Creating a PHP Settings Class

By Dennis Pallett

Abstract

In this tutorial you will be shown how to create a settings class which can handle different formats (PHP, INI, XML and YAML), using polymorphism

Introduction

A config.php or a settings.xml file is a very common thing for most PHP scripts, and it's usually where all the script settings (e.g. database information) are stored. The easiest way is to simply use a simple PHP script as a config file, but this may not be the best way, and it's certainly not the most user-friendly way.

In this tutorial we'll have a look at creating a Settings class which can handle any type of config format. I'll take you through the steps necessary to handle four different formats (PHP, INI, XML and YAML), but it's very easy to add more formats.

Let's get started.

Writing the base class

Because our Settings class needs to support multiple formats it's best to write a base class first, and then have separate specific classes. In PHP 5 we would make this base class an abstract class, but since we're using PHP4, it'll just be a normal class.

The basic skeleton of our base class looks like this:

```
Class Settings {
	var $_settings = array();

	function get($var) {
		trigger_error ('Not yet implemented', E_USER_ERROR);
	}

	function load() {
		trigger_error ('Not yet implemented', E_USER_ERROR);
	}
}
```

As you can see we've defined two methods: get and load. The get() method is used to get the value of a certain setting, and the load() method is used to load the settings. We've also defined a single property, $_settings, which will hold the loaded settings.

We can't implement the load() method yet, because each format requires a different way of loading. But we can implement the get() method already, so let's do that. See the example below:

```
function get($var) {
        $var = explode('.', $var);

       $result = $this->_settings;
        foreach ($var as $key) {
               if (!isset($result[$key])) { return false; }

              $result = $result[$key];
        }

       return $result;
 }
```

The get() method first splits the passed varname into separate parts. This makes it possible to have multidimensional settings, for example db.name, db.host, etc.

Now that we've got a working get() method, all we need to do is implement the load() method, but we're not going to do that in the base class. We'll use separate child classes for that.

Let's start with the first one, the PHP settings class.

PHP Settings Class

The PHP settings class is a child class of the base settings class, and therefore needs to extend the base class, like this:

```
Class Settings_PHP Extends Settings {

}
```

Since we are inheriting from the base class, the get() method already works, so all we need to do now is write the load() method. What the load() method needs to do is include the config file, and get the necessary settings into the $_settings array. The config.php file looks like this:

```
<?php
 $db = array();

// Enter your database name here:
 $db['name'] = 'test';

// Enter the hostname of your MySQL server:
 $db['host'] = 'localhost';

?>
```

The first thing the load() method needs to do is check whether the config file exists, like this:

```
function load ($file) {
	if (file_exists($file) == false) { return false; }

}
```

Then we need to include the config file, using the standard include() function, like this:

```
function load ($file) {
	if (file_exists($file) == false) { return false; }

	// Include file
	include ($file);
}
```

Now that the config file has been included, the config variables have also been included. But we still need to get the config variables into the $_settings array. This can be done with the get_defined_vars() function, which returns an associative array of all the variables that exist in the current scope.

The following piece of code will import the config variables into the $_settings array:

```
// Get declared variables
unset($file);
$vars = get_defined_vars();

// Add to settings array
foreach ($vars as $key => $val) {
	if ($key == 'this') continue;

	$this->_settings[$key] = $val;
}
```

The above code first gets all the defined vars, and then imports them in the $_settings array. We make sure to skip on the $this variable, since it refers to the object (and therefore it's not a config variable).

Our complete load() method now looks like this:

```
function load ($file) {
	if (file_exists($file) == false) { return false; }

	// Include file
	include ($file);

	// Get declared variables
	unset($file);
	$vars = get_defined_vars();

	// Add to settings array
	foreach ($vars as $key => $val) {
		if ($key == 'this') continue;

		$this->_settings[$key] = $val;
```

```
        }
}
```

The following code demonstrates the use of the Settings_PHP class:

```
<?php
 include ('settings.php');

// Load settings (PHP)
 $settings = new Settings_PHP;
 $settings->load('config.php');

echo 'PHP: ' . $settings->get('db.host') . '';
 ?>
```

Now let's take a look at using an INI file as a config file.

INI Settings Class

Just like with the PHP settings class, let's start with an empty INI class that extends the base class:

```
Class Settings_INI Extends Settings {

}
```

Now we need to create the load() method again, but this time we'll have to create a load() method that can read INI files. The following will do the trick:

```
function load ($file) {
        if (file_exists($file) == false) { return false; }
        $this->_settings = parse_ini_file ($file, true);
}
```

In the above code snippet we use the parse_ini_file() to read in our config.ini file, which looks like this:

```
[db]
 name = test
 host = localhost
```

Using the INI class is almost exactly the same as using the PHP class:

```
// Load settings (INI)
 $settings = new Settings_INI;
 $settings->load('config.ini');

echo 'INI: ' . $settings->get('db.host') . '';
```

As you can see all we needed to change was the first line, and this makes changing between different setting formats extremely flexible. You don't have to change any code in your script, except for one line.

Let's create the XML settings class now.

XML Settings Class

Like the previous two classes, let's start with an empty class again:

```
Class Settings_XML Extends Settings {

}
```

Since we are going to use an XML file as a config file now, we will have to use an XML parser. Since XML parsers are fairly standard stuff, I'm just going to use the XML Lib by Keith Devens, available at http://keithdevens.com/software/phpxml.

The load() method of our XML class will have to parse an XML config file that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
 <settings>
        <db>
                <name>test</name>
                <host>localhost</host>
        </db>
 </settings>
```

The following code will do exactly what we want:

```
function load ($file) {
        if (file_exists($file) == false) { return false; }

       include ('xmllib.php');
        $xml = file_get_contents($file);
        $data = XML_unserialize($xml);

       $this->_settings = $data['settings'];
 }
```

In the above example we use the XML library to unserialize the XML (changing it into a normal PHP array), and then get the settings from the unserialized XML.

Again, to use the XML class, all we have to do is change one line:

```
// Load settings (XML)
 $settings = New Settings_XML;
 $settings->load('config.xml');

echo 'XML: ' . $settings->get('db.host') . '';
```

Just like the previous two times everything is exactly the same, except for the first line.

Now let's create the final config format: YAML.

YAML Settings Class

YAML is a relatively new data format, and uses mostly white-space to define data structures. More information can be found at http://www.yaml.org/, but we'll just be using the Spyc library to parse YAML files into PHP structures.

Let's start with the empty class:

```
Class Settings_YAML Extends Settings {

}
```

The load() method now needs to parse a YAML config file, which looks like this:

```
db:
   name: test
   host: localhost
```

(*Note that the indention is very important.*)

With the Spyc library parsing YAML is no problem, and it takes only three lines to import the settings:

```
function load ($file) {
        if (file_exists($file) == false) { return false; }

       include ('spyc.php');
        $this->_settings = Spyc::YAMLLoad($file);
 }
```

As you can see we used the Spyc library to parse the YAML, and then passed the results to the $_settings array.

You can probably guess how to use the YAML settings class, and all we need to do again is change only one single line.

Conclusion

In this tutorial I've shown you how to create a Settings class that can handle four different formats, but the great thing about our Settings class is that it's extremely flexible. If you're halfway through your project, and suddenly want to start using a different format it's no problem. All you have to do is change a single line, and the rest of your code is still good to go.

And that's the main point of this tutorial. We've effectively decoupled our Settings class from the rest of the project. Our settings class doesn't need to know about the rest of the script, and the rest of the script doesn't need to know which Settings class we're using, as long as there is a load() and get() method. This is called 'polymorphism', and can be a very powerful tool.

If you want to have a look at the full Settings Class, click here to download it.

If you have any questions or comments on this tutorial, feel free to leave them below. Also join us at PHPit Forums!

About this PDF