A moodle Assignment submission/feedback & Webservice plugin tutorial

By Samuel Deane.

Contents

[Overview 1](#_Toc405977639)

[What plugin to choose? 2](#_Toc405977640)

[Setting up a plugin development environment, 2](#_Toc405977641)

[Getting a copy of moodle 2](#_Toc405977642)

[PHP issues 2](#_Toc405977643)

[The text editor to use 3](#_Toc405977644)

[Folder layout and finding the APIs 3](#_Toc405977645)

[How to apply the API 4](#_Toc405977646)

[Install a plugin 4](#_Toc405977647)

[Submission plugins 4](#_Toc405977648)

[General Plugin layout 5](#_Toc405977649)

[Feedback plugins 5](#_Toc405977650)

[Webservices 5](#_Toc405977651)

# Overview

This tutorial is designed to provide additional information to complete the development information presented on the moodle site (<https://docs.moodle.org/dev/Main_Page>). For the most part the information is aimed towards the development of Assignment Submission and Feedback plugins and web service plugins, though it may be extendable to many of the other plugin types supported by moodle. This tutorial is 10 pages long and cover the following types of :

* What plugin type to choose,
* Help with general plugin development environment including some general moodle/PHP hints and getting started,
* greater explanation of the moodle function hooks,
* extended API lists (referencing the API functions where they are defined) and
* some layout guides

# What plugin to choose?

The type of plugin to choose depends the functionality that you want to introduce to moodle. A full list of possible plugin types is available at <https://docs.moodle.org/dev/Plugins>. Of these there are 5 mentionable types

* Activity Modules – As the name implies Activity modules are for introducing a new type of activity to moodle, for example this might be something similar to the core moodle Activity modules that already exists such as a Forum, Quiz or Assignment
* Assignment Submission plugin – The M in moodle stands for modularity so if you want to keep all the benefits of the Assignment Activity module but want to extend it by introducing a new submission method or have come form of submissions processing, this is the plugin type to choose.
* Assignment Feedback Plugin – again keeping all the benefits of the Assignment Activity module but want to add some form or processing on the feedback side such as pdf feedback, some limited grading scheme, then this is the plugin type to choose.
* Local if nothing else fits and this is more of simple one of solution then choose the core plugin
* Web services, whilst technically a local plugin it is worth mentioning that is it possible to design plugins who can allow external interaction with some plugins and moodle components (though quite limited). It should be noted that some moodle provides do not allow external modification of files and furthermore Moodle itself recommends that only private and draft area files should be uploaded.

# Setting up a plugin development environment,

## Getting a copy of moodle

First get a copy of moodle from the moodle download page, it is recommended that you choose a version of moodle that is the same as the one you are developing for.

For those developing on Mac or Windows there are installer packages that comes with XAMPP (apache and PHP)) configured to run moodle.

* <https://download.moodle.org/windows/>
* <https://download.moodle.org/macosx/>

Of course moodle can also be downloaded separately at <https://download.moodle.org/releases/latest/>

Note that older and other release are found on the right hand side

## PHP issues

### Php not working at all

I seem to recall that one of the versions I downloaded did not have PHP turned on by default, this can be fixed by uncommenting (remove the ‘#’) on the following lines in the apache\conf\extra\httpd-xampp.conf configuration file (note that for other releases these lines appear in apache\conf\httpd.conf):

LoadFile "<install\_dir>/server/php/php5ts.dll"

LoadFile ""<install\_dir>/server/php/libpq.dll"

LoadModule php5\_module "<install\_dir>/server/php/php5apache2\_4.dll"

More information on Moodle’s PHP see: <https://docs.moodle.org/28/en/PHP>

### Speed up php

To make php faster to respond to changes, comment out the lines under [opcache] at the bottom of php.ini located in the PHP folder of the server. As the name indicates this function caches php operations so commenting out these lines causes the server to re-process the code anew each time.

Note that changes will only take place after the sever has been restarted

### Php webserver 500 error

There are multiple reasons why a 500 error may arise among these includes code that references a function that does not exists. Even if the function with the error is not even called by the Webservice function a 500 error will still be returned. In short know that requiring a file with bad code can lead to such an error even if it is not used.

## The text editors and IDEs

Moodle is written in PHP, so it is highly recommended to choose a text editor that has built in support for PHP. A recent poll lists the following popular PHP development editors:

* PhpStorm (got 40% of the vote though is paid software)
* Sublime Text (got 19% of the vote, is free and well supported, tends to get a good rap for most languages)
* NetBeans (got 15% of the vote, is free, PHP is fairly well supported)
* Zend Studio
* Notepad++
* Eclipse with PDT
* Dreamweaver
* Aptana
* Vim
* phpDesigner

See <http://www.sitepoint.com/best-php-ide-2014-survey-results/> for the complete results of the pole, with comparisons and some personal reviews.

Since most large IDEs come with PHP support, the editor/IDE to choose comes down to personal preference. I primarily used NetBeans as the PHP development environment assisted by Notepad++ as I had them both installed and was familiar with them from previous PHP development.

## Folder layout and finding the APIs

The moodle documentation only provides glimpses into the API, there are no complete lists of functions nor are there comprehensive examples.

So to make plugin development simpler it is highly recommend to have some knowledge of the layout of moodle folders so that you can identify where relevant functionality that may be required is located. More often than not the definitions of the functions provides significantly more information that found on the moodle site, furthermore some functions are often missing from those presented. The presented list of APIs can be found on the moodle website at <https://docs.moodle.org/dev/Core_APIs>. However the site is often a little slow so it might be advantageous to keep a copy on your machine and add some bookmarks to them.

The actual folder layout of moodle is fairly self-explanatory with most related files under their appropriate folder name under the moodle directory.

Some of the more notable folders include:

[mod] – holds all the plugins

[enrol] – holds all the enrolment information (use to get student info)

[lib] – holds all the libraries for all the services such as

[filestorage] … defines methods to store and retrieves files stored in the moodle file system

[filebrowser] … more secure though less functional version of above

[form] … defines all the form elements

[dml] … defines all the sql call functions based on the sql db used (highly recommended to read when adding calling the db)

### Notable files

In these folders most of the relevant functions are often located in the foldername.php, locallib.php or externallib.php (for components with web service functions) files in the respective folders. As stated above, more information can be attained in viewing these functions at their declartion.

* Index.php the main entry into moodle … the default page to access moodle
* config.php – holds some of the generic configuration information, such as library paths and site preferences, accessed through the global variable **$CFG**. Also install.php defines additional information attributes
* \lib\dml\moodel\_database.php – the moodle database functions accessed by the global variable **$DB,** For IDE recognition add the line /\* @var **$DB** moodle\_database \*/ after requiring the moodel\_database.php file.
* \lib\filestorage\file\_storage.php – the moodle file storage functions (often denoted as the variable **$fs,** For IDE recognition add the line /\* @var **$fs** file\_storage \*/ after requiring the file\_storage.php file,

This is often need as the ‘**$fs** = get\_file\_storage(); ‘ does not specify the type.

* \lib\moodlelib.php - Defines many of the core functions of moodle
* \lib\filelib.php –Defines functions for file handling such as draft areas for files that is areas to temporally store files while working on them in the browser also allows user to quickly cancel modifications as those they made are to the draft area and not to the files themselves.

Note **$USER** is not specified in full in one area, though its attributes can be seen in the user table on the moodle database. However some relevant functions are located in the \lib\moodlelib.php on lines 1740-2013.

Moodle in its modular design heavily relies on extending classes so it is often advantageous to view the parent classes of the plugin to be implemented. For instance the assignment submission plugins are extensions of the assign\_submission\_plugin (located in submissionplugin.php) which itself extends the assign\_plugin (located in assignmentplugin.php).

## Tips on development

* To get the most out of IDE/editor it is recommend to include the entire moodle directory in the external library/include path for the project. This will allow the editor to auto complete and fetch information on functions outside the directory tree of the project itself.
* In conjunction with the previous point, telling the IDE what the variable type, i.e /\* @var $db moodle\_database \*/, allows it to provide hints (drop down menus of possible functions) for variables that are not explicitly defined. Just remember to require\_once() the class file i.e require\_once("$CFG->libdir/dml/moodel\_database.php") and known that moodle does not use these hints in its own code.

see **Notable files** for further examples

* Sometimes it is difficult to identify where the definition of a function is, even after back tracking through the require\_once statements. Other times you might know where the function is defined but not how to use it. In situations like these I have found it beneficial to use a text search program (TextCrawler or similar) to find the definition and uses of functions.
* Often the best examples for using functions can be found in the test files located in the ‘tests’ folder in the same folder as the file where the function is declared i.e for the file storage functions see \lib\filestorage\tests

# Submission plugins

Moodle has some information on assignment submission plugins, which can be found at <https://docs.moodle.org/dev/Assign_submission_plugins>

Moodle states that:

An assignment submission plugin is used to display custom form fields to a student when they are editing their assignment submission. It also has full control over the display the submitted assignment to graders and students. Plugins participate in all assignment features including backup/restore, upgrades from 2.2, offline grading, group assignments and blind marking.

## Install a plugin

To install a plugin, just place in the appropriate directory, for

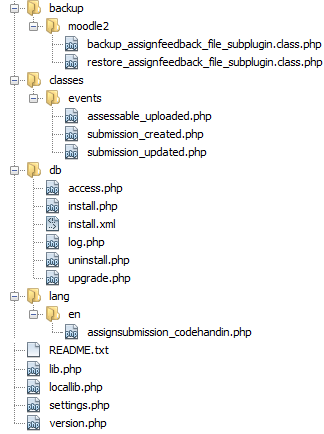
* assignment submission plugins: mod\assign\submission\
* assignment feedback plugins: mod\assign\feedback\
* web services: \local\

Login to moodle as an administrator and click ‘plugins overview’ under Site administration/plugins on the left had sidebar - this will show all installed plugins.

Then To install/update a plugin simply click ‘Check for available updates’ and Follow the install …

## General Plugin layout

The submission plugin has the following layout



<- defines classes that backup and restore the plugins data and structure

… back files

… restore files

<- defines event classes that perform some action on the trigger (the name)

… do something after an assessable component is uploaded

… do something after a submission is created

… do something after a submission is updated

<- the capabilities of users to view, add and submit (can use assign capabilities!)

<- run custom code on installation of the plugin (usually don’t have to change)

<- defines the database tables for the plugin\*

<- used to log add, update, view and view all actions to the db

<- run custom code on uninstallation of the plugin (usually don’t have to change)

<- run custom code on upgrade of the plugin

<- defines all the language tags

<- README file about the plugin (not technically required but good to have)

<- functions required by the API i.e. used to allow the web service to download files

<- where all core functionality of the plugin is located

<- defines the settings page/form for the plugin

<- the version of the plugin

Backup & classes/events will not be covered, see the file submission assignment plugin for an example.

### db/Access.php,

Defines the additional privileges of users for the plugin, this may not be required as the submission plugin is fairly comprehensive.

Take the following example:

$capabilities = array( // inherit capabilities from mod/assign

'mod/codehandin:addinstance' => array( # capability name

‘riskbitmask' => RISK\_XSS, # ... add scripted content\*

'captype' => 'write', # can write data (or ‘read’)

'contextlevel' => CONTEXT\_COURSE, # at the context level\*

'archetypes' => array( # array of users/levels\*

'editingteacher' => CAP\_ALLOW, # teacher can add instance

'manager' => CAP\_ALLOW # manager can add instance

),

'clonepermissionsfrom' => 'moodle/course:manageactivities' # include parent permissions

),

\* see \moodle\lib\accesslib.php for other CAP\_, CONTEXT\_ and RISK\_ types

### Install.php

Run custom code on installation of the plugin (usually don’t have to change)

Good for populating lists based on local environments i.e. for a programming plugin … check available languages … setup external compile etc. For things that are generic and won’t need to be changed, otherwise hard coded or define (i.e. define('ROLENAME\_ORIGINAL', 0);) if needed.

### Locallib.php & settings.php

Define the core functionality of the plugin

… Additions to the setup of an assignment instance

Most other parts are self-explanatory

# Feedback plugins

# Webservice

<https://docs.moodle.org/28/en/Using_web_services>

See <https://docs.moodle.org/dev/Web_services_%282.0_onwards%29>

<https://docs.moodle.org/dev/Web_services_files_handling>

resource

### 500 errors hints

There are multiple ways a 500 or server not found error can be generated however don’t be fooled into thinking that a 500 always means that the server cannot be found as often they can result from errors in the server code such as a Call to an undefined method

To find errors on the server the first thing to do is make sure all wsfunctions exist and the file can be passed. A simple way to do this is to go to

Home / ► Site administration / ► Plugins / ► Web services / ► External services,

click functions for the web service under development and then fix any errors that are shown. Of course remember to save each change and refresh the server. If changes are not applied immediately see \_\_\_ for turning page caching off.

Also note that this will not check the parsing validiility of required functions (those functions that are defined on other .phps)

All \_paramters functions in the web service must begin with

‘return new external\_function\_parameters( ‘ , however no such wrapper function is required for returned results?

# General PHP info

## Object and Arrays

stdClass is PHP's generic empty class, kind of like Object in Java or object in Python. However this can also be interpreted as an array by casting it.

Arrays operate under the key value pair arrangement and can be defined as Array(‘key’=>’value’);

To helpful functions are array\_keys($anArray) which returns all the keys in an array helpful when grabbing ids from an array that contains multiple objects/subarrays which have an id.

foreach (array\_expression as $value)

statement

foreach (array\_expression as $key => $value)

statement

## Accessing methods and values

Scope Resolution Operator (::), allows access to static, constant, and overridden properties or methods of a class:

Self::to access a function within the same class

className::functionName($args) to access a static function in another class

Sometimes does not work so the class must be instantiated first i.e $aClass = new className();

Then call className::functionName($args).

# General SQL info

Moodle returns results in the form of a key-value pair, this can cause a problem if the records returned are not uniquely identified, a surety in cases where LEFT JOIN/RIGHT JOINs are performed. To overcome this problem it is necessary to introduce a unique number for each row, sometimes called a record number:

SELECT @rn:=@rn+1 AS rank,

… rest of the selected statement

FROM (… other tables) JOIN (SELECT @rn := 0 FROM DUAL) AS sub

// assignid/ ... two zip files one named by the assignment id (holds test files) and the other named by the username (holds submission files)

// new fs

//'savepath' 'downloadFolder' 'filePath'

// C:/.../ assignid/s/ submissionid.zip

// assignid/a/ assignmentid.zip

// then unzip like old fs:

//'savepath' 'downloadFolder' 'filePath'

// C:/.../ assignid/s/ submissionid/file.\*

// assignid/a/ testid/file.\*