

# PHP

## Building a Site Engine Using PHP, Part 3

Contributed by James Murray

2004-06-28

[ Send Me Similar Content When Posted ]

[ Add Developer Shed Headlines To Your Site ]



DISCUSS



NEWS



SEND



PRINT



PDF

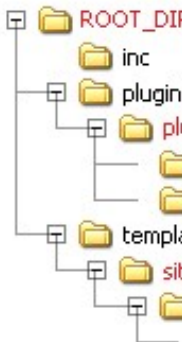


advertisement

In the third article of the series, I'll show you how the database and directories should be set up. I'll also talk about the authentication methods for such a project so that multiple sites can run off the same users table in the database, while not accurate and secure from break-ins. (For part 2, [see here](#).)

### *Give Me Direction*

The file system is very important to the site engine. If a directory isn't named right, the engine won't know it's there, in designed it, my ability should be pretty good). With the help of my friends "Windows Explorer" and "Adobe Photoshop"



Fig

The best way to explain this is to say that the directories labeled in black are the ones that must be labeled the way they a optional folder in case you have images, CSS, JavaScript, or whatever you want to put in it that's relevant to your theme

The "ROOT\_DIR" directory can be named whatever you would like to be named as long as the server is set up to know

The "inc" folder should always be named "inc". This is where all the engine files reside.

The “plug-ins” directory should always be named “plug-ins”. This is the directory that we’ll be putting all our plug-ins

The “plug-in\_name” directory should be named accordingly, based on what plug-in is contained in it. This is also known as the engine knows the difference by the name of the directory it is in. There needs to be one directory for each plug-in.

The “blocks” directory should always be named “blocks”. This is where all the blocks that depend on the plug-in, whose

The “modules” block directory should always be named “modules”. This is where all the modules that depend on the plug-in

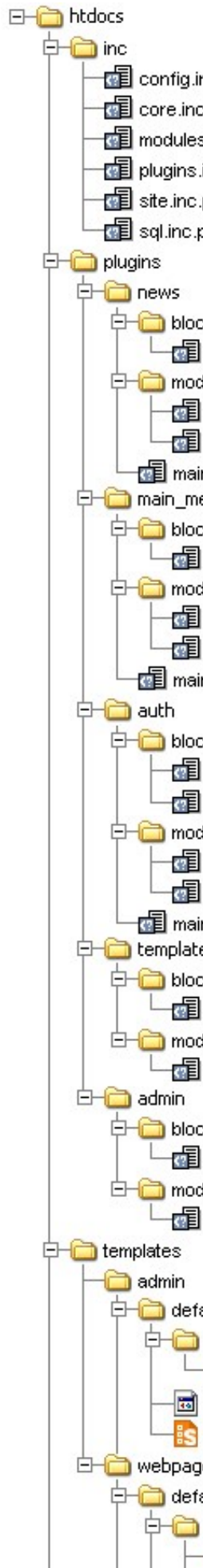
The “templates” directory should always be named “templates”. This is where all the files that define the templates for each

The “site\_name” directory should be the same as the host that is defined for the particular site. If your site is hosted at www.url.com the site is hosted at www.url.com the host would be "url", therefore the directory’s name would be “url”. There needs to be one

The “template\_name” directory should be named the same as the name of the template. If your template is named “Default

The “images” directory can be named whatever you want it to be named. I just included it in the diagram as an example

Here is an example of what the directory would look like for the engine running 2 sites, one of the sites having two templates



**Note.** The two directories that are marked with an asterisk (Fig. 1) can actually be renamed to anything you want, although module systems

That sums it all up for the file system. It's actually pretty straight forward and painless when you actually start to work w

The database is just as easy and logical as the file system — actually it's a lot easier. The first thing I'd like to show you

**Block\_location:** This table contains all the data that directs the blocks on where to load, when to load, and what site to lo

**Blocks:** This table contains all the data for the block's title, filename, plug-in dependence, module dependence, authenti

**Group\_users:** This table contains all the data that specifies what groups a user belongs to and on which site.

**Groups:** This table contains all the data that defines the group's names.

**Module\_status:** This table contains all the data that regulates what modules are initialized and for which site.

**Modules:** This table contains all the data that specifies a module's name, filename, directory, author name, version numb

**Plugin\_status:** This table contains all the data that regulates what plug-ins are initialized and for which site.

**Plugins:** This table contains all the data that specifies a plug-in's name, filename, directory, author name, version numb

**Sites:** This table contains all the data that defines each site's name and host.

**Template\_users:** This table contains all the data that specifies what template is currently being used, and by what user (i

**Templates:** This table contains all the data that defines each template's name, site, file, and status.

**Users:** This table contains all the data that defines each user's name, password, and what site it is registered on.

Here is the actual SQL dump of the site engine's database. If you would like to, can load this into a query and set up you

```
# Table: 'block_location'
#
CREATE TABLE `block_location` (
  `bl_ID` int(11) NOT NULL auto_increment,
  `block_ID` int(11) default '0',
  `block_row` int(11) default '0',
  `block_col` int(11) default '0',
  `block_page` varchar(255) default 'all',
  `site_ID` int(11) default '1',
```

```
`user_ID` int(11) default '0',

PRIMARY KEY (`bl_ID`),

UNIQUE KEY `bl_ID` (`bl_ID`),

KEY `bl_ID_2` (`bl_ID`)

) TYPE=MyISAM;

# Table: 'blocks'
#
CREATE TABLE `blocks` (

  `block_ID` int(11) NOT NULL auto_increment,

  `block_title` varchar(255) default 'Block Title',

  `block_file` varchar(255) default '0',

  `plugin_ID` int(11) default '0',

  `group_ID` int(11) default '0',

  `site_ID` int(11) default NULL,

  `mod_ID` int(11) default NULL,

  PRIMARY KEY (`block_ID`),

  UNIQUE KEY `block_ID` (`block_ID`),

  KEY `block_ID_2` (`block_ID`)

) TYPE=MyISAM;

# Table: 'group_users'
#
CREATE TABLE `group_users` (

  `gu_ID` int(11) NOT NULL auto_increment,

  `group_ID` int(11) default '0',

  `user_ID` int(11) default '0',

  `site_ID` int(11) default '0',
```

```
PRIMARY KEY (`gu_ID`),  
UNIQUE KEY `gu_ID` (`gu_ID`),  
KEY `gu_ID_2` (`gu_ID`)  
)  
TYPE=MyISAM;  
  
# Table: 'groups'  
#  
CREATE TABLE `groups` (  
  `group_ID` int(11) NOT NULL auto_increment,  
  `group_name` varchar(255) default NULL,  
  PRIMARY KEY (`group_ID`),  
  UNIQUE KEY `group_ID` (`group_ID`),  
  KEY `group_ID_2` (`group_ID`)  
)  
TYPE=MyISAM;  
  
# Table: 'module_status'  
#  
CREATE TABLE `module_status` (  
  `ms_ID` int(11) NOT NULL auto_increment,  
  `mod_ID` int(11) default '0',  
  `mod_status` varchar(255) default 'not_initialized',  
  `site_ID` int(11) default '0',  
  PRIMARY KEY (`ms_ID`),  
  UNIQUE KEY `ms_ID` (`ms_ID`),  
  KEY `ms_ID_2` (`ms_ID`)  
)  
TYPE=MyISAM;
```

```
# Table: 'modules'
#
CREATE TABLE `modules` (
  `mod_ID` int(11) NOT NULL auto_increment,
  `mod_name` varchar(255) default NULL,
  `mod_dir` varchar(255) default NULL,
  `mod_file` varchar(255) default NULL,
  `mod_author` varchar(255) default NULL,
  `mod_version` varchar(255) default '1.0',
  `mod_type` varchar(255) default 'public',
  `plugin_ID` int(11) default NULL,
  PRIMARY KEY (`mod_ID`),
  UNIQUE KEY `mod_ID` (`mod_ID`),
  KEY `mod_ID_2` (`mod_ID`)
) TYPE=MyISAM;

# Table: 'plugin_status'
#
CREATE TABLE `plugin_status` (
  `ps_ID` int(11) NOT NULL auto_increment,
  `plugin_ID` int(11) default '0',
  `plugin_status` varchar(255) default 'not_initialized',
  `site_ID` int(11) default '0',
  PRIMARY KEY (`ps_ID`),
  UNIQUE KEY `ps_ID` (`ps_ID`),
  KEY `ps_ID_2` (`ps_ID`)
) TYPE=MyISAM;
```

```
# Table: 'plugins'
#
CREATE TABLE `plugins` (
  `plugin_ID` int(11) NOT NULL auto_increment,
  `plugin_name` varchar(255) default '0',
  `plugin_dir` varchar(255) default 'plugins',
  `plugin_file` varchar(255) default '0',
  `plugin_author` varchar(255) default '0',
  `plugin_version` varchar(255) default '0',
  `plugin_type` varchar(255) default 'private',
  `plugin_priority` int(11) default '0',
  PRIMARY KEY (`plugin_ID`),
  UNIQUE KEY `plugin_ID` (`plugin_ID`),
  KEY `plugin_ID_2` (`plugin_ID`)
) TYPE=MyISAM;

# Table: 'sites'
#
CREATE TABLE `sites` (
  `site_ID` int(11) NOT NULL auto_increment,
  `site_name` varchar(255) NOT NULL default '0',
  `site_host` varchar(255) default '0',
  PRIMARY KEY (`site_ID`),
  UNIQUE KEY `site_ID` (`site_ID`),
  KEY `site_ID_2` (`site_ID`)
) TYPE=MyISAM;
```

```
# Table: 'template_users'
#
CREATE TABLE `template_users` (
  `tu_ID` int(11) NOT NULL auto_increment,
  `user_ID` int(11) default '0',
  `t_ID` int(11) default '0',
  `site_ID` int(11) default '0',
  PRIMARY KEY (`tu_ID`),
  UNIQUE KEY `tu_ID` (`tu_ID`),
  KEY `tu_ID_2` (`tu_ID`)
) TYPE=MyISAM;

# Table: 'templates'
#
CREATE TABLE `templates` (
  `t_ID` int(11) NOT NULL auto_increment,
  `t_name` varchar(255) default NULL,
  `t_file` varchar(255) default NULL,
  `t_status` varchar(255) default '0',
  `site_ID` int(11) default NULL,
  PRIMARY KEY (`t_ID`),
  UNIQUE KEY `t_ID` (`t_ID`),
  KEY `t_ID_2` (`t_ID`)
) TYPE=MyISAM;

# Table: 'users'
#
CREATE TABLE `users` (
  `user_ID` int(11) NOT NULL auto_increment,
```

```

`user_name` varchar(255) default NULL,
`user_pass` varchar(255) default NULL,
`site_ID` int(11) default '1',
PRIMARY KEY (`user_ID`),
UNIQUE KEY `user_ID` (`user_ID`,`user_name`),
KEY `user_ID_2` (`user_ID`)
) TYPE=MyISAM;

```

Now it's about time to talk about the block system. Don't worry if the concept escapes you; it's taken me a lot of time and about how it works so you don't have to pull your hair out. The block system can be very confusing if you don't pay close

The block system is very intricate, so take your time when working with it or it can get *very* out of hand. At the beginning you to declare your variables at the beginning of a class with public or private. We want to define all ours as public so the

```

public $bcount;
public $blocks_list;
public $columns;
public $rows;
public $cur_block;
public $cur_tags;

```

As you can see, we'll use nine variables through out the blocks plug-in. \$bcount is a variable that will hold the number of columns that's in our template. \$rows is basically the same as \$columns, but instead of holding an array of columns, it's the block that's currently being parsed, in our script. \$cur\_tags is the array that will hold all the block information that we need. It's an array that holds all the information about our blocks, information such as the block's titles, content, what column

Now run our SQL query to select all the blocks that will be shown on the current page. Start by making our function. Name it

```
function blocks(){
```

Now declare all the variables that we'll use as global. That way we'll be able to use all the data from the other classes in our script just yet.

```

global $sql,$plugins,$modules,$site,$user;

$this->bcount=0;

```

Now that we're started on the function, we want to start a loop to make part of our query; this particular loop is going to get the value of each group ID. Then it will generate the part of our query that makes it so that only blocks from that group ID generates a query or part of a query, it's usually a good idea to echo the query to the browser to make sure it's generating

```

$i=0;
$b_grp="";
while($i<$count=count($user->user['gids'])){
    $b_grp.="blocks.group_ID = '{$user->user['gids'][$i]}'";
}

```

```

    $i++;
    if($i==$count){
        $b_grp.=" ";
    }else{
        $b_grp.=" OR ";
    }
}
}

```

Now it's time for the meat of the blocks plug-in, the main SQL query. This query is fine tuned to select only the block in blocks to be selected, the blocks must have a plug-in that's been initialized; they must have a module that's been initialized; they must have authentication levels as the user requesting them; and they must be set to show up on the current page, or all pages. Another example can see in the following query.

```

$blocks=$sql->_query("SELECT
    `blocks`.`block_ID`,
    `blocks`.`block_title`,
    `blocks`.`block_file`,
    `block_location`.`block_col`,
    `block_location`.`block_row`,
    `plugins`.`plugin_dir`,
    `plugins`.`plugin_file`
FROM
    `block_location`,
    `blocks`,
    `plugin_status`,
    `plugins`,
    `module_status`,
    `modules`
WHERE
    (`blocks`.`block_ID` = `block_location`.`block_ID`)
    (`blocks`.`plugin_ID` = `plugin_status`.`plugin_ID`)
    (`plugins`.`plugin_ID` = `plugin_status`.`plugin_ID`)
    (`plugin_status`.`plugin_status` = 'initialized')
    (`blocks`.`mod_ID` = `module_status`.`mod_ID`)
    (`modules`.`mod_ID` = `module_status`.`mod_ID`)
    (`module_status`.`mod_status` = 'initialized')
    $bgs
    (`block_location`.`site_ID` = '{ $site->site['ID'] }')
    ((`block_location`.`block_page` = 'all') OR
    (`block_location`.`block_page` = '{ $site->site['ID'] }'))
ORDER BY
    `block_location`.`block_col`,
    `block_location`.`block_row`");

```

After the query returns the results in the `$blocks` variable, get the information from it with the `_fetch_object()` method and put it into another array that will be a little more user friendly to us later on in the project. The reason the new array is easier to work with is that it's set as `$blocks_list[1][2]` (read as blocks list column 1 row 2) as the 2nd block down in the 1st column.

```

while($block=$sql->_fetch_object($blocks)){
    if(file_exists("{ $block->plugin_dir }/{ $block->plugin_file }/block_{$block->block_ID}.php")){

```

```

$this->blocks_list['column'.$block->block_col][$block->block_row]['block_pa
    $this->blocks_list['column'.$block->block_col][$block->block_row]
    $this->blocks_list['column'.$block->block_col][$block->block_row]
}else{
    $this->blocks_list['column'.$block->block_col][$block->block_row]
    $this->blocks_list['column'.$block->block_col][$block->block_row]
    $this->blocks_list['column'.$block->block_col][$block->block_row]
}
}
}

```

For example, if I were to replace the variables in the following code with their values, for the 1st block in the 2nd column

```

$this->blocks_list['column2'][1]['block_path']="plugins/test/blocks/test.bl
$this->blocks_list['column2'][1]['block_ID']= "4";
$this->blocks_list['column2'][1]['block_title']="Test Block";

```

The reason we use the actual words “column1”, “column2”, and “column3” instead of just using the column numbers is to build the `$blocks_list` array, we’ll go ahead and call the function `parse_blocks()` which actually puts the block into the new article.

```

    $this->parse_blocks();
}

```

The `parse_blocks` function is an important part of the blocks plug-in, if not the most important. What it does is it loops through each row. After that it calls the block’s template data from the template plug-in, parses it all into an array that contains strings and the column name that the data will be put into. After the array is built we send all the data to the template plug-in where it gets used.

This function is actually a fairly simple process. First we name the function, and set up our global variables.

```

function parse_blocks(){
    global $template,$plugins;
}

```

Then we check to make sure the `$blocks_list` array have been built without errors

```

if(is_array($this->blocks_list)){

```

If the array looks like it’s all right, we then get the keys from the array that contains all the column names that is produced. We then use those keys to select each element from the `$columns` array, that will then make yet another array based on the keys from the `$columns` array. We then loop through each block in the `$blocks_list` array.

```

    $this->columns=array_keys($template->columns);
    foreach($this->columns as $column){
        $this->rows=array_keys($this->blocks_list[$column]);
        foreach($this->rows as $row){

```

The easiest way to work with the current block at this level is to set it as the value of another easier to work with variable

```

    $this->cur_block = $this->blocks_list[$column][$row];

```

After that’s all taken care of, we’ll go ahead and get “down and dirty” with our block data. First, set up a temporary array and the block ID.

```
$this->cur_tags=array( 'block_title'=>$this->cur_block[ 'b
```

To get the file that contains the block's content, we'll have to include it and set it to a variable. That will contain the data that will show up at the top of the page instead of in the block. To fix this we turn to output buffering. Traditionally output buffering is stopped to the browser. In this case we'll still tell it to stop buffering, but instead of having it send the data to the browser, we'll store it in the current buffer. This way we get all the contents of our include without it sending the data directly to the browser. After we

```
ob_start();
@include($this->cur_block[ 'block_path' ]);
$bdata=$template->replace($this->cur_tags,ob_get_clean());
$this->cur_tags[ 'block_content' ]=$bdata;
```

Don't worry it's winding down to the end of the blocks plug-in. now we have all our block data right where we want it. We'll use a function which will fill in all the tags in our block template with the data in the \$cur\_tags array. Then it will assign it to the \$cur\_block function that builds the columns and outputs it all to the browser in the, in the templates plug-in.

```
$this->cur_column[$column].=$template->replace($this->cur_tags,$bdata);
unset($this->cur_block,$this->cur_tags,$bdata);
$this->bcount++;
    }
}
$template->build_cols($this->cur_column);
}
}
}
```

The user authentication is relatively easy to work with — easier than the blocks. The most important thing about the user authentication is what everyone wants to get too. Setting cookies can be a security risk, if you don't use them correctly. With the following code for every site that you host with the site engine not the mention you make one array that you can use anywhere in the engine.

First we need to define our class and the variables we'll be using. \$usr will hold all the information about a user in an array. \$password is the user's password that is passed to the script by the login form.

```
class user{

    public $user;

    public $username;
    public $userpass;
```

Unlike the other classes in the site engine, we want to name the actual login function something different than the class name. We'll name it `__check_login()`. Also set up the global variables we'll be using and set up the password to match the md5's password to it.

```
function __check_login(){
    global $sql,$site;

    $pass = md5($this->userpass)
```

Now get all our user's information from the database. Put the query function into the `fetch_row()` function to kill two birds with one stone. If it was a successful login. Then set the information in the `$user` with the corresponding information that we got from the database.

```

        $cl=$sql->_fetch_row($sql->_query("SELECT `users`.`user_ID`,`users`.`user_name`,`users`.`site_ID`FROM`users`WHERE (`users`.`site_ID` = '{ $site->site['ID'] }') AND (`users`.`user_ID` = '{ $user->user['ID'] }')");
    if(is_array($cl)){
        $this->user['ID'] = $cl[0];
        $this->user['name'] = $cl[1];
        $this->user['pass'] = $cl[2];
        $this->user['site'] = $cl[3];
    }

```

Next we come to the group users table in the database; we wouldn't be able to do the next step without it. After the user's information is set, we need to find out what authentication groups the user belongs to. Since all users belong to at least two user groups, the result would be an array. During the while loop, set the values to the `user['gids']` array.

```

        $gids=$sql->_query("SELECT `groups`.`group_ID`FROM`group_users`,`groups`WHERE (`group_users`.`user_ID` = '{ $this->user['ID'] }') AND (`group_users`.`site_ID` = '{ $site->site['ID'] }')");
        while($gid=$sql->_fetch_row($gids)){
            $this->user['gids'][] = $gid[0];
        }
    }

```

After the while loop, check to see if the `user['gids']` has any values in it, if it doesn't, assign the group ID's for the user.

```

        if(!isset($this->user['gids'])){
            $this->user['gids'][] = "1";
            $this->user['gids'][] = "2";
        }
    }

```

Finally after the group ID's are taken care of, end the *if* statement that we started when checking to see if the results from the database are the same as the information for a visitor to the user array. This doesn't bar the user from trying to login again; it just sets the user array to the correct information.

```

        }else{
            $this->user['ID'] = 0;
            $this->user['name'] = "Guest";
            $this->user['site'] = $site->site['ID'];
            $this->user['gids'][] = "1";
            $this->user['gids'][] = "2";
        }
    }
}

```

### ***Final Thoughts***

If you thought the blocks plug-in was confusing, just keep looking at it and thinking of it line by line. Follow the variables and see how they work. The authentication plug-in also shows a few nice ways to work with and utilize arrays in your everyday scripts.

The file system can be changed to your likings -- just remember if you change it, that you'll have to update the plug-in. It's a good idea to make a test plug-in, a test module, and a test block. That way you can follow it by replacing the variables with your own.

In the next article, I'll be telling you about the template system and how to get your actual engine up and running in diff hopefully by then you'll be able to see fully what it does, how it does it, and why it does those things. By then you should