
BlueShoes Concepts For:

error & exception handling

Andrej Arn
Sam Blume

Version:
1.0, updated 2004-04-11



blueshoes

Intro:

Murphy's Law: everything that can happen will happen.

There are many reasons for exceptional circumstances. Full disk, no write permissions, connections maxed out to name a few.

It is important for every piece of software to have secure code where errors are caught. At the same time the code should still be readable and not blown up with 2/3 error catching code.

Index:

1. try/catch in PHP4
2. bool FALSE and a text error string
3. exceptions
4. throw php warnings and errors
5. additional reading

Try/catch in PHP4

Doesn't exist? Right. Only PHP5 comes with this very nice control flow structure. But it can be faked more or less.

PHP5 Code:

```
try {
    doCriticalThing();
    doOtherCriticalThing();
    //...
} catch ($e) {
    //do something.
}
```

PHP4 Code:

```
$isOk = FALSE;
do {
    if (!doCriticalThing()) break;
    if (!doOtherCriticalThing()) break;
    //...
    $isOk = TRUE;
} while (FALSE);
if (!$isOk) {
    //do something.
}
```

We use a do-while block – although it looks like a loop, it's not, it'll never loop. It's the do-block we are using to surround the code as substitute for the missing try-block in PHP4. On the first occurrence of an error we break out of the do-block (similar to try-catch) run into the if-statement (that is used instead of the catch). But the main difference is we won't have an exception object to pass on. We'll see another example further down that will return a Bs_Exception.

bool FALSE and a Text Error String

Often it makes sense to just return boolean FALSE in a function/method when something fails. That's simple and quick, and the user can check the return with:

```
if (!$myObject->saveFile()) {
    //it failed
}
```

But how about telling WHAT failed? An error message would be neat. Every class that extends Bs_Object inherits the following methods: setError(), getLastError(), getLastErrors(), getErrors()

We can use that when coding our class:

```
class Foo extends Bs_Object {
    /**
     * @access public
     * @return bool (TRUE on success, FALSE on failure.)
     */
    function saveFile() {
        $fileFullPath = '/var/www/foo.txt';
        $fileContent = 'foo bar';
        if (file_exists($fileFullPath)) {
            $this->setError('Already exists: '.$fileFullPath,
                'ERROR');
            return FALSE;
        }
        //save file, blah
        return TRUE;
    }
}
```

The methods mentioned can also be used as global functions, without object. then they are prefixed with "bs_", so:

```
bs_setError()
bs_getLastError()
bs_getLastErrors()
bs_getErrors()
```

A poor man's namespace.

See the class core/lang/Bs_Error.class.php for further information.

And back to our userland code:

```
$myObject =& new Foo();
if (!$myObject->saveFile()) {
    echo "Saving Failed, error was: " . $myObject->getLastError();
}
```

And the output, in case of an error, will be like:

```
Saving Failed, error was: In errorexception1.php [foo::seterror
near line 15] ERROR: Already exists: /var/www/foo.txt
```

It tells us the reason, plus names the file and line where the error occurred. Strong yet simple, isn't it?

getLastError() returns the message of the last call to setError().
 getLastErrors() returns all errors that have not been fetched yet using getLastError() or getLastErrors(), while getErrors() returns all errors that have ever been set.

The errors are not separated for every object; all share the same error collection. It makes sense if you think about it.

Exceptions

Instead of setting an error message and returning FALSE you can "throw" an exception à la Java. An exception is an object like any other object. It knows about the problem that occurred.

Imagine a class structure in a code like this:

```
class PhoneBook extends Bs_Object {

    var $db;

    /**
     * constructor
     */
    function PhoneBook() {
        $this->db =& new Database();
    }

    /**
     * @access public
     * @param string $string (what you are looking for)
     * @return array (the result)
     * @throws bs_exception
     */
    function search($string) {
        $result = $this->db->executeSql("SELECT * FROM table WHERE field LIKE '" .
addSlashes($string) . "'");
        if (isEx($result)) {
            $result->stackTrace('i was here', __FILE__, __LINE__);
        }
        return $result;
    }
}

class Database extends Bs_Object {

    /**
     * @access public
     * @param string $sqlCode
     * @return array (the result)
     * @throws bs_exception
     */
    function executeSql($sqlCode) {
        if (!$this->isConnected()) {
            return new Bs_Exception('Not connected!', __FILE__, __LINE__);
        } elseif ($this->isBusy()) {
            return new Bs_Exception('I am busy!', __FILE__, __LINE__);
        } else {
            //run the query
            return $result;
        }
    }

    function isConnected() { return TRUE; }
    function isBusy() { return TRUE; }
}
```

We have a PhoneBook class to perform searches, which uses a Database class to do the hard work. When the database has a problem, it throws an exception. In the PhoneBook class we can care about it, and act based on the exception, or we can just pass it along to the user. Userland code:

```
$phoneBook =& new PhoneBook();
$result = $phoneBook->search("Alanis Morissette");
if (isEx($result)) {
    $result->stackDump('die');
} else {
    //go on
}
```

The stackDump() output:

```

                                     _STACK_TRACE_0_
-----
Message      : I am busy!
File         : \usr\local\lib\php\blueshoes-
4.6\documentation\manuals\php\errorexception2.php
Line        : 18
Timestamp   : 2004/04/11 16:29:44

                                     _STACK_TRACE_1_
-----
Message      : i was here
File         : \usr\local\lib\php\blueshoes-
4.6\documentation\manuals\php\errorexception2.php
Line        : 57
Timestamp   : 2004/04/11 16:29:44
```

See the class `core/lang/Bs_Exception.class.php` for further information.

Throw PHP notices, warnings and errors

PHP itself throws different kind of errors: notices, warnings, errors and parse errors. The developer then defines what to do with those. Log it to a file and/or display it on the screen or ignore them. See the setting "error_reporting" in the php.ini file. PHP also lets you set that option on the fly using `ini_set()`.

You can use the same technique as PHP does using PHP's `trigger_error`-function:
`trigger_error(message, type)`

Example:

```
class Database {  
  
    /**  
     * @access public  
     * @param string $host  
     * @param string $user  
     * @param string $pass  
     * @return bool TRUE on success, FALSE on failure  
     */  
    function connect($host, $user, $pass) {  
        if (empty($pass)) {  
            //that is no good. warn the user, but go on.  
            trigger_error("You should set a password for the  
database connection!", E_USER_WARNING);  
        }  
        $this->resource = mysql_connect($host, $user, $pass);  
        if (!$this->resource) return FALSE;  
        return TRUE;  
    }  
}  
  
$db =& new Database();  
$status = $db->connect('localhost', 'root', '');
```

On dev machines warnings should be displayed on the screen. On live machines. They should be logged to a file. Thus, while developing, the coder will see this message on the screen:

Warning: You should set a password for the database connection! in
c:\usr\local\lib\php\blueshoes-4.6\documentation\manuals\php\errorexception3.php on line 18

Additional reading

- File php.ini: the part "Error handling and logging" and especially the setting "error_reporting".
- The global.conf.php file from BlueShoes, there we overwrite the error_reporting setting.
- The lass core/lang/ Bs_Error.class.php
- The class core/lang/ Bs_Exception.class.php
- The PHP manual for try/catch when it's updated for PHP5.