



A tutorial explaining how simple component for Joomla! CMS 1.0 works, and how to create a component. One may wonder why to write about version 1.0 of Joomla!, when version 1.5 is almost ready (as of January 2007). I think version 1.0 is still a very good framework: one can get to know how Joomla! works, extend the PHP knowledge, and prepare for Joomla! 1.5, which is a lot different and more advanced. And since there rather will not be upgrade possible from 1.0 to 1.5, just migration (hopefully a fast and easy one, thanks to great developers), there may be still a lot of existing web sites working on Joomla! 1.0 for quite some time.

Official Joomla! web site contains a lot of useful information:

help.joomla.org > [Developer](#) > [Components](#)

[Joomla! Development Wiki](#)

There are of course tutorials of [Joseph LeBlanc](#) , and let's not forget about the most important part - security:

[Guide to more secure components](#)

1. What is a component ?

Component is mini-application working in Joomla! enviroment. There are a few component included with Joomla! installation: com_banners, com_contact, com_content, com_weblinks and others. These core components cannot be uninstalled, of course. Components show information in frontend and backend, as Joomla! does.

2. Simplest component

The very simplest component has only two files:

very_simple.php

very_simple.xml

Download zip file and install this sample component as usual in Joomla! Then create menu position pointing to the component. If there were no errors during installation the component should work now - it should display either "you are not logged in" or "welcome back" if you are logged in from frontend.

Joomla! installer creates two folders for the component (com_simplest), and will put these files there:

administrator/components/com_simplest/simplest.xml

components/com_simplest/simplest.php

The xml file is the "definition" file for the component - it contains details about it, list of files, SQL queries to perform during installation and after uninstalling it.

The php file is the actual frontend of a component.

3. Typical component files

The discussed component - JPortfSimple - has the following files:

1. admin.jportfsimple.php
2. admin.jportfsimple.html.php
3. toolbar.jportfsimple.php
4. toolbar.jportfsimple.html.php
5. install.jportfsimple.php
6. uninstall.jportfsimple.php
7. jportfsimple.xml
8. jportfsimple.class.php
9. jportfsimple.php
10. jportfsimple.html.php
11. CSS / jp.css
12. HELP / english.html
13. IMAGES
14. LANG / english.php

During component installation there are two folders created within Joomla!:

in administrator/components/com_jportfsimple - files 1 to 7 are copied here, this is so called *backend* and
in components/com_jportfsimple - files 8, 9, 10 are copied here, this is *frontend*.

Location of remaining folders with files (11 to 14) depends on a developer. Location of all the above files and folders is defined in XML file, which is read during installation.

4. XML file

jportfsimple.xml

```
1. <?xml version="1.0" ?>
2. <mosinstall type="component">
3. <name>jportfsimple</name>
4. <creationDate>2007/01/01</creationDate>
5. <author>Konrad Gretkiewicz</author>
6. <copyright>This component in released under the GNU/GPL License</copyright>
7. <authorEmail> kgretk@anetus.com</authorEmail>
8. <authorUrl>www.anetus.com</authorUrl>
9. <version>0.2</version>
10. <files>
11. <filename>index.html</filename>
12. <filename>jportfsimple.php</filename>
13. <filename>jportfsimple.html.php</filename>
14. <filename>jportfsimple.class.php</filename>
15. <filename>license.txt</filename>
16. <filename>_readme.txt</filename>
17. <filename>css/index.html</filename>
18. <filename>css/jp.css</filename>
19. <filename>images/index.html</filename>
20. <filename>images/no_image.jpg</filename>
21. <filename>lang/index.html</filename>
22. <filename>lang/english.php</filename>
23. </files>
24. <install>
25. <queries>
26. <query>
27. CREATE TABLE IF NOT EXISTS `#__jportfsimple_conf` (
28. `id` INT NOT NULL AUTO_INCREMENT,
29. `version` VARCHAR(20) NOT NULL,
30. `base_path` VARCHAR(50) NOT NULL,
31. `title` VARCHAR(100) NOT NULL,
32. `description` TEXT NOT NULL,
33. `css_file` VARCHAR(50) NOT NULL,
34. PRIMARY KEY (`id`)
35. ) TYPE=MyISAM
36. </query>
37. <query>
38. CREATE TABLE IF NOT EXISTS `#__jportfsimple_categories` (
39. `id` INT NOT NULL AUTO_INCREMENT,
40. `cat_name` TEXT NOT NULL,
41. `cat_info` TEXT NOT NULL,
42. `cat_path` TEXT NOT NULL,
43. `meta_desc` TEXT NOT NULL,
44. `meta_keywords` TEXT NOT NULL,
45. `cat_image` TEXT NOT NULL,
46. `cat_grp` int(11) NOT NULL default '0',
47. `ordering` int(11) NOT NULL default '0',
48. `access` TINYINT(1) NOT NULL default '0',
49. `published` TINYINT(1) NOT NULL,
50. PRIMARY KEY (`id`)
51. ) TYPE=MyISAM
52. </query>
53. <query>
54. CREATE TABLE IF NOT EXISTS `#__jportfsimple_projects` (
55. `id` INT NOT NULL AUTO_INCREMENT,
56. `catid` TEXT NOT NULL,
57. `name` TEXT NOT NULL,
58. `description` TEXT NOT NULL,
59. `meta_desc` TEXT NOT NULL,
60. `meta_keywords` TEXT NOT NULL,
61. `proj_image` TEXT NOT NULL,
62. `proj_images_path` TEXT NOT NULL,
63. `userid` int(11) NOT NULL default '0',
64. `date` datetime NOT NULL,
65. `ordering` int(11) NOT NULL default '0',
66. `access` TINYINT(1) NOT NULL default '0',
67. `published` TINYINT(1) NOT NULL,
68. PRIMARY KEY (`id`)
69. ) TYPE=MyISAM
70. </query>
71. <query>
72. INSERT INTO `#__jportfsimple_conf` VALUES (1,'0.2','images/stories/','JPortSimple','Sample description.','jp.css');
73. </query>
74. </queries>
75. </install>
76. <uninstall>
77. <queries>
78. <query>
```

```

79. DROP TABLE IF EXISTS `#__jportfsimple_conf`;
80. </query>
81. <query>
82. DROP TABLE IF EXISTS `#__jportfsimple_categories`;
83. </query>
84. <query>
85. DROP TABLE IF EXISTS `#__jportfsimple_projects`;
86. </query>
87. </queries>
88. </uninstall>
89. <installfile>
90. <filename>install.jportfsimple.php</filename>
91. </installfile>
92. <uninstallfile>
93. <filename>uninstall.jportfsimple.php</filename>
94. </uninstallfile>
95. <administration>
96. <menu>jportfsimple</menu>
97. <submenu>
98. <menu act="categories">Categories</menu>
99. <menu act="projects">Projects</menu>
100. <menu act="configure">Configuration</menu>
101. <menu act="info">About</menu>
102. </submenu>
103. <files>
104. <filename>index.html</filename>
105. <filename>admin.jportfsimple.php</filename>
106. <filename>admin.jportfsimple.html.php</filename>
107. <filename>toolbar.jportfsimple.php</filename>
108. <filename>toolbar.jportfsimple.html.php</filename>
109. <filename>logo.gif</filename>
110. <filename>help/index.html</filename>
111. <filename>help/english.html</filename>
112. </files>
113. </administration>
114. <params>
115. <param name="back_button" type="list" default="" label="Back Button" description="Show/Hide a Back Button, that returns you
to the previous page">
116. <option value="">Use Global</option>
117. <option value="0">Hide</option>
118. <option value="1">Show</option>
119. </param>
120. <param name="item_navigation" type="list" default="" label="Navigation Bar" description="Show/Hide the Navigation bar">
121. <option value="">Use Global</option>
122. <option value="0">Hide</option>
123. <option value="1">Show</option>
124. </param>
125. <param name="display_num" type="list" default="50" label="Display Number" description="Number of items to be displayed by
default">
126. <option value="6">6</option>
127. <option value="10">10</option>
128. <option value="16">16</option>
129. <option value="20">20</option>
130. <option value="25">25</option>
131. <option value="30">30</option>
132. <option value="50">50</option>
133. </param>
134. </params>
135. </mosinstall>

```

lines 1 to 9 - basic component's information: name, author, version

10 to 23 - frontend files

24 to 75 - SQL queries creating new tables (CREATE) with necessary structure in database. Here 3 tables are created: jos_jportfsimple_conf (few configuration values), jos_jportfsimple_categories (for categories) and jos_jportfsimple_projects (for projects). #_ is replaced during installation with table prefix defined for your site (usually jos_). Last query (line 81) loads sample configuration values.

76 to 88 - SQL queries run during uninstalling the component, removing old tables (DROP)

89 to 94 - defines files to run after installing and uninstalling component

96 to 102 - created backend menu, in Components

103 to 112 - backend files

114 to 134 - defines parameter which can be modified in menu position for given component

5. Frontend

Class file defines classes used in the component.

```

jportfsimple.class.php
9. // no direct access

```

```

10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12.
13. class jportfsimpleConf extends mosDBTable {
14.     var $id = null;
15.     var $version = null;
16.     var $base_path = null;
17.     var $title = null;
18.     var $description = null;
19.     var $css_file = null;
20.
21.     function jportfsimpleConf(&$db){
22.         $this->mosDBTable('#__jportfsimple_conf', 'id', $db);
23.     }
24. }
25.
26. class jportfsimpleCategories extends mosDBTable {
27.     var $id = null;
28.     var $cat_name = null;
29.     var $cat_info = null;
30.     var $cat_path = null;
31.     var $meta_desc = null;
32.     var $meta_keywords = null;
33.     var $cat_image = null;
34.     var $cat_grp = null;
35.     var $ordering = null;
36.     var $access = null;
37.     var $published = null;
38.
39.     function jportfsimpleCategories(&$db){
40.         $this->mosDBTable('#__jportfsimple_categories', 'id', $db);
41.     }
42. }
43.
44. class jportfsimpleProjects extends mosDBTable {
45.     var $id = null;
46.     var $catid = null;
47.     var $name = null;
48.     var $description = null;
49.     var $meta_desc = null;
50.     var $meta_keywords = null;
51.     var $proj_image = null;
52.     var $proj_images_path = null;
53.     var $userid = null;
54.     var $date = null;
55.     var $ordering = null;
56.     var $access = null;
57.     var $published = null;
58.
59.     function jportfsimpleProjects(&$db){
60.         $this->mosDBTable('#__jportfsimple_projects', 'id', $db);
61.     }
62. }

```

There are 3 classes defined, extending Joomla's class mosDBTable, each pointing to it's database table. Of course names of all fields here and in xml file (and database) must be the same.

Main frontend file - **jportfsimple.php** - contains all logic for the component. It should not contain any output instructions, output should be done via .html.php file.

jportfsimple.php

```

9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12. global $mosConfig_live_site, $mosConfig_absolute_path, $mosConfig_lang, $database;
13.
14. // load .class.php and .html.php files for the component
15. require_once( $mainframe->getPath( 'class' ) );
16. require_once( $mainframe->getPath( 'front_html' ) );
17.
18. // load language file
19. if (file_exists( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/'.$mosConfig_lang.'.php'))
20.     include_once( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/'.$mosConfig_lang.'.php');
21. else
22.     if (file_exists( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/english.php'))
23.         include_once( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/english.php');
24.
25. // load configuration from database
26. $database->setQuery('SELECT * FROM #__jportfsimple_conf' );
27. $conf_rows = $database -> loadObjectList();
28. if ($database -> getErrorNum()) {
29.     echo $database -> stderr();
30.     return false;
31. }
32. $jpConf=$conf_rows[0];
33.
34. // add CSS file for the component
35. $mainframe->addCustomHeadTag( '<link href="'.$mosConfig_live_site.'/components/com_jportfsimple/css/'.$jpConf->css_file.'"
36.     rel="stylesheet" type="text/css" />' );
37.
38. // get option and category
39. $option = trim( mosGetParam( $_REQUEST, 'option' ) );

```

```

39. $cat = (int) ( mosGetParam( $_REQUEST, 'cat' ));
40.
41. if (!$cat)
42. {
43.     // if category not defined display all
44.     JP_categories();
45. }
46. else
47. {
48.     //display one category or one project
49.
50.     $project = (int) ( mosGetParam( $_REQUEST, 'project' ));
51.
52.     if ($project)
53.     {
54.         // display one project
55.         JP_project( $cat, $project);
56.     }
57.     else
58.     {
59.         // no project so display all in category
60.         JP_one_cat( $cat );
61.     }
62. }

```

First part of frontend file does the following:

line 15, 16 - loads the required class and frontend html (jportfsimple.html.php) files

19 to 23 - loads language file, default is english.php in com_jportfsimple/lang folder. The name of the file must be the same as the language file name for your Joomla! site

26 to 32 - reads data from database table containing configuration parameters. They are stored in \$jpConf object.

line 35 - using Joomla function addCustomHeadTag the component loads the CSS file, what allows to style component display without changing whole site's CSS.

38 to 39 - function mosGetParam should be always used to get any input from user, with trim or (int) as additional security

41 to 62 - main frontend "logic": if category (\$cat) is not defined, then display all categories, else try to read project number, and if it is defined, display one project, else display all projects in given category. That's it! Of course in case of more complicated component, here you would have case statement, with more variables.

Next, let's look at each function.

jportfsimple.php

```

65. /**
66. * Function displaying all categories
67. */
68. function JP_categories()
69. {
70.     global $database, $option, $mainframe, $jpConf;
71.     // get categories
72.     $database->setQuery('SELECT * FROM #__jportfsimple_categories ORDER BY ordering' );
73.     $rows = $database -> loadObjectList();
74.     if ($database -> getErrorNum()) {
75.         echo $database -> stderr();
76.         return false;
77.     }
78.     // set page title
79.     $mainframe->setPageTitle( $jpConf->title );
80.
81.     display_categories( $option, $rows );
82. }

```

First function loads all categories from database table (lines 71 to 77), then sets component title as web page title, and calls display function.

jportfsimple.php

```

84. /**
85. * Function displaying one category
86. */
87. function JP_one_cat( $cat )
88. {
89.     global $database, $mainframe, $option, $itemid, $mosConfig_absolute_path, $jpConf;
90.
91.     // get parameters associated with menu
92.     $params = new stdClass();
93.     if ( $itemid ) {

```

```

94.     $menu = new mosMenu( $database );
95.     $menu->load( $itemid );
96.     $params = new mosParameters( $menu->params );
97. } else {
98.     $menu = '';
99.     $params = new mosParameters( '' );
100. }
101. // define parameters, if not set already
102. $params->def('back_button', $mainframe->getCfg( 'back_button' ) );
103. $params->def('item_navigation', $mainframe->getCfg( 'item_navigation' ));
104. $params->def('display_num', 50);
105.
106. // get category
107. $database->setQuery('SELECT * FROM #__jportfsimple_categories WHERE id = '.$cat );
108. $categ = $database -> loadObjectList();
109. if ($database -> getErrorNum()) {
110.     echo $database -> stderr();
111.     return false;
112. }
113.
114. if ($categ)
115. {
116.     if ($categ[0]->published)
117.     {
118.         // get projects in the category
119.         $database->setQuery('SELECT * FROM #__jportfsimple_projects WHERE catid='.$cat.' AND published=1 ORDER BY ordering ' );
120.         $rows = $database -> loadObjectList();
121.         if ($database -> getErrorNum()) {
122.             echo $database -> stderr();
123.             return false;
124.         }
125.         // get limit values (needed for pagination)
126.         $limit = intval( mosGetParam( $_REQUEST, 'limit', '' ) );
127.         $limitstart = intval( mosGetParam( $_REQUEST, 'limitstart', 0 ) );
128.
129.         $total = count($rows);
130.         $limit = $limit ? $limit : $params->get( 'display_num' ) ;
131.         if ( $total <= $limit )
132.         {
133.             $limitstart = 0;
134.             $limit = $total;
135.         }
136.         //load pagination class
137.         require_once( $mosConfig_absolute_path . '/includes/pageNavigation.php' );
138.         $pageNav = new mosPageNav( $total, $limitstart, $limit );
139.
140.         $pp=$categ[0]->cat_name;
141.         // add category name to pathway
142.         $mainframe->appendPathWay($pp);
143.         // set page title
144.         $mainframe->setPageTitle( $jpConf->title.' - '.$categ[0]->cat_name );
145.         // set meta tags
146.         $mainframe->appendMetaTag( 'description', $categ[0]->meta_desc );
147.         $mainframe->appendMetaTag( 'keywords', $categ[0]->meta_keywords );
148.
149.         display_one_cat($categ[0]->id, $categ[0]->cat_name, $categ[0]->cat_info, $categ[0]->cat_path, $rows, $params, $pageNav );
150.     }
151.     else echo _COM_JP_ALB_NOT_PUBLISHED;
152. }
153. else echo _COM_JP_ALB_NOT_PUBLISHED;
154. }

```

Lines 91 to 104 - reads parameters associated with component's menu position. These are the parameters defined at the end of xml file.

106 to 112 - reads category data from db.

114 to 117 - checks if there is a category with given id, and if it's published

118 to 124 - reads projects in given category

126 to 138 - gets pagination parameters (limit, limitstart), loads required class and created \$pageNav object needed later

140 to 142 - adds category name to pathway

line 144 - adds component title and category name to web site title bar

146 to 147 - adds meta tags (keywords and description)

line 149 - all logic done, call display function

jportfsimple.php

```

156. /**
157. * Function displaying one project
158. */
159. function JP_project( $cat, $project )

```

```

160. {
161.     global $database, $mainframe, $option, $itemid, $jpConf;
162.
163.     // get parameters associated with menu
164.     $params = new stdClass();
165.     if ( $itemid ) {
166.         $menu = new mosMenu( $database );
167.         $menu->load( $itemid );
168.         $params = new mosParameters( $menu->params );
169.     } else {
170.         $menu = '';
171.         $params = new mosParameters( '' );
172.     }
173.     // define parameters, if not set already
174.     $params->def('back_button', $mainframe->getCfg( 'back_button' ) );
175.     $params->def('item_navigation', $mainframe->getCfg( 'item_navigation' ) );
176.
177.     // get category
178.     $database->setQuery('SELECT * FROM #__jportfsimple_categories WHERE id = '.$cat );
179.     $categ = $database -> loadObjectList();
180.     if ( $database -> getErrorNum() ) {
181.         echo $database -> stderr();
182.         return false;
183.     }
184.
185.     // get project
186.     $database->setQuery('SELECT * FROM #__jportfsimple_projects WHERE catid='.$cat.' AND id = '.$project.' AND published=1');
187.     $proj = $database -> loadObjectList();
188.     if ( $database -> getErrorNum() ) {
189.         echo $database -> stderr();
190.         return false;
191.     }
192.
193.     if ( $proj )
194.         if ( $proj[0]->published )
195.             {
196.
197.                 // add category name to pathway
198.                 $pp="

```

Frontend view - **jportfsimple.html.php** - contains all output functions.

jportfsimple.html.php

```

9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12. /**
13. * Displaying all categories
14. */
15. function display_categories( $option, &$rows )
16. {
17.     global $jpConf, $itemid, $mosConfig_live_site;
18.     ?>
19.
20.     <div id="jp_front">
21.         <div id="jp_fronttitle">
22.             <?php echo $jpConf->title; ?>
23.         </div>
24.         <?php if ( $jpConf->description ) { ?>
25.             <div id="jp_frontdesc">
26.                 <?php echo $jpConf->description; ?>
27.             </div>
28.         <?php } ?>
29.         <div id="jp_frontcategories">
30.
31.             <?php
32.             // if no rows then display: No categories defined !
33.             if ( !$rows ) echo _COM_JP_NO_CAT;
34.             else
35.             foreach( $rows as $row )
36.             {
37.                 if ( $row->published ) {
38.                     $link='<a href="'.self::relToAbs( 'index.php?option='.$option.'&cat='.$row->id.'&Itemid='.$itemid).'>';
39.                     echo '<div class="jp_frontcategory">';
40.                     echo '<div class="jp_frontcatname">';

```

```

41.     echo $link.$row->cat_name.'</a><br />';
42.     echo '</div>';
43.     echo '<div class="jp_frontcatimage">';
44.     if ($row->cat_image)
45.         echo $link.'</a>';
46.
47.     echo '</div>';
48.     echo '<div class="jp_frontcatinfo">';
49.     echo $row->cat_info;
50.     echo '</div>';
51.     echo '</div>';
52. }
53. }
54. echo '</div>';
55. bottom();
56. echo '</div>';
57. }

```

Lines 20 to 28 - shows component title and description

line 33 - shows warning if there are no categories

35 to 54 - loop through categories and shows their name, description and image, if exists

in line 38 function **sefRelToAbs** is used so links work when SEF is enabled

As you can see, almost each displayed element is surrounded with <div> element, each with specific id or class. This way component display is not based on tables, but uses CSS.

jpportfsimple.html.php

```

59. /**
60. * Displaying all projects in one category
61. */
62. function display_one_cat( $catid, $catname, $catinfo, $catpath, &$rows, &$params, &$pageNav )
63. {
64.     global $jpConf, $option, $itemid, $mosConfig_absolute_path, $mosConfig_live_site;
65.
66.     echo '<div id="jp_front">';
67.     echo '<div id="jp_cattitle">'. _COM_JP_CAT_NAME.$catname.'</div>';
68.     if ($catinfo)
69.         echo '<div id="jp_catinfo">'. $catinfo.'</div>';
70.
71.     echo '<div id="jp_onecat">';
72.     // if no rows then display: No projects defined !
73.     if (!$rows) echo _COM_JP_NO_PROJ;
74.
75.     // go through all projects on given page
76.     for($i=$pageNav->limitstart; $i < ($pageNav->limitstart+$pageNav->limit) && $i < $pageNav->total; $i++)
77.     if ($rows[$i])
78.     {
79.         $src=$jpConf->base_path.$catpath.'/'. $rows[$i]->proj_image;
80.         // if image defined exist, then show it, or show no_image.jpg
81.         if (!file_exists($mosConfig_absolute_path.'/'. $src) | is_dir($mosConfig_absolute_path.'/'. $src))
82.         {
83.             $src=$mosConfig_live_site.'/components/com_jpportfsimple/images/no_image.jpg';
84.         }
85.         $link2=sefRelToAbs('index.php?option='.$option.'&cat='.$catid.'&project='.$rows[$i]->id.'&itemid='.$itemid);
86.         $prname=$rows[$i]->name;
87.         echo '<div class="jp_onecat_proj">';
88.         echo '<div class="jp_onecat_img"><div class="jp_onecat_img2"><a href="'.$link2.'"></a></div></div>';
90.         echo '<div class="jp_onecat_name"><a href="'.$link2.'">'. $prname.'</a></div>';
91.         echo '</div>';
92.     }
93.     echo '</div>';
94.
95.     // if item navigation enabled, then show page number
96.     if ( $params->get('item_navigation') && !($pageNav->limit == $pageNav->total) )
97.     {
98.         echo '<div class="jp_pagination">';
99.         $link='index.php?option='.$option.'&cat='.$catid.'&itemid='.$itemid;
100.        echo $pageNav->writePagesLinks( $link );
101.        echo '</div>';
102.    }
103.
104.    // if back button enabled, show it
105.    if ( $params->get('back_button') )
106.    {
107.        echo '<div class="jp_back">';
108.        mosHTML::BackButton ( $params );
109.        echo '</div>';
110.    }
111.    bottom();
112.    echo '</div>';
113. }

```

line 76 - uses limit and limitstart (from \$pageNav) to loop through all projects on given page

95 to 102 - shows page numbers, if it should

104 to 110 - shows back button, if enabled

jportfsimple.html.php

```
115. /**
116. * Displaying one project
117. */
118. function display_project( $catid, $catname, $catpath, &$amp;proj, &$amp;params )
119. {
120.     global $database, $option,$Itemid, $jpConf, $mosConfig_absolute_path, $mosConfig_live_site;
121.
122.     echo '<div id="jp_front">';
123.     echo '<div id="jp_cattitle">'. _COM_JP_CAT_NAME.$catname.'</div>';
124.     echo '<div id="jp_projtop" >'. _COM_JP_PROJ_NAME.$proj[0]->name.'</div>';
125.
126.     echo '<div id="jp_projcont" >';
127.         echo '<div id="jp_projimage" >';
128.             $src=$jpConf->base_path.$catpath.'/'. $proj[0]->proj_image;
129.             // if image defined exist, then show it, or show no_image.jpg
130.             if (!file_exists($mosConfig_absolute_path.'/'. $src) | is_dir($mosConfig_absolute_path.'/'. $src))
131.             {
132.                 $src=$mosConfig_live_site.'/components/com_jportfsimple/images/no_image.jpg';
133.             }
134.             echo '';
135.         echo '</div>';
136.         echo '<div id="jp_projdesc" >';
137.             echo $proj[0]->description.'<br />';
138.         echo '</div>';
139.     echo '</div>';
140.
141.     // if back button enabled, show it
142.     if ( $params->get( 'back_button' ) )
143.     {
144.         echo '<div class="jp_back">';
145.             mosHTML::BackButton ( $params );
146.         echo '</div>';
147.     }
148.
149.     bottom();
150.     echo '</div>';
151. }
152. }
153.
154. /**
155. * Displaying component footer
156. */
157. function bottom()
158. {
159.     ?>
160.     <div align="right" style="float:right; width:60%; line-height:6px; padding:3px;">
161.     <span class="small">
162.     <br /><br /><br />
163.     <a href="http://www.anetus.com/_r/jportfolio-component-for-joomla-cms" target="_blank" title="JPortfolio - component for Joomla! CMS">jp</a>
164.     </span>
165.     </div>
166.     <?php
167.     }
168.
169.     ?>
170. }
```

Function display_project does exactly what the name suggests - it displays one project.

Last one: bottom() - this is the function showing component footer, something like link to component web page and/or copyright information. It is not necessary to have, of course ;)

6. Remaining frontend files

There are few more files in frontend folder: language file, CSS file, and images.

Language file - lang/english.php:

lang/english.php

```
9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12. // main info
13.
14. DEFINE('_COM_JP_VERSION','0.2');
15. DEFINE('_COM_JP_DESC',' JPortfSimple - example of Joomla! component');
```

It's always good to provide language file, instead of using strings inside component, so the component can be translated easier.

CSS file - css/jp.css:

```
jp.css
1. /* CSS for JPortSimple
2. *
3. * jp.css
4. */
5.
6. /*****
7. /* frontpage */
8.
9. #jportfront {
10. position:relative;
11. float:left;
12. width:99.5%!important;
13. width:99.7%;
14. margin:0px;
15. }
16.
17. #jportfronttitle, #jportcattitle {
18. position:relative;
19. font-weight:bold;
20. font-size: 16px;
21. border-bottom: 2px solid #ccc;
22. float:left;
23. width:94%!important;
24. width:97%;
25. padding:10px;
26. margin:1%;
27. }
28.
29. and so on
30. ....
```

CSS file contains all classes used in divs in the component.

So far I haven't found a tool which would make checking language or css files easier. Now whatever change in the code is done - a text constant added or changed, or CSS class name changed - one has to **remember** to add/remove/change the respective one in the language or CSS file.

Joomla! 1.0 Component Tutorial - Part 2: Back-end



Part 2 of the component tutorial explains all files creating backend interface for my sample JPortSimple component:

1. admin.jportfsimple.php
2. admin.jportfsimple.html.php
3. toolbar.jportfsimple.php
4. toolbar.jportfsimple.html.php
5. install.jportfsimple.php
6. uninstall.jportfsimple.php
7. jportfsimple.xml

which are placed in administrator/components/com_jportfsimple folder during installation.

Well, .xml file was explained earlier (see p. 4).

7. Backend

Similarly to frontend, one file is responsible for "logic" of the component (it's backend interface) - **admin.jportfsimple.php**, and second file is displaying the information (**admin.jportfsimple.html.php**).

```
admin.jportfsimple.php
9. // no direct access
10. defined('_VALID_MOS') or die('Direct Access to this location is not allowed.');
```

```

25. $sact = trim( mosGetParam( $_REQUEST, 'act', '' ));
26.
27. // get selected objects
28. $sid = josGetArrayInts( 'cid' );
29.
30. // load language file
31. if (file_exists( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/.'.$mosConfig_lang.'.php'))
32.     include_once( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/.'.$mosConfig_lang.'.php');
33. else
34.     if (file_exists( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/english.php'))
35.         include_once( $mosConfig_absolute_path.'/components/com_jportfsimple/lang/english.php');
36.
37. // get configuration parameters
38. $database->setQuery('SELECT * FROM #__jportfsimple_conf' );
39. $conf_rows = $database -> loadObjectList();
40. if ($database -> getErrorNum()) {
41.     echo $database -> stderr();
42.     return false;
43. }
44. if ($conf_rows) {
45.     $jpConf = $conf_rows[0];
46.     $base_path = $jpConf->base_path;
47. }

```

First part of frontend file does the following:

lines 12 to 16 - acl check - does logged in user has access to the component ?

19 to 21 - loads the required class and backend html (admin.jportfsimple.html.php) files

23 to 28 - gets variables defining what to do (\$sact, \$task) and with which object (\$id)

30 to 35 - loads language file, default is english.php in com_jportfsimple/lang folder

37 to 47 - reads data from database table containing configuration parameters. They are stored in \$jpConf object, and location of images in \$base_path

Main backend "logic":

admin.jportfsimple.php

```

50. switch( $sact )
51. {
52.     case 'configure':
53.         switch($task) {
54.             case 'save':
55.                 Conf_save( $option );
56.                 break;
57.             default:
58.                 Conf_list( $option );
59.                 break;
60.         }
61.         break;
62.
63.     case 'categories':
64.         switch ( $task ) {
65.             case 'save' :
66.                 Cat_save( $option );
67.                 break;
68.             case 'cancel' :
69.                 Cat_cancel( $option, $sact);
70.                 break;
71.             case 'edit' :
72.                 Cat_edit( $option, $id );
73.                 break;
74.             case 'new' :
75.                 $id = '';
76.                 Cat_edit( $option, $id );
77.                 break;
78.             case 'delete' :
79.                 Cat_del( $option, $id );
80.                 break;
81.             case 'publish' :
82.                 Cat_publish( $option, '1', $id );
83.                 break;
84.             case 'unpublish' :
85.                 Cat_publish( $option, '0', $id );
86.                 break;
87.             case 'orderup':
88.                 ordercat( intval( $id[0] ), -1, $option, $sact );
89.                 break;
90.             case 'orderdown':
91.                 ordercat( intval( $id[0] ), 1, $option, $sact );
92.                 break;
93.
94.             default:
95.                 Cat_list($option);
96.                 break;
97.         }
98.     }

```

```

99.     break;
100.
101.     case 'projects':
102.     switch ( $task ) {
103.         case 'save' :
104.             Proj_save( $option );
105.             break;
106.         case 'cancel' :
107.             Proj_cancel( $option, $act);
108.             break;
109.         case 'edit' :
110.             Proj_edit( $option, $id );
111.             break;
112.         case 'new' :
113.             $id = '';
114.             Proj_edit( $option, $id );
115.             break;
116.         case 'delete' :
117.             Proj_del( $option, $id );
118.             break;
119.         case 'publish' :
120.             Proj_publish( $option, '1', $id );
121.             break;
122.         case 'unpublish' :
123.             Proj_publish( $option, '0', $id );
124.             break;
125.         case 'orderup':
126.             orderproj( intval( $id[0] ), -1, $option, $act );
127.             break;
128.         case 'orderdown':
129.             orderproj( intval( $id[0] ), 1, $option, $act );
130.             break;
131.
132.         default:
133.             Proj_list( $option );
134.             break;
135.     }
136.     break;
137.
138.
139. case 'info':
140.     HTML_jportfsimple::Info( $option );
141.     break;
142.
143. default:
144.
145. break;
146. }

```

Variable **\$act** get it's value from chosen menu. You remember this part of xml file:

```

<menu>jportfsimple</menu>
<submenu>
<menu act="categories">Categories</menu>
<menu act="projects">Projects</menu>
<menu act="configure">Configuration</menu>
<menu act="info">About</menu>
</submenu>

```

So for example, if Categories was clicked, **\$act** would be 'categories', and then variable **\$task** will be checked (line 64). Initially **\$task** is empty, so the default action is performed: function **Cat_list** in line 95. Other functions are called when you click toolbar buttons.

Configuration saving function:

admin.jportfsimple.php

```

148. /**
149.  * Configuration saving
150.  */
151. function Conf_save( $option )
152. {
153.     global $database,$mosConfig_absolute_path, $jfiles;
154.
155.     // no . in path, and must end in slash
156.     if (!(strpos($POST['base_path'],'..')==FALSE)) { $err2=_COM_JP_ERROR; $POST['base_path']='images/stories/'; }
157.     if (substr($POST['base_path'],-1,1)!=='/') $POST['base_path']=$POST['base_path'].'/' ;
158.
159.     $row = new jportfsimpleConf($database);
160.
161.     // bind it to the table
162.     if (!$row -> bind($POST)) {
163.         echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
164.         exit();
165.     }
166.     // store it in the db
167.     if (!$row -> store()) {
168.         echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
169.         exit();
170.     }
171.
172.     mosRedirect( 'index2.php?option='.$option.'&act=configure', 'Configuration Saved'.$err2 );
173. }

```

Function called after editing configuration parameters, when "Save" is clicked.

First some specific for this component checks - if base_path variable contains two dots, then it's changed to default one; if it does not end with slash, one is added.

Lines 159 to 172 - whichever object in Joomla! is being saved (from those defined in .class file) similar actions are performed:

- new object is created (line 159)
- bind - copies values from \$_POST to the above object - if for example you have a field in a form with a name which is not in object's class, then this part will throw an error (line 162)
- storing to database - here also an error may show up, if for example you added field to .class file, but you didn't update database table
- if no errors, then component redirects to the page depending on **act** variable in mosRedirect function, with nice message "Configuration Saved"

Configuration listing function:

```
admin.jportfsimple.php
175. /**
176. * Configuration listing
177. */
178. function Conf_list( $option )
179. {
180.
181.     HTML_jportfsimple::Conf_list( $option );
182.     HTML_jportfsimple::bottom();
183. }
```

It just calls display function, since configuration data is already in \$jpConf variable (see line 45).

Function publishing a category:

```
admin.jportfimple.php
186. /**
187. * Category publishing
188. */
189. function Cat_publish( $option, $publish=1, $cid )
190. {
191.     global $database, $my;
192.     if (!is_array( $cid ) || count( $cid ) < 1) {
193.         $action = $publish ? 'publish' : 'unpublish';
194.         echo "<script> alert('Select an item to $action'); window.history.go(-1);</script>\n";
195.         exit;
196.     }
197.     $cids = implode( ',', $cid );
198.     $database->setQuery( 'UPDATE #__jportfsimple_categories SET published=.'.$publish.' WHERE id IN ( '.$cids.' )');
199.
200.
201.     if (!$database->query()) {
202.         echo "<script> alert('".$database->getErrorMessage()."'); window.history.go(-1); </script>\n";
203.         exit();
204.     }
205.     mosRedirect( 'index2.php?option='.$option.'&act=categories' );
206. }
```

First it checks if at least one category is checked (line 192), if not - it displays popup window with error message (this is one exception of the rule to not use output commands in this file). Then it runs a query to update 'published' column in jos_jportfsimple_categories table (line 198), and if no errors (line 201) it redirects back to categories page.

Category saving function:

```
admin.jportfsimple.php
208. /**
209. * Category saving
210. */
211. function Cat_save( $option )
212. {
213.     global $database;
214.     $row = new jportfsimpleCategories($database);
215.     // bind it to the table
216.     if (!$row -> bind($_POST)) {
217.         echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
218.         exit();
219.     }
220.     // store it in the db
221.     if (!$row -> store()) {
222.         echo "<script> alert('".$row->getError()."'); window.history.go(-1); </script>\n";
223.         exit();
224.     }
225.     $row->updateOrder();

```

```

226.     mosRedirect( 'index2.php?option='.$option.'&act=categories', 'Saved' );
227. }

```

As you see, it's almost the same as configuration saving function earlier, just one additional function updating order of all categories (line 225).

Category deleting function:

admin.jportfsimple.php

```

229. /**
230. * Category deleting
231. */
232. function Cat_del( $option, $cid )
233. {
234.     global $database;
235.     if (!is_array($cid) || count($cid) < 1) {
236.         echo "<script> alert('Select an item to delete'); window.history.go(-1);</script>\n";
237.         exit();
238.     }
239.     if (count($cid))
240.     {
241.         $ids = implode(',', $cid);
242.         $database->setQuery('DELETE FROM #__jportfsimple_categories WHERE id IN ( '.$ids.' )');
243.     }
244.     if (!$database->query()) {
245.         echo "<script> alert('".$database->getErrorMessage()."'); window.history.go(-1); </script>\n";
246.     }
247.     mosRedirect( 'index2.php?option='.$option.'&act=categories' );
248. }

```

Similar to category publishing function, just different query is used (line 242).

Category editing function. As you perhaps noticed (line 72 and 76), the same function is used for creating new category and for editing existing one.

admin.jportfsimple.php

```

250. /**
251. * Category editing
252. */
253. function Cat_edit( $option, $uid )
254. {
255.     global $database;
256.
257.     // build the html select list for ordering
258.     $query = 'SELECT ordering AS value, cat_name AS text FROM #__jportfsimple_categories ORDER BY ordering';
259.     $lists['ordering'] = mosAdminMenus::SpecificOrdering( $row, $uid, $query, 1 );
260.
261.     $row = new jportfsimpleCategories($database);
262.     if($uid){
263.         $row -> load($uid[0]);
264.     }
265.
266.     HTML_jportfsimple::Cat_edit( $option, $row, $lists );
267. }

```

Lines 258, 259 are needed only if you want to be able to change order of categories during editing. You would also need 'ordering' field in database table. Actual category editing part is in lines 261 to 266: first new object is needed (\$row), then if it is in fact editing of an existing category (\$uid is not empty) load function loads it from the database. Finally, edit page is displayed (line 266).

Cancellation of editing:

admin.jportfsimple.php

```

269. /**
270. * Category editing cancel
271. */
272. function Cat_cancel( $option, $act )
273. {
274.
275.     mosRedirect( 'index2.php?option='.$option.'&act='.$act );
276. }

```

Just redirecting after pressing "Cancel".

Category listing:

admin.jportfsimple.php

```

278. /**
279. * Category listing
280. */

```

```

281. function Cat_list( $option )
282. {
283.     global $database, $mainframe, $mosConfig_absolute_path, $mosConfig_list_limit;
284.
285.     $limit      = intval( $mainframe->getUserStateFromRequest( 'viewlistlimit', 'limit', $mosConfig_list_limit ) );
286.     $limitstart = intval( $mainframe->getUserStateFromRequest( "view{$option}limitstart", 'limitstart', 0 ) );
287.     require_once( $mosConfig_absolute_path . '/administrator/includes/pageNavigation.php' );
288.
289.     $query = 'SELECT COUNT(*) FROM #__jportfsimple_categories';
290.     $database->setQuery( $query );
291.     $total = $database->loadResult();
292.
293.     $pageNav = new mosPageNav( $total, $limitstart, $limit );
294.
295.     $database->setQuery('SELECT * FROM #__jportfsimple_categories ORDER BY ordering LIMIT '.$pageNav->limitstart.',
    '.$pageNav->limit );
296.     $rows = $database -> loadObjectList();
297.     if ( $database -> getErrorNum() ) {
298.         echo $database -> stderr();
299.         return false;
300.     }
301.
302.     HTML_jportfsimple::Cat_list( $option, $rows, $pageNav );
303.     HTML_jportfsimple::bottom();
304. }

```

Main function listing all categories.

Lines 285 to 287 - gets current page and number of categories per page, selected by user, and loads required class

289 to 291 - first query, loading just the total number of existing categories (SELECT COUNT), needed for pagination

293 - creates new \$pageNav object

295 to 300 - second query, loading just the categories on current page

302, 303 - displaying functions are called

Project functions next are very similar to the above function dealing with categories. I will list only main differences instead of full rest of the code.

In Proj_publish, Proj_save and Proj_del (project publishing, saving and deleting functions) redirecting function in last line goes to projects page:

admin.jportfsimple.php

```

325. mosRedirect( 'index2.php?option='.$option.'&act=projects' );

```

Project editing function:

admin.jportfimple.php

```

369. /**
370.  * Project editing
371.  */
372. function Proj_edit( $option, $uid )
373. {
374.     global $database, $base_path;
375.     $proj = new jportfsimpleProjects($database);
376.     if($uid)
377.     {
378.         $proj -> load($uid[0]);
379.     }
380.     else $proj->date = date( 'Y-m-d H:i:s' ); // creation date, 2006-09-01 10:00:00
381.
382.     $database->setQuery('SELECT * FROM #__jportfsimple_categories ORDER BY cat_name');
383.     $rows = $database -> loadObjectList();
384.     if ( $database -> getErrorNum() ) {
385.         echo $database -> stderr();
386.         return false;
387.     }
388.
389.     foreach ( $rows as $row ) { $urow2[]=$row->id; $urow[]=$row->cat_name; }
390.
391.     $i=0; $j=0; // $j for Select category text in case of new item
392.     if (!$uid)
393.     {
394.         $categories[$i] = mosHTML::makeOption( '0', _COM_JP_SEL_ALBUM );
395.         $i++; $j=1;
396.     }
397.     if (count($rows)>0)
398.     {
399.         foreach ( $urow as $row )
400.         {
401.             $categories[$i] = mosHTML::makeOption( $urow2[$i-$j], $urow[$i-$j]);
402.             $i++;
403.         }
403.     }

```

```

404.
405.     $javascript = '';
406.
407.     $category = mosHTML::selectList( $categories, 'catid', 'class="inputbox" size="'. $size ." ". $javascript, 'value', 'text',
    $proj->catid );
408.     $lists['category'] = $category;
409.
410.     // build the html select list for ordering
411.     $query = 'SELECT ordering AS value, name AS text '
412.     . ' FROM #__jportfsimple_projects'
413.     . ' WHERE catid = '.$proj->catid
414.     . ' ORDER BY ordering';
415.     $lists['ordering'] = mosAdminMenus::SpecificOrdering( $proj, $uid, $query, 1 );
416.
417.     foreach ( $rows as $c ) { $c2[$c->id]=$c; }
418.
419.     HTML_jportfsimple::Proj_edit( $option, $proj, $base_path, $c2, $lists);
420. }

```

Lines 376 to 380 - if editing existing project, load it's data, or set a creation date for new project

382 to 408 - this part is used to create a list of categories in HTML select form, for easier selecting a category the project should belong to, instead of entering a category name manually. The challenge here is to select a name, but return a category id. First it gets list of categories, then it creates a **\$categories** array with HTML code for select function, with category id and a name (lines 389 - 402), finally it finishes HTML preparation. It uses functions from Joomla's mosHTML class.

410 to 415 - builds another HTML select list, this one for ordering of a project

419 - calles display function

Project listing function is also similar to category listing, added is the above code for creating HTML select function for filtering projects by a category.

Last two functions: ordercat and orderproj - are used whenever reorder arrow is clicked.

admin.jportfsimple.php

```

494. /**
495.  * Category ordering
496.  */
497.  function ordercat( $uid, $inc, $option, $act )
498.  {
499.     global $database;
500.
501.     $row = new jportfsimpleCategories( $database );
502.     $row->load( $uid );
503.     $row->updateOrder();
504.     $row->move( $inc, 'published >= 0' );
505.     $row->updateOrder();
506.
507.     mosRedirect( 'index2.php?option='.$option.'&act='.$act);
508. }

```

This is nothing else like using internal Joomla function (updateOrder in this case) to reorder objects.

Now the display file - **admin.jportfsimple.html.php** - which displays the information provided by the "logic" file, earlier, or shows forms and input fields to enter new information.

All backed functions are members of HTML_jportfsimple class:

admin.jportfsimple.html.php

```

9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12.
13. class HTML_jportfsimple{

```

Configuration listing function is an example of using tabs, and also editor:

admin.jportfsimple.html.php

```

30. /**
31.  * Configuration listing
32.  */
33.  function Conf_list( $option )
34.  {
35.     global $jpConf;
36.     ?>
37.
38.     <script language="javascript" type="text/javascript">

```

```

39.         function submitbutton(pressbutton) {
40.             <?php getEditorContents( 'editor', 'description' ); ?>
41.             submitform( pressbutton );
42.         }
43.     </script>
44.
45.     <script language="Javascript" src="js/dhtml.js"></script>
46.
47.     <table class="adminheading">
48.         <tr>
49.             <th><?php echo _COM_JP_CONFIG_INFO ?></th>
50.         </tr>
51.     </table>
52.
53.     <table cellpadding="4" cellspacing="0" border="0" width="100%">
54.     <tr>
55.     <td width="5%" class="tabpadding"> </td>
56.     <td id="tab1" class="offtab" onClick="dhtml.cycleTab(this.id)">Settings</td>
57.     <td id="tab2" class="offtab" onClick="dhtml.cycleTab(this.id)">CSS</td>
58.     <td width="5%" class="tabpadding"> </td>
59.     </tr>
60. </table>
61.
62. <form action="index2.php" method="post" name="adminForm">
63. <div id="page1" class="pagetext">
64.
65. <table cellpadding="4" cellspacing="0" border="0" width="90%" class="adminlist">
66. <tr>
67.     <th class="title" width="10%"><?php echo _COM_JP_CONFIG_PARAMETER ?></th>
68.     <th class="title" width="20%"><?php echo _COM_JP_CONFIG_VALUE ?></th>
69.     <th class="title"><?php echo _COM_JP_CONFIG_DESCR ?></th>
70. </tr>
71. <tr>
72. <tr>
73.     <td><?php echo _COM_JP_CONFIG_BASEPATH ?> </td>
74.     <td><input size="30" name="base_path" value="<?php echo $jpConf->base_path; ?>"> </td>
75.     <td><?php echo _COM_JP_CONFIG_BASEPATH_DESCR ?> </td>
76. </tr>
77. <tr>
78. <tr>
79.     <td><?php echo _COM_JP_CONFIG_TITLE ?> </td>
80.     <td><input size="30" name="title" value="<?php echo $jpConf->title; ?>"> </td>
81.     <td><?php echo _COM_JP_CONFIG_TITLE_DESCR ?> </td>
82. </tr>
83. <tr>
84. <tr>
85.     <td valign="top" ><?php echo _COM_JP_CONFIG_FRONT_DESCR ?> </td>
86.     <td align="center" colspan="2">
87.     <?php
88.     editorArea( 'editor', $jpConf->description , 'description', '600', '220', '60', '40' ) ;
89.     ?>
90.     </td>
91. </tr>
92. </table>
93. </div>
94.
95. <div id="page2" class="pagetext">
96.
97. <table cellpadding="4" cellspacing="0" border="0" width="90%" class="adminlist">
98.
99. <tr>
100.     <th class="title" width="10%"><?php echo _COM_JP_CONFIG_PARAMETER ?></th>
101.     <th class="title" width="20%"><?php echo _COM_JP_CONFIG_VALUE ?></th>
102.     <th class="title"><?php echo _COM_JP_CONFIG_DESCR ?></th>
103. </tr>
104. <tr>
105. <tr>
106.     <td valign="top" ><?php echo _COM_JP_CONFIG_CSS ?> </td>
107.     <td><input size="30" name="css_file" value="<?php echo $jpConf->css_file; ?>"> </td>
108.     <td valign="top" ><?php echo _COM_JP_CONFIG_CSS_DESCR ?> </td>
109. </tr>
110. </table>
111. </div>
112.
113. <input type="hidden" name="option" value="<?php echo $option; ?>" />
114. <input type="hidden" name="task" value="" />
115. <?php if ($jpConf->id) { ?>
116. <input type="hidden" name="id" value="<?php echo $jpConf->id ?>" />
117. <?php } ?>
118. <input type="hidden" name="act" value="configure" />
119. </form>
120.
121. <script language="javascript" type="text/javascript">dhtml.cycleTab('tab1');</script>
122.
123. <?php
124. }

```

Lines 38 to 43 - are required for using WYSIWYG editor in a form (actually to get back the content of an editor). Editor is loaded in line 88. Notice the name of the editor field: 'editor' - if you'd like to use more than one editor, their names should be unique.

45 - this is needed for tabs. Their displayed names are defined in lines 56, 57, first open tab is defined in line 121, and then each tab content is displayed within a div. So here:

tab1 - name: Settings, div id=page1 (line 63, 93)

tab2 - name: CSS, div id=page2 (line 95, 111)

As you see, each input field must have it's name, which must be the same as defined in class file, depending on object which is edited.

The "hidden" fields (lines 113 to 118) are used to pass other variables, like \$option or \$act, so the backend "knows" where to go after you press "Save". Variable \$task is empty initially - it's value is assigned when actual button in pressed. Variable \$id is needed so the right record is saved. In case of configuration there is always only 1 record.

Categories listing function:

```
admin.jporthsimple.html.php
126. /**
127. * Category listing
128. */
129. function Cat_list( $option, &$rows, &$pageNav )
130. {
131. ?>
132.
133. <form action="index2.php" method="post" name="adminForm">
134.
135. <table class="adminheading">
136.     <tr>
137.         <th><?php echo _COM_JP_ALBUMS_INFO ?></th>
138.     </tr>
139. </table>
140.
141. <table cellpadding="4" cellspacing="0" border="0" width="100%" class="adminlist">
142. <tr>
143.     <th width="20%"><input type="checkbox" name="toggle" value="" onclick="checkAll(<?php echo count($rows); ?>);" /></th>
144.     <th class="title" width="20%"><?php echo _COM_JP_ALBUM_NAME ?></th>
145.     <th class="title"><?php echo _COM_JP_ALBUM_PATH ?></th>
146.     <th class="title"><?php echo _COM_JP_ALBUM_INFO ?></th>
147.     <th width="5%" colspan="2" align="center"><?php echo _COM_JP_ORDER; ?></th>
148.     <th width="7%"></th>
149.     <th width="7%"><?php echo _COM_JP_CONFIG_PUBLISHED; ?> </th>
150. </tr>
151.
152. <?php
153. if (count($rows)<1) echo ' <tr><td colspan="8" ><center><b>'._COM_JP_NO_CAT.'</b></center></td></tr>';
154.
155.     $k = 0;
156.     for ($i=0, $n=count( $rows ); $i < $n; $i++)
157.     {
158.         $row = &$rows[$i];
159.
160.     ?>
161.     <tr class="<?php echo "row$k"; ?>">
162.         <td><input type="checkbox" id="cb<?php echo $i;?>" name="cid[]" value="<?php echo $row->id; ?>"
163.         onclick="isChecked(this.checked);" /></td>
164.         <td><a href="#edit" onclick="hideMainMenu();return listItemTask('cb<?php echo $i;?>', 'edit')"><?php echo $row->cat_name;
165.         ?></a></td>
166.         <td><?php echo $row->cat_path; ?></td>
167.         <td><?php echo substr($row->cat_info,0,30); ?></td>
168.         <td>
169.             <?php echo $pageNav->orderUpIcon( $i, 1 ); ?>
170.         </td>
171.         <td>
172.             <?php echo $pageNav->orderDownIcon( $i, $n, 1 ); ?>
173.         </td>
174.         <td align="center">
175.         <?php
176.             if ($row->published == "1") {
177.                 echo " <a href=\"javascript: void(0);\" onClick=\"return listItemTask('cb$i','unpublish')\"><img
178.                 src=\"images/publish_g.png\" border=\"0\" /></a>";
179.             } else {
180.                 echo " <a href=\"javascript: void(0);\" onClick=\"return listItemTask('cb$i','publish')\"><img src=\"images/publish_x.png\"
181.                 border=\"0\" /></a>";
182.             }
183.         ?>
184.         </td>
185.     </tr>
186.     <?php $k = 1 - $k; ?>
187. </tr>
188. </table>
189.
190. <?php echo $pageNav->getListFooter(); ?>
191.
192. <input type="hidden" name="option" value="<?php echo $option; ?>" />
193. <input type="hidden" name="task" value="" />
194. <input type="hidden" name="boxchecked" value="0" />
195. <input type="hidden" name="hidemainmenu" value="0" />
196. <input type="hidden" name="act" value="categories" />
197. </form>
198.
199. <?php
200. }
201. }
```

Here or in any similar listing function, few blocks are used:

lines 141 to 150 - start of table, and column headers

153 - if no category defined yet, display warning

156 - loop through all categories

162 - checkbox for selecting categories

163 - displays category name with Javascript returning 'edit' value for \$task variable

167, 170 - displays reordering icons

176 to 180 - displays 'published' or 'unpublished' icon

191 - displays select list - how many categories per page to display

Next function - category editing - is similar to configuration editing. Additionally it has Javascript doing basic field validation. It displays popup warning if category name or folder are not entered.

```
admin.jportfsimple.html.php
220. <script language="javascript" type="text/javascript">
221.     function submitbutton(pressbutton) {
222.         var form = document.adminForm;
223.         if (pressbutton == 'cancel') {
224.             submitform( pressbutton );
225.             return;
226.         }
227.
228.         // do fields validation
229.         if (form.cat_name.value == ''){
230.             alert( "Category must have a name" );
231.         } else if (form.cat_path.value == ''){
232.             alert( "You must select a category folder" );
233.         } else {
234.             <?php getEditorContents( 'editor', 'description' ); ?>
235.             submitform( pressbutton );
236.         }
237.     }
238. </script>
```

Project listing and project editing functions are similar to the above ones.

Last function is 'Info' which displays information about the component. It shows logo, short description and version as defined in language file constants (_COM_JP_DESC, _COM_JP_VER1) and then it includes HTML file from help folder.

```
admin.jportfsimple.html.php
575. if (file_exists( $mosConfig_absolute_path.'/administrator/components/com_jportfsimple/help/'. $mosConfig_lang.'.html'))
576.     include_once( $mosConfig_absolute_path.'/administrator/components/com_jportfsimple/help/'. $mosConfig_lang.'.html');
577. else
578.     if (file_exists( $mosConfig_absolute_path.'/administrator/components/com_jportfsimple/help/english.html'))
579.         include_once( $mosConfig_absolute_path.'/administrator/components/com_jportfsimple/help/english.html');
```

If information in other language is not available, it will show english.html by default. It could be done differently - whole information could be also entered in one constant in language file, but I found it easier to update this way.

8. Backend - toolbars

Toolbar is defined with help of two files:

1. toolbar.jportfsimple.php
2. toolbar.jportfsimple.html.php

```
toolbar.jportfsimple.php
9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12. require_once( $mainframe->getPath( 'toolbar_html' ) );
13. $task = mosGetParam( $_REQUEST, 'task', '' );
14. $act = mosGetParam( $_REQUEST, 'act', '' );
15.
16. switch ( $task ) {
17.
18.     case 'edit':
```

```

19.     case 'new':
20.         menujportfsimple::EDIT_MENU();
21.         break;
22.
23.     default:
24.         switch($act)
25.         {
26.             case "configure":
27.                 case "css":
28.                     menujportfsimple::CONFIGURE_MENU();
29.                     break;
30.
31.             case "info":
32.                 menujportfsimple::INFO_MENU();
33.                 break;
34.
35.             case "categories":
36.             case "projects":
37.                 menujportfsimple::DEFAULT_MENU();
38.                 break;
39.
40.             default:
41.
42.                 break;
43.         }
44.         break;
45.     }

```

Depending on \$act and \$task it calls certain menu function from *menujportfsimple* class, which displays the right toolbar.

toolbar.jportfsimple.html.php

```

9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12.
13.
14. class menujportfsimple{
15.
16.     function DEFAULT_MENU() {
17.
18.         mosMenuBar::startTable();
19.         mosMenuBar::publish('publish');
20.         mosMenuBar::unpublish('unpublish');
21.         mosMenuBar::divider();
22.         mosMenuBar::addNew('new');
23.         mosMenuBar::editList('edit', 'Edit');
24.         mosMenuBar::deleteList( ' ', 'delete', 'Remove' );
25.         mosMenuBar::divider();
26.         mosMenuBar::endTable();
27.     }
28.
29.     function EDIT_MENU() {
30.
31.         mosMenuBar::startTable();
32.         mosMenuBar::cancel();
33.         mosMenuBar::save('save');
34.         mosMenuBar::endTable();
35.     }
36.
37.     function CONFIGURE_MENU() {
38.
39.         mosMenuBar::startTable();
40.         mosMenuBar::save('save');
41.         mosMenuBar::endTable();
42.     }
43.
44.     function INFO_MENU() {
45.
46.         mosMenuBar::startTable();
47.         mosMenuBar::endTable();
48.     }
49. }

```

Each `_MENU` function uses `startTable` and `endTable` from `mosMenuBar` class at the beginning and at the end of a toolbar. Then it defines actual buttons using functions like: `addNew`, `editList` or `save`.

9. Backend - install, uninstall

Last two files contain functions which are run after installing or uninstalling a component.

1. install.jportfsimple.php
2. uninstall.jportfsimple.php

Both are almost identical, only the function name inside is different, and the constant used to display information.

install.jportfsimple.php

```
9. // no direct access
10. defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' );
11.
12. function com_install() {
13.     global $mosConfig_live_site, $mosConfig_absolute_path, $mosConfig_lang;
14.     if (file_exists( $mosConfig_absolute_path."/components/com_jportfsimple/lang/" . $mosConfig_lang.".php"))
15.         include_once( $mosConfig_absolute_path."/components/com_jportfsimple/lang/" . $mosConfig_lang.".php");
16.     else
17.         if (file_exists( $mosConfig_absolute_path."/components/com_jportfsimple/lang/english.php"))
18.             include_once( $mosConfig_absolute_path."/components/com_jportfsimple/lang/english.php");
19.
20.     echo _COM_JP_INFO_TOP;
21.     echo _COM_JP_INSTALL1;
22.
23.     echo _COM_JP_INFO_BOTTOM;
24. }
```

10. Summary

In the above tutorial I tried to explain the structure and functions of sample component written for Joomla! CMS v. 1.0.12.