10
Panels and Widgets – Part One

A **panel** is an area in your design which can contain a number of widgets. These widgets can be installed by simply "dropping" them into place using the admin area.

A **widget** is a small visual object such as a poll, login box, search box, news scroller, and so on, which you may want to place on your site.

Items placed in a panel can be seen on every page of the site.

In this chapter, we will learn how to:

- Create the panel plugin
- Create the content snippet widget
- Add widgets to panels
- Display panels and widgets on the front-end

This is another two-part chapter.

This first chapter will develop panels and widgets enough that they can be created and seen on the front-end.

The next chapter will enhance this foundation and let you customize the widgets, and choose which pages the panels and widgets should be visible on.

Creating the panel plugin

Usually when creating a website, a header, footer, and sidebar will be written into the theme which are specific to the company the website is for.

The header would include the company name, logo, and maybe a collage of pertinent images.

The footer would include the company name, trademarks, maybe a contact number.

The sidebar would include contact details of the company.

If you replace those three areas with panels (named "header", "footer", "sidebar"), each of which can contain a widget which provides the required HTML for the company-specific details, then this allows you to create a generic web design which can be customized for a specific company, and yet be reused by another customer if you wish.

The Content Snippet widget is just that—it's a snippet of HTML content which you want to have shared among all your pages.

Create the directory `/ww.plugins/panels` and add this `plugin.php` to it:

```
<?php
$plugin=array(
  'name'=>'Panels',
  'description'=>
    'Allows content sections to be displayed throughout the site.',
  'admin'=>array(
    'menu'=>array(
      'Misc>Panels'=>'index'
    )
  ),
  'frontend'=>array(
    'template_functions'=>array(
      'PANEL'=>array(
        'function' => 'panels_show'
      )
    )
  ),
  'version'=>4
);
function panels_show($vars){
}
```

We'll leave the front-end function blank for now.

Notice that we've added a template function to the configuration array. Because we are talking about adding panels to specific parts of the site design, we need to add code to the theme's template files to say where the panels go. We do this by adding the `PANEL` function to Smarty, which will call `panels_show()` when it is used in the design. The `template_functions` array will be explained later.

An example of its use:

```
<html>
  <body>
    {{$PANEL name="header"}}
    <p>page content goes here</p>
    {{$PANEL name="footer"}}
  </body>
</html>
```

At the moment, our CMS doesn't actually add that function to Smarty, but we'll get to that.

Notice as well that the configuration set the version number to `4`. This is because there were four database table revisions I made while developing the plugin.

I've combined all four into one step here. Add the `/ww.plugins/panels/upgrade.php` file:

```
<?php
if($version<4){ // panels table
  dbQuery('CREATE TABLE IF NOT EXISTS `panels` (
    `id` int(11) NOT NULL auto_increment,
    `name` text,
    `body` text,
    `visibility` text,
    `disabled` smallint default 0,
    PRIMARY KEY  (`id`)
    ) ENGINE=MyISAM DEFAULT CHARSET=utf8');
  $version=4;
}
```

The panels table includes these fields:

`id`	Internal ID of the panel.
`name`	The name of the panel. This is used in the design template. For example: `{{PANEL name="footer"}}`
`body`	A JSON array containing details of any contained widgets.
`visibility`	A list of pages which this panel is visible on. If left blank, it's visible on all of them.
`disabled`	This allows you to "turn off" a panel so it's not visible at all on the front-end, yet keeps its settings.

The reason we detail the contained widgets in a JSON array instead of a related database table is that database access is usually slower than simply reading a text file, especially if there are quite a few records.

In any one panel, there would usually be only two or three widgets. Running a database search for that number of items is silly when you can include their details in the container panel's result instead.

Registering a panel

A panel is tied explicitly to a name, which is referred to in a template. When you create the template, you write into it where you want the panel to appear by using that name.

There are two ways to know the names of the panels that are contained in any template:

1. Decide beforehand what names are allowed.
2. Parse the templates to extract the names.

The first option is simply unreasonable. If we do decide on a certain list of allowed names, we may end up with a list of ten or more panel names, where any design may pick and choose from the list. However, we then still have the problem of showing only the active panel names to the administrator.

It is more reasonable to not restrict the list of names, but instead extract the names that the designer chose from the template itself.

The extraction itself also poses a problem. Do we parse the actual template file itself? If so, we would be writing the equivalent of a Smarty compiler.

Why not let Smarty do the work itself? The solution I've come to use is that the panel names in any template are figured out by viewing the template in the front-end and using Smarty to then register any unknown panels in a database table.

This means that by viewing the front-end of the site before you go to the administration page, we populate the table of panels with a list of actual panels that are in use by the site.

While this is not ideal, in actual usage, it's sufficient. Most website designs involve creating the pages before worrying about panels. This means that the list of panels is complete before the administrator goes to populate them.

So, first we need to edit the template and add in the panel. Open up the `/ww.skins/basic/h/_default.html` file and add in the following highlighted lines:

```
    <div id="wrapper">
      <div id="menu-wrapper">{{MENU
        direction="horizontal"}}</div>
      {{PANEL name="right"}}
      <div id="page-wrapper">{{$PAGECONTENT}}</div>
      {{PANEL name="footer"}}
    </div>
```

I've also edited the CSS file so the **right** panel, when generated, will be floated to the right-hand side and the `#page-wrapper` element leaves enough margin space for it. It's not essential for me to print that here.

Note that in the `plugin.php` file, we had this section:

```
  'frontend'=>array(
    'template_functions'=>array(
      'PANEL'=>array(
        'function' => 'panels_show'
      )
    )
  ),
```

We will use the `template_functions` array to add custom functions to Smarty, which will be executed at the appropriate place in the template.

Edit `/ww.incs/common.php`, and add the highlighted code to the `smarty_setup` function:

```
    $smarty->register_function('MENU', 'menu_show_fg');
    foreach($GLOBALS['PLUGINS'] as $pname=>$plugin){
      if(isset($plugin['frontend']['template_functions'])){
        foreach($plugin['frontend']['template_functions']
            as $fname=>$vals){
          $smarty->register_function($fname,$vals['function']);
        }
      }
    }
    return $smarty;
  }
```

Now any `template_functions` array items will be added to Smarty and can be called from the template itself.

Edit the `/ww.plugins/panels/plugin.php` file, and replace the `panels_show()` function:

```
function panels_show($vars){
  $name=isset($vars['name'])?$vars['name']:'';
  // { load panel data
  $p=dbRow('select visibility,disabled,body from panels where
      name="'.addslashes($name).'" limit 1');
  if(!$p){
    dbQuery("insert into panels (name,body)
        values('".addslashes($name)."','{\"widgets\":[]}')");
    return '';
  }
  // }
  // { is the panel visible?
  // }
  // { get the panel content
  $widgets=json_decode($p['body']);
  if(!count($widgets->widgets))return '';
  // }
}
```

This is a skeleton function. A lot of it has been left out. At the moment, all that it does is to verify that the panel is in the database and if not, the panel is added to the database.

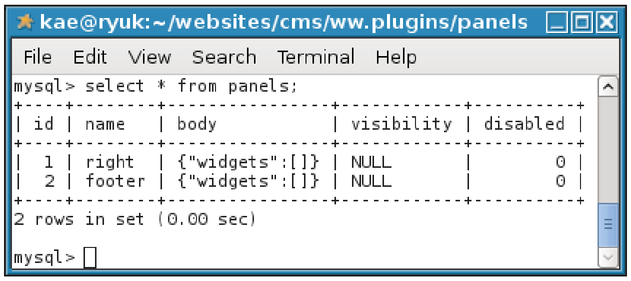
When you view the page in the browser, there is no visible difference. The HTML source has not been changed. It's as if the `{{PANEL}}` lines weren't in the template at all.

This is because if a panel is empty, it is pointless having an empty space in the page.

It is possible to have CSS change the layout of the page depending on whether the panel exists or not. That is outside the scope of the book, but if you need to know how to do it, read the following article which I wrote a few months ago:

`http://blog.webworks.ie/2010/04/27/creating-optional-columns-in-website-layouts/`

Even though there is no visible difference in the HTML, the database table has been populated:



kae@ryuk:~/websites/cms/ww.plugins/panels
File Edit View Search Terminal Help

```
mysql> select * from panels;
+----+--------+-----------------+------------+----------+
| id | name   | body            | visibility | disabled |
+----+--------+-----------------+------------+----------+
|  1 | right  | {"widgets":[]} | NULL       |        0 |
|  2 | footer | {"widgets":[]} | NULL       |        0 |
+----+--------+-----------------+------------+----------+
2 rows in set (0.00 sec)

mysql> 
```

By decoding the JSON in the body field, the function knew there were no contained widgets and returned an empty string.

The panel admin area

Now we need to populate the panels.

The panel admin section will be easy to explain the look of, but is kind of complex underneath it.

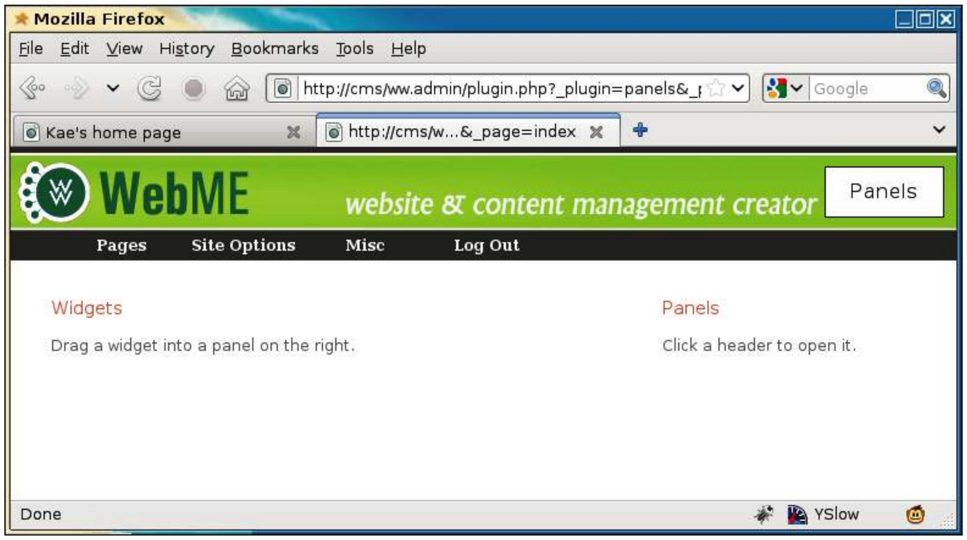
The page will be a two-column layout, with a wide column on the left-hand side holding a list of all available widgets, and a narrow column on the right-hand side listing the available panels.

We will start by building that visual.

Create the `/ww.plugins/panels/admin` directory, and in there, place an `index.php` file:

```
<?php
echo '<table style="width:95%"><tr>';
echo '<td><h3>Widgets</h3><p>Drag a widget into a panel on '
  .'the right.</p><div id="widgets"></div>'
  .'<br style="clear:both" /></td>';
echo '<td style="width:220px"><h3>Panels</h3><p>Click a '
  .'header to open it.</p><div id="panels"></div>'
  .'<br style="clear:both" /></td></tr>';
echo '</table>';
```

When viewed, we get a general idea of how this will work.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/ww.admin/plugin.php?_plugin=panels&_
Google
Kae's home page
http://cms/w...&_page=index
WebME
website & content management creator
Panels
Pages Site Options Misc Log Out
Done
YSlow

Widgets

Drag a widget into a panel on the right.

Panels

Click a header to open it.

Next, we will show the panels that we inserted into the database in the previous section of the chapter.

Showing panels

Add the following code to the `admin/index.php` file that we just edited:

```
echo '<link rel="stylesheet" type="text/css"
    href="/ww.plugins/panels/admin/css.css" />';
// { panel and widget data
echo '<script>';
// { panels
echo 'ww.panels=[';
$ps=array();
$rs=dbAll('select * from panels order by name');
foreach($rs as $r)$ps[]='{id:'.$r['id'].',disabled:'
  .$r['disabled'].',name:"'.$r['name'].'",widgets:'
  .$r['body'].'}';
echo join(',',$ps);
echo '];';
// }
// { widgets
echo 'ww.widgets=[];';
// }
// { widget forms
echo 'ww.widgetForms={};';
// }
// }
?>
</script>
<script src="/ww.plugins/panels/admin/js.js"></script>
```

When viewed in a browser, the plugin now generates the following HTML:

```
<h1>panels</h1>
<table style="width:95%">
  <tr>
    <td><h3>Widgets</h3><p>Drag a widget into a panel on the
      right.</p><div id="widgets"></div>
      <br style="clear:both" /></td>
    <td style="width:220px"><h3>Panels</h3><p>Click a header
      to open it.</p><div id="panels"></div>
      <br style="clear:both" /></td>
  </tr>
</table>
<link rel="stylesheet" type="text/css"
    href="/ww.plugins/panels/admin/css.css" />
<script>
ww.panels=[
  {id:2,disabled:0,name:"footer",widgets:{"widgets":[]}},
  {id:1,disabled:0,name:"right",widgets:{"widgets":[]}}
];
ww.widgets=[];
ww.widgetForms={};
</script>
<script src="/ww.plugins/panels/admin/js.js"></script>
```

And now we can add some JavaScript to generate the panel wrappers. Create the file `/ww.plugins/panels/admin/js.js`:

```
function panels_init(panel_column){
  for(var i=0;i<ww_panels.length;++i){
    var p=ww_panels[i];
    $('<div class="panel-wrapper '
        +(p.disabled?'disabled':'enabled')+'" id="panel'
        +p.id+'">'
        +'<h4><span class="name">'+p.name+'</span></h4>'
        +'<span class="controls" style="display:none">'
          +'<a title="remove panel" href="'
```

```
              +'javascript:panel_remove('
              +i+');" class="remove">remove</a>, '
            +'<a href="javascript:panel_visibility('
              +p.id+');" class="visibility">visibility</a>, '
            +'<a href="javascript:panel_toggle_disabled('
              +i+');" class="disabled">'
              +(p.disabled?'disabled':'enabled')+'</a>'
          +'</span></div>'
        )
        .data('widgets',p.widgets.widgets)
        .appendTo(panel_column);
    }
  }
  $(function(){
    var panel_column=$('#panels');
    panels_init(panel_column);
    $('<span class="panel-opener">&darr;</span>')
      .appendTo('.panel-wrapper h4')
      .click(function(){
        var $this=$(this);
        var panel=$this.closest('div');
        if($('.panel-body',panel).length){
          $('.controls',panel).css('display','none');
          return $('.panel-body',panel).remove();
        }
        $('.controls',panel).css('display','block');
      });
  });
```

So first, we get the `#panels` wrapper and run `panels_init` on it. This function builds up a simple element with a few links inside it:

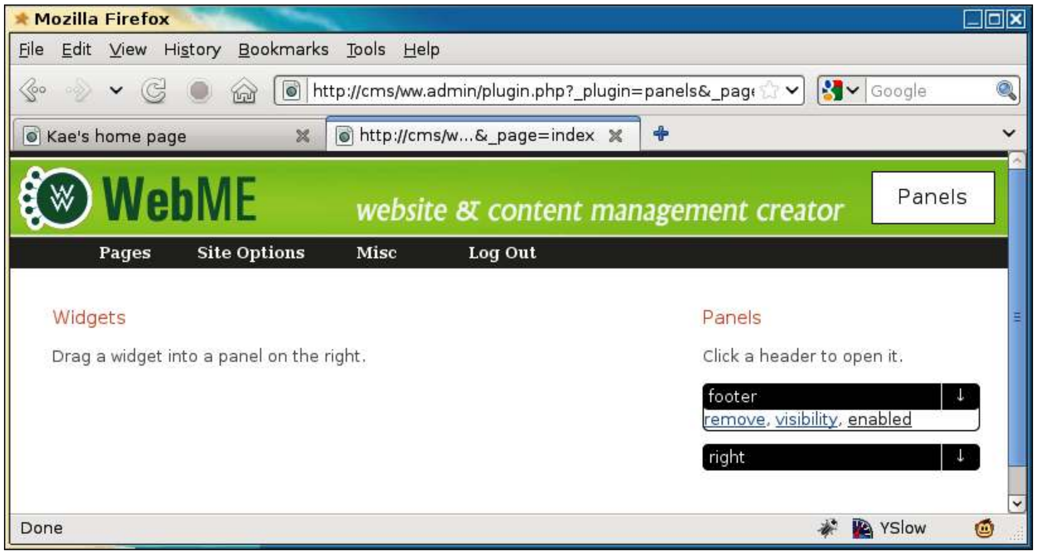
`remove`	This link is for deleting the panel if your template doesn't use it or you just want to empty it quickly.
`visibility`	This will be used to decide what pages the panel is visible on. If you want this panel to only show on the front page, for example, you would use this.
`enabled`	This link lets you turn off the whole panel and its contained widgets so you can work on it in the admin area but it's not visible in the front-end.

Notice the usage of `.data('widgets',p.widgets.widgets)`—this saves the contained widget data to the panel element itself. We'll make use of that soon.

The panels start with their bodies hidden and only their names visible (in `<h4>` elements).

After `panels_init()` is finished, we then add a down-arrow link to each of those `<h4>` elements, which when clicked will toggle the panel body's visibility.

Here's what the page looks like now, with one of the panels opened up:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/ww.admin/plugin.php?_plugin=panels&_page
Google
Kae's home page
http://cms/w...&_page=index
WebME
website & content management creator
Panels
Pages Site Options Misc Log Out

Widgets

Drag a widget into a panel on the right.

Panels

Click a header to open it.

footer

remove, visibility, enabled

right

Before we write the code for those links, we will start building the widget code—otherwise, there'd be no visible proof that the links are working.

Creating the content snippet plugin

In order to demonstrate widgets, we will build a simple plugin called content snippet. This plugin will manage small snippets of code which can be displayed anywhere that a panel is.

In my own work, I use content snippets to add editable footers, headers, and side panel content sections (for addresses, contact details, and so on) to design templates. This allows the customer to update details without needing access to the template files themselves.

Create the directory `/ww.plugins/content-snippet`, and add the following `plugin.php` file in it:

```
<?php
$plugin=array(
  'name' => 'Content Snippets',
  'admin' => array(
```

Download from Wow! eBook <www.wowebook.com>

```
    'widget' => array(
      'form_url' => '/ww.plugins/content-snippet/admin/'
        .'widget-form.php'
    )
  ),
  'description' => 'Add small static HTML snippets to any '
    .'panel - address, slogan, footer, image, etc.',
  'frontend' => array(
    'widget' => 'contentsnippet_show'
  ),
  'version' => '1'
);
function contentsnippet_show($vars=null){
  require_once SCRIPTBASE.'ww.plugins/content-snippet/'
    .'frontend/index.php';
  return content_snippet_show($vars);
}
```

Inside the admin array, we have a widget section. The `form_url` parameter points to the address of a PHP script which will be used to configure the panel.

And inside the front-end array, we have a corresponding widget section which points at a function which will display the widget.

We will need a database table for this plugin, so create the `upgrade.php` file:

```
<?php
if($version=='0'){ // add table
  dbQuery('create table if not exists content_snippets( id'
    .' int auto_increment not null primary key, html text)'
    .'default charset=utf8;');
  $version=1;
}
```

All we need for this panel is to record some HTML, which is then displayed as-is on the front-end.

After you finish writing the files, go to the **Plugins** area of the CMS and enable this new plugin, then go back to the **Panels** area.

Adding widgets to panels

We now have a very simple plugin skeleton ready to go. All we need to do is add it to a panel, configure it (by adding HTML), and then show it on the front-end.

Showing widgets

First, we need to show the list of available widgets.

Edit the file `/ww.plugins/panels/admin/index.php` and change the widgets section to this:

```
// { widgets
echo 'ww_widgets=[';
$ws=array();
foreach($PLUGINS as $n=>$p){
  if(isset($p['frontend']['widget']))$ws[]='{type:"'.$n
    .'",description:"'.addslashes($p['description']).'"}';
}
echo join(',',$ws);
echo '];';
// }
```

That will output the following to our browser:

```
ww_widgets=[{type:"content-snippet",description:"Add small static HTML
snippets to any panel - address, slogan, footer, image, etc."}];
```

Now we edit the `admin/js.js` file to show these widgets in the left-hand side column. This will involve a few small changes, so we'll step through them one at a time.

First, add the highlighted lines to the `$(function){` section:

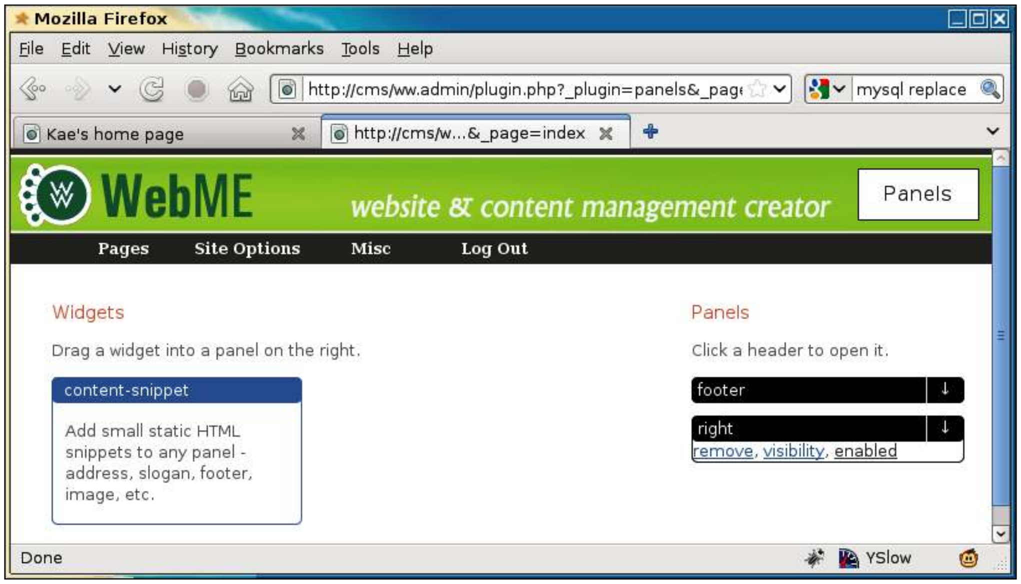
```
  panels_init(panel_column);
  var widget_column=$('#widgets');
  ww_widgetsByName={};
  widgets_init(widget_column);
  $('<span class="panel-opener">&darr;</span>')
```

Then add the `widgets_init()` function:

```
function widgets_init(widget_column){
  for(var i=0;i<ww_widgets.length;++i){
    var p=ww_widgets[i];
    $('<div class="widget-wrapper"><h4>'+p.type
        +'</h4><p>'+p.description+'</p></div>')
      .appendTo(widget_column)
      .data('widget',p);
    ww_widgetsByName[p.type]=p;
  }
}
```

This takes the global `ww_widgets` array and builds little box elements out of the data, attaching the widget data to the boxes, and then adding the boxes to the left column of the main table.

The page looks much better now, as you can see in the next screenshot:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/ww.admin/plugin.php?_plugin=panels&_page
mysql replace
Kae's home page
http://cms/w...&_page=index
WebME
website & content management creator
Panels
Pages Site Options Misc Log Out
Done
YSlow

Widgets

Drag a widget into a panel on the right.

content-snippet

Add small static HTML snippets to any panel - address, slogan, footer, image, etc.

Panels

Click a header to open it.

footer

right

remove, visibility, enabled

The next step is to take widgets from the left-hand side of the page and associate them with panels on the right-hand side of the page.

Dragging widgets into panels

Drag-and-drop is handled by using jQuery UI's Sortable plugin, which allows you to drag items from one list (the widgets list on the left-hand side) and drop into another list (the list of contained widgets in each panel on the right).

Edit the `$(function(){` section of `js.js` again and add these highlighted lines:

```
    $('.controls',panel).css('display','block');
    var widgets_container=$('<div class="panel-body">'
      +'</div>');
    widgets_container.appendTo(panel);
    $('<br style="clear:both" />')
      .appendTo(panel);
    $('.panel-body').sortable({
    });
  });
```

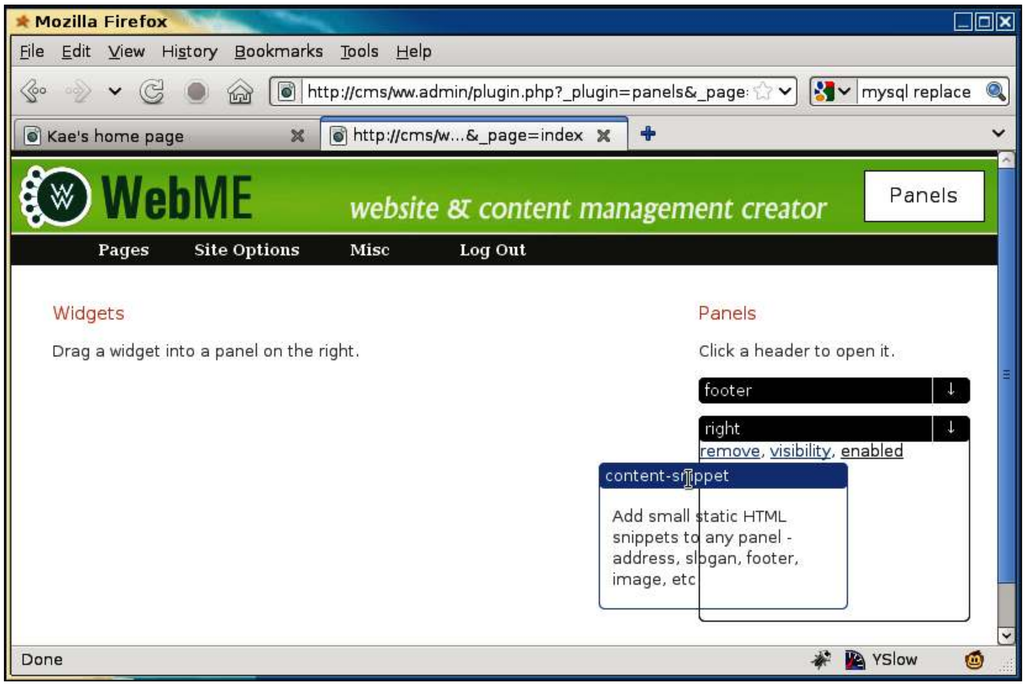
```
    $('#widgets').sortable({
      'connectWith':'.panel-body',
      'stop':function(ev,ui){
        var item=ui.item;
        var panel=item.closest('.panel-wrapper');
        if(!panel.length)return $(this).sortable('cancel');
      }
    })
    $('<br style="clear:both" />').appendTo(widget_column);
  });
```

First we add a `panel-body` element to the right-hand side panels, which will hold the widgets.

Next, we make the contents of the right-hand side `pane-body` elements `sortable`, so we can link to them with the left-hand side column widgets.

Finally, we make the left-hand side column `sortable`, linking to the `panel-body` elements on the right-hand side.

With this done, we can now drag widgets into the panels:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/ww.admin/plugin.php?_plugin=panels&_page
mysql replace
Kae's home page
http://cms/w...&_page=index
WebME
website & content management creator
Panels
Pages Site Options Misc Log Out
Widgets
Drag a widget into a panel on the right.
Panels
Click a header to open it.
footer
right
remove, visibility, enabled
content-snippet
Add small static HTML snippets to any panel - address, slogan, footer, image, etc
Done
YSlow

Unfortunately, this is not very useful, as the widget has been removed from the left-hand side column, and therefore can't be reused. For example, if you wanted to use a content snippet in each panel, you can't do that now.

So, what we need to do is "clone" the dragged item (or at least its data properties) and place the clone in the panel, then cancel the drag so the original dragged widget goes back to the **right** panel.

Edit the same file again, and add these highlighted lines:

```
'stop':function(ev,ui){
  var item=ui.item;
  var panel=item.closest('.panel-wrapper');
  if(!panel.length)return $(this).sortable('cancel');
  var p=ww_widgetsByName[$('h4',ui.item).text()];
  var clone=buildRightWidget({'type':p.type});
  panel.find('.panel-body').append(clone);
  $(this).sortable('cancel');
}
```

The above code calls a function `buildRightWidget()` with the widget name (the name of the plugin used to create the widget), and the resulting element is added to the panel instead of the actual dragged widget.

The dragged widget is then returned to its original place (by cancelling the sortable's drag) where it can be used again.

Here's the function `buildRightWidget()`, to be added to the file:

```
function buildRightWidget(p){
  var widget=$('<div class="widget-wrapper '
      +(p.disabled?'disabled':'enabled')+'"></div>')
    .data('widget',p);
  var h4=$('<h4></h4>')
    .appendTo(widget);
  var name=p.name||p.type;
  $('<input type="checkbox" class="widget_header_visibility"'
      +' title="tick this to show the widget title on the'
      +' front-end" />')
    .click(widget_header_visibility)
    .appendTo(h4);
  $('<span class="name">'+name+'</span>')
    .click(widget_rename)
    .appendTo(h4);
  $('<span class="panel-opener">&darr;</span>')
    .appendTo(h4)
```

```
      .click(showWidgetForm);
    return widget;
  }
```

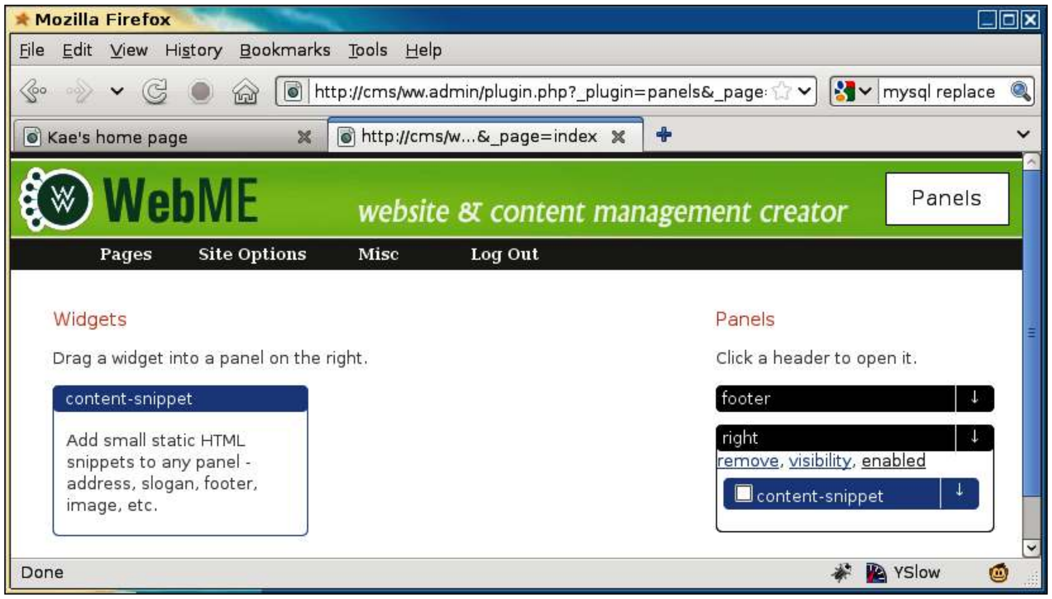
This code creates another wrapper similar to the panels wrappers, with the title of the widget visible.

There are a number of functions called with callbacks, for doing things such as renaming the widget, showing the widget name in the front-end, or showing the widget form.

We'll get to those. In the meantime, add some "stub" functions as placeholders:

```
  function showWidgetForm(){}
  function widget_header_visibility(){}
  function widget_rename(){}
```

Now, after dragging, we have a visual similar to the following screenshot:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/ww.admin/plugin.php?_plugin=panels&_page:
mysql replace
Kae's home page
http://cms/w...&_page=index
WebME
website & content management creator
Panels
Pages Site Options Misc Log Out
Done
YSlow

Widgets

Drag a widget into a panel on the right.

content-snippet

Add small static HTML snippets to any panel - address, slogan, footer, image, etc.

Panels

Click a header to open it.

footer

right

remove, visibility, enabled

content-snippet

We'll do one more thing in the admin area, then we can show the widget in the front-end.

Saving panel contents

Whenever a new widget is added to a panel, we need to rebuild the panel's body JSON string (in the panels table in the database) and save it to the server.

Edit the `js.js` file again, and add the following highlighted lines:

```
    widgets_container.appendTo(panel);
    $('<br style="clear:both" />').appendTo(panel);
    $('.panel-body').sortable({
      'stop':function(){
        updateWidgets($(this).closest('.panel-wrapper'));
      }
    });
  });
```

This code runs `updateWidgets()` whenever a widget in the **right** panel is moved around (rearranged, for instance).

Add the following highlighted code as well:

```
    var clone=buildRightWidget({'type':p.type});
    panel.find('.panel-body').append(clone);
    $(this).sortable('cancel');
    updateWidgets(panel);
  }
})
```

This adds a behavior such that when the widget is dropped into a panel body, the function `updateWidgets()` is run.

Here is the `updateWidgets()` function:

```
function updateWidgets(panel){
  var id=panel[0].id.replace(/panel/,'');
  var w_els=$('.widget-wrapper',panel);
  var widgets=[];
  for(var i=0;i<w_els.length;++i){
    widgets.push($(w_els[i]).data('widget'));
  }
  panel.data('widgets',widgets);
  var json=json_encode({'widgets':widgets});
  $.post('/ww.plugins/panels/admin/save.php',{
    'id':id,
    'data':json
  });
}
```

This function takes a panel as its parameter. It then searches the panel for any contained widgets, adds all of their contained "widget" data to its own "widgets" array, and sends that to the server to be saved.

There is no built-in JSON encoder in jQuery, so you'll need to add one.

Edit `/ww.admin/j/admin.js` and add this:

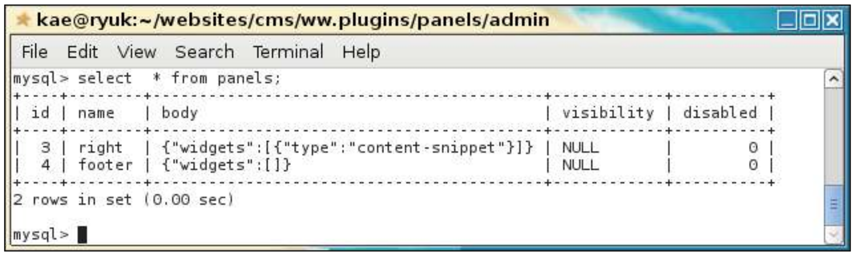
```
function typeOf(value) {
  // from http://javascript.crockford.com/remedial.html
  var s = typeof value;
  if (s === 'object') {
    if (value) {
      if (value instanceof Array) {
        s = 'array';
      }
    } else {
      s = 'null';
    }
  }
  return s;
}
function json_encode(obj){
  switch(typeOf(obj)){
    case 'string':
      return '"'+obj.replace(/(["\\])/g,'\\$1')+'"';
    case 'array':
      return '['+obj.map(json_encode).join(',')+']';
    case 'object':
      var string=[];
      for(var property in obj)string.push(
          json_encode(property)+':'
          +json_encode(obj[property]));
      return '{'+string.join(',')+'}';
    case 'number':
      if(isFinite(obj))break;
    case false:
      return 'null';
  }
  return String(obj);
}
```

The first function, `typeOf()`, is there because JavaScript's built-in `typeof` keyword doesn't differentiate between objects and arrays, and those are very different in JSON!

Here is the server-side file that saves it— /ww.plugins/panels/admin/save.php.

```
<?php
require $_SERVER['DOCUMENT_ROOT'].'/ww.admin/admin_libs.php';
$id=(int)$_REQUEST['id'];
$widgets=addslashes($_REQUEST['data']);
dbQuery("update panels set body='$widgets' where id=$id");
```

Now after dragging a content snippet widget into a panel, here is the database table:



```
mysql> select  * from panels;
+----+--------+------------------------------------------+------------+----------+
| id | name   | body                                     | visibility | disabled |
+----+--------+------------------------------------------+------------+----------+
|  3 | right  | {"widgets":[{"type":"content-snippet"}]} | NULL       |        0 |
|  4 | footer | {"widgets":[]}                           | NULL       |        0 |
+----+--------+------------------------------------------+------------+----------+
2 rows in set (0.00 sec)

mysql> 
```

We can now record what widgets are in what panels.

While interesting details of the widgets, such as customizing widget contents, are yet to be recorded, this is enough to display some stubs on the front-end, which we'll do next.

Showing panels on the front-end

We have the panel widgets in the database now, so let's write some code to extract and render them.

Edit /ww.plugins/panels/plugin.php and add the following highlighted code to the end of the panels_show() function:

```
// { show the panel content
$h='';
global $PLUGINS;
foreach($widgets->widgets as $widget){
  if(isset($PLUGINS[$widget->type])){
    if(
      isset($PLUGINS[$widget->type]['frontend']['widget'])
    ){
      $h.=$PLUGINS[$widget->type]['frontend']['widget']
        ($widget);
    }
    else $h.='<em>plugin "'
```

```
          .htmlspecialchars($widget->type)
          .'" does not have a widget interface.</em>';
      }
      else $h.='<em>missing plugin "'
        .htmlspecialchars($widget->type)
        .'".</em>';
    }
    // }
    $name=preg_replace('/[^a-z0-9\-]/','-',$name);
    return '<div class="panel panel-'.$name.'">'.$h.'</div>';
    // }
  }
```

The highlighted line does the trick. In the `plugin.php` for the content snippet plugin, we had this section:

```
    'frontend' => array(
      'widget' => 'contentsnippet_show'
    ),
```

What the highlighted line does is to call the function `contentsnippet_show()` with a parameter `$widget` which is set to the contents of the panel's `$widgets` array at that point.

So, anything that's saved in the admin area for that widget is passed to the function on the front-end as an array. We'll see more on this later.

The function `contentsnippet_show()` loads up `frontend/index.php` and then returns the result of a call to its contained `content_snippet_show()`.

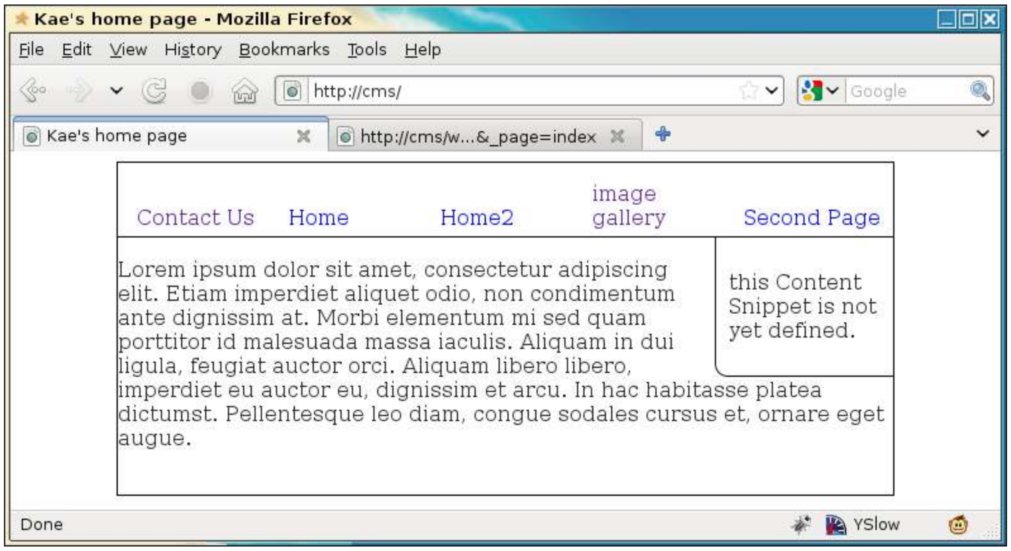
The functions have similar names—the only difference being the '_'. The one without the '_' is a stub, in case the other is not actually required. There is no point loading up a lot of code if only a little of it is actually used.

Create the directory `/ww.plugins/content-snippet/frontend`, and add the file `index.php` to it:

```
  <?php
  function content_snippet_show($vars){
    if(is_object($vars) && isset($vars->id) && $vars->id){
      $html=dbOne('select html from content_snippets
          where id='.$vars->id,'html');
      if($html)return $html;
    }
    return '<p>this Content Snippet is not yet defined.</p>';
  }
```

You can see that the function expects the parameter `$vars` to be an object with a variable `$id` in it.

We have not set that variable yet, which would correspond to a table entry in the database, so the alternative failure string is returned instead.


Kae's home page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://cms/
Google
Kae's home page
http://cms/w...&_page=index
Done
YSlow

Contact Us Home Home2 image gallery Second Page

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam imperdiet aliquet odio, non condimentum ante dignissim at. Morbi elementum mi sed quam porttitor id malesuada massa iaculis. Aliquam in dui ligula, feugiat auctor orci. Aliquam libero libero, imperdiet eu auctor eu, dignissim et arcu. In hac habitasse platea dictumst. Pellentesque leo diam, congue sodales cursus et, ornare eget augue.

this Content Snippet is not yet defined.

If you remember, we added two panels to the template—a **right** panel, visible in this screenshot, and a **footer** panel. The **footer** panel is not visible, because we haven't added anything to it yet.

Summary

In this chapter, we built the basics of a panels and widgets system.

Widgets are a way to add huge flexibility to any web design that has panels built into it.

In the next chapter, we will finish the system, letting the admin customize the widgets, and choose what pages the widgets and panels are visible on.