A Gentle Introduction to the Zend Framework

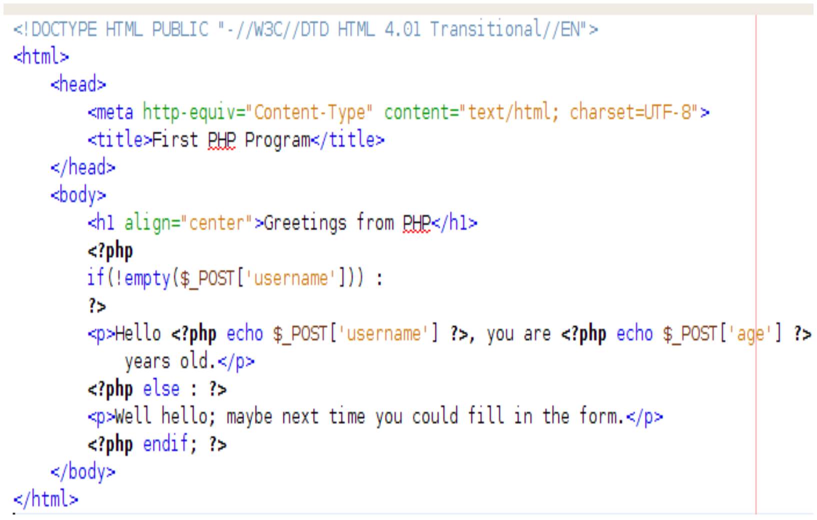
This tutorial provides an introduction to the Zend Framework. It assumes readers have experience in writing simple PHP scripts that provide web-access to a database.

1. First a digression – Smarty templates: the separation of code (model and control) and display (view)
2. Using simple Zend components in isolation
3. Zend_DB – help with data persistence
4. Model View Control and related issues
5. Zend's MVC framework

The examples have been built for an Ubuntu system with a local Apache server using NetBeans 6.9.1 and Zend 1.10. The NetBeans projects are available for download. Details of how to set up the environment are appended to this document along with links to tutorials at NetBeans and Oracle. Some examples use a MySQL server hosted on the same machine as the Apache server; others use a remote Oracle server accessed via Oracle's instant_client and its associated libraries.

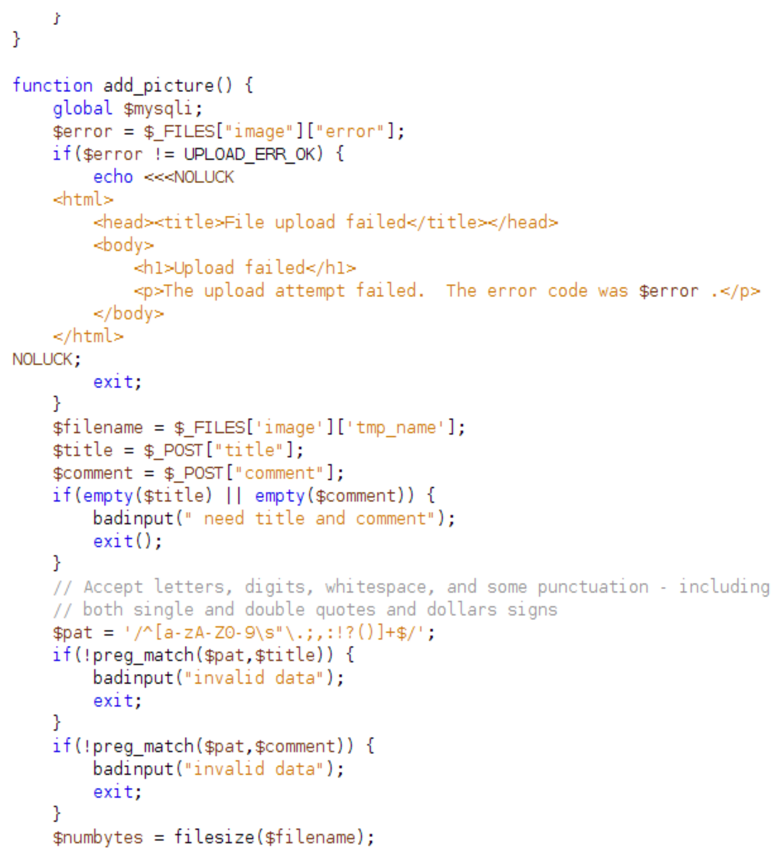
1 Smarty

The first PHP program that you wrote probably achieved the supposed ideal of just a little code embedded in a HTML page, such as this variant of Ramus Lerdorf's example script for handling input from a form for entering a user's name and age:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>First PHP Program</title>
    </head>
    <body>
        <h1 align="center">Greetings from PHP</h1>
        <?php
        if(!empty($_POST['username'])) :
        ?>
        <p>Hello <?php echo $_POST['username'] ?>, you are <?php echo $_POST['age'] ?>
            years old.</p>
        <?php else : ?>
        <p>Well hello; maybe next time you could fill in the form.</p>
        <?php endif; ?>
    </body>
</html>
```

But when you started to create real PHP applications you were using large files with lots of PHP code containing output statements that generated the HTML, as illustrated in the following fragment of a script dealing with file uploads:



```
    }
}

function add_picture() {
    global $mysqli;
    $error = $_FILES["image"]["error"];
    if($error != UPLOAD_ERR_OK) {
        echo <<<NOLUCK
    <html>
        <head><title>File upload failed</title></head>
        <body>
            <h1>Upload failed</h1>
            <p>The upload attempt failed.  The error code was $error .</p>
        </body>
    </html>
NOLUCK;
        exit;
    }
    $filename = $_FILES['image']['tmp_name'];
    $title = $_POST["title"];
    $comment = $_POST["comment"];
    if(empty($title) || empty($comment)) {
        badinput(" need title and comment");
        exit();
    }
    // Accept letters, digits, whitespace, and some punctuation - including
    // both single and double quotes and dollars signs
    $pat = '/^[a-zA-Z0-9\s"\.;,:!?()]+$/';
    if(!preg_match($pat,$title)) {
        badinput("invalid data");
        exit;
    }
    if(!preg_match($pat,$comment)) {
        badinput("invalid data");
        exit;
    }
    $numbytes = filesize($filename);
```

Code for data processing just does not mix well with display. They really are separate issues.

Similar problems had arisen with other technologies, such as the Java web technologies. The original Java servlets (~1997) were all code with data display handled via println statements – and as such they were not popular. It was difficult to envisage how pages would appear given just a collection of println statements. Java Server Pages (JSPs) were invented (~1999) to try to overcome the problems. JSPs (which are "compiled" into servlet classes) were an attempt to achieve the ideal of a little code embedded in a HTML document. The code in JSPs was initially as scriptlets – little fragments of embedded Java – later, with the Java Standard Tag Library (JSTL), XML style markup tags were used to define the code. (Although implemented quite differently, JSTL tags act a bit like macros that expand out to large chunks of Java code when the JSP is compiled to yield a servlet.)

But the problems remain. Little scraps of code (scriptlet or JSTL tag style) embedded in HTML work fine for "hello world" demonstrations, but with real applications it's more like little scraps of HTML embedded in lots of code.

Eventually, Java developers settled on a division of responsibility. Java code in servlets and application defined classes was used to select the data to be displayed; JSPs with scriptlet coding or preferably JSTL tags were used to display these data. The servlet that had been written by the developer ran first. All the complex coding is done in the servlets and helper classes. The servlet code builds data structures to hold the data that are to be output and has control code that determines which JSP display script is to run (there could be JSP scripts for successful outcomes and scripts that deal with various error reports). The servlet forwarded the structures with data to the selected JSP (actually, forwarded the data to another servlet that had been compiled from the JSP). Any code in the JSP would be very simple - "*display this data element here in this HTML div*", "*loop through this data collection outputting successive rows of a list or table*".

A similar division of responsibility can be achieved in PHP applications with PHP scripts and classes used to select the data that are then displayed using Smarty classes working with template files. Taking a slightly simplified view, the PHP script will create a hash-map of (name, value) pairs inside a "smarty" object and then invoke the display method of that object using a provided template.

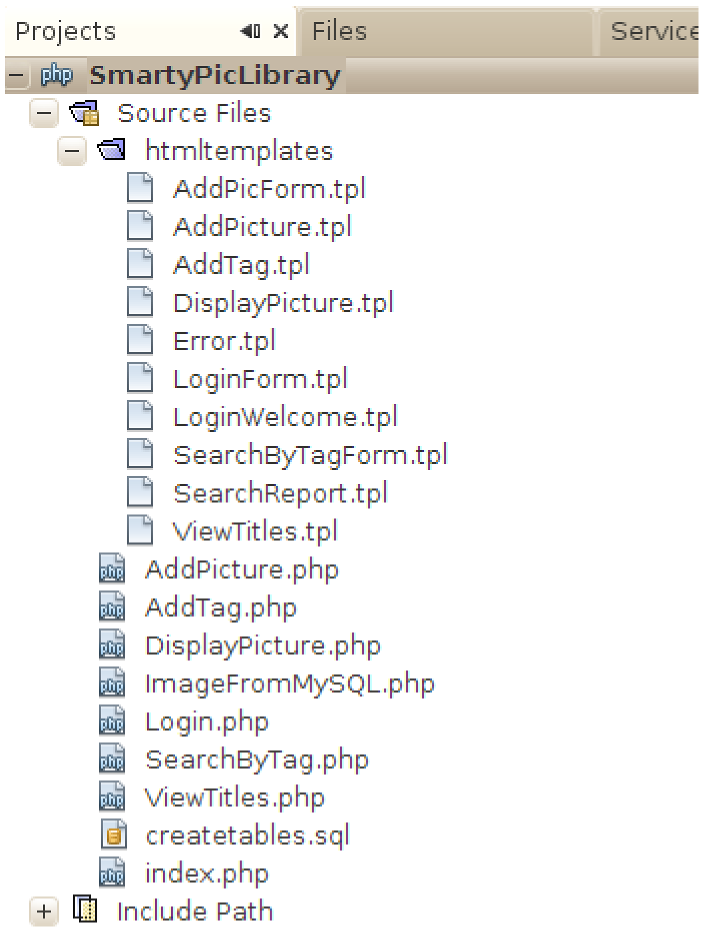
A Smarty template file consists of HTML markup and static content along with some limited scripting (in its own scripting language) that deals with tasks like "*display this data element here in this HTML div*", "*loop through this data collection outputting successive rows of a list or table*". Smarty template files are "compiled" into PHP classes (just like JSPs being compiled into servlets).

The main PHP script, and associated class files, are not encumbered with any print statements or blocks of predefined HTML. The Smarty template file is a pretty much standard HTML file that can be edited by a specialised HTML editor (provided the editor is configured to ignore the Smarty code fragments). PHP code is for control and data access; HTML markup and Smarty script fragments handle display. View is separated from control and model.

1.1 Smarty example – SmartyPicLibrary

The example application is available as the SmartyPicLibrary project in the download zip file. A conventional PHP version is described in the CSCI110 lecture notes; this new version uses Smarty for the form pages and response pages.

The application allows an owner to create a picture gallery, uploading pictures to a MySQL database. Pictures in the collection have title and comment information, and also "tags" held in a separate table. Uploading of pictures is restricted to the owner; other visitors can view the titles of pictures, search for pictures by tag, view pictures and add tags. The application is comprised of a number of PHP scripts; most handle "GET" requests by displaying a form, with input from that form being returned in "POST" requests for processing.



```
Projects    Files    Service
− SmartyPicLibrary
  − Source Files
    − htmltemplates
        AddPicForm.tpl
        AddPicture.tpl
        AddTag.tpl
        DisplayPicture.tpl
        Error.tpl
        LoginForm.tpl
        LoginWelcome.tpl
        SearchByTagForm.tpl
        SearchReport.tpl
        ViewTitles.tpl
      AddPicture.php
      AddTag.php
      DisplayPicture.php
      ImageFromMySQL.php
      Login.php
      SearchByTag.php
      ViewTitles.php
      createtables.sql
      index.php
  + Include Path
```

PHP scripts have two points of interaction with the Smarty system. There is an initial setup step where a “Smarty” object is configured; it needs a place to put the code that it will generate from the template HTML files:

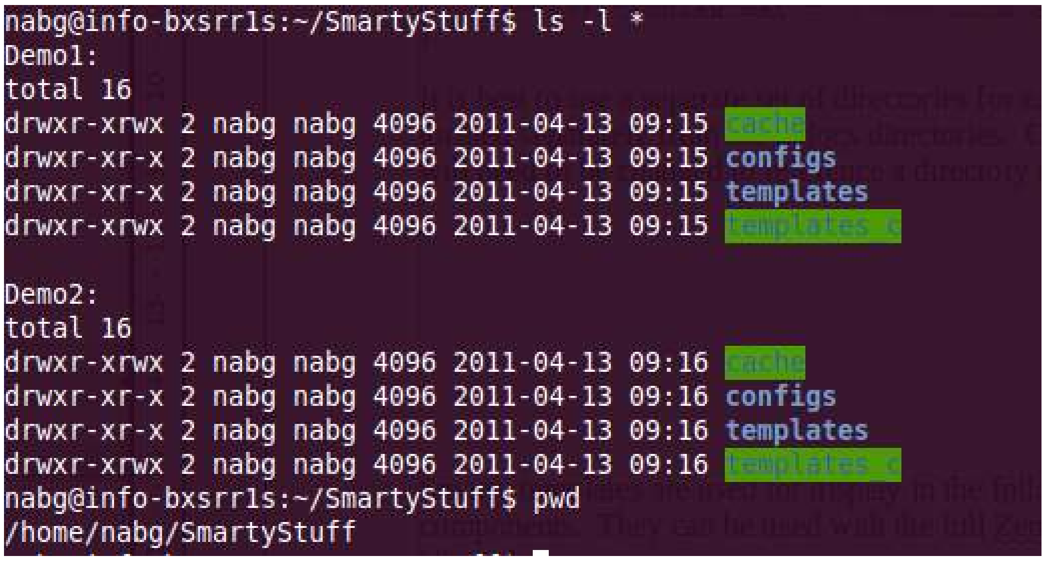
```
<?php

require('/usr/local/lib/php/Smarty/Smarty.class.php');

// Global variables
$smarty = new Smarty();
$mysqli = 0;
$script = $_SERVER["PHP_SELF"];

function smartysetup() {
    global $smarty;
    $smarty->template_dir = '/home/nabg/SmartyStuff/Demo1/templates';
// The cache and templates_c directories need to be writeable
// by www-data (i.e. the Apache server process)
    $smarty->compile_dir = '/home/nabg/SmartyStuff/Demo1/templates_c';
    $smarty->cache_dir = '/home/nabg/SmartyStuff/Demo1/cache';
    $smarty->config_dir = '/home/nabg/SmartyStuff/Demo1/configs';
}
```

It is best to use a separate set of directories for each application that uses Smarty; these should be located separately from the htdocs directories. Obviously, the example code in the download files will need to be changed to reference a directory that you create on your own system. The cache and templates_c directories are for the code that the Smarty template engine creates and should be writeable by the Apache process. The other two directories are for more advanced uses, such as extending the set of templates that the Smarty engine employs.



```
nabg@info-bxsrr1s:~/SmartyStuff$ ls -l *
Demo1:
total 16
drwxr-xrwx 2 nabg nabg 4096 2011-04-13 09:15 cache
drwxr-xr-x 2 nabg nabg 4096 2011-04-13 09:15 configs
drwxr-xr-x 2 nabg nabg 4096 2011-04-13 09:15 templates
drwxr-xrwx 2 nabg nabg 4096 2011-04-13 09:15 templates_c

Demo2:
total 16
drwxr-xrwx 2 nabg nabg 4096 2011-04-13 09:16 cache
drwxr-xr-x 2 nabg nabg 4096 2011-04-13 09:16 configs
drwxr-xr-x 2 nabg nabg 4096 2011-04-13 09:16 templates
drwxr-xrwx 2 nabg nabg 4096 2011-04-13 09:16 templates_c
nabg@info-bxsrr1s:~/SmartyStuff$ pwd
/home/nabg/SmartyStuff
```

The other point of interaction between your PHP script and Smarty is where you forward the data that are to be displayed and start the Smarty display process.

The example Picture Library application has a script that allows a user to enter a tag, and that then uses this tag to retrieve the titles and identifiers that have been characterised with that tag. The results page from this search should be a table of links that will allow retrieval and display of the actual pictures. The PHP script runs the search request and assembles the resulting data for display in a Smarty template. These interactions are illustrated in the following code fragments:

```php
function display_search_form() {
    global $script;
    global $smarty;
    $smarty->assign('script', $script);
    $smarty->display('./htmltemplates/SearchByTagForm.tpl');
}

function dosearch() {
    global $mysqli;
    global $smarty;

    $usertag = $_POST["searchtag"];

    $stmt = $mysqli->Prepare("SELECT ident,title FROM nabg.Picys "
          . "where ident in (select Picid from nabg.PicyTags where Tagstr=?)");

    $stmt->bind_param('s', $usertag);
    $stmt->execute();
    $stmt->bind_result($id, $title);

    $matches = array();
    while ($stmt->fetch()) {
        $matches[] = array($id, $title);
    }

    $mysqli->close();
    $smarty->assign('matches', $matches);
    $smarty->assign('usertag', $usertag);
    $smarty->display('./htmltemplates/SearchReport.tpl');
}
```