EdRepo System Overview

Basic Concepts

EdRepo is designed to be a light- to mid-weight collection for storing, viewing, and organizing digital content. It has a particular emphasis on handling collections suited for educators, with and number of features (and, more importantly, metadata support for) primarily of use to educators. If using EdRepo for collections which do not have an educational focus, a number of aspects of EdRepo will most likely become irrelevant, although the system would still be usable.

EdRepo organizes information into "modules" and "materials." Modules are the basis of EdRepo's organization, and activities such as search or browsing the collection, adding new content, and editing content are centered around modules. A module is a collection of metadata and materials which are intended to be used together to form a single entity. A material is simply a file with metadata attached to it describing the file. For example, a module might be a lesson plan for introducing PHP programming to students. The module would therefore contain information such as a title, author, goals and objectives for the module (such as what students should learn), information about how long it is expected to take to teach the lesson, and four materials: a presentation (such as a PowerPoint file), an example PHP file for students to look at and analyze, a word processing file containing a lab for students, and a PDF file with basic PHP reference information as a hand-out for the class. Notice that, although the module is made of many files, and each file has its own metadata (the PDF might have a different creator than presentation; for example), the files are intended to be used together. Instead of having to add files to a single large archive and upload that, EdRepo's module-material system allows users to add as many physical files or links to a single container.

It should be noted that although files were used in the above example, EdRepo does not require that materials be uploaded files or that materials even reside inside the EdRepo system. Materials can either be file uploaded to EdRepo or links to any content available through the Internet. This allows, for example, someone who has made a number of YouTube videos demonstrating properties of physics to create a module about demonstrating physics and simply link to the YouTube videos. A module may contain materials of either or both types.

EdRepo is also modular in design, and supports different storage back-ends for modules and materials. This allows collection administrators to select the back-end storage system best suited for their collection. A small collection might want a simple, possibly feature-limited back-end which is easy to set up and has no special dependencies, while a larger, more complex collection may prefer a feature-full back-end running through a MySQL server, and be willing to invest in the time necessary to set up the database server. Not every back-end must support every possible feature, and the front-end is designed to be flexible enough to work with only the features actually supported by the back-end in use, providing back-ends support a minimum, limited set of features. The front-end communicates with the back-end through a data-manager, which is a collection of standard functions all back-ends must accept (although they may choose to do nothing with

functions they do not support other than return "Not Implemented"). The data-manager is contained within a file in the back-end, and selecting different data-managers selects different back-ends. Note that, while simple and efficient, this design does mean that only one back-end can be used at a time, and that EdRepo itself does not provide any method to synchronize or migrate between different back-ends automatically. Back-end writers are encouraged to write tools to help automate the process of moving data between back-ends, but EdRepo itself makes to attempt to provide this capability. Therefore, collection administrators must carefully select the back-end they intend to use which installing EdRepo, since it may be difficult and/or time consuming to switch back-ends later while preserving the collection's contents.

System Organization

EdRepo's code is organized into files centered around a single "task" or, in the case of more complex tasks (such as uploading or editing modules), into several files handling parts of the task. In the case of the latter, the files are located in their own subdirectory of the main install path, to keep closely related files together and away from other, less related files.

Files which are not intended to be accessed directly by users (for example, files with nothing but functions which do not print to a page, or configuration files), are stored in the "lib/" subdirectory of the main install path or a subdirectory of "lib/". This allows system administrators to easily set up rules to prevent their web server from displaying content not intended for view or download by users. Simply disallow access by browsers to the "lib" subdirectory(s) while still allowing PHP files to import files from those directories. This can be done easily in the Apache web server through the use of .htaccess files.

Files are named after the task they perform. Subtasks, such as confirmations, are handled by the same file which handles the main task, and are triggered by passing different "action" parameters to the file. For example, "moderate.php" file is used to handle module moderation tasks. It performs three different subtasks: displaying modules for moderation, approving modules, and denying modules. The default action is to display modules waiting for moderation. If a different action is passed (through an HTTP GET or POST) however, this file will perform the specified subtask, such as approving a module waiting for moderation, instead of the default task.

Every file in the front-end must handle validating logged in users and logging them off if necessary. Users log into through the "loginLogout.php" file (which also allows users to voluntarily log out) and are logged into the system via the back-end in use. Once logged in, users receive an "authentication token" which is stored as a cookie in their browser. If a user accesses a front-end through a browser which stores an authentication token, the page being accessed must read the token, check to see if it corresponds to a valid login (as determined by the back-end in use) and act accordingly. If the authentication token is deemed valid by the back-end, the front-end assumes the user is logged in. If the authentication token is not deemed valid, the front-end file must act as though the user is logged out and also unset the authentication token stored in the user's browser. This is necessary because back-ends may internally expire the authentication tokens and so invalid

authentication tokens from expired logins may be passed to the front-end, which needs to clear them to prevent their continued use. Front-ends therefore contain a function "logout" which handles the needed force-logout. This function handles both informing the back-end that the authentication token is being logged out, and removing the authentication token cookie from the user's browser.

Within the "lib/" subdirectories are three directories of particular importance: backends/, config/, and look/. The config/ directory contains files which contain collection-wide configuration settings. Users may edit these files by hand to configure their instillation of EdRepo, or EdRepo may provide a way to configure these files through its web interface.

The look/ directory contains themes for EdRepo. EdRepo supports simple themes which are made up of a CSS file to skin the HTML EdRepo outputs, simple icons used to indicate success or failure, and a header and footer, which are simply HTML pages which EdRepo will automatically print to the top and bottom of pages it displays. Collections may install as many themes as they wish, with each being contained under a different subdirectory of look/. The file look.php is used to select the theme in use and contains a single variable, $LOOK_DIR, which is the name of the subdirectory containing the theme the collection maintainer wishes to use. Changing this variable will change the theme used.

The backends/ subdirectory contains all possible storage back-ends which can be used by EdRepo. For example, a collection may come with two storage back-ends: a MySQL database back-end, for use with MySQL databases, and an XML flat-file back-end with reduced features for suited for small collections of systems which do not have access to a MySQL server. Administrators may install as many storage back-ends as they wish, but only one can be active at a time and EdRepo does not make any attempt to automatically sync of migrate data between different back-ends. Back-ends are connected to EdRepo's front-end (the files which actually display information to users) through the back-end's data-manager, which is a file named "datamanager.php" with a specific set of functions used by the front-end. Back-ends may internally work however they wish, but they must interface with the front-end only through the data-manager.

The active back-end is selected through the "backends/backend.php" file, which has a single entry, which simply requires the data-manager of the back-end to use. The imported data-manager is then responsible for importing any additional files required by the back-end. The back-end code may also be written directly into the data-manager if back-end writers prefer.

Back-ends are intended to be self-sufficient, and contain all settings, support files, and stored files specific to them within their own subdirectory. For example, a MySQL database back-end would be expected to store database configuration information (such as usernames and passwords to the databases it uses) within its subdirectory of backends/, and if it does not store material files themselves in the database, provide its own subdirectory for storing material files.

Standard Conventions

The following standard conventions are used throughout EdRepo's code, and knowing them may make reading the code easier. In addition, it is encouraged new coders either adhere to these conventions, or come up with better ones and re-write all parts of EdRepo to use the new conventions.

Front-end files check for subtasks through the "action" POST of GET parameter.

Most front-end pages perform multiple subtasks, with a default subtask if no subtask is specifically specified. The general layout of these files is as follows:

```
$action="default_action_name"; //Se the default action.
if(isset($_REQUEST["action"])) { //Was an action specified?
        $action=$_REQUEST["action"]; //Set the action to the specified action.
/* Actions which require special handling/initialization are handled here. */
...
if($action=="default_action_name") {
        //Do something
} elseif($action=="subaction1") {
        //Do something
}
... //Continue elseif($action==...) for every possible subaction.
} else { //Unknown action specified, given error.
        echo 'Error: Unknown action specified.';
}
```

Some (or all) subactions may require that a user is logged in and possibly even a specific minimum user level. In these cases, files generally check to make sure the user is logged in and at the required level before checking for actions dependent on those criteria (basically, an if(user if logged in and high enough privilege level) block around the above if($action==..) {} elseif(){} else {} block shown above).

Front-end files check for a logged in file through the presence of the $userInformation variable.

The first thing front-end files do is to determine if the user accessing the page is logged in with a valid authentication token. If they are, then the front-end fills the variable $userInformation with information about the user, so other parts of the front-end can use this information. If the user is not logged in, or the front/back-end determines that the user's authentication token given is invalid, than the $userInformation variable is unset (no longer exists). Therefore, other parts of the front-end check to see if a user is logged in or not simply by whether the $userInformation variable exists or not. If it doesn't, than it is assumed the user is not logged in. If it is set, that it is assumed the user is logged in and the $userInformation variable is an array with standard user information (as returned by the back-end).