File API

Moodle 2.0 introduced a new file system for Moodle. The API is specifically for internal files Related to the this API is the repository plugins functionality which deals with file stores outside the site’s files system.

Internal Files are stored in a combination of physical files and a database table record.

File API (cont)

- Many of the File API’s functions are defined in the ***/lib/filelib.php*** file.
- Additional functionality is defined in the `filestorage` class (***/lib/filestorage/file_storage.php***) for low-level file activities,such as retrieving the file contents.
- Also the `filebrowser` class (***/lib/filebrowser/file_browser.php***) for browsing files in code.

File API (cont)

The usual process when working with files is:

- To create a draft area for the user;
- If the files are existing files, as in the case of they being about to be replaced or edited, to copy the files into the draft area.
- Once the user commits to saving the file, to copy the draft file(s) back to the plugin's file area;
- When any user attempts to access the file(s), to deliver the file after doing relevant access checks.

File API (cont)

Plugin's files need to be stored in plugin’s file area, similar to directories but not quite the same. A file area is defined by:

- A `contextid` – the context in which the file is being saved.
- The plugin’s Frankenstyle name;
- A file area type – this is for the plugin to determine. For example, you could have an images type and another for non-image files. Most simple plugins just have one type;
- A unique `itemid` – again something the plugin determines, it might be the record id of a related record in another table; otherwise use 0.

File API (cont)

To create a link to a file, plugins generally call `moodle_url::make_pluginfile_url()` with the following parameters:

- `contextid`
- `component – i.e. plugin`
- `area`
- `itemid`
- `pathname`
- `filename`
- `forcedownload`

The resultant link will be in the format:

/**pluginfile.php**/$contextid/$component/$filearea/arbitrary/extra/information/filename.ext

File API (cont)

- Moodle has several files relating to the delivery of files: **file.php**, **draftfile.php**, **userfile.php**, and **pluginfile.php** which is the one in the link.
- **pluginfile.php** will call a function `<pluginname>_pluginfile()` in the plugin's **`lib.php`** when requested as so all plugins that store and deliver files have to define this callback function.

File API (cont)

The callback should expect the following parameters:

- `course` – the course object (will be null if not applicable)
- `cm` – the course module object (will be null if not applicable)
- `context` – the context based on the context id – could be system context
- `filearea` – the name of the file area
- `args` –itemid, path – the function decides what it does with these
- `forcedownload` – whether or not to force download
- `options` – additional options affecting the file serving, the function decides what to do with these; usually, it is not defined

Return `false` if the file is not found. Otherwise, send the file using the File API's `send_stored_file()` function after processing.

File API (cont)

The file request processing will:

- Check context is relevant
- Check the file area
- Check user is logged in – if applicable
- Check for any required capabilities
- Identify `$itemid`, `$filepath`, `$filename` from the forward slashed separated `$args` parameter
- Attempt to retrieve the file from the file store and return results – either `false` or using the `send_stored_file()` function

Links

- ‘File API’ page (https://docs.moodle.org/dev/File_API).
- Repository plugins (https://docs.moodle.org/dev/Repository_plugins)
- ‘File API Internals’ page (https://docs.moodle.org/dev/File_API_internals).

File Form Elements

Moodle provides 3 custom elements to work with file uploads – they are closely tied to the **File API**

- `filepicker` – a more appropriate replacement for the HTML file element, but really useful to get a file that is to be processed and then discarded, such as a CSV upload.
- `filemanager` – the recommended way to get uploaded files and save them to the appropriate file area.
- `editor` – this is actually a specialised text-area with an HTML editor (so not strictly a file element), but it allows the use of files such as images, sounds, and video within that HTML (not covered in this course).

Element Options

File elements have an optional `options` array parameter and the possible options are:

- **subdirs** (not applicable for file picker) – whether to include subdirectories
- **maxbytes** – maximum file size
- **areamaxbytes** (not applicable for file picker) – maximum file area size
- **maxfiles** (not applicable for file picker) – the maximum number of files
- **accepted_types** – Defaults to '*', which is all files. You can use file extensions, e.g. `array('.txt', '.jpg', 'audio')` or, alternatively, file types, such as `array('audio', 'video', 'document')`
- return_types – Value is one or a combination of file 'types' constants – `FILE_EXTERNAL`, `FILE_INTERNAL`, `FILE_REFERENCE` or `FILE_CONTROLLED_LINK` – will be discussed later, we use `FILE_INTERNAL` which is a file saved internally in Moodle.

The editor also has other specific editor options.

File Types

These constants are actually defined in the **/repository/lib.php** file.

- Files that are uploaded to Moodle's internal file system are of the type `FILE_INTERNAL`.
- Where the file remains in the external repository and is accessed from there as a link, it is defined as of the `FILE_EXTERNAL` type.
- A variation of this type is the `FILE_REFERENCE` type, which refers to files that remain in the external repository, but may be cached locally.
- `FILE_CONTROLLED_LINK` type was introduced in Version 3.3. The documentation explains that 'the file remains in the external repository' – not very clear as exactly what this is.

filepicker

- Adding the element to the form:

```
$mform->addElement('filepicker', 'elementname',
$elementprompt, $attributes, $options);
```

- When the form has been submitted, get the file content:

```
$content = $mform->get_file_content($elementname);
```

  NB: The underlying PHP call is `file_get_contents()`, so the contents are returned as a string. Watch out if this is a CSV file and you want to process it.

- Get the filename:

```
$name = $mform->get_new_filename($elementname);
```

filemanager

Closely tied to the File API, process is:

1. Get a draft file area id with `file_get_submitted_draft_itemid()`. The value is ultimately used to populate the value of the `filemanager` element.
2. Prepare the draft file area using the id obtained in step 1 by calling `file_prepare_draft_area()`, existing files need be copied into the area before form is displayed, to be discussed later
3. On submission, the files are moved back to the file storage with `file_save_draft_area_files()`