest;
+----+------------+-------------+
| ID | firstField | secondField |
+----+------------+-------------+
|  1 | FIRST      | second      |
|  2 | FIRST      | second      |
|  3 | FIRST      | second      |
|  4 | FIRST      | second      |
+----+------------+-------------+
4 rows in set (0.00 sec)
```

Now we're gonna select data in different ways. The Db class offers some useful methods to get data in the structure we want, using just one method call.

getOne() offers to get just one cell value, like this:

```
          \/
   +----+------------+-------------+
   | ID | firstField | secondField |
   +----+------------+-------------+
   |  1 | FIRST      | second      |
=> |  2 | FIRST      | second      | <=
   |  3 | FIRST      | second      |
   |  4 | FIRST      | second      |
   +----+------------+-------------+
          /\
```

The code:

```
$sql = "SELECT firstField FROM test.BsMysqlTest WHERE ID = 2";
$data = $bsDb->getOne($sql);
if ($bsDb->isException($data)) {
  $data->stackTrace('', __FILE__, __LINE__, 'fatal');
  $data->stackDump('alert');
  break;
} else {
  dump($data);
}
```

The result on your screen:

```
string(5) "FIRST"
```

getRow () offers to get one full row, like this:

```
   +----+------------+-------------+
   | ID | firstField | secondField |
   +----+------------+-------------+
   |  1 | FIRST      | second      |
```

```
=> |  2 | FIRST      | second      | <=
   |  3 | FIRST      | second      |
   |  4 | FIRST      | second      |
   +----+------------+-------------+
```

Here's the code:

```
$sql = "SELECT * FROM test.BsMysqlTest WHERE ID = 2";
$data = $bsDb->getRow($sql);
if ($bsDb->isException($data)) {
  $data->stackTrace('', __FILE__, __LINE__, 'fatal');
  $data->stackDump('alert');
  break;
} else {
  dump($data);
}
```

And that's what you get on your screen:

```
array(3) {
  ["ID"]=>string(1) "2"
  ["firstField"]=>string(5) "FIRST"
  ["secondField"]=>string(6) "second"
}
```

If you prefer to have a vector instead of a hash, you can pass the constant BS_DB_FETCHMODE_ORDERED as 3rd argument to getRow(). Default is always BS_DB_FETCHMODE_ASSOC. It's always useful to have the field names, imo.

getCol() does the same with a column, like this:

```
          \/
   +----+------------+-------------+
   | ID | firstField | secondField |
   +----+------------+-------------+
   |  1 | FIRST      | second      |
   |  2 | FIRST      | second      |
   |  3 | FIRST      | second      |
   |  4 | FIRST      | second      |
   +----+------------+-------------+
          /\
```

The code:

```
$sql = "SELECT firstField FROM test.BsMysqlTest WHERE ID < 5";
$data = $bsDb->getCol($sql);
if ($bsDb->isException($data)) {
  $data->stackTrace('', __FILE__, __LINE__, 'fatal');
  $data->stackDump('alert');
  break;
} else {
  dump($data);
}
```

The result:

```
array(4) {
  [0]=>string(5) "FIRST"
  [1]=>string(5) "FIRST"
  [2]=>string(5) "FIRST"
  [3]=>string(5) "FIRST"
}
```

In that getCol() example we don't know the ID. The Array index (0-3) ist just a numeric value. If we want the ID, we're going to use the next one.

getAssoc() fetches the entire data into a hash using the first column as the key. Btw: if you want more information about these methods, go and read the manual inside the class itself. It's well documented there. This is just to get you going.

The code:

```
$sql = "SELECT * FROM test.BsMysqlTest WHERE ID < 3";
$data = $bsDb->getAssoc($sql);
if ($bsDb->isException($data)) {
  $data->stackTrace('', __FILE__, __LINE__, 'fatal');
  $data->stackDump('alert');
  break;
} else {
  dump($data);
}
```

The output:

```
array(2) {
  [1]=>array(2) {
    [0]=>  string(5) "FIRST"
    [1]=>  string(6) "second"
  }
  [2]=>array(2) {
    [0]=>  string(5) "FIRST"
    [1]=>  string(6) "second"
  }
}
```

If you change the call to getAssoc() to getAssoc($sql, TRUE, TRUE)

Your output will be

```
array(2) {
  [1]=>array(2) {
    ["firstField"]=>  string(5) "FIRST"
    ["secondField"]=>  string(6) "second"
  }
  [2]=>array(2) {
    ["firstField"]=>  string(5) "FIRST"
    ["secondField"]=>  string(6) "second"
  }
}
```

There are also getAssoc2() and getAll() and some others to fetch data, and tons of other methods, for example to read table and db information.

References

1 ADOdb, see http://php.weblogs.com/ADOdb