Development of a PostNuke Module

Code: portal-prog-pn-en

Original files

url: http://tecfa.unige.ch/guides/tie/html/portal-prog-pn-en/portal-prog-pn-en.html
url: http://tecfa.unige.ch/guides/tie/pdf/files/portal-prog-pn-en.pdf

Authors and version

- Vivian Synteta - Daniel K. Schneider

- Version: 0.1 (modified: 3/9/02)

Prerequisites

MySQL and PHP (some info in french below)

Module technique précédent: mysql-intro
Module technique précédent: php-mysql

Abstract

This module is about the basics on developing a module for the weblog/CMS called "Postnuke"

Objectives

- Describe the architecture of PostNuke
- Explain what is a module, show it's architecture
- Show the steps that one needs to develop a "postnuke" module
- Give some "tips"

Aknowledgements

- This module is strongly inspired from the official documentation provided through the Postnuke portal.

Disclaimer

- Original version is in French, apologies for any mistakes as this is a quick translation to english to distribute to bigger audiences the information.

1. Table of Contents

1. Table of Contents 3
2. Introduction 4
   - **2.1 "Everything is in the documentation or almost everything :)" 5**
   - **2.2 Steps to follow before the real action 6**
3. Development 7
   - **3.1 Hints 7**
   - **3.2 Important files 7**
   - **3.3 Structure of a PN module postnuke 8**
4. Module installation and initialisation 10
   - **4.1 Describe all SQL tables in "pntables.php" file 10**
   - **4.2 Create or delete SQL tables inside "pninit.php" file 11**
   - **4.3 Test (and verify the SQL tables inside the DB !) 12**
5. Admin and user interface 13
   - **5.1 Separate GUI from API functions 13**
   - **5.2 Everything in functions or classes 13**
   - **5.3 Use official API functions (doc!) 14**
   - **5.4 For SQL use the ADODB library (doc!) 14**
   - **5.5 Manage exceptions (doc!) 14**
   - **5.6 Security and permissions 15**
   - **5.7 For HTML use "pnHTML" object and it's methods (doc!) 16**
6. Multilingual modules 17

2. Introduction

- Postnuke = community portal (weblog)/CMS with modular architecture.
- A module is:
  - an application that augments the functionalities of the portal and integrates easily in the core structure (ex: forum, Top10).
  - a directory that contains files with functions :)
- A module can be:
  - independent (ex: forum): performs tasks that are independent from other tasks/ modules of the portal and which uses the portal as the display engine (and which shares the users and permissions system)
  - dependent (ex: Top10): from other modules and which uses the content (database info) from other modules of the portal and manipulates it
- A module can be displayed:
  - in a "block" and/or
  - in the central display area of the portal
- A module has (usually, but not always) 2 interfaces:
  - an administration interface to administer/configure the module and which integrates in the global administration tool of the system
  - a user interface for the user to "run" the module

2.1 "Everything is in the documentation or almost everything :)"

From the very beginning you need the following:

- install a "test" portal (otherwise you risk to destroy your official portal)
- search and print all the official documents (most recent versions):

some hints for searching:

url: http://www.postnuke.com/index.php
url: http://mods.postnuke.com/
url: http://support.postnuke.com/

Most important documents:

url: Postnuke Module Developers Guide
url: Postnuke API Command Reference
url: Postnuke ADODB Documentation

2.2 Steps to follow before the real action

- Print all documentation and read it carefully!!!
- Get inspired from other modules (only the ones that are properly developped :)
- Decide for the functionalities of your module and verify that it doesn't exist already! (that, will save a lot of trouble ...)
- Choose a name (short, all in small letters, and of course one that makes sense) and register it in the official PN portal in order to avoid conflicts with the name of other modules
- Decide for the type of your module: "item" or "utility"

item = independent modules that manipulate their own content (ex: news, FAQ, downloads)
utility= modules that manipulate the content of other modules (ex: comments, ratings) and who use usually "hooks" (interactions among modules)

- Design your module

> Separate "user" from "admin" functions
> Separate "GUI" from "API" functions (display from operations)
> Decide whether you want to have "blocks"

- Think carefully about the structure of the database tables that your module will need if this is the case (take your time as this is VERY important!)
- Start programming (according the official doc and our tips that follow :)
- Ask for help (*Module Support Forums*)
- Document your code (!), test and debug, package your module (zip, tar)

3. Development

Some examples to get inspired from:

url: Template **(example module included in the official distribution)**
url: joinproject **(simple with only basic functionalities)**
url: vquiz **(more complicated)**

3.1 Hints

- Choose names (for functions, variables, SQL tables) that start with the name of the module (ex: vquiz_questions).
- Organize your code into functions (no "main" part!). Call to functions is done through a long URL like:

```
 index.php?module=joinproject&type=admin&func=main
Parameters:
module(=module_name),type(='user'_or_'admin'),func(=function_name)
```

3.2 Important files

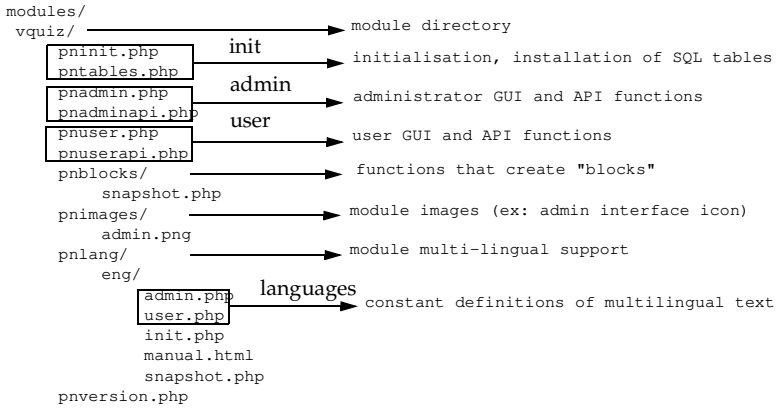
- ***Installation and initialisation:*** pninit.php, pntables.php
- ***Admin Interface:*** pnadmin.php, pnadminapi.php
- ***User Interface:*** pnuser.php, pnuserapi.php
- ***Language management:*** (dir) pnlang/(eng, fra, deu, ...)/admin.php,user.php

3.3 Structure of a PN module postnuke

- All modules are under the "modules" directory
- All files of the same module have to be under the same directory that carries the same name with that of the module
- The internal structure of this directory has to resemble to the structure below:

```
modules/
 vquiz/
     pninit.php
     pntables.php
     pnadmin.php
     pnadminapi.php
     pnuser.php
     pnuserapi.php
     pnblocks/
         snapshot.php
     pnimages/
         admin.png
     pnlang/
         eng/
             admin.php
             user.php
             init.php
             manual.html
             snapshot.php
     pnversion.php
```

Details:



4. Module installation and initialisation

PN provides an interface to initialise, activate/deactivate or remove a module (it's SQL tables) and the files that contribute to this are: "pntables.php" et "pninit.php".

4.1 Describe all SQL tables in "pntables.php" file

In this file, there is a function that fills in a PN core array called "pntable" with the name of all the tables used in a module and their fields

(example below: table "pn_joinproject_members" with 4 fields)

```
function joinproject_pntables() {
    // Initialise table array
    $pntable = array();
    // Get the name for the template item table.
    $members = pnConfigGetVar('prefix') . '_joinproject_members';
    // Set the table name
    $pntable['joinproject_members'] = $members;
    // Set the column names.
    $pntable['joinproject_members_column'] =
array('memberid'   => $members . '.memberid',
      'projid'     => $members . '.projid',
      'membername' => $members . '.membername',
      'accepted'   => $members . '.accepted');
    // Return the table information
    return $pntable; }
```


4.2 Create or delete SQL tables inside "pninit.php" file

There are 2 functions : init() and delete()

Inside init() we create the tables needed by the module inside the PN database, by calling simple SQL queries

```
function joinproject_init() {
    list($dbconn) = pnDBGetConn();
    $pntable = pnDBGetTables();
    $memberstable = $pntable['joinproject_members'];
    $memberscolumn = &$pntable['joinproject_members_column'];
    $sql1 = "CREATE TABLE $memberstable (
            $memberscolumn[memberid] tinyint unsigned NOT NULL auto_increment,
            $memberscolumn[projid] tinyint unsigned NOT NULL default '',
            $memberscolumn[membername] varchar(255) NOT NULL default '',
            $memberscolumn[accepted] tinyint(1) NOT NULL default '0',
            PRIMARY KEY(memberid))";
   $dbconn->Execute($sql1);
    // Check for an error with the database code, and if so set an
    // appropriate error message and return
    if ($dbconn->ErrorNo() != 0) {
        pnSessionSetVar('errormsg', _CREATETABLEFAILED);
        return false;
    }
}
```


Inside delete() we delete the SQL tables, that are connected to the module, from the PN database, applying the SQL query: "DROP TABLE"

```
function joinproject_delete()
{
    list($dbconn) = pnDBGetConn();
    $pntable = pnDBGetTables();
    $sql1 = "DROP TABLE $pntable[joinproject_members]";
    $dbconn->Execute($sql1);
    // Check for an error with the database code, and if so set an
    // appropriate error message and return
    if ($dbconn->ErrorNo() != 0) {
        pnSessionSetVar('errormsg', _DROPTABLEFAILED);
        return false;
    }
    // Deletion successful
    return true;
}
```

4.3 Test (and verify the SQL tables inside the DB !)

Administration -> Modules -> List -> "MyModule" ->

Initialise,

Activate/Deactivate,

Remove

5. Admin and user interface

Admin Interface:
- access to the administration of the module through portal's "administration"

User Interface:
- either through a "menu block" (ex: "Main Menu") where we add a line that calls our module: Title=My New Module, URL={mymodule}
- either by creating a new "block" that has the name of the module
- and so many other possibilities

5.1 Separate GUI from API functions

GUI: pnadmin.php, pnuser.php

API: pnadminapi.php, pnuserapi.php

5.2 Everything in functions or classes

Function names inside *pnadmin.php* et *pnadminapi.php*:

- mymodule_admin_main(), mymodule_admin_function()

Function names inside *pnuser.php* et *pnuserapi.php*:

- mymodule_user_main(), mymodule_user_function()

5.3 Use official API functions (doc!)

to call a function: pnModURL()

```
pnModURL('joinproject', 'user', 'main')
```

to redirect a page: pnRedirect()

```
pnRedirect(pnModURL('joinproject', 'admin', 'main'));
```

and many others ...

5.4 For SQL use the ADODB library (doc!)

PN uses ADODB library for SQL and below one can find the most basic pieces of code needed:

```
list($dbconn) = pnDBGetConn();
$pntable = pnDBGetTables();
$projectstable = $pntable['joinproject_projects'];
$result1=$dbconn->Execute("SELECT * FROM $projectstable");
```

5.5 Manage exceptions (doc!)

PN has an internal system to manage exceptions ("system" and "user") and it is at your interest to use it.

5.6 Security and permissions

PN has a very elaborated system to manage permissions (see online manual). For this reason, there is an API function and it is better to be called in the very beginning of every function in order to avoid security holes:

```
if (!pnSecAuthAction(0, 'joinproject::', '::', ACCESS_ADMIN)) {
  $output->Text(_TEMPLATENOAUTH);
  return $output->GetOutput(); }
```

There are several levels of access (according to official doc):

```
ACCESS_NONE No access
ACCESS_OVERVIEW Allowed to get an overview of the content
ACCESS_READ Allowed to read the content
ACCESS_COMMENT Allowed to comment on the content
ACCESS_MODERATE Allowed to moderate the content
ACCESS_EDIT Allowed to edit the content
ACCESS_ADD Allowed to add content
ACCESS_DELETE Allowed to delete content
ACCESS_ADMIN Full access
```


5.7 For HTML use "pnHTML" object and it's methods (doc!)

NEVER use "echo" or "print", instead use "pnHTML" methods that will take care of the HTML tags (and thus guarantee clean, correct and compliant HTML).

There are many methods for all kinds of HTML elements, some are listed below:

```
$output = new pnHTML();
$output->Start();
$output->End();
$output->TableStart();
$output->TableEnd();
$output->TableAddRow();
$output->Text();
$output->Title();
$output->BoldText();
$output->FormStart();
$output->FormEnd();
$output->FormText();
$output->FormTextArea();
$output->FormHidden();
$output->FormList();
$output->FormSubmit();
$output->PrintPage();
$output->setInputMode();
$output->setOutputMode();
$output->Redirect();
$output->LineBreak();
$output->URL();
return $output->GetOutput();
```

6. Multilingual modules

To make your module "international" (i.e., multilingual :)

- inside your code, instead of "row" text, use constants (ex: _HELLO)
- under module's "pnlang" directory, make as many sub-directories as the languages needed (ex: "eng", "fra") and add files "admin.php" and "user.php" to each one.

Exemple 6-1: File mymodule/pnlang/fra/admin.php

```
<?php
define("_MODULETITLE","Join Project module");
define("_ADMININTERFACE","Interface administrateur");
define("_LISTOFPROJECTS","Liste des projets");
?>
```

Exemple 6-2: File mymodule/pnlang/fra/user.php

```
<?php
define("_MODULETITLE","Join Project");
define("_JOINAPROJECT","Joindre un projet");
define("_LISTOFPROJECTS","Liste des projets");
?>
```

Exemple 6-3: Utilisation de ces constants dans pnadmin.php ou pnuser.php

```
$output->Title(_MODULETITLE);
```