Bs_SimpleObjPersister:
make your objects persistable

Sam Blume
Andrej Arn

blueshoes

SAY YOUR APPLICATION MANAGES ADDRESSES OF EMPLOYEES. INSTEAD OF FETCHING THE DATA OF EMPLOYEES EVERYWHERE IN YOUR CODE WHERE YOU NEED SOME INFORMATION, IT IS WISE TO CREATE AN 'EMPLOYEE' CLASS YOU CAN LOAD AND STORE. THIS WAY A CHANGE TO THE DATA STRUCTURE FOR EXAMPLE CAN BE DONE VERY EASILY, ONLY YOUR CLASS NEEDS A SMALL CHANGE.

Is able to store and load a PHP Object to and from a DB.

NOTE: Only MySQL is currently supported. Although the class is designed to use a generic Bs-Db-Agent, the only interface coded so fare, that supports the full interface, is Bs_MySql.class.php.

Source Location:

Class: core/storage/objectpersister/Bs_SimpleObjPersister.class.php
Example: core/storage/objectpersister/examples/simpleObjPersister.php

Featurs:

- No major modifications of the class that is to be persisted. Integration is fairly transparent and painless.
- Creates the table in the DB **on the fly** (if table is missing) and creates the corresponding table columns for the class attributes that will be stored (=persisted).
- Support for unique key names (string is generated by PHP's uniqid().)
- During development, you can add new class attributes and the table columns are added **automatically** to the table. (For security reasons no columns are deleted. Unneeded created columns must be deleted manually)
- During development, you can upgrade an attributes to an other type E.g. string(20) to string(40) / integer to blob a.s.o. (For security reasons downgrading must be done manually)
- Callback methods to the persisted object before and after each load/store/delete. *Callback methods are methods to the persisted object that are called before and after a store or load of the object. It's possible to handle relational things this way.*
- Strong type-checking. (Can be turned off to save the CPU overhead but is on per default, good while coding and for debugging).
- Plugable crypting function. (With integrated default crypter).
- Support for indexes

Prepare Your Object to be persistable:

Let us assume we had the following class called 'Foo' that we'd like to be persistable.
NOTE: As result we want only the first 2 class attributes to be persisted; '$dummy' is *not* to be persisted in this sample:

```
class Foo {
  var $attr_1 = 'A string to persist';  // persist (string)
  var $attr_2 = 0;                      // persist (integer)

  var $dummy = '';                      // don't persist
}
```

A) The fist thing we have to do is to identify the attribute in the class that we'll be using as ID (also referred to as the "Primary Key"). If your class hasn't got an ID (like the sample above), then you must add one. Let's add the attribute **$ID**.

```
class Foo {
  var $ID = 0; // Following attribute will be our ID (primary key)

  var $attr_1 = 'A string to persist';  // persist (string)
  var $attr_2 = 0;                      // persist (integer)

  var $dummy = '';                      // don't persist
}
```

B) We now need a way to tell the Bs_SimpleObjPersister which attributes are to be saved. And how we want the data to be stored into the DB (e.g. as string or integer, with or without index, cryptified or readable a.s.o.).

This is accomplished be adding the '**callback**'-method **bs_sop_getHints()** to your class, that has to return the persist hint-hash.

```
class Foo {
  var $ID = 0; // Following attribute will be our ID (primary key)

  var $attr_1 = 'A string to persist';  // persist (string)
  var $attr_2 = 0;                      // persist (integer)

  var $dummy = '';                      // don't persist

  function bs_sop_getHints() {
    static $hint_hash = array (
        'primary' => array (
           'ID' => array('name'=>'id', 'type'=>'auto_increment'),
         ),
        'fields' => array (
           'attr_1' => array('name'=>'str', 'metaType'=>'string', 'size'=>40 ),
           'attr_2' => array('name'=>'num', 'metaType'=>'integer'),
         )
      );
     return $hint_hash;
  }
}
```

That's it! All instances of class 'Foo' are persistable now (read on).

Set-up the SimpleObjPersister:

Now let's set-up the SimpleObjPersister. If you have **one** database to connect too, you will only need **one** SimpleObjPersister instance; the rule is one SimpleObjPersister for each database.

First we need an interface to the DB. This is accomplished by creating a so called BS-DB-Agent. The BS DB-Agent is created using the 'factory'-function getDbObject(). In the following case an instance of Bs_MySql.class.php will be returned as DB-Agent. It is then assigning it to the SimpleObjPersister with setDbObject().

```
//------------------------------------------------------------------------------
// A) First thing we do here, is to include the DB-Factory that will
//    return us a DB-Agent. The Bs_SimpleObjPersister needs a DB-Agent to
//    'talk' to the underlying DB.
require_once($_SERVER['DOCUMENT_ROOT']       . '../global.conf.php');
require_once($GLOBALS['APP']['path']['core'] . 'db/Bs_Db.class.php');

//------------------------------------------------------------------------------
// B) Then setup the connection parameters, that we want to pass to the
//    DB-Factory. In this case it's going to return a mySQL DB-Agent
$dsn = array (
  'name'=>'test', 'host'=>'localhost', 'port'=>'3306', 'socket'=>'',
  'user'=>'root', 'pass'=>'', 'syntax'=>'mysql', 'type'=>'mysql'
);

//------------------------------------------------------------------------------
// C) Get the DB-Agent. If the return is a Bs_Exception echo error and die.
if (isEx($dbAgent =& getDbObject($dsn))) {
  $dbAgent->stackDump('echo');
  die();
}

//------------------------------------------------------------------------------
// D) Now create the Bs_SimpleObjPersister and pass the DB-Agent.
//    It's now ready to be used ...
require_once($GLOBALS['APP']['path']['core'] .
  'storage/objectpersister/Bs_SimpleObjPersister.class.php');

$objPersister = new Bs_SimpleObjPersister();
$objPersister->setDbObject($dbAgent);
```

Using the SimpleObjPersister:

We are now ready to persist any object that has been made persistable like the class 'Foo'.
Let's try to **store** an instance of 'Foo':

```
$myFoo = new Foo();
$myFoo->attr_1 = 'This is new!';
$objPersister->store($myFoo);
```

If MySql was setup in the default manner, you should see a new table called 'Foo' in the test database of MySql. The table was created on the fly! It will contain 3 columns: ID (auto_increment), attr_1 (string 40) and attr_2 (int) with 1 record.

Now let us try to **load** the object we just stored:

```
$myFoo = new Foo();
$myFoo->ID = 1;
if ($objPersister->load($myFoo)) {
  echo $myFoo->attr_1;          // Will print 'This is new!';
}
```

Besides a singular load there are also following methods to load an array of objects :

- loadByIds($classname, $ids)
- loadByWhere($classname, $whereClause)
- loadAll($classname)

The first parameter is either the class name (string) or an instance of the class to load. As return you'll receive a hash-array of loaded objects where the primary-key value is used as hash-key. On ERROR you'll receive FALSE and can fetch the errors using
$objPersister->getLastErrors()

Where to go from here:

This was only a very basic example. The SimpleObjPersister can do much more.
A good idea is to check the API doc, available at http://developer.blueshoes.org/phpdoc/

If you want to make your objects not only persistable but also editable, check the FormItAble class and its HOWTO: http://www.blueshoes.org/howtos/Bs_FormItAble.howTo.pdf