9
Image Gallery Plugin

When the time comes to display images in a website, whether it is a family album, a portfolio, or a series of product shots, there are a few ways to do this:

- You can insert each image manually using a rich-text editor to build up a large table
- You can select the images one-by-one from a list of images that exists on your server
- You can upload your images into a directory and have them automatically converted into a gallery

Option one can be done by using the rich-text editor—CKeditor, which we've already integrated into pages—but it's horribly tedious work building a gallery that way.

Option two would take some work to achieve, as we would need to create a list of all the images first and then create a method to select the images, store them in a database, and then, finally, create the gallery. Even then, selecting images one-by-one is probably more work than an admin should need to do.

Option three is perfect—the admin simply uploads images into a folder and a gallery is automatically created. It's even easier than that, as we will see, because we don't even need the admin to realize a directory is in use.

In this chapter, we will create an Image Gallery plugin, allowing an admin to upload a lot of images and have them automatically converted to either a slide-show gallery or a tabular gallery (we'll offer the choice).

We will not need to extend the plugin architecture any further for this one, so there will be minimal core engine editing.

Plugin configuration

Create the directory `/ww.plugins/image-gallery`, and in it, create the `plugin.php` file:

```
<?php
$kfm_do_not_save_session=true;
require_once SCRIPTBASE.'j/kfm/api/api.php';
require_once SCRIPTBASE.'j/kfm/initialise.php';

$plugin=array(
  'name' => 'Image Gallery',
    'page_type' => array(
      'image-gallery' => 'image_gallery_admin_page_form'
    )
  ),
  'description' => 'Allows a directory of images to be
      shown as a gallery.',
  'frontend' => array(
    'page_type' => array(
      'image-gallery' => 'image_gallery_frontend'
    )
  )
);

function image_gallery_admin_page_form($page,$vars){
  require dirname(__FILE__).'/admin/index.php';
  return $c;
}
function image_gallery_frontend($PAGEDATA){
  require dirname(__FILE__).'/frontend/show.php';
  return image_gallery_show($PAGEDATA);
}
```

Earlier in the book, we introduced KFM, which is an online file manager. Instead of writing a whole new file management system every time we need to manage uploads or other file stuff, it makes sense to use what we already have.

The first three lines of the `plugin.php` load up KFM in a way that can be used by the server without needing to have a pop-up window for the client.

The first line tells KFM not to bother recording or checking its database for session data. This vastly speeds up interaction with it. As we are not interested in reusing the session, it is fine to ignore it.

The second loads up some useful functions that are not used within KFM, but would be useful for external systems. For example, we will use the `kfm_api_getDirectoryId()` function, which translates a human-readable directory (such as `images/page-2`) to an internal KFM ID.

We then initialize KFM, loading the various classes and building up its base information.

The rest of the `plugin.php` is standard fare by now—we create a page type, "Image Gallery", and its associated helper functions.

Now, log in to your admin area and enable the plugin, then go to the Pages section.

Page Admin tabs

As we did with the Forms plugin, let's create a skeleton tabs list first before we fill them in.

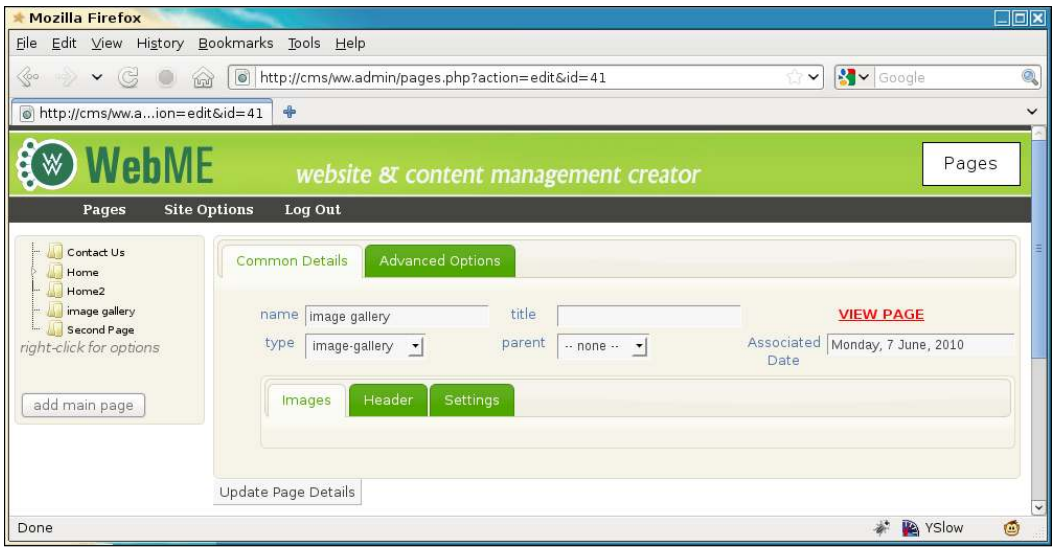
Create the directory `/ww.plugins/image-gallery/admin` and add the file `index.php` to it:

```
<?php
$c='<div class="tabs">';
// { table of contents
$c.='<ul><li><a href="#image-gallery-images">Images</a></li>'
  .'<li><a href="#image-gallery-header">Header</a></li>'
  .'<li><a href="#image-gallery-settings">Settings</a></li></ul>';
// }
// { images
$c.='<div id="image-gallery-images">';
$c.='</div>';
// }
// { header
$c.='<div id="image-gallery-header">';
$c.=ckeditor('body',$page['body']);
$c.='</div>';
// }
// { settings
$c.='<div id="image-gallery-settings">';
$c.='</div>';
// }
$c.='</div>';
$c.='<link rel="stylesheet"
    href="/ww.plugins/image-gallery/admin/admin.css" />';
```

As before, the only tab that we've fully completed is the **Header** one, which is simply a CKeditor object. This tab will appear in just about every page admin, so it makes sense to simply copy or paste it each time.

The other two tabs will be fleshed out shortly, and I won't explain the `admin.css` file (it's just style—download it from Packt's archived files for this chapter).

When viewed, this looks totally bare:



Notice that in the Forms plugin, we placed the **Header** tab first. In this one, it is the second tab.

The reason for this is that once a form is created, it is unlikely to be changed much, so if an admin is going to that page it is probably to adjust the header text, so we make that immediately available.

In an image gallery, however, the most likely reason an admin visits the page is to upload new images or delete old ones, so we make that one the first tab.

Initial settings

Before we get to work on the upload tab, we need to add some settings so the gallery knows how to behave.

Edit the `index.php` file again and add the following code after the opening `<?php`; the existing lines (beginning and end) have been highlighted:

```
<?php
// { initialise variables
$gvars=array(
  'image_gallery_directory'    =>'',
  'image_gallery_x'            =>3,
  'image_gallery_y'            =>2,
  'image_gallery_autostart'    =>0,
  'image_gallery_slidedelay'   =>5000,
  'image_gallery_thumbsize'    =>150,
  'image_gallery_captionlength'=>100,
  'image_gallery_type'         =>'ad-gallery'
);
foreach($gvars as $n=>$v)if(isset($vars[$n]))$gvars[$n]=$vars[$n];
// }
$c='<div class="tabs">';
```

This reads the page variables and if any of the `$gvars` variables are not defined, the page variable of that name is created and set to the defaults set here.

Default	Function
`image_gallery_directory`	The directory that contains the images.
`image_gallery_x`	If the gallery type is a tabular one, then x is the width in cells of that table.
`image_gallery_y`	This is the height in rows of the images table.
`image_gallery_autostart`	If the gallery type is a slide-show, then this indicates where it should start sliding when the page loads.
`image_gallery_slidedelay`	How many milliseconds between each page slide.
`image_gallery_thumbsize`	This is the size of the image thumbnails.
`image_gallery_captionlength`	How many characters to show of the image's caption before cutting it off.
`image_gallery_type`	What type of gallery to use.

As we can save these variables directly into the page variables object, we don't need to provide an external database table for them or even to save them into the site's `$DBVARS` array (that should really only be used for data that is site-wide and not page-specific).

Uploading the Images

As we discussed earlier, the easiest way to manage images is to have them all uploaded to a single directory.

It is not necessary for the admin to know what directory that is. Whenever I do anything that involves uploading user-sourced files, it is always into the KFM-controlled files area, so that the files can be manipulated in more than one way.

We will upload the files into a directory `/f/image-gallery/page-`*n*, where the 'n' is a number corresponding to the page ID.

Let's build the **Images** tab. The code is medium long, so I'll describe it in a few blocks. In total, it should replace the current images comment block in the source:

```
// { images
$c.='<div id="image-gallery-images">';
if(!$gvars['image_gallery_directory'] || !is_dir(
    SCRIPTBASE.'f/'.$gvars['image_gallery_directory'])){
  mkdir(SCRIPTBASE.'f/image-galleries');
  $gvars['image_gallery_directory']=
      '/image-galleries/page-'.$page['id'];
  mkdir(SCRIPTBASE.'f/'.$gvars['image_gallery_directory']);
}
```

Here's how it goes:

The first thing that we do is check if the `image_gallery_directory` option is set and whether the directory actually exists.

If not, the option is set to `/image-galleries/page-` plus the page ID and this directory is then created.

```
$dir_id=kfm_api_getDirectoryId(preg_replace('/^\//','',
    $gvars['image_gallery_directory']));
$images=kfm_loadFiles($dir_id);
$images=$images['files'];
$n=count($images);
```

Next, we get the internal KFM ID of that directory (the ID is created if it doesn't already exist), and then load up all files in that directory.

`$n` is set to the number of files found.

```
$c.='<iframe src="/ww.plugins/image-gallery/admin/'
  .'uploader.php?image_gallery_directory='
  .urlencode($gvars['image_gallery_directory'])
  .'" style="width:400px;height:50px;'
```

```
    .'border:0;overflow:hidden"></iframe>'
       .'<script>window.kfm={alert:function(){}};'
       .'window.kfm_vars={};function x_kfm_loadFiles(){}'
       .'function kfm_dir_openNode(){'
       .'document.location=document.location;}</script>';
```

Because all the tabs are contained in the page form, we can't have a sub-form to handle image uploads. So, we create an `<iframe>` to handle the upload.

This `<iframe>` will submit its files to KFM's `upload.php` file, which will handle their upload.

Upon a successful upload, KFM calls two functions, `x_kfm_loadFiles()` and `kfm_dir_openNode()`. We create dummy versions of these so there are no errors, and use the `kfm_dir_openNode()` call to reload the page to show the new images.

We'll create the `<iframe>`'s file after we finish this tab.

```
if($n){
  $c.='<div id="image-gallery-wrapper">';
  for($i=0;$i<$n;$i++){
    $c.='<div><img src="/kfmget/'.$images[$i]['id']
    .',width=64,height=64" title="'
    .str_replace('\\\\n','<br />',$images[$i]['caption'])
    .'" /><br /><input type="checkbox" id="image-gallery-'
    .'dchk-'.$images[$i]['id'].'" /><a href="javascript:;"'
    .' id="image-gallery-dbtn-'.$images[$i]['id']
    .'">delete</a></div>';
  }
  $c.='</div>';
}
```

If images were found in the directory, then we display the images, shrunk down to a maximum width of 64x64. The `/kfmget/...` bit will be explained shortly.

After the image is displayed, we add a check-box and **delete** link to delete the image. We'll add behaviors to those shortly.

```
else{
  $c.='<em>no images yet. please upload some.</em>';
}
$c.='</div>';
// }
```

Finally, we handle the case where there are no images, by simply asking for them to be uploaded.

Handling the uploads

In the same directory, `/ww.plugins/image-gallery/admin`, create the file `uploader.php`:

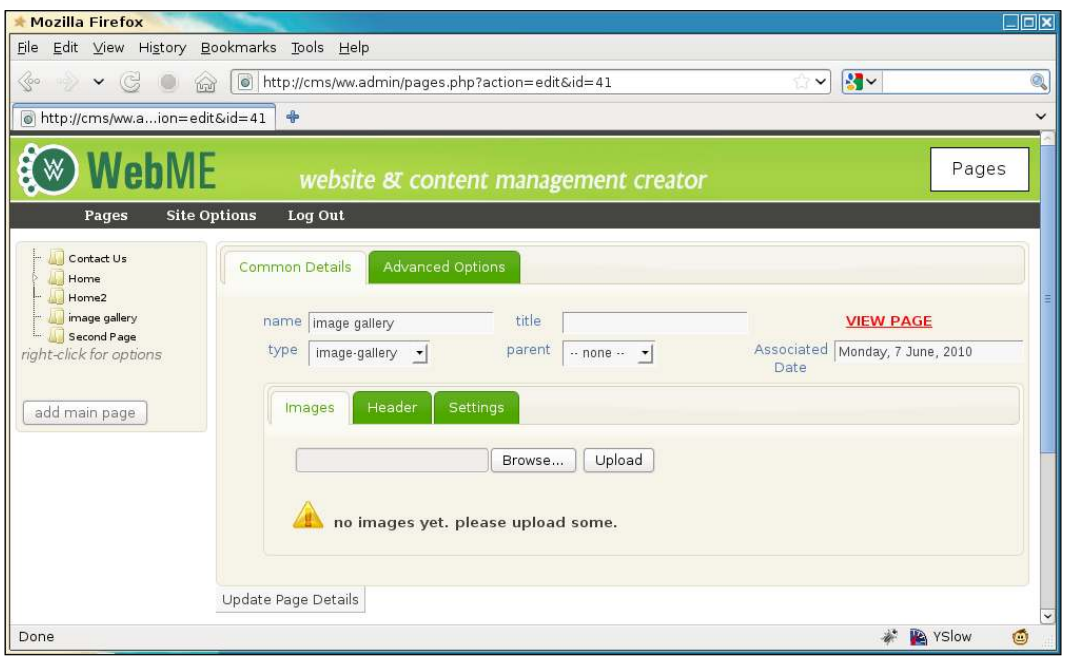
```
<?php
$dir=$_REQUEST['image_gallery_directory'];

echo '<form action="/j/kfm/upload.php" method="POST"
      enctype="multipart/form-data">
  <input type="file" name="kfm_file[]" multiple="multiple" />
  <input type="hidden" name="MAX_FILE_SIZE" value="9999999999"
     />
  <input type="hidden" name="directory_name"
      value="'.htmlspecialchars($dir).'" />
  <input type="submit" name="upload" value="Upload" />
  </form>';
```

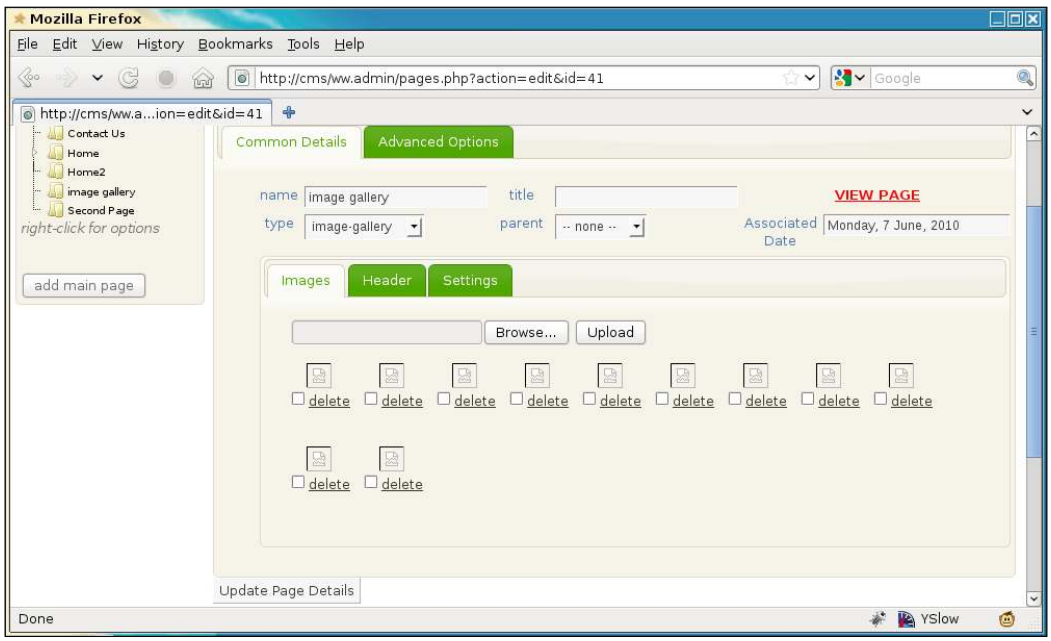
This is a very simple upload form.

Note the use of `multiple="multiple"` in the file input box. Modern browsers will allow you to upload multiple files, while older browsers will still work, but one image at a time.

With that in place, we can now view the tab:



Uploading images will work, as you can verify by looking in your `/f/image-galleries/page-n` directory, but will appear broken:



The reason for this is that the images are displayed using a `mod_rewrite` URL that we have not yet defined.

Adding a kfmget mod_rewrite rule

Here is what we provided in the source (for example):

```
<img src="/kfmget/13,width=64,height=64" ... />
```

The long version of this is:

```
/j/kfm/get.php?id=13,width=64,height=64
```

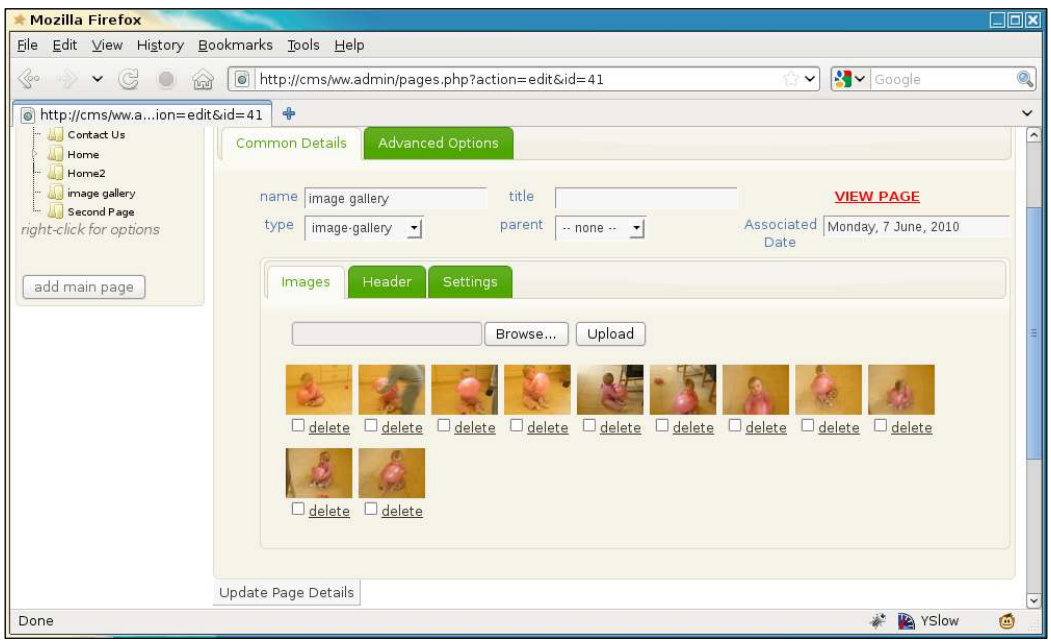
KFM understands that you want the file with ID `13` to be displayed and its size is constrained to 64x64.

Edit `/.htaccess` and add the following highlighted line:

```
RewriteEngine on
RewriteRule ^kfmget/(.*)$ /j/kfm/get.php?id=$1 [L]
RewriteRule ^([^./]{3}[^.]*)$ /index.php?page=$1 [QSA,L]
```

We place the rule before the main page rule because that one matches everything and `mod_rewrite` would not get to the `kfmget` rule.

The reason we use the shortcut in the first place is that by eliminating the `?` symbol, we allow the file to be cached. If you work a lot with images, it can be a drain on your network (and your wits) to constantly reload images that you were only viewing a moment ago.



Much better.

Now, let's work on the **delete** links.

Deleting images

In the previous screenshot, you saw that each image had a check-box and link. The check-box must be ticked before the **delete** link is clicked to verify that the admin meant to delete the image.

Add the following highlighted line to the end of the `index.php` file:

```
$c.='</div>';
$c.='<link rel="stylesheet"
    href="/ww.plugins/image-gallery/admin/admin.css" />';
$c.='<script
    src="/ww.plugins/image-gallery/admin/js.js"></script>';
```

Then create the `/ww.plugins/image-gallery/admin/js.js` file:

```
$('#image-gallery-wrapper a').bind('click',function(){
  var $this=$(this);
  var id=$this[0].id.replace('image-gallery-dbtn-','');
  if(!$('#image-gallery-dchk-'+id+':checked').length){
    alert('you must tick the box before deleting');
    return;
  }
  $.get('/j/kfm/rpc.php?action=delete_file&id='
      +id,function(ret){
    $this.closest('div').remove();
  });
});
```

First, we bind the click event to each of the delete links.

When clicked, we verify that the check-box was checked or return an alert explaining that it needs to be checked.

Finally, we call KFM through an **RPC** (**Remote Procedure Call**) to delete the file.

The RPC file is not yet a part of the official KFM distribution but was always on the plans for version 2, so here's the first implementation of `/j/kfm/rpc.php`:

```
<?php
require 'initialise.php';

switch($_REQUEST['action']){
  case 'delete_file': // {
    $id=(int)$_REQUEST['id'];
    $file=kfmFile::getInstance($id);
    if($file){
      $file->delete();
      echo 'ok';
      exit;
    }
    else die('file does not exist');
  // }
}
```

Over time, that file will grow to include all sorts of RPC commands.

With this in place, we have completed the **Images** tab.

Before we get to the **Settings** tab, we will create the front-end part of the plugin.

Upload some images so we've something to look at and then let's get to work.

Front-end gallery display

There are many ways to show lists of images on the front-end.

If you look in the "Media" section of `http://plugins.jquery.com/`, you will find many galleries and other ways of representing multiple images.

When given the choice, most people in my experience want a gallery where a list of thumbnails is shown and clicking or hovering on one of them shows a larger version of the image.

The plugin we will use here is ad-gallery (`http://coffeescripter.com/code/ad-gallery/`) but we are writing the CMS plugin such that we can easily switch to another jQuery plugin by changing the "type" select-box in the admin area.

Create the directory `/ww.plugins/image-gallery/j/ad-gallery` (create the `j` first, obviously) and then download the JS and CSS files (in the **Downloads** section of `http://coffeescripter.com/code/ad-gallery/`) to there.

Create the `/ww.plugins/image-gallery/frontend` directory, and in there, create the file `show.php`:

```
<?php
function image_gallery_show($PAGEDATA){
  $gvars=$PAGEDATA->vars;
  // {
  global $plugins_to_load;
  $c=$PAGEDATA->render();
  $start=isset($_REQUEST['start'])?(int)$_REQUEST['start']:0;
  if(!$start)$start=0;
  $vars=array(
    'image_gallery_directory'    =>'',
    'image_gallery_x'            =>3,
    'image_gallery_y'            =>2,
    'image_gallery_autostart'    =>0,
    'image_gallery_slidedelay'   =>5000,
    'image_gallery_thumbsize'    =>150,
    'image_gallery_captionlength'=>100,
    'image_gallery_type'         =>'ad-gallery'
  );
  foreach($gvars as $n=>$v)
    if($gvars->$n!='')$vars[$n]=$gvars->$n;
  $imagesPerPage=
    $vars['image_gallery_x']*$vars['image_gallery_y'];
  if($vars['image_gallery_directory']=='')
    $vars['image_gallery_directory']
```

```
    ='/image-galleries/page-'.$PAGEDATA->id;
  // }
  $dir_id=kfm_api_getDirectoryId(preg_replace('/^\//','',
      $vars['image_gallery_directory']));
  $images=kfm_loadFiles($dir_id);
  $images=$images['files'];
  $n=count($images);
  if($n){
    switch($vars['image_gallery_type']){
      case 'ad-gallery':
        require dirname(__FILE__).'/gallery-type-ad.php';
        break;
      default:
        return $c.'<em>unknown gallery type "'
          .htmlspecialchars($vars['image_gallery_type'])
          .'"</em>';
    }
    return $c;
  }else{
    return $c.'<em>gallery "'.$vars['image_gallery_directory']
      .'" not found.</em>';
  }
}
```

This script acts as a controller for the gallery, making sure default variables are set before including the requested gallery type's script.

Notice the `switch` statement. If you want to add more jQuery gallery plugins, you can add them here.

Let's create the file `/ww.plugins/image-gallery/frontend/gallery-type-ad.php`. We'll build it up a bit at a time:

```
<?php
$c.='<style type="text/css">img.ad-loader{
  width:16px !important;height:16px !important;}</style>
  <div style="visibility:hidden" class="ad-gallery">
    <div class="ad-image-wrapper"> </div>
    <div class="ad-controls"> </div>
    <div class="ad-nav"> <div class="ad-thumbs">
      <ul class="ad-thumb-list">';
for($i=0;$i<$n;$i++){
  $c.='<li> <a href="/kfmget/'.$images[$i]['id'].'">
    <img src="/kfmget/'.$images[$i]['id'].',width='
      .$vars['image_gallery_thumbsize'].',height='
```

```
        .$vars['image_gallery_thumbsize'].'" title="'
        .str_replace('\\\\n','<br />',$images[$i]['caption'])
        .'"> </a> </li>';
}
$c.='</ul> </div> </div> </div>';
```

First, we display the list of thumbnails, similar to how it was done in the admin area.

The styles and element structure are the `ad-gallery` plugin.

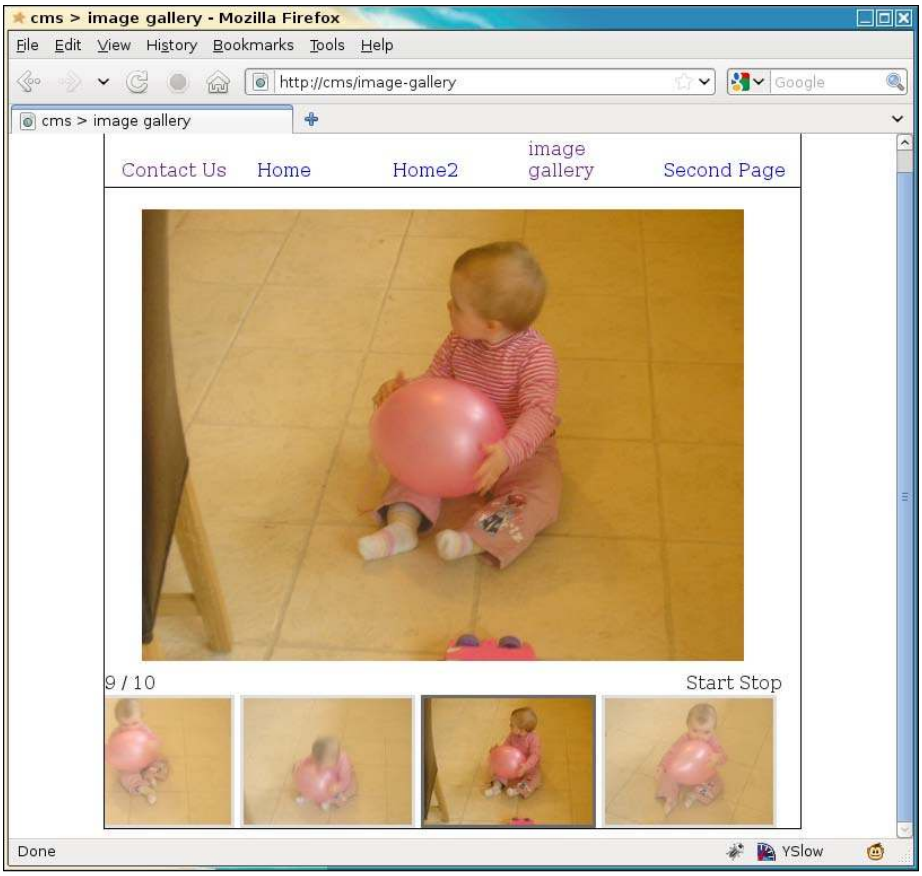
```
$c.='<script src="/ww.plugins/image-gallery/j/ad-gallery/'
  .'jquery.ad-gallery.js"></script>'
  .'<style type="text/css">@import "/ww.plugins/image-'
  .'gallery/j/ad-gallery/jquery.ad-gallery.css";'
  .'.ad-gallery .ad-image-wrapper{ height: 400px;}'
  .'</style>';
```

Next, we import the `ad-gallery` plugin and its CSS file.

```
$c.='<script>
$(function(){
  $(".ad-gallery").adGallery({
    animate_first_image:true,
    callbacks:{
      "init":function(){
        $("div.ad-gallery").css("visibility","visible");
      }
    },
    loader_image:"/i/throbber.gif",
    slideshow:{';
$slideshowvars=array();
if($vars['image_gallery_autostart']){
  $slideshowvars[]='enable:true';
  $slideshowvars[]='autostart:true';
}
$sp=(int)$vars['image_gallery_slidedelay'];
if($sp)$slideshowvars[]='speed:'.$sp;
$c.=join(',',$slideshowvars);
$c.='}
  });
});</script>';
```

Finally, we build up the start-up function that will be called when the page loads.

And you can then view the page in your browser:



This gallery should cover the most basic needs, but if you wanted to use a different gallery type, it should be simple enough to add it to this.

We'll demonstrate this by building up a simple gallery in a grid fashion.

Settings tab

First, we need to write the **Settings** tab code so we can configure it. Edit the file `/ww.plugins/image-gallery/admin/index.php` and let's replace the `settings` comment block, starting with this:

```
// { settings
$c.='<div id="image-gallery-settings">';
$c.='<table><tr><th>Image Directory</th><td><select '
```

```
    .'id="image_gallery_directory" '
    .'name="page_vars[image_gallery_directory]">'
    .'<option value="/">/</option>';
  foreach(image_gallery_get_subdirs(SCRIPTBASE.'f','') as $d){
    $c.='<option value="'.htmlspecialchars($d).'"';
    if($d==$gvars['image_gallery_directory'])
      $c.=' selected="selected"';
    $c.='>'.htmlspecialchars($d).'</option>';
  }
  $c.='</select></td>';
  $c.='<td colspan="2"><a style="background:#ff0;'
    .'font-weight:bold;color:red;display:block;'
    .'text-align:center;" '
    .'href="#page_vars[image_gallery_directory]" '
    .'onclick="javascript:window.open(\'/j/kfm/?startup_folder='
    .'\'+$(\'#image_gallery_directory\').attr(\'value\')'
    .',\'kfm\',\'modal,width=800,height=600\');">Manage '
    .'Images</a></td></tr>';
```

This first section allows finer control over where the files are uploaded to.

After that, follow the next steps:

First, we create a select-box containing all the directories in the user uploads section (`/f`), using the `image_gallery_get_subdirs()`, which we'll define in a moment.

Next, we add a link that lets you open KFM straight to that directory, so you can edit the images with more control than what was in the first tab.

```
  // { columns
  $c.='<tr><th>Columns</th><td><input '
    .'name="page_vars[image_gallery_x]" value="'
    .(int)$gvars['image_gallery_x'].'" /></td>';
  // }
  // { gallery type
  $c.='<th>Gallery Type</th><td><select '
    .'name="page_vars[image_gallery_type]">';
  $types=array('ad-gallery','simple gallery');
  foreach($types as $t){
    $c.='<option value="'.$t.'"';
    if(isset($gvars['image_gallery_type']) &&
        $gvars['image_gallery_type']==$t)
      $c.=' selected="selected"';
    $c.='>'.$t.'</option>';
  }
```

```
$c.='</select></td></tr>';
// }
```

Next, we add an input box for columns (in case the type you choose is a grid-style gallery) and a drop-down select-box to choose the gallery type.

In the `$types` array, you name the types just as the `switch` in `show.php` on the front-end expects to find them. I've named our new one "simple gallery".

```
// { rows
$c.='<tr><th>Rows</th><td><input '
  .'name="page_vars[image_gallery_y]" value="'
  .(int)$gvars['image_gallery_y'].'" /></td>';
// }
// { autostart the slideshow
$c.='<th>Autostart slide-show</th><td><select '
  .'name="page_vars[image_gallery_autostart]"><option '
  .'value="0">No</option><option value="1"';
if($gvars['image_gallery_autostart'])
  $c.=' selected="selected"';
$c.='>Yes</option></select></td></tr>';
// }
```

Next, we add an input for the rows for grid-style galleries, followed by a select-box to choose whether slide-shows should be auto-started or not.

```
// { caption length
$cl=(int)@$gvars['image_gallery_captionlength'];
$cl=$cl?$cl:100;
$c.='<tr><th>Caption Length</th><td><input '
  .'name="page_vars[image_gallery_captionlength]" value="'
  .$cl.'" /></td>';
// }
// { slide delay
$sd=(int)@$gvars['image_gallery_slidedelay'];
$c.='<th>Slide Delay</th><td><input name="'
  .'page_vars[image_gallery_slidedelay]" class="small" '
  .'value="'.$sd.'" />ms</td></tr>';
// }
```

Next, we ask for the caption length. We set its default to 100 and if we are using a slide-show, we ask for the slide-show delay (default value is set to 5000 ms).

You can set an image's caption by either editing its embedded data before uploading it, or by using KFM to edit the caption after it's uploaded (right-click on the image | **edit** | **change caption**).

```
// { thumb size
$ts=(int)@$gvars['image_gallery_thumbsize'];
$ts=$ts?$ts:150;
$c.='<tr><th>Thumb Size</th><td><input name="'
  .'page_vars[image_gallery_thumbsize]" value="'.$ts
  .'" /></td></tr>';
// }
$c.='</table>';
$c.='</div>';
// }
```

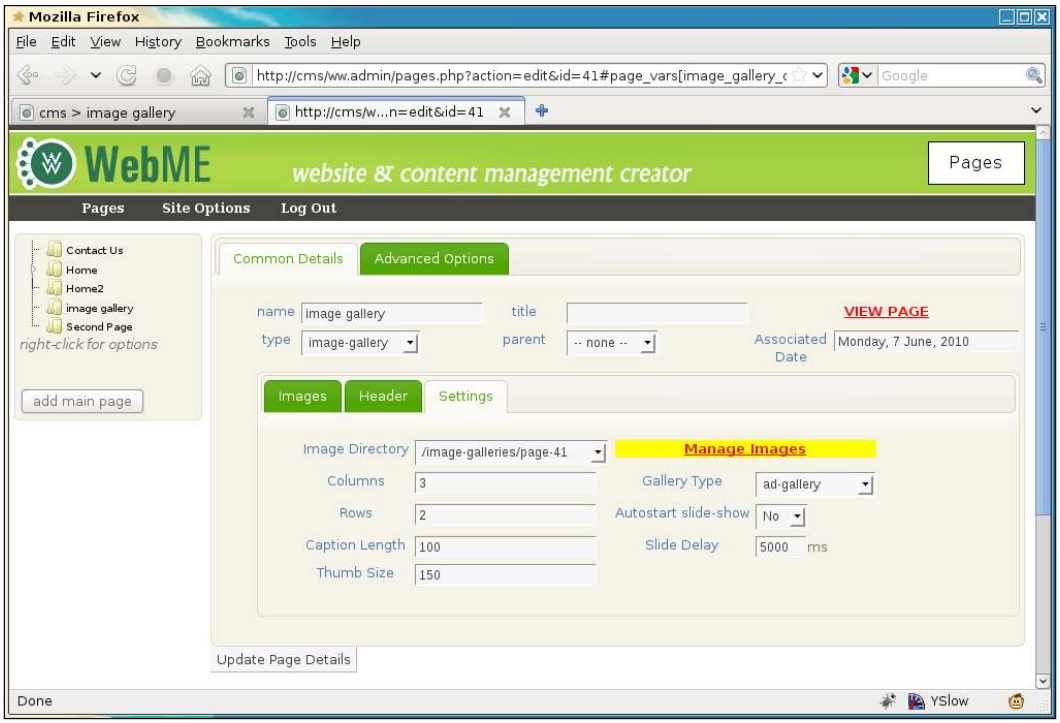
Finally, we ask for the thumb-size, and then close up the tab.

Okay, before we can view the tab, we need to create that missing function. Add this at the top of the file (after the `<?php`):

```
function image_gallery_get_subdirs($base,$dir){
  $arr=array();
  $D=new DirectoryIterator($base.$dir);
  $ds=array();
  foreach($D as $dname){
    $d=$dname.'';
    if($d{0}=='.')continue;
    if(!is_dir($base.$dir.'/'.$d))continue;
    $ds[]=$d;
  }
  asort($ds);
  foreach($ds as $d){
    $arr[]=$dir.'/'.$d;
    $arr=array_merge($arr,image_gallery_get_subdirs(
      $base,$dir.'/'.$d));
  }
  return $arr;
}
```

This function recursively builds up a list of the directories contained in the user uploads section and returns it.

Finally, we can view the tab:



That's the admin area completed.

Now, we can get back to the front and finish our grid-based gallery.

Grid-based gallery

We've already added the ad-gallery. Now let's create our grid-based gallery that will be called `simple gallery`.

Edit `/ww.plugins/image-gallery/frontend/show.php` and add the following highlighted lines to the switch:

```
    break;
case 'simple gallery':
    require dirname(__FILE__).'/gallery-type-simple.php';
    break;
default:
```

After that, create `/ww.plugins/image-gallery/frontend/gallery-type-simple.php`, again explained in parts, as follows:

```
<?php
$c.='<table id="image_gallery" class="image_gallery">';
if($n>$imagesPerPage){
  $prespage=$PAGEDATA->getRelativeURL();
  // { prev
    $c.='<th class="prev" style="text-align:left" '
      .'id="image_gallery_prev_wrapper">';
    if($start>0){
      $l=$start-$imagesPerPage;
      if($l<0)$l=0;
      $c.='<a href="'.$prespage.'?start='.$l.'">&lt;-- '
        .'prev</a>';
    }
    $c.='</th>';
  // }
  for($l=1;$l<$vars['image_gallery_x']-1;++$l)$c.='<th></th>';
  // { next
    $c.='<th class="next" style="text-align:right" '
      .'id="image_gallery_next_wrapper">';
    if($start+$imagesPerPage<$n){
      $l=$start+$imagesPerPage;
      $c.='<a href="'.$prespage.'?start='.$l.'">next '
        .'--&gt;</a>';
    }
    $c.='</th>';
  // }
}
```

This first section sets up pagination. If we have columns and rows set to 3 and 2, then there are six images per page.

If there are more than six images in the set, we need to provide navigation to those images.

This block figures out what page we're on and whether there is more to come.

```
$all=array();
$s=$start+$vars['image_gallery_x']*$vars['image_gallery_y'];
if($s>$n)$s=$n;
for($i=$start;$i<$s;++$i){
  $cap=$images[$i]['caption'];
  if(strlen($cap)>$vars['image_gallery_captionlength'])
```

```
      $cap=substr($cap,0,$vars['image_gallery_captionlength']-3)
      .'...';
    $all[]=array(
      'url'=>'/kfmget/'.$images[$i]['id'],
      'thumb'=>'/kfmget/'.$images[$i]['id'].',width='
        .$vars['image_gallery_thumbsize'].',height='
        .$vars['image_gallery_thumbsize'],
      'title'=>$images[$i]['caption'],
      'caption'=>str_replace('\\\\n',"<br />",
        htmlspecialchars($cap))
    );
  }
```

Next, we build up an array of the visible images, including details such as caption, link to original image, address of thumbnail, and so on.

```
  for($row=0;$row<$vars['image_gallery_y'];++$row){
    $c.='<tr>';
    for($col=0;$col<$vars['image_gallery_x'];++$col){
      $i=$row*$vars['image_gallery_x']+$col;
      $c.='<td id="igCell_'.$row.'_'.$col.'">';
      if(isset($all[$i]))$c.='<div style="text-align:center" '
        .'class="gallery_image"><a href="'.$all[$i]['url']
        .'"><img src="'.$all[$i]['thumb'].'" />'
        .'<br style="clear:both" /><span class="caption">'
        .$all[$i]['caption'].'</span></a></div>';
      $c.='</td>';
    }
    $c.='</tr>';
  }
  $c.='</table>';
```

Finally, we generate the table of images.

This can be enhanced by generating jQuery to manage the pagination but as this is just a demonstration of having two gallery methods for the one CMS plugin, it's not necessary to go through that trouble.

In the admin area, go to the **Settings** tab and change the gallery type to **simple gallery** and click **Update** to save it.

Now, when viewed in a browser, the page looks like this:

And when the **next -->** link is clicked, it changes to this:

Notice that the URL has been changed to add a `start` parameter and the pagination has changed. There is no **next -->** and a **<-- prev** link is added.

Also, the bottom photo on the left-hand side has a caption on it.

That's it—a completed Image Gallery plugin.

Summary

In this chapter, we created an Image Gallery plugin for the CMS, which lets you upload multiple images to a directory and choose from a list of gallery types how you want to show the images on the front-end.

In the next chapter, we will build the basics of the Panels plugin. A **Panel** is basically a "wrapper", within which widgets can be placed by the admin. It greatly extends the customizability of a site design.