HDMS Developers Guide

Introduction

This note is a developers guide for the HDMS Document Management System. It is intended for someone who needs to modify the system, so it describes the 'nuts and bolts' of the system to give a head start rather than trying to decode all of the source code from scratch.

I hope it helps. Try emailing grahamjones@physics.org for more help if necessary.

User Interface

HDMS is a web application, so the user interface is via web browser. The user interface is simple, and uses very little fancy technology - a bit of javascript for some of the search functionality, but that is about it - most of it is standard html.

Server Side

The server code is written in php and uses the CakePHP framework in order to limit the amount of code that is unique to hdms. It uses a MySQL database for storing the metadata about the documents and their revisions. The document files themselves are stored in the server's filesystem (not the database).

Database Structure

The following database tables are used by the HDMS application:

- Main Data Tables
  - settings - high level settings for the application - enable email notifications, pdf generator url etc.
  - docs - one record per document - doc title, number, doc type etc.
  - revisions - one record per document revision - revision status, author, revision comment etc.
  - route_lists - defines a route list - associates a number of route_list_entries to a revision.
  - route_list_entries - Identifies a user who is requested to review/approve a document - records the user's response (approve/reject) and their comment to accompany the response.
- User related tables
  - users : list of users (name, role, email address, encrypted password etc.)
  - roles : list of hdms roles for users (user, administrator, disabled)
  - positions : list of positions within organisation (staff, director etc.)

  - notifications : list of notifications sent to users.

- Ancillary Tables
  - facilities : list of facilities (business units)
  - doc_types : list of document types (policy, procedure etc.)
  - doc_subtypes : list of document subtypes (Finance, Governance etc.)
  - doc_statuses : list of document statuses (draft, rejected, issued)
  - route_list_statuses : list of route list statuses (submitted, completed etc.)
  - responses : list of responses to route lists (none, approve, reject)

Code Structure

The server code is written in PHP (v5.5 but older versions will probably work), which is a very common language for server code, so many people will be able to read it. To make future maintenance easier hdms uses the CakePHP framework for writing web applications, and it uses the standard file location and naming conventions for the framework.

This means that all code associated with the hdms application (rather than the cakephp framework) is in hdms/app. Within hdms/app, the following directories contain relevant hdms code: * Config : database.php defines how to connect to the database - very important!

- Model : Models are used to interface to database tables, so the Doc.php model is used to interface to the 'docs' table. The Models define the relationships between tables as well as containing useful re-usable code where it is useful to put it in a model rather than controller.

- Controller : Controllers are called when a user accesses a url. For example, a user accessing hdms/docs/homepage will call the 'homepage' function in the DocsController.php controller file.

  ```
  The controller interprets the data sent by the user, uses the
  relevant model(s) to extract the required data from the databse
  and sends the data to a View which will present the data to the
  user.
  ```

- View : Views accept data from a controller and present the data as html to the user. The view which is presented to the user is defined by a CakePHP convention.

  For example if the user requests hdms/docs/homepage, the Controller/DocsController.php file will be accessed and the homepage() function called. Once the homepage() controller function is complete, the View/Docs/homepage.ctp view will be used to present the data to the user.

- webroot : webroot is used to serve static data to the user. For example accessing /hdms/img/ca_logo.png will return the file

app/webroot/img/ca_logo.png. This is used primarily for cascading style sheets to define font sizes, colours etc (webroot/css directory); javascript code (webroot/js directory), or images (webroot/img directory).

Changing Things

A few examples of how to change things, which may help:

- Text in hdms page headers and footers:
  Look in hdms/app/View/Layouts/default.ctp - it is probably there.
- Text in main part of a HDMS page: Unless it is coming from the database, it will be in the View file, which can be identified by analysing the url used to access the page.

  The url has the form `/hdms/<controller>/<action>...` The View file used to generate the page will be `hdms/app/View/<controller>/<action>.ctp`.
- Data Presented on a Page: You need to make sure the controller sends the right data to the View. Find the right controller by analysing the url used to access the page.

  The url has the form `/hdms/<controller>/<action>...` The Controlller file used to send data to the view to produce the page will be `hdms/app/Controller/<controller>Controller.php`. The function within the controller file that is called is ().

  Once you have added the data from the controller, you need to update the view file to make it display it. The view file will be `hdms/app/View/<controller>/<action>.ctp`.
- Font Sizes, font spacing or colours: Look in hdms/app/webroot/css/hat_generic.css - it is there for most pages, although the homepage is more complicated so has its own styles in hdms/app/webroot/css/homepage.css