Plugin API documentation

Plugin API documentation table of contents:

Plugin directory structure

Plugins are installed into %root/plugins directory, the typical plugin directory structure is as follows:

%pluginname

- admin_files
    - %pluginname
    - lang
    - include
- classes
- css
- install
    - sample_templates
    - sql
- smarty_camp_plugins
- template_engine

admin_files/%pluginname

Contains files which will be used in the admin interface. For further information see the Info file description/Menu section.

admin_files/lang

Contains language files for localizer

admin_files/include

Contains hook files. For further information see the Hooks section.

install/sample_templates

Contains files which are used to display this plugin in sample templates.

install/sql

Contains files needed for installation of database changes. How to apply database changes

smarty_camp_plugins

Contains smarty extensions.

template_engine

TODO: Mugur

Function and class reference

function **camp_get_plugin_path**($pluginName)

returns standard plugin path for $pluginName

CampMail

CampMail is a static classed which is providing mailing functions to Newscoop and it's plugins.

static boolean **ValidateAddress**($p_email)

returns true if $p_email is a valid email address

static void **MailMime**($recipients, $text=false, $html=false, $hdrs)

$recipients - list of recipients divided by semicolon

$text - Body of the text message

$html - Body of the HTML message, if both $text and $html are set, both contents will also be sent to $recipients.

$hdrs - Message headers

Sends an email to list of $recipients.

FormProcessor

This class provides functionality to build an form using Pear Quickform.

static void function ParseArr2Form(&$form, &$mask, $side='client')

&$form - object, reference to HTML_QuickForm object

&$mask - array, reference to array defining to form elements

$side - string, side where the validation should happen

CampContext

Todo: Mugur

CampContext::_get() and CampContext::_set() methods search for plugin defined objects in the plugins directoriesCampContext

CampContext::registerObjectType() registers new plugin custom objectsCampContext

CampContext::registerListObject() registers new plugin list objects

Hooks

What is a hook

Hook, in the scope of plugin API, is a way to show part of the plugin in an existing core Newscoop admin screens.

Which screens are able to use hooks

Hooks are currently implemented in following PHP admin files:

issues/edit.php
sections/edit.php
articles/edit_html.php
system_pref/index.php
system_pref/do_edit.php
pub/pub_form.php

How to use hooks

Hooks are stored in newscoopRoot/plugins/yourPlugin/admin-files/include directory. When you want to make a hook on article edit screen, you create subdirectory "articles" and under this subdirectory you create a file "edit_html.php" which will be loaded everytime when someone opens an article edit screen.

New directory structure would look like this:

newscoopRoot/plugins/yourPlugin/admin-files/include/articles/edit_html.php

Info file description

There should be $info array declared at the beginning of the %pluginname.info.php.

Basic information

string **Name**

This part of $info variable array describes the plugin name. It should be the same as the plugin's containing folder.

string **Version**

Describes which Newscoop version is supported by a plugin.

string **Label**

Label is being used as the plugin name in the plugin manager.

string **Description**

This is being used as the plugin description in the plugin manager

string **Handler functions**

Those are function which are being called at a specific time during the plugin handling. They should also be defined in the info file.

install

This function is called after the plugin is install, it should run any SQL install scripts.

1. CampInstallationBaseHelper::ImportDB(CS_PATH_PLUGINS.DIR_SEP.'%pluginName'.DIR_SEP.'install'.DIR_SEP.'sql'.DIR_SEP.'%filename.sql', $error_queries);

enable

This function is called when plugin is enabled at the plugin administration. Plugin enable will work even if this function is not defined, it is an automatic process, plugin developer do not need to enable the plugin in code manually.

disable

This function is called when plugin is disabled at the plugin administration. Plugin disable will work even if this function is not defined, it is an automatic process, plugin developer do not need to disable the plugin in code manually.

update

This function is called when a plugin is being updated, usually to run an upgrade SQL script.

uninstall

This function is called when a plugin is being uninstalled. The plugin should undo any database changes which it made during or after installation.

```
1. 'install' => 'plugin_%pluginname_install',
2. 'enable'  => 'plugin_%pluginname_install',
3. 'update'  => 'plugin_%pluginname_update',
4. 'disable' => 'plugin_%pluginname_disable',
5. 'uninstall' => 'plugin_%pluginname_uninstall'
```

array **localizer**

Contains configuration for Localizer.

- id - id for localizer, should be "plugin_%pluginname"
- path - path for localizer files, root starts at plugin's root directory
- screen_name - User friendly name

```
1. 'localizer' => array(
2.         'id' => 'plugin_%pluginname',
3.         'path' => '/plugins/%pluginname//*',
4.         'screen_name' => '%pluginname'
5. )
```

Menu *(array)*

This part of the configuration describes the menu structure of the plugin. This menu structure will be created under the "Plugins" main menu. This configuration property is designed to be a nested array.

string **Name**

Internal menu name.

string **Label**

This is the label of the menu. Localizer strings will be added automatically.

string **Icon**

Path to a Icon which will be displayed in the menu. The path's root is at plugin's root folder so if the path is "/css/%pluginname.png", the full path would be "newscoop_root/admin_files/plugins/%pluginname/css/%pluginname.png".

string **Path**

Path to a .php file which will be loaded after click on the menu item.

string **Permission**

Name of permission which is required to show this menu item.

array **Sub**

Submenu structure, it contains the same configuration element as a main menu structure.

Example for menu and submenu:

```
'menu' => array(
    'name' => '%pluginname',
    'label' => '%pluginlabel',
    'icon' => '/css/pluginMenuMain.png',
    'sub' => array(
        array(
            'permission' => 'plugin_%pluginname_admin',
            'path' => "%pluginname/admin/index.php",
            'label' => 'Administer',
            'icon' => 'css/pluginMenuAdmin.png'
        ),
        array(
            'permission' => 'plugin_%pluginname_user',
            'path' => "%pluginname/admin/user.php",
            'label' => 'Use plugin',
            'icon' => 'css/pluginUserAdmin.png'
        )
    )
)
```

array **no_menu_scripts**

No menu script sections marks file, to which the menu and standard Newscoop admin header is not attached. This is useful specially for AJAX files.

```
'no_menu_scripts' => array(
    '/plugin_name/admin/edit.php',
    '/plugin_name/admin/edit_item.php'
),
```

Configuration and permissions

array **userDefaultConfig**

This is the configuration that will be used in case that user doesn't have any configuration saved. It contains name-value configuration settings.

```
 'userDefaultConfig' => array(
    'name' => 'Valule'
)
```

array **permissions**

This configuration contains name value collection consisting of permission name and permission description/

```
'permissions' => array(
    'plugin_%pluginname_somepermission' => 'Some permission for %pluginname'
)
```

Template engine

```

'template_engine' => array(
    'objecttypes' => array(
        array('%pluginname' => array('class' => '%pluginname')),
        array('%pluginname' => array('class' => '%pluginnameItem')),
    ),
    'listobjects' => array(
        array('%pluginname' => array('class' => '%pluginname', 'list' => '%pluginname', 'url_id'=>'inv')),
        array('%pluginnameitems' => array('class' => '%pluginname', 'list' => '%pluginname', 'url_id'=>'inv_it')),
    ),
    'init' => 'plugin_%pluginname_init'
```

12.   )
13.);

Installer for plugins on the plugin managment page:

- using a package format to upload just one file, which will be extracted to a tree
- adding 'install' section to the info.php file, which will do the install process: coping files, adding tables etc.

old doc:

The 3.1 plugin API was implemented as follows:

- function camp_get_plugin_path($pluginName) returns the standard plugin path for a certain plugin name
- class CampPlugin for managing plugins (initialization, list, default config etc.)
- CampPlugin::createPluginMenu() for initialization of the plugin menu in the admin interface
- CampPlugin::getPluginsInfo() returns the plugin related user rights
- new user right 'plugin_manager'
- class FormProcessor (needs explanation)
- class CampMail (needs explanation)
- plugin default user rights (needs explanation)
- plugin template directories (needs explanation)
- plugin custom actions (needs explanation)
- CampContext::_get() and CampContext::_set() methods search for plugin defined objects in the plugins directories
- CampContext::registerObjectType() registers new plugin custom objects
- CampContext::registerListObject() registers new plugin list objects

Each plugin has an %pluginname.info.php file in it's base folder, which descripes a list of items:

- name, version, label, description
- menu: extentions to the admin menu
- userDefaultConfig: userright names for liveuser
- permissions: for the admin users
- no_menu_scripts: those admin pages which don't need the menu (like in admin.php, may should replaced by using an post/get parameter)
- template_engine: to init the template engine