

Using old Addons with PHProjekt 5

Old addons do no longer work when You upgrade your PHProjekt to version 5. There are two main reasons for this:

- The config now uses defines instead of variables
- The layout is different now and depends heavily on CSS

In order to use Your addons from PHProjekt 4 with the new version 5, a compatibility mode was created. You can activate it by setting

```
define('PHPR_COMPATIBILITY_MODE', '1');
```

in your `config.inc.php`. For information on how addons work with version 5, see below. Just this, for now: You do not need `<html>`, `<head>` and `<body>`. If Your addon depends on this, consider using an `<iframe>`.

If Your addon depends on a particular skin, Albrechts suggestion is to copy the skin files into the addon directly and include them from there.

Creating Addons for PHProjekt 5

This tutorial shows you how to write addons that benefit from the new PHProjekt 5 features. For all explanations the default skin is used.

1. Quickstart

The simple way to make an addon a script is just adding a directory to the addon subdirectory in the PHProjekt tree. The name of Your addon should not start with an "_" and should not be "CVS", since those names are ignored.

In the Directory, you will need a file named "index.php" that is automatically called from PHProjekt later. After that, log out of PHProjekt and log in again and You should see Your addon in the navigation already among the other PHProjekt modules.

For the code, You need not care about <html>, <head> or <body> at all, PHProjekt will do that for You. Everything you print to the screen will end up in the content <div>-area.

Now you got a simple addon that shows up in the navigation and lets you get back to other PHProjekt modules easily.

2. Fancy Titlebar

The plain addon looks a bit "naked" without the usual titlebar from the PHProjekt modules. So we will add a plain one. Insert the following code in Your main script:

```
require_once($path_pre.'lib/dbman_lib.inc.php');
$module='Myaddon';
$tabs=array();
echo get_tabs_area($tabs);
```

where "Myaddon" is the name of Your addon of course¹. `$path_pre` is already set by the addon script itself. We need the included file for the tab area, we currently have no tabs. This generates the titlebar, the name of the addon in the top left corner and a help button in the top right corner. (The help button will redirect you to the PHProjekt wiki)

So far so good. If you just plan to have an addon with one page (or a list view and a detail view), You're done here. If you plan something more complicated, you can now add some tabs to your titlebar. You have two options - add tabs to the left hand side, right beneath the addon name or add buttons to the right hand side near the help button.

Change the above line containing "`$tabs=array();`" to this:

```
$script="addon.php?addon=Myaddon";
$tabs['view']=array(
    'href'=>$script.'mode=view',
    'text'=>'View',
    'position'=>'left',
    'id'=>'view1',
    'target'=>'_self',
    'title'=>'view');
$tabs['keywords']=array(
    'href'=>$script.'mode=keywords',
    'text'=>'Keywords',
    'position'=>'left',
    'id'=>'keyw1',
    'target'=>'_self',
    'title'=>'keywords');
$tabs['export']=array(
    'href'=>$script.'mode=export',
    'text'=>'Export',
    'position'=>'right',
    'id'=>'expl',
    'target'=>'_self',
    'title'=>'export');
```

¹ It's difficult to add a name consisting of more than one word: On one Hand, the space for the name is limited by CSS and you can not have spaces, on the other hand you can not add fancy encodings for the whitespace since PHProjekt also uses the name as an index

This leaves you with two tabs on the left side reading "*View*" and "*Keywords*" and a button on the right side labelled "*Export*". All of them link to the current window, all of them call the main addon script passing the parameter "mode".

Now you have a fancy titlebar that fits in with the PHProjekt design (and more important, it heeds your chosen skin). You can now add different interface scripts into one addon menu entry.

3. Default list view

If you have an addon that is following the usual PHProjekt structure, you have a list view and a form view, plus a search bar and a button for new entries. In this section we will add a default list view to the addon.

Prior to coding in the addon we need at least two things:

- A database table that contains our records². To utilize PHProjekt mechanisms, this table needs at least the usual fields:

```
CREATE TABLE `myaddon` (  
  `ID` int(8) NOT NULL auto_increment,  
  `title` varchar(2500) default '',  
  `von` int(8) default NULL,  
  `gruppe` int(6) default NULL,  
  `acc_read` text,  
  `acc_write` text,  
  `parent` int(8) default NULL,  
  PRIMARY KEY (`ID`)  
);
```

Explanation:

- ID is the primary identifier.
 - von is the owner of the record
 - gruppe is the group the record belongs to
 - acc_read and acc_write are access fields denoting who may access the record besides the owner and whether those persons may write to it
 - parent may contain the parent record when a tree view is to be used.
 - title is optional, but I will use it as an example here.
(PHProjekt uses a mixture of German and English variable names, this is a legacy)
- An entry in the db_manager table (Footnote: Again, heed the prefix). You can add more fields later on, but the easiest way to access the addon in the module designer is to add at least one field into the table so that the addon will show up in the list.

```
INSERT INTO `db_manager` VALUES (  
  db_table      = 'myaddon',  
  db_name       = 'title',  
  form_name     = 'Title',  
  form_type     = 'text',  
  form_tooltip  = 'Name of the Object',  
  form_pos      = 1,  
  form_colspan  = 1,  
  form_rowspan  = 1,  
  list_pos      = 1,  
  filter_show   = '1');
```

This leaves us with an entry looking up records from the table "myaddon", column

² If you use a database prefix, you need to prefix the table name when you enter a query

"title" named "Title" in the form being of type "text"³. The item is the first column in the list view, spans one cell in the form view, is the first item in the form view (which we don't have yet) and will show up in the filter. If you wish to have more than one field in your list view (or form, later), you can add fields with the module designer as admin user. Keep in mind, though, that the module designer adds fields to your table as well, so you should not add them manually in the database as that will result in an error.

Now that the prerequisites are met, we can add the code.

```
if (SID) $hidden[session_name()] = session_id();
$form_fields=array();
$out="";

$tablename['Myaddon'] = 'myaddon';

$buttons[]=array(
    'type'=>'form_start',
    'hidden'=>$hidden,
    'name'=>'frm',
    'onsubmit'=>'');
$buttons[]=array(
    'type'=>'link',
    'href'=>$script.'mode=forms',
    'text'=>__( 'New' ),'active'=>false);

$out .=get_buttons_area($buttons).
    "<div class='hline'></div>";

$getstring="addon=bookmarks";
$fields=build_array('Myaddon', $ID, 'view');
$tab=build_table(
    array(),
    'Myaddon',
    $where,
    $page,
    $perpage);
$out .= get_all_filterBars('Myaddon', $tab);
echo $out;
```

First we add a hidden `sessionid` field to the form that will follow so that the addon will work when the user deactivates cookies. After that we tell PHProjekt that the name of the table associated to our addon is "myaddon" (We already told PHProjekt that "Myaddon" is the name of your addon). Then we add two entries into the `$buttons` array - a form starting tag, since we will have a form for the filter, and a "New" button for generating a new object. Again `$script` is used to call the addon script. After that we can get the buttons to display. We add a thin line below to separate the buttons from the filter part. `$getstring` needs to be set, so that the list entries can be linked correctly

3 The module-designer type "text", not the MySQL type "text"

later. `$fields` is a global variable that controls the table name and the index column. `build_table()` generates the table that contains the currently visible records. Its first argument can contain additional fields that should be fetched from the table, `$where`, `$page` and `$perpage` control the ownership, accessibility and amount of records fetched. After we got the records, we generate the appropriate filter bar from it with `get_all_filter_bars()`. Note that you can control which fields can be filtered by adjusting the appropriate field in the module designer. Finally we can echo everything.

Without records, you should now see the navigation, the fancy titlebar, the bar with the "New" button, the bars with the filters and the list titles containing the field names. After you added a form for generation and alteration of the records, you have a fully functional add-on