

Moodle forms

Moodle forms have been created based on [PEAR HTML_QuickForm](#) and technically is a fork of the library, as it contains Moodle fixes and changes. Original PEAR package is a bit outdated and it's not actively maintained for a long time.

Form definition

To create a new form, you need to define a new class that extends moodleform. Typically those classes are put in a separate file named with `_form.php` postfix. We will create a class `mod_php_submission_form` for student to upload his submission. We will put the code under classes directory as `mod/php/classes/submission_form.php`, so Moodle will automatically load the class when required. Our form will have an input field for title and allow student to either copy & paste a submission into a text field or attach a .php file. Just to show some forms functionality, we will create a select box where format can be choosen (text or file upload) so appropriate form element will be enabled (and other one disabled).

We define the structure of our form by overriding a method called `definition()`.

Moodle forms: `submission_form.php`

```
class mod_php_submission_form extends moodleform
{
    function definition()
    {
        $mform = $this->_form;

        $current = $this->_customdata['current'];

        $mform->addElement('header', 'general', get_string('submission', 'workshop')\
);

        $mform->addElement('text', 'title', get_string('submissiontitle', 'workshop'\
));
        $mform->setType('title', PARAM_TEXT);
        $mform->addRule('title', null, 'required', null, 'client');

        $choices = array('text', 'file');
        $default = 'text';
        $mform->addElement('select', 'format_choice', "Choose format", $choices);
        $mform->setDefault('format_choice', $default);
        $mform->setType('format_choice', PARAM_ALPHA);

        $mform->addElement('textarea', 'content_editor', "Paste your code", 'rows="2\
0" cols="70"');
        $mform->setType('content_editor', PARAM_RAW);
        $mform->disabledIf('content_editor', 'format_choice', 'noteq', '0');

        $mform->addElement('static', 'filemanagerinfo', "Instead of pasting the code\
, you can choose ".
            "'file' format and attach a file below.");

        $mform->addElement('filemanager', 'attachment_filemanager', "Upload file her\
e");
        $mform->disabledIf('attachment_filemanager', 'format_choice', 'noteq', '1');

        $mform->addElement('hidden', 'id', $current->id);
        $mform->setType('id', PARAM_INT);

        $mform->addElement('hidden', 'cmid', $current->cmid);
        $mform->setType('cmid', PARAM_INT);

        $this->add_action_buttons();

        $this->set_data($current);
    }
}
```

```
}
}
```

One of the properties of the extended class is `$this->_form`. This is a `MoodleQuickForm` class that extends a class from `QuickForm` library. We will assign it to the variable `$mform` as we will use it a lot. To add a new element we use `addElement` method. The first parameter is always a quickform element type and the second is a name (think about it like about unique id) for the element. The third and following arguments depend on which type is being created, have a look at [Moodle form documentation](#).

With `$mform->setType()` we set the type of the value that should come from HTML form. This information will be used for validation, you can use any of the `PARAM_*` constants here.

The first argument in `setType()` and similar methods is the name (id) of the element - it should match the first parameter of `addElement()`. You can probably guess what `setDefault()` does.

One of the interesting `$mform` methods is `disabledIf()`. It can disable/enable an element based on some condition. The first parameter is (again) an id of the element we want to enable/disable. The second is the element the value we depend on, then condition and finally a value to compare to. So the line:

```
$mform->disabledIf('content_editor', 'format_choice', 'noteq', '0');
```

will read: disable 'content_editor' when the value of 'format_choice' is not equal to 0. The client-side JavaScript will be taken care off by Moodle. Be warned that `disabledIf` does not work with all elements, e.g. see [MDL-29701](#).

`add_action_buttons()` is just a simple helper method that adds submit & cancel buttons to the form.

Validation

The form elements will be processed according to the type we have set with `setType()` method. For instance, if form field is defined as `PARAM_INT` and user inputs non-integer value (say: "Hello") then you will receive the validated data as integer "0".

On top of that, you can add your custom form validation code by implementing validation method. Let's make sure that source code is provided by either filling in the text box or by attaching a file.

Moodle forms: submission_form.php

```
function validation($data, $files) {
    global $USER;

    $errors = parent::validation($data, $files);

    $usercontext = context_user::instance($USER->id);
    $files = array();
    if(isset($data['attachment_filemanager'])) {
        $fs = get_file_storage();
        $files = $fs->get_area_files($usercontext->id, 'user', 'draft', $data['a\
ttachment_filemanager']);
    }

    // Make sure that either file or pasted code was submitted.
    if ((!empty($data['content_editor']) && count($files) > 1) ||
        (empty($data['content_editor']) && count($files) <= 1)
    ) {
        $errors['format_choice'] = 'Either submit a file or paste the code.';
    }

    return $errors;
}
```

Start with calling a parent validation function, in case there are some checks implemented there (there are none for `moodleform` class):

```
parent::validation($data, $files);
```

Then implement validation logic. If there are any errors, return them in array - one key/value for one error. The key must match one of the element names in the form - Moodle will display an error next to the element form. validation() method should return an empty array if there are no errors.

Form processing

The basic workflow is as follows:

Moodle forms: view.php

```
$mform = new mod_php_submission_form();
if ($mform->is_cancelled()) {
    // form cancelled, redirect
    redirect(new moodle_url('view.php', array()));
    return;
} else if (($data = $mform->get_data())) {
    // form has been submitted
    mod_php_save_submission($data);
} else {
    // Form has not been submitted or there was an error
    // Just display the form
    $mform->set_data(array('id' => $id));
    $mform->display();
}
```

The form is by default submitted to the same PHP script where it was displayed. If the URL for our module is mod/php/view.php?id=3 then after form submit, a POST request will be sent to mod/php/view.php. id parameter will be lost, so we define id as a hidden integer in the form and then set it with \$mform->set_data(array('id' => \$id));. Keep in mind that Moodle does not make a difference between GET & POST data - by using

```
$id = required_param('id', PARAM_INT);
```

we will get id parameter from either URL (GET) or POST data.

The \$data will contain the values submitted from the HTML form rendered, already validated and sanitized. We need to persist the user's submission data to DB, for now all that we need to know is handled by function mod_php_save_submission().

Files processing

Handling files attached to a Moodle form is not as straight-forward as one may think. There is [some documentation](#) already available, here we will cover only part of it. We will go deeper into Moodle File API later on.

When a form with filemanager is submitted, Moodle will store the file(s) in moodle data and create new entries in mdl_files table. The file is stored as component "user" and filearea "draft". Both are columns in mdl_files table. First of all we need to find the file by retrieving itemid of file(s) with:

```
$draftitemid = file_get_submitted_draft_itemid('attachment_filemanager');
```

itemid - another column in mdl_files is used instead of primary key id, because file manager can handle more than one file. \$draftitemid will refer to all the files uploaded.

Now that we know what do we need to save, we will put files under correct area in Moodle files. There is a helper function that will copy the files from draft into area specified by us:

```
file_save_draft_area_files($draftitemid, $cm->context->id, 'mod_php', 'submission', 0);
```

\$id is a context id to which the files are to be attached.

Third argument is component - “mod_php” in our case. All files with component equal to “mod_php” will be handled by our custom module.

Fourth argument is a filearea - we are free to put there whatever we want here. We could create several different file areas to logically split files we’re dealing with into groups. For handling user submissions, we will name our file area “submission”.

Finally, within the same file area, we could have several submissions. For example we could allow student to upload first version of assignment, then second, etc. For tracking those, we would use a different value for itemid. In our case we will only let one set of files at the time, so we’ll set it to 0. In mdl_files there is a column called userid, which will keep track of who uploaded the file.

Editing submission

Let’s tackle another piece of functionality now - if user already submitted their data, we want them to edit existing submission. So the form should show up as empty the first time around but if the submission was already done, we allow the editing.

The data for current user submission will be saved in our custom table, for now all we need to know is that we have a function mod_php_get_my_submission() to retrieve it.

Files need to be handled separately - there is no link in our submission table to user submitted file. Instead our files are found by searching mdl_files for matching context, component (mod_php), file area (submission) and item id (0).

To have files available for editing, we need to copy them from real storage into draft area with file_prepare_draft_area() function. We then simply use set_data() to set existing values in the rendered form.

Moodle forms: view.php - editing

```
$submission = mod_php_get_my_submission();
// ...
if ($submission) {
    $draftitemid = 0;
    file_prepare_draft_area($draftitemid, $cm->context->id, 'mod_php', 'subm\
ission', 0);
    $mform->set_data(array('content_editor' => $submission->code,
        'title' => $submission->title, 'attachment_filemanager' => $draftite\
mid));
}
```

Pay attention to the first argument of file_prepare_draft_area() - \$draftitemid. It’s passed by the reference and set to correct value inside the function.

That’s it - we have just created a functional Moodle form, see [the chapter5 code](#). For more information about the topic, have a look at [Moodle documentation](#). There are also few good hints to read about [designing usable forms](#).

▼ Title header

Title*

Choose format

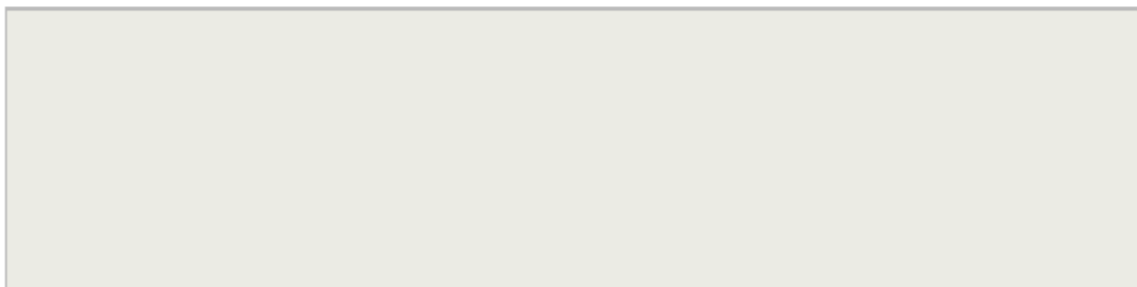
text ▼

Paste your code

```
echo 'Hello world';
```

Instead of pasting the code, you can choose 'file' format and attach a file below.

Upload file here



Save changes

Cancel

Moodle form