

# WWW

Daniel C. Bastos  
dbastos@math.utoledo.edu

November 20, 2004

## Abstract

This is an introduction to WWW Programming. When language matters, PHP is used; However, this is not a PHP tutorial. This introduction is solely to guide an oral presentation. The key point of this presentation is to demonstrate how browsers exchange information with web servers, so the HTTP protocol is mentioned. Other things introduced are templates, a methodology and some ideas to package programs.

## 1 Introduction

The WWW is a system which must consist of, at least, a browser and a webserver. By “browser”, we mean a program which requests files to a webserver. By “webserver”, we mean a program which answers browser’s requests. Browsers and web servers must be able to communicate with each other and this requires a common language. The “language” the WWW uses is known as the HTTP protocol. Think of “language” when you hear the word “protocol”, in this context. Some web servers are equipped with “engines” — PHP and Perl are two examples. Such engines were written to extend the capabilities of web servers, and so giving the WWW some dynamism. We will use the terms “browser”, “webserver”, “protocol” and “engines” as previously described.

## 2 Watching an HTTP conversation

When a human, using a browser, types `www.cnn.com`, for example, the browser being used will issue — after so many other requirements such as DNS resolutions — a request to a webserver responsible for `www.cnn.com`. After receiving and understanding the request, the webserver in question will answer, in a similar way, to the browser and possibly satisfying the browser’s demand. A common answer is something like “I don’t have what you look for”. The message “404 File not found” is usually familiar to WWW users.

By using the software `http-0.1.tar.gz`<sup>1</sup>, one can interactively watch a conversation between a browser and a webserver, one request at a time. An example of a query that `http-0.1.tar.gz` would perform is:

```
GET / HTTP/1.0
Host: cnn.com
Accept: /*/*
User-Agent: danux-bot[dbastos@slashlog.org]/0.1
Connection: close
```

This request gets sent through the network and is read by a webserver responsible for `cnn.com` which could answer as follows:

```
HTTP/1.1 302 Found
Date: Thu, 09 Dec 2004 03:29:24 GMT
Server: Apache
Location: http://www.cnn.com/
Content-Length: 262
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.cnn.com/">here</a>.</p>
<hr />
<address>Apache Server at cnn.com Port 80</address>
</body></html>
```

The answer given by the webserver includes an HTML file after an HTTP set of headers with information about the file sent back to the browser. A browser usually strips the header and displays the user only the text after the empty line; This is the way that a browser can distinguish the header from the HTTP document — the HTML document, in this case.

### 3 Engines, Webservers and Browsers

A PHP engine, like most engines, works with a webserver, but some other engines work with a browser — an example of a browser's engine is Javascript.

There is an agreement between the webserver and the engine: which one is responsible for what. For instance, when the webserver notices that a request is made to a `.php` file, it has to forward the request to the PHP

---

<sup>1</sup>Download <http://math.utoledo.edu/~dbastos/files/http-0.1.tar.gz>.

engine and wait for an answer. When an answer is received, the webserver is free to send it to the browser. One may conclude that a webserver needs solely to be able to send files to browsers and forward requests to engines, if appropriate.

The communication between a webserver and an engine is not based on HTTP conversations. A webserver is free to choose its method of communication with its engines and engine programmers must then respect it when they write softwares to extend the webserver's capabilities.

## 4 Composing answers at runtime

If one wants to create files on-the-fly and send them to browsers, an engine will be necessary. One, then, needs to understand how to instruct the engine do the task of generating files; That is, to learn to write programs using some language, such as PHP.

PHP programs can be embedded in HTML language, so a PHP program may contain HTML and PHP code. When PHP instructions are written, they must be enclosed by PHP tags. An example is:

```
<html>
  <body> <?php print 'Hello World'; ?> </body>
</html>
```

After read by the engine, the program above would become a file without the PHP code and ready to be sent to the browser.

```
<html>
  <body> Hello World </body>
</html>
```

A browser issues an HTTP request such as “Give me the file `hello.php`”. The webserver respects its agreement and forwards the request to the engine. The PHP engine reads `hello.php` and performs its instructions which is to generate some text and sends it back to the webserver which in turn sends the text back to the browser which displays the page saying “Hello World”.

## 5 Human interaction

Programs are usually written to take orders from humans and to generate answers to satisfy them. Browsers, then, were written to allow humans to pass more information other than which file to retrieve. When humans, for instance, fill forms on the web and submit them, they are using this mechanism. A form is a way to send information to the other side, which

many times is supposed to reach the engine and not exactly the webserver. However, in order to reach the engine, the webserver must be able to forward the information.

Forms are written in HTML and usually operate using the HTTP methods called “GET” and “POST”. There are other methods, but these two are the most popular ones. By using `http-0.1.tar.gz`, one can interactively send GET and POST requests to webserver on the Internet.

There is one difference between POST and GET: the extra information one can send to a webserver — or to an engine — can be attached to the URL or it can be sent as an HTTP document after the header. The former method is GET and the latter, POST. Examples:

```
GET /index.html?animal=lion&fruit=orange&language=c HTTP/1.0
Host: www.cnn.com
Accept: */*
User-Agent: danux-bot[dbastos@slashlog.org]/0.1
Connection: close
```

The extra information is attached to the URL. The webserver would forward the extra information to the engine, if appropriate.

```
POST /index.html HTTP/1.0
Host: www.cnn.com
Accept: */*
User-Agent: danux-bot[dbastos@slashlog.org]/0.1
Connection: close
Content-type: application/x-www-form-urlencoded
Content-length: 35
```

```
animal=lion&fruit=orange&language=c
```

Now the extra information is attached to the HTTP document. The header now only contains the file to be retrieved and clearly the method changed to POST. The difference between GET and POST, then, is minimal. However, this offers a big advantage to the WWW; For instance, with POST, browsers are now able to upload files. The URL limits the amount of information, so a large enough file would be truncated.

We should be sure to notice the addition of two more fields added to the HTTP header request when using POST. They are: “Content-type” and “Content-length”. The latter tells the webserver how much of extra information is to be found after the request header.

Each field on an HTTP header has a limit of characters. One cannot encode and upload a file by appending a sequence of bytes on the URL if it exceeds the limit. By allowing a document to be sent from the browser to the webserver, using POST, one can then send a whole file to the webserver, not to mention other information.

In case of the file upload, the file would be encoded — by using, for example, an algorithm such as base 64 — and the webserver or the engine, would extract the sequence of bytes, decode it — using the proper algorithm — and save it to disk. The file is then uploaded. Information about the file such as name, file size etc are passed as variables.

The package `http-0.1.tar.gz` demonstrates how variables — attached to the URL or composing an HTTP document — come from the HTTP request to PHP variables. As an example, the following HTTP request issues a GET method with some extra information.

```
GET /~dbastos/tmp/echo.php?animal=bear&fruit=orange&drink=coke
HTTP/1.0
Host: math.utoledo.edu
Accept: */*
User-Agent: danux-bot[dbastos@slashlog.org]/0.1
Connection: close
```

A PHP script hosted by `math.utoledo.edu` is programmed to display the extra information attached to the URL or included in the HTTP document sent possibly by the browser.

Hello. I'm a PHP program.

```
<?php
if( count($_GET) > 0) {
    print 'I found extra information on the URL<br>\n';
    print_r($_GET);
}

if( count($_POST) > 0) {
    print 'I found extra information in the HTTP document<br>\n';
    print_r($_POST);
}
?>
```

After issuing the request, the `echo.php` program displays:

Hello. I'm a PHP script.

```
I found extra information on the URL<br>
Array
(
    [animal] => bear
    [fruit] => orange
    [drink] => coke
)
```

The browser issues the HTTP request; The file requested is a PHP program, so the webserver forwards the request to the PHP engine which

extracts the extra information, feeds it to the program `echo.php` which obtains the information through PHP arrays — such as `$_GET` and `$_POST` — and does something with it; In this case, it simply prints them out. After PHP processing is done, the engine feeds the text back to the web-server which sends to the browser for display.

## 6 Writing PHP programs

In order to write programs, one needs to clearly understand each step that is required to perform the task. For instance, if a programmer wishes to open, read and print a file which is present locally in the file system of the computer, then a programmer must be able to:

1. Open a file.
2. Check if operation succeeded.
3. Read byte after byte until the end of the file.
4. Check if operation succeeded.
5. Stop reading when necessary.
6. Print the data.

The PHP engine offers a collection of functions which one can take advantage to write programs. To accomplish the task above, one uses a function called `fopen()` to get a file descriptor from the operating system.

A file descriptor is a number which identifies, for example, an open file in the file system. This number is useful, for example, for a function called `fread()`. The function `fread()` takes two arguments: a file descriptor and an integer. The latter argument is the amount of bytes to be read at once. Here's a snippet:

```
$fd = @fopen('/etc/passwd', 'r');
if ( !is_resource($fd) )
    err('Could not open file\n');

while ( !feof($fd) )
    $buffer .= @fread($fd, 32);

print $buffer; exit;
```

## 7 Talking to a database

Using PHP, one can talk to a database such as MySQL by using PHP functions that will do each step of the necessary set of steps. In order to perform a query, one must be able to:

1. Connect to the database program.
2. Check if operation succeeded.
3. Tell the database program what database will be used.
4. Check if operation succeeded.
5. Tell the database program to execute the task.
6. Check if operation succeeded.
7. Read byte after byte until the end of the file.
8. Stop reading when necessary.
9. Print the data.

Here is a snippet:

```
$fd = @mysql_connect('cnn.com', 'melissa', 'mumble');
if ( !is_resource($fd) )
    err('Could not connect to the database program\n');

$i = @mysql_select_db('book_store');
if ($i === false)
    err('Could not select database\n');

$s = 'SELECT title FROM books';
$r = @mysql_query($s);
if ($r === false)
    err('Could not execute query\n');

while ( ($t=mysql_fetch_row($r)) )
    print $t[0] . '<br>';

@mysql_close($fd); exit;
```

## 8 Arrays

In the previous section, an array was returned by `mysql_fetch_row()`. Many other functions will return an array, or will take one or more arrays as arguments. One must be able to deal with arrays in many computer languages. PHP is not an exception.

### 8.1 Creating PHP arrays

PHP offers a language constructor called `array()`. One may create an array by assigning a variable to `array()`.

```
$a = array();
```

Other ways:

```
/* numeric indices */
$a[0] = 'some string';

/* string indices. associative arrays */
$b['name'] = 'Michelle';
```

In PHP, an array item may be another array.

```
$c = array();
$c[0] = array();
$c[0]['key'] = 1;
$c[0]['name'] = 'Arts';
$c[1] = array();
$c[1]['key'] = 2;
$c[1]['name'] = 'Sciences';
$c[2] = array();
$c[2]['key'] = 2;
$c[2]['name'] = 'Finances';
```

A way to see what the last array looks like is to call `print_r($c)`.

```
Array
(
    [0] => Array
        (
            [key] => 1
            [name] => Arts
        )

    [1] => Array
        (
            [key] => 2
            [name] => Sciences
        )

    [2] => Array
        (
            [key] => 2
            [name] => Finances
        )
)
```

PHP still offers other ways to create arrays. The next way described is to use an empty index and the engine will index it automatically by using numbers starting at 0.

```

$i = 0;
for ( ;; ) {
    $c[] = ++$i;
    if ($i == 5)
        break;
}

print_r($c);

```

Which, by calling `print_r()`, would look like:

```

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)

```

## 8.2 Looping through arrays with PHP

There are a few ways to loop through an array, using PHP. Here, we present a couple of different ones. A simple `for()` loop is first used and then a `while()` loop together with the constructors `list()` and `each()`. When not using array indices, it may be necessary to set the array pointer to the beginning, by using `reset()`.

```

$e = count($t);

print 'Print array elements (1)\n';
for( $i=0; $i < $e; ++$i)
    print $t[$i] . '\n';

end($t);

print 'Print array elements (2)\n';
while( list($k, $v) = each($t) )
    print $v . '\n'; /* $v == $t[$k] */

print 'Print array elements (3)\n';
reset($t);

while( list($k, $v) = each($t) )
    print $t[$k] . '\n'; /* $v == $t[$k] */

```

The last snippet would generate the following text:

Print array elements (1)

1  
2  
3  
4  
5

Print array elements (2)

5

Print array elements (3)

1  
2  
3  
4  
5

The `end()` constructor, moves the array pointer to the last item of the array. By using `each()`, one must be sure that the pointer points to the first item, by using `reset()`, or the array may not be entirely traversed.

## 9 Helping designers

In previous sections, we wrote programs which execute steps and print information to be displayed by a browser. On a real program, the amount of steps and the amount of information to be displayed can be overwhelming.

Consider this situation: a programmer writes hundreds of line of PHP code with hundreds of lines of HTML, in the usual way. A programmer is usually working along with a designer who will increment the design of the page, format fonts, tables etc. When the programmer is done, the software is working with a skeleton of the HTML and no colors, no tables, no professional look on the website. So, the designer takes all the programs — the files that compose the software — and starts to change them accordingly to have a nice look and feel. Many times, designers feel overwhelmed by the complexity of the code and sometimes accidents happen and PHP code is messed up, destroyed etc.

One solution for this situation is to use a template engine. PHP already is a template engine, so what we mean is to write or to use another template engine on top of PHP. A PHP program is a file composed of instructions when instructions are needed and composed of text when instructions are not needed. So, the idea of using another template engine to create another layer of abstraction is criticized by some programmers as “excessive”. On the other hand, some projects may fail because of the lack of understanding on how to work with a template engine such as PHP itself. In these cases, the creation of another layer of abstraction may be helpful.

One may also consider the difficulty of display PHP code on a program such as “DreamWeaver”. Some HTML editors do not display the PHP code because they are enclosed by tags `<?php ... ?>` and this is considered HTML by many editors, and so nothing is displayed. It gets hard, then, to understand HTML that is supposed to be displayed in case some expression is true or false — this is the case of `ifs` and `elses` —, not to mention loops.

By using a second template engine such as “Smarty”<sup>2</sup>, one can add simple tags made of plain text which will clearly tell the designer how the flow of information will be displayed.

## 10 Operating a template engine

By looking at an example of a Smarty template, one clearly sees that PHP wouldn’t consider the template code because it is not PHP code. It may look like PHP code, but since it is not enclosed by `<?php ... ?>` tags, it won’t be executed by the PHP engine. An example:

```
Hello, {$name}. <br>
You have exactly {$n} messages.<p>

<ul>
{foreach key=k item=m from=$messages}
  <li>Subject: {$m.subject}. From <i>{$m.from}</i>.
{/foreach}
</ul>
```

If you feed the text above to the PHP engine, it will return the exact same text because it realizes that there’s nothing to be executed. What this means that in order for the text above be useful, something must happen before sending the template to the PHP engine. What must happen before is the parsing of the template engine, which in this case is Smarty.

Smarty is PHP code responsible to read the template — such as the example above — and rewrite it using the PHP syntax. So after Smarty does its job, the text would look like:

```
Hello, <?php echo $this->_tpl_vars['name']; ?>. <br>
You have exactly <?php echo $this->_tpl_vars['n']; ?>
messages.<p>

<ul>
<?php if (count((array)$this->_tpl_vars['messages'])):
  foreach ((array)$this->_tpl_vars['messages'] as
    $this->_tpl_vars['k'] => $this->_tpl_vars['m']):
?>
```

---

<sup>2</sup>See <http://smarty.php.net/>.

```
<li>Subject: <?php echo $this->_tpl_vars['m']['subject']; ?>.
    From <i><?php echo $this->_tpl_vars['m']['from']; ?></i>.

<?php endforeach; endif; ?>
</ul>
```

After this is done — and all Smarty steps is transparent to the programmer —, Smarty saves the output to a `file.php` on the local disk, then it invokes that file and asks the PHP engine to do its job. Then Smarty saves what the PHP engine returns and then it finally sends the output to the browser — at this stage, only text or HTML is displayed.

This way, PHP code is somewhat separated from HTML and it helps designers by not offering an overwhelming amount of HTML mixed with PHP and it keeps designers away from breaking PHP code and other things such as messing up the indentation.

One interesting feature is that it only generates PHP code once, unless changes in the template or in the PHP code are made. It creates an archived of compiled PHP code, ready to run by the PHP engine, avoiding overhead of compilation.

The package that demonstrates Smarty is `mail-0.1.tar.gz`<sup>3</sup>. This package is not a webmail, it is simply an example of how to write Smarty template to work with PHP.

## 11 The Node Methodology

When you think of writing a software for the WWW, lots of things come into action: the organization of the code, which files will store functions, which functions are specific to the application, which functions are used by every application you write and so on.

The Node Methodology is a set of ideas that one can use and have a standard way of writing WWW applications. It may be useful to work with a team, for instance, where every member knows exactly the ways that the code gets written and parts of the system may be merged with minimal effort. Another advantage of the methodology is that it is easy to spot which part of the program needs change, because of the standard and organized way the code is written.

Before the details of the methodology, it may be helpful to take a look at a very simple application written under the Node Methodology. By looking at the code and reading the requirements and suggestions of the methodology, it should be easier to understand why the Node Methodology is the way it is. The package `sort-0.1.tar.gz`<sup>4</sup> was written to show only the requirements of the methodology. A real application could use many

---

<sup>3</sup>Download <http://math.utoledo.edu/~dbastos/files/mail-0.1.tar.gz>

<sup>4</sup>Download <http://math.utoledo.edu/~dbastos/files/sort-0.1.tar.gz>

other suggestions, but here the point is simply to show what is required by the methodology.

Any Node Application must consist of two files: `index.php` and `library.php`. We display `index.php` next:

```
<?php
require('./library.php');

$m = get_node();

switch ( $m ) {

case 'sort':
    $s = $_POST['text'];
    if ( ! strlen($s) )
        _exit('Nothing to sort');

    $t = array();
    $t = explode('\n', $s);
    d ( isort($t, 'strings') );
    break;

default:
    print '<html> <title> sort v0.1 </title>';
    print '<form action=index.php method=post>';
    print '<input type=hidden name=node value=sort>';
    print 'Type your values separated by newline<br>';
    print '<textarea cols=50 rows=10 wrap=virtual name=text>';
    print '</textarea><p>';
    print '<input type=submit value=Sort>';
    print '</form>';
    break;
}
?>
```

Now we display `library.php`:

```
<?php
function get_node() {
    $_POST = array_merge($_GET, $_POST);
    $node = $_POST['node'];

    if ( strlen($node) )
        return $node;

    return 'home';
}

function d($v) { print '<pre>'; print_r($v); print '</pre>'; }
```

```

function _exit($msg) { print $msg; exit; }

function isort($t) {

    if (!is_array($t))
        return -1;

    $c = count($t);

    /* sort in-place */
    for ($j=1; $j < $c; ++$j) {
        $key=$t[$j] + 0; $i=$j-1;

        while ( ($i >= 0) && ( $t[$i] > $key) )
            $t[$i+1] = $t[$i--];

        $t[$i+1]=$key;
    }

    return $t;
}

/* INCLUDE USER LOCAL LIBRARY */
if ( file_exists('./library_user.php') )
    include('./library_user.php');

?>

```

The first step of the application is to ask for a list of values — numbers — and the second step is to press the button “Sort”. The result is either an error message or an ascended sorted list.

The Node Methodology dictates the flow of the application. This means that no matter what your application does — sort values, send emails, search in a database, connect to an IRC server — the flow is always the same: (1) requests go through index.php, (2) the right node is spotted, (3) instructions are executed.

EVERY request is made to the index.php file. The index.php MAY execute steps before it handles the request. Then it handles the request based on the NODE variable passed to the system. In case no NODE variable was passed, the default “HOME” is used.

This rule allows a programmer to work on a very specific step of a complex — tens or hundreds of nodes (or steps) — because if a programmer is working on an application such as sort-0.1.tar.gz and she or he wants to make a change at some part of the application, it's only necessary to find out which NODE needs the change and move right to the place where that NODE is processed by index.php.

## 12 Packaging software

A methodology is also necessary to package software. A few rules and a program that check these rules may help programmers to package their software and make it easy to test installations. The package `packager-0.2.tar.gz`<sup>5</sup> will automatically create a package of a software if these rules are respected:

1. Your software must live in a directory named `name-m.n/`
2. All files that compose the software must live in `name-m.n/`
3. All files that compose the software must be listed in `FILES`
4. The package name must be written in the first line of `PACKAGE`
5. The package version must be written in the first line of `VERSION`

We demonstrate an example of working with `packager-0.2`.

```
%ls sort-0.1/
ChangeLog      PACKAGE      index.php
FILES          VERSION      library.php

%cat sort-0.1/FILES
./ChangeLog
./FILES
./PACKAGE
./VERSION
./index.php
./library.php

%packager sort-0.1/
sort-0.1/./ChangeLog
sort-0.1/./FILES
sort-0.1/./PACKAGE
sort-0.1/./VERSION
sort-0.1/./index.php
sort-0.1/./library.php
-rw-r--r--  1 dbastos  dbastos  1401 Dec 11 05:52 sort-0.1.tar.gz
```

Now, one may use `scp` command to release the package to the public.

```
scp sort-0.1.tar.gz dbastos@math.utoledo.edu:~/html/files/
```

---

<sup>5</sup>Download <http://math.utoledo.edu/~dbastos/files/packager-0.2.tar.gz>