

Learning *Online* Network with CAPA

*Domain Coordination Tutorial And Manual*

August 28, 2008

LON-CAPA Group

Michigan State University

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Domain Configuration</b>	<b>3</b>
2.1	Log-in Screen Customization . . . . .	3
2.2	Setting E-mail Addresses for Administrators and Support . . . . .	4
2.3	Default Color Schemes . . . . .	5
2.4	Setting Domain Defaults: Authentication, Language and Time zone . . . . .	6
2.5	Setting Default Quotas for User Portfolios . . . . .	6
2.6	Identity Management: Searching for Users in an Institutional Directory . . . . .	6
2.7	Identity Management: Creating New Users . . . . .	7
2.8	Identity Management: Modifying Existing Users . . . . .	8
2.9	Identity Management: Automated Updates of User Information . . . . .	8
2.10	Course Catalogs . . . . .	9
2.11	Automated Enrollment in Official Courses . . . . .	10
2.12	Scantron Data Formats . . . . .	12
<b>3</b>	<b>Domain Management</b>	<b>13</b>
3.1	Creating Domain Coordinators . . . . .	13
3.2	Scheduling of Scripts Run Periodically . . . . .	13
<b>4</b>	<b>Integration with Institutional Systems</b>	<b>14</b>
4.1	Institutional Authentication . . . . .	14
4.2	Institutional User Categories/Affiliations . . . . .	16
4.3	Format Rule Definitions and Checks: Usernames and IDs . . . . .	17
4.4	Institutional Directory Information . . . . .	21
4.5	Search Filters for Official Course Categories . . . . .	29

# 1 Introduction

The Domain Coordination Manual includes both a description of tasks which require standard LON-CAPA interaction via a web browser, with the Domain Coordinator role active, but also tasks which require command line access to the primary library server in a domain. In some cases one individual may complete both these types of task, whereas in others a separate Systems Administrator may need to be called upon for tasks completed from the command line.

With the Domain Coordinator role active, the Main Menu provides a Domain Coordinator with access to the functionality needed to complete the routine tasks which a Domain Coordinator carries out, including creation of courses and assignment of course coordinator and author roles. Courses can be created interactively by completing a web form, or can be created in batch mode by uploading a file containing course specifications for one or more courses. Addition of users/roles can similarly be carried out interactively, or by uploading a text file of users. The Domain Coordinator may also periodically need to modify domain settings via the Domain Configuration menu.

LON-CAPA provides hooks to permit integration with institutional information systems to support the following procedures:

- Authentication via an institutional authentication service (e.g., LDAP)
- Automatic updates of classlists
- Automatic updates of user information
- Retrieval of institutional user information for individual users
- Searches for users at the institution
- Automatic import of student photos from an institutional repository

There are two perl modules included in LON-CAPA (*localauth.pm* and *localenroll.pm*, both located in */home/httpd/lib/perl*) which need to be customized to provide the integration with institutional systems. Although customization and testing will involve a systems administrator, and programmer(s) at an institution, a Domain Coordinator is likely to be involved in the design and testing phases of the integration, if not in the actual implementation. Set-up of a standalone LON-CAPA instance on a separate server is advised for the purposes of implementing and testing institutional integration, before enabling the new functionality on the production system.

Besides the integration described above, at present customization of *localenroll.pm* is also needed to define user population terms/titles used at an institution (e.g., Faculty, Staff, Students), and also to define official course categories (e.g., Year, Semester, Department, Number) which can be used as filters when searching the Course Catalog.

## 2 Domain Configuration

### 2.1 Log-in Screen Customization

The login page can be customized for your domain, by:

- uploading custom image files
- changing colors of text, links or backgrounds
- enabling/disabling display of specific links

Logos displayed in the login page configuration table are scaled down from the full size used in the login-page itself.

The following elements are configurable:

- Header image at the top of the page
- Main Logo centered in the upper part of the main panel
- Domain logo in the lower left corner of the main panel
- Header above the login panel - can also be set to use text ("Log in") instead of an image.
- Background colors for the page itself, the main panel, and the left (side) panel.
- Text color used for text on the page
- Enable/disable display of three links:
  - Course Catalog, for a catalog of courses
  - Admin E-mail, for the e-mail address of the administrator
  - New User, for users to create their own accounts
- Default colors for links in the page, depending on status: either active, visited or default (if neither apply).

## 2.2 Setting E-mail Addresses for Administrators and Support

LON-CAPA will send automatic e-mail to administrators/support staff under certain circumstances. The contact information data table can be used to provide e-mail addresses for receipt of these e-mails and to configure which types of e-mail should be sent to each address.

The types of e-mail are:

- Error reports - whenever a server encounters a 500 error (Internal Server Error), Apache will handle that event by displaying an error report form which the affected user can complete and submit. The submission, which contains session information besides any information provided by the user, will be sent as an e-mail.
- Package update alerts - the CHECKRPMS script run every other day will generate e-mail if it detects that package updates are needed.

- Helpdesk requests - clicking the Help link displayed at the right side of the inline navigation bar at the top of a LON-CAPA page (unless the Remote Control is active) will display a Help Menu which includes an "Ask helpdesk" link. The "Ask helpdesk" link provides access to a web form which a user will complete and submit to request LON-CAPA support. The submission, which contains information about the user's browser, besides information provided by the user, will be sent as an e-mail.

Definition of the default Admin email address and the default Support E-mail address saved from the "Contact Information" screen supercede any definitions made when ./UPDATE is run to update to a new version of LON-CAPA. Addresses entered the first time ./UPDATE was run on the primary library server for the domain (i.e., when LON-CAPA was first installed) will continue to apply until the first "Save" of the Contact Information settings has occurred in the domain.

## 2.3 Default Color Schemes

Default color schemes can be set for the domain for four types of user context:

- Student
- Course Coordinator
- Author/Co-author/Assistant Author
- Domain roles (Domain Coordinator)

In each case the following can be set or modified:

- Header image displayed in place of the inline navigation controls when the page is viewed with the Remote Control active.
- Color used for text in the LON-CAPA interface
- Background colors used for the page itself, and the inline navigation (if shown) and the page header below it, and a border to the header (not used).
- Default colors for links in the page, depending on status: either active, visited or default (if neither apply).

Individual users can override any default settings you establish for the domain via the "Change Color Scheme" link in their individual Preferences screen. In the absence of individual preferences, any domain defaults you set will be used whenever users from your domain are using LON-CAPA, regardless of which domain owns the server hosting the session.

## 2.4 Setting Domain Defaults: Authentication, Language and Time zone

Prior to LON-CAPA 2.7, default language and authentication type/argument were defined in the domain's entry in the domain.tab file. Those settings will continue to be used by servers in your domain until you have displayed and saved the Default authentication/language/timezone data table. Once that has been done, whenever values need to be determined for these settings in the domain they will be retrieved from the configuration.db file on the primary library server in your domain, which is where information saved from the "Domain Configuration" data tables is stored. Any information in the domain.tab file will no longer be consulted, except by servers running pre-2.7 versions of LON-CAPA.

Default domain configurations can be assigned for:

- default language used by users in your domain, unless overridden by a user preference
- default authentication type for new users in the domain. You will need to set the default authentication if you intend to allow a user to create a LON-CAPA account if the user successfully authenticated via a central service at your institution (e.g., Kerberos), but is without a LON-CAPA account. The default authentication is also the default offered when Course Coordinators or Authors create new accounts, assuming user creation is permitted in these contexts.
- default timezone - this will be the timezone used when showing any times in your domain, unless overridden at a course level, by a course-wide timezone. The timezones available are mostly in the form Continent/City, although for the USA there are some in the form America/State/City as well as EST5EDT, CST6CDT, MST7MDT, PST8PDT and HST (for Eastern, Central, Mountain, Pacific and Hawaii Timezones, which adjust for daylight savings as appropriate). If no default timezone is set times will be displayed according to the timezone of the server hosting the user's LON-CAPA session.

## 2.5 Setting Default Quotas for User Portfolios

Each user in your domain will receive an individual portfolio space to which files may be uploaded. Students can submit items from their portfolio to meet the requirements of assignments in their courses. The default quotas you set for users in your domain will be overridden by any quota you set for individual users.

Default quotas (in Mb) can be set to vary by institutional affiliation, as defined for your domain (e.g., Faculty, Adjunct, Staff, Student). If a user is affiliated with more than one group, whichever default quota is largest for the different groups is the one which applies. Institutional types need to be defined in a customized version of `&inst.usertypes()` in the `localenroll.pm` module installed on the primary server in your domain. If no types have been defined, then a single default quota will apply for all users from the domain.

## 2.6 Identity Management: Searching for Users in an Institutional Directory

LON-CAPA provides a mechanism to search for users by complete username, last name, or last name,first name (or fragments of each). Accounts within the LON-CAPA domain itself

can be searched, or alternatively, an institutional directory may be searched, if you are able to implement a conduit to directory information to support such searches. This will be done by customizing the `&getuserinfo()` routine in `localenroll.pm`. This routine has a dual role in LON-CAPA. Not only will it support searches where the user is not exactly known, but it can also be used to retrieve institutional records for a specific username or student/employee ID. This latter mode of use is appropriate when adding a new user to LON-CAPA.

Settings for directory searches are:

- Set directory searches as available or unavailable in the domain
- Set whether users from other LON-CAPA domains can use the institutional directory search to search for users. Note: this only applies to institutional directory data, users with privileges to add users to a course will always be able to search the LON-CAPA user database for other domains.
- For searches by users within your domain, you can limit the users who can use directory search based on institutional status
- Set which types of search method - username, last name or last name,first name are allowed for institutional searches
- Set which types of search latitude - exact match, begins with or contains - are allowed

In the case of a "contains" type search at least three characters must be entered by the user as the search term. Searches are case insensitive.

## 2.7 Identity Management: Creating New Users

Identity management in a LON-CAPA domain is dependent on settings made for user creation and user modification. Of particular concern is the potential for assignment of usernames in a format used by your institution when the username does not yet exist. In such a case, authentication is likely to be set to be "internal", and should a real user be created in the future, and be enrolled in a course by auto-enrollment, the user would either be unable to authenticate (using LON-CAPA log-in page), or would be authenticated by SSO, and have access to the original user's roles and associated information.

It is important therefore to establish format rules for new usernames so the only users created with institutional-type usernames are the real users themselves with the appropriate authentication type (Kerberos or localauth). Even without format rules, the Domain Coordinator can set who can create new users, and the authentication types that may be set in different context.

The domain-wide options available for user creation are:

- Activate/deactivate operation of format rule(s) for usernames
- Activate/deactivate operation of format rule(s) for student/employee IDs
- Activate/deactivate operation of format rule(s) which prohibit self-created accounts using certain types of e-mail address as the username.
- Control which types of username (official or non-official) may be used when creating new users in course or author context

- Control which types of user may create their own accounts in LON-CAPA
- Control which types of authentication may be used when assigning authentication to new users in author, course or domain context

The format rules themselves are defined by customizing the following routines in `localenroll.pm`:

- usernames: `&username_rules()` and `&username_check()`
- IDs: `&id_rules()` and `&id_check()`
- self-created accounts: `&selfcreate_rules()` and `&selfcreate_check()`

The first two of these - username and ID check, when enforced, require that if a username and/or ID of the activated formats is to be used in LON-CAPA, they must exist in the institutional directory. If they exist, the corresponding user information (first name, middle name, last name, e-mail address) will be used when creating the new user account. If they do not exist, account creation will not occur.

The third one operates in the opposite manner - if a user attempts to self-create an account employing a username with an e-mail address in a format which matches the rule, the action does not proceed, and the user is directed to create an account with the corresponding institutional log-in. In this case account creation can only occur once the user has authenticated using that login.

## 2.8 Identity Management: Modifying Existing Users

Configuring settings which apply to modification of existing user information (names, e-mail address, student/employee ID) forms a part of LON-CAPA identity management. Authors and Course Coordinators both have access to "Manage Users" which permits them to assign roles to users appropriate to the context. In addition to the ability to assign roles, the ability to modify existing user information may be conferred in this context depending on the target user's role(s). The types of user information which are modifiable in the different contexts is also configurable.

If you have chosen to permit users to self-create their accounts, you can also set which fields in their user records they may set, in cases where the corresponding fields retrieved from the institutional directory are blank. Users may receive different settings depending on their institutional status(es). Institutional status types available are the ones defined for the domain in a customized version of the `&inst_usertypes()` routine in the `localenroll.pm` module installed on the primary server in your domain.

## 2.9 Identity Management: Automated Updates of User Information

An auto-update, run as a regular process, can update user information stored in LON-CAPA for all users in a domain, for whom institutional directory information is available. Which user records are updated can be controlled by institutional status (e.g., Faculty, Staff, Student etc.). If a user is affiliated with more than group, then the attributes which can be updated will be the cumulative set for the different groups to which the user belongs.



If users are not affiliated with any institutional group, they can be accommodated within the default "Other users" group which is provided automatically. If no status types are defined for your domain, this default group is entitled "All users".

Settings for auto-update are:

- Set auto-enrollment as active or inactive in the domain.
- Set whether user information changes should propagate to data stored in classlist database files for the separate courses in which the user has an active student role.
- Set which of the following attributes: first name, middle name, last name, generation, e-mail address, student/employee ID should be updated within LON-CAPA if a different version to the one currently stored is retrieved from the institutional directory.

In order for Autoupdate to work, the `&allusers_info()` routine in `localenroll.pm` needs to be customized and a conduit established to institutional data. In addition, if you wish to differentiate between institutional user types in your LON-CAPA domain the `&inst_usertypes()` routine in `localenroll.pm` will need to be customized to correspond with the types used at your institution. These types are then used to populate the "User population" column in each of the "Updateable user information" row(s) in the Auto-update data table in "Domain Configuration".

Warnings will be written to the Auto-update log file found in `/home/httpd/perl/logs` if a possible username change is detected. Although the username is the unique identifier in LON-CAPA, the student/employee ID operates as an additional, mostly unique identifier. At present LON-CAPA does not support username changes. For users who switch username (assuming institutional authentication will no longer authenticate the user's old username) the recommendation is to convert the authentication type in LON-CAPA for the user to "internal", set an initial password, make sure that permanent e-mail is set for the user, then e-mail the user and ask them to use the "[Forgot password?](#)" link on the log-in page to change the password to something secure.

## 2.10 Course Catalogs

LON-CAPA courses can be both self-cataloging, and also manually cataloged.

Self-cataloging uses the institutional course code assigned to the course when it is first created, or when the course is modified by a Domain Coordinator via "Modify a course". If a course has no institutional code it will not appear in the category: Official courses (with institutional codes).

A hierarchy of categories and sub-categories can be defined which are independent of institutional course codes. These categories might be used to catalog courses in the domain to which the "official courses" designation does not apply, or they might be used to provide alternative ways of cataloging official courses.

Besides definition of the hierarchy of categories and sub-categories, the "cataloging of courses" screen provides two options to be set for the domain by a Domain Coordinator, which control:

- Who may hide a course from the course catalog, if the course would ordinarily appear by virtue of having an institutional course code, or having been assigned to a custom category/sub-category.

- Who may assign a custom category/subcategory to a course

In both cases, the choice is between a Domain Coordinator and a Course Coordinator. For the former, hiding of courses and assignment of categories will be via "Modify Course", while for the latter these operations will be via "Modify Parameters" ("Set Course Environment" option in sub-menu).

Definition of custom categories is by the Domain Coordinator. The "Cataloging of courses" interface allows custom categories and sub-categories to be defined and reordered. There is one category listed at the top level in the hierarchy which behaves differently to the others - the category: "Official courses (with institutional codes)". This is the category which is used for self-cataloged courses: the option is to either display or not display.

Although sub-categories can not be defined via this interface for this "system" category (unlike the other "custom" categories), customization of two routines in `localenroll.pm` - `&instcode_defaults()` and `&instcode_format()` are used to automatically generate linked select boxes which will be displayed when the course catalog is shown to allow users to include limits to their searches for official courses. If these routines have not been customized, the catalog for official courses will display all courses with institutional codes, which have not been specifically hidden.

When `&instcode_format()` has been customized it will populate perl structures (hashes and arrays) which LON-CAPA will use to generate the Javascript code embedded in the course catalog page which is used in the functioning of the linked select boxes. The contents of the hashes and arrays are determined from the complete list of institutional course codes used in the domain. For example at MSU, the following linked select boxes are displayed for the official courses catalog: Year, Semester, Department, Number.

The course catalog is useful in domains where Course Coordinators have opted to allow self-enrollment in their courses. In such cases, students can include a flag to only display courses allowing self enrollment when they display courses from the catalog. Course Coordinators will be advised, when enabling self-enrollment, if a courses is currently unlisted in the course catalog (and therefore difficult for students to locate), and the action to take to rectify the reason why there is no listing, which could be because:

- the course is hidden
- the course has no institutional code
- the course has not been assigned to any custom categories
- the course has an institutional code, but display of the official courses catalog is disabled

## 2.11 Automated Enrollment in Official Courses

If your institution can provide access to roster information for courses using LON-CAPA then your domain can offer automated enrollment once `localenroll.pm` has been customized on each of the library servers in your domain which serves as a home server for one or more courses. The required customization is the creation of connections to rosters for institutional course sections providing enrollment to each LON-CAPA course. These connections can involve queries of database tables, in real time, or can involve retrieval from a data source which is only updated periodically.

There are two configuration options:

- Set auto-enrollment as active or inactive in the domain.
- Set the username:domain used in the notification messages sent when changes in enrollment occur as a result of auto-enrollment updates. By setting these to a specific user (you might create one for this purpose), you can view all auto-enrollment change messages for the entire domain by viewing the contents of the sent messages folder for that user.

Auto-enrollment settings for each course consist of items set by a Domain Coordinator within the "Modify Course" area, and those items set in a course context from the "[Automated Enrollment Management](#)" link in the "Manage Users" menu. This link is only displayed if auto-enrollment has been set to be active in the domain.

The items which must be set by a Domain Coordinator include:

- institutional code (used in mapping institutional rosters to LON-CAPA courses)
- default authentication
- course owner

Your institution may have policies in place to control who may have access to student information contained within course rosters. In such cases, assignment of an appropriate course owner in LON-CAPA may facilitate access to institutional rosters. When a course is first created the initial Course Coordinator chosen will be identified as the course owner. If, instead a course is created using an uploaded XML course description file, the XML file will include tags to define the username and domain of the course owner.

Settings which control auto-enrollment which are modifiable by a Course Coordinator are described in the ?? help page.

The auto-enrollment process will update user information (name, e-mail address, studentID etc.) for any student who is being added to the course, but will not change it in other case (i.e., drops, section switches) should there be a difference between the values in the institutional roster and the values in the LON-CAPA classlist. To change user information for students already in the course, the Auto-update process must be run (see below).

Warnings will be written to the Auto-enrollment log file found in /home/httpd/perl/logs if a possible username change is detected. Although the username is the unique identifier in LON-CAPA, the studentID operates as an additional, mostly unique identifier. The same studentID may not be assigned to more than one user - if an existing studentID is assigned to a different user, the change will not occur, unless forced (there's a checkbox for that). A blank studentID can be assigned, and in this case multiple users can share the same studentID. StudentIDs are used for LON-CAPA grading of scantron sheets, and are also used to detect changes in username by the auto-enrollment and auto-update processes.

At present LON-CAPA does not support username changes, although this functionality will be supported in the future. In the meantime, what you must do for users who switch username mid-semester (assuming institutional authentication will no longer authenticate the user's old username) is to convert the authentication type in LON-CAPA for the user to "internal", set an initial password, make sure that permanent e-mail is set for the user, then e-mail the user the initial password, and ask them to use the "[Forgot password?](#)" link on the log-in page to change the password to something secure.

## 2.12 Scantron Data Formats

Where scantron exams are used in a course, the format of data in the file generated from the processing of bubbled-in scantron sheets may vary between institutions. The format definitions available when performing scantron grading in LON-CAPA were originally listed in the scantronformat.tab file, stored in /home/httpd/lonTabs, which might have been modified locally on each server.

Starting with LON-CAPA 2.7, scantron format information is read from either a custom.tab file, or a default.tab file both of which belong to the special domain configuration user (\$dom-domainconfig, where \$dom is the name of the domain) and which are automatically published into resource space.

For LON-CAPA installations older than 2.7, when the primary library server for the domain has been updated, a Domain Coordinator should display the "Scantron format file" configuration page via "Domain Configuration". The first time this page is displayed, a default.tab (a copy of the standard LON-CAPA scantronformat.tab file), and a custom.tab (if the scantronform.tab file currently on the server differs from the standard file) will be copied and published. Thereafter any changes to scantron format files to be used for grading scantron exams in courses from the domain will be made via the Domain Configuration menu. Any scantronform.tab files in /home/httpd/lonTab directories on servers in the domain will no longer be used.

The settings available via "Scantron format file" support upload of a new custom file, or deletion of an existing custom file (in which case grading will default to use of the default.tab file). An uploaded scantron format file contains one or more lines of colon-separated values for the parameters in the following order:

name:description:CODEtype:CODEstart:CODElength:IDstart:IDlength:Qstart:Qlength:Qoff:Qon:PaperID:PaperIDlength:FirstName:FirstNa

- name is the internal identifier used within LON-CAPA
- description is the text displayed for each option in the the "Format of data file" drop-down in the Scantron grading screen. The user will choose the appropriate format for the scantron file currently being used for scantron grading.
- CODEtype can be either 'none' 'letter' 'number'
- Qon can be either:
  - the symbol that says a bubble has been selected, or
  - 'letter' (for when the selected letter appears), or
  - 'number' for when a number indicating the selected letter appears

As an example, below are four different format lines: the first two were used at MSU prior to 2006; the last two have been used since then.

- msunocode:MSU without any CODE:none:0:0:57:9:77:10: :1:5:5:51:1:41:10
- msucode:MSU with CODE in separate location:letter:69:6:57:9:77:10: :1:5:5:51:1:41:10
- msucodelet:MSU with CODE in separate location (letter format):-1:69:6:57:9:77:1: :letter:5:5:51:1:41:10
- msucodenum:MSU with CODE in separate location (number format):-1:69:6:57:9:77:1: :number:5:5:51:1:41:10

## 3 Domain Management

### 3.1 Creating Domain Coordinators

When LON-CAPA was first installed for a domain, an initial library server will have been set up, and the command

`perl loncom/build/make_domain_coordinator.pl` will have been run (as root) in the `loncapa-X.Y.Z` directory created when the LON-CAPA tarball was uncompressed.

This command will have created a new Linux user, who will be filesystem authenticated when logging into LON-CAPA, and who will have been assigned the Domain Coordinator role.

A Domain Coordinator can add users to the domain and assign any role in the domain with the exception of the Domain Coordinator role. To assign the Domain Coordinator role to other users, someone with root privileges on a library server in the domain can:

- run `perl make_domain_coordinator.pl` to create a new user (filesystem authenticated). `make_domain_coordinator.pl` will fail if:
  - the user already has a Linux account, or
  - the username is already in use for an existing LON-CAPA user in the domain.
- assign the Domain Coordinator role to an existing LON-CAPA user by running the following command in the `loncapa-X.Y.Z` directory
  - `perl loncom/build/add_domain_coordinator_privilege.pl [USERNAME:DOMAIN] [DCDOMAIN]`
    - \* where `USERNAME:DOMAIN` are the username and domain of an existing user, who is to be granted Domain Coordinator privileges,
    - \* `DCDOMAIN` is the domain to be coordinated. Note: `DCDOMAIN` must be a domain for which the server where the command is run is a library server.

### 3.2 Scheduling of Scripts Run Periodically

When LON-CAPA is installed a file named `loncapa` is written to `/etc/cron.d`. The frequency and timing of execution of scripts included in this `loncapa` crontab file can be modified to suit the needs of your domain. The scripts, which are all run as the user 'www', are as follows:

- `/home/httpd/perl/loncron` run daily at 5.10 am This updates the list of servers in the LON-CAPA cluster to which your domain belongs. All servers in this list should be contactable, and will have the ability to host user sessions for users in the domain, as well as being available, if designated as library servers, to return responses to remote searches for files housed there which have been contributed to the LON-CAPA content repository. Connections to all servers are re-evaluated by `loncron`, in case some machines had become unavailable within the last 24 hours, and had therefore been flagged as temporarily offline.

- */usr/local/loncapa/bin/CHECKRPMS* runs every other day at 3.10 am. This file automates the process of checking for available updates to LON-CAPA systems. The *distprobe* script, installed as a part of LON-CAPA, is used to determine the Linux distribution installed on the server, which in turn dictates which utility (*yum*, *up2date*, *you* or *rug*) is called to perform the package check.
- */home/httpd/perl/searchcat.pl* run every other day at 1.10 am traverses the LON-CAPA resource directory in a domain and gathers metadata which are entered into a SQL database. The script will repopulate and refresh the metadata database used for the searching the resource catalog. The script also refreshes and repopulates database tables used to store metadata for publicly accessible portfolio files, and user information needed for user searches in a LON-CAPA domain.
- */home/httpd/perl/cleanup\_database.pl* run daily at 2.13 am drops tables from the LON-CAPA MySQL database if their comment is 'temporary' and they have not been modified in a given time (default is 2 days).
- */home/httpd/perl/cleanup\_file\_caches.pl* run daily at 1.05 am removes temporary files from the LON-CAPA print spool, the multidownload zip spool, and userfiles cache.
- */home/httpd/perl/Autoenroll.pl* run daily at 1.30 am updates classlists for any LON-CAPA courses for which auto-enrollment is active, if enabled in the domain. A conduit needs to have been established to institutional course roster information.
- */home/httpd/perl/Autoupdate.pl* run daily at 3.30 am can reconcile first name, last name etc. information stored in LON-CAPA with authoritative data available from an institutional directory. A conduit needs to have been established to the institutional data source.

## 4 Integration with Institutional Systems

### 4.1 Institutional Authentication

When a user is assigned an authentication type of “Local authentication” , the perl module */home/httpd/lib/perl/localauth.pm* will be used to evaluate the user’s credentials. The documentation included in the stub provided with a LON-CAPA installation describes the basic operation of *localauth.pm*

The *localauth* routine receives four arguments (in the order: two required, one optional, another required).

1. the username the user types in.
2. the password the user typed in.
3. optional information stored when the authentication mechanism was specified for the user (“Local authentication with argument: ....“)
4. the domain the user typed in.

The routine will return 1 if the user is authenticated and 0 otherwise, and it can optionally return a negative value for an error condition. This negative value will be logged along with the username used in the failed authentication which resulted in the error condition.

A common use of `localauth.pm` is to connect with an LDAP service.

```
package localauth;
use strict;
use Net::LDAP;
use Net::LDAPS;
sub localauth {
    my ($username,$password) = @_;
    my $ldap_host_name = ''; # insert the host name of your ldap
server, e.g., ldap.msu.edu
    my $ldap_ca_file_name = ''; # insert the ldap certificate
filename - include absolute path
    # certificate is required if you wish to encrypt the password.
    # e.g., /home/http/perl/lib/local/ldap.certificate
    my $ldap_search_base = ''; # ldap search base, this might
be set to 'o=msu.edu'.
    my $ldap = Net::LDAPS->new(
        $ldap_host_name,
        verify => 'require', # 'require' -> a certificate
is needed, -> 'none' if no certificate used
        cafile => $ldap_ca_file_name,
    );
    if (!(defined($ldap))) {
        return (0);
    }
    $ldap->bind;
    my $search_string = '(uid='.$username.')';
    my $mesg = $ldap->search (
        base => $ldap_search_base,
        filter => $search_string,
        attrs => ['dn'] ,
    );
    if ($mesg->code) {
        $ldap->unbind;
        $ldap->disconnect;
        return (0);
    }
    my @entries = $mesg->all_entries;
    if (@entries > 0) {
        $ldap->unbind;
        $ldap->disconnect;
        return (0);
    }
}
```

```

    }
    $mesg = $ldap->bind (
        dn => $entries[0]->dn,
        password => $password,
    );
    $ldap->unbind;
    $ldap->disconnect;
    if ($mesg->code) {
        return (0)
    }
    return (1);
}
1;

```

## 4.2 Institutional User Categories/Affiliations

Users in a domain can be assigned one or more institutional affiliations by the Autoupdate process which reconciles user information in LON-CAPA with institutional directory information. User type (or affiliation) can determine such things as (a) default portfolio quota, (b) the types of user information which may be updated in different contexts, (c) whether a user can self-enroll in a course. The possible institutional types in a domain are defined by *inst\_usertypes()*. Examples of institutional types might be: Faculty, Adjunct, Staff, Student etc. In addition to any types defined in *inst\_usertypes()*, a type “other” will also be available for assignment to users who do not fall in any of the recognized categories of user. In the absence of any defined user categories, the type “other” applies to all users from a domain.

### **inst\_usertypes**

The routine accepts three arguments:

1. \$dom - domain
2. \$usertypes - reference to hash which will contain key = value, where keys are institution affiliation types (e.g., Faculty, Student etc.) and values are titles (e.g., Faculty/Academic Staff)
3. \$order - reference to array which will contain the order in which institutional types should be shown when displaying data tables (e.g., default quotas or updateable user fields (see Domain Configuration menu)

The routine returns 'ok' if no errors occurred.

At MSU there are six different categories of users.

```

sub inst_usertypes {
    my ($dom,$usertypes,$order) = @_;
    my $outcome = 'ok';
    %{$usertypes} = (

```



```

        Faculty => 'Faculty/Academic Staff',
        Staff => 'Support Staff',
        Student => 'Student',
        Assistant => 'Assistant',
        StaffAff => 'Affiliate',
        StuAff => 'Student Affiliate'
    );
    @{$order}=('Faculty','Staff','Student','Assistant','StaffAff','StuAff');
    return $outcome;
}

```

### 4.3 Format Rule Definitions and Checks: Usernames and IDs

Format restrictions for usernames and student/employeeIDs for an institution, and formats which may *not* be used for e-mail addresses used as usernames when users self-create accounts are defined in three subroutines in `localenroll.pm`: `username_rules()`, `id_rules()`, and `selfcreate_rules()`. The three routines accept a similar set of arguments, and return 'ok' in each case, if no error occurred.

**username\_rules** - Incoming data: three arguments

1. `$dom` - domain

2. `$ruleshash` - reference to hash containing rules (a hash of a hash)

keys of top level hash are short names (e.g., `netid`, `noncredit`); for each key, value is a hash.

- `desc =i` long name for rule
- `rule =i` description of rule
- `authtype =i` (`krb5`, `krb4`, `int`, or `loc`) authentication type for rule
- `authparm =i` authentication parameter for rule
- `authparmfixed =i` 1 if `authparm` used when creating user for rule must be `authparm`
- `authmsg =i` Message to display describing authentication to use for this rule

3. `$rulesorder` - reference to array containing rule names in order to be displayed

At MSU, a NetID consists of eight characters or less, and will be authenticated by Kerberos (version 5) in the MSU.EDU realm. The rule itself is defined in `username_rules()`, and the code which checks for compliance is in `username_check()`:

```

sub username_rules {
    my ($dom,$ruleshash,$rulesorder) = @_;
    %{$ruleshash} = (
        netid => {

```

```

        name => 'MSU NetID',
        desc => 'Eight characters or less',
        authtype => 'krb5',
        authparm => 'MSU.EDU',
        authparmfixed => '',
        authmsg => 'A new user with a username which
        matches a valid MSU NetID will log-in using the
        MSU Net ID and MSU Net password.',
    }
);
@{$rulesorder} = ('netid');
return 'ok';
}

```

**id\_rules** - Incoming data: three arguments

1. \$dom - domain
2. \$ruleshash - reference to hash containing rules (a hash of a hash) keys of top level hash are short names (e.g., studentID, employeeID); for each key, value is a hash
  - desc =<sub>i</sub> long name for rule
  - rule =<sub>i</sub> description of rule
3. \$rulesorder - reference to array containing rule names in order to be displayed

At MSU, studentIDs and employee IDs are eight digits prefaced by A or Z. The rule itself is defined in *id\_rules()*, and the code which checks for compliance is in *id\_check()*:

```

sub id_rules {
    my ($dom,$ruleshash,$rulesorder) = @_;
    %{$ruleshash} = (
        studentID => {
            name => 'MSU student PID',
            desc => 'Letter A or a, followed by eight digits',
        },
        facstaffID => {
            name => 'MSU faculty/staff ID',
            desc => 'Letter Z or z, followed by eight digits',
        },
    );
    @{$rulesorder} = ('studentID','facstaffID');
    return 'ok';
}

```

**selfcreate\_rules** - Incoming data: three arguments

1. \$dom - domain
2. \$ruleshash - reference to hash containing rules (a hash of a hash)  
keys of top level hash are short names (e.g., msuemail); for each key, value is a hash
  - desc = long name for rule
  - rule = description of rule
3. \$rulesorder - reference to array containing rule names in order to be displayed

At MSU all users receive a Net ID (e.g., *sparty*), and a corresponding e-mail account: *sparty@msu.edu*. So, at MSU the rules for e-mail addresses to be used as LON-CAPA usernames prohibit e-mails such as *sparty@msu.edu*. In such cases, the user should log-in with the sparty Net ID/password and request account creation for the username: *sparty*. The rule itself is defined in *selfcreate\_rules()*, and the code which checks for compliance is in *selfcreate\_check()*:

```
sub selfcreate_rules {
    my ($dom,$ruleshash,$rulesorder) = @_;
    %{$ruleshash} = (
        msuemail => {
            name => 'MSU e-mail address ',
            desc => 'netid@msu.edu',
        },
    );
    @{$rulesorder} = ('msuemail');
    return 'ok';
}
```

The corresponding routines which check for compliance with rules enabled via Domain Configuration-> User Creation are *username\_check()*, *id\_check()*, and *selfcreate\_check()*. The three routines accept a similar set of four arguments, and return 'ok' in each case, if no error occurred.

1. \$dom - domain (scalar)
2. \$uname (username\_check()), \$id (id\_check()) or \$selfcreatenam (selfcreate\_check())  
- proposed username, id or self-created username being compared against rules (scalar)
3. \$to\_check (reference to array of rule names to check)
4. \$resultshash (reference to hash of results) hash of results for rule checked  
keys are rule names - values are: 1 or 0 (for matched or unmatched)

The routines used for checking rule compliance at MSU are as follows:

### **username\_check**

```
sub username_check {
```

```

my ($dom,$uname,$to_check,$resultshash) = @_;
my $outcome;
if (ref($to_check) eq 'ARRAY') {
    foreach my $item (@{$to_check}) {
        if ($item eq 'netid') {
            if ($uname =~ /\w{2,8}$/) {
                $resultshash->{$item} = 1;
            } else {
                $resultshash->{$item} = 0;
            }
        }
    }
    $outcome = 'ok';
}
return $outcome;
}

```

### id\_check

```

sub id_check {
    my ($dom,$id,$to_check,$resultshash) = @_;
    my $outcome;
    if (ref($to_check) eq 'ARRAY') {
        foreach my $item (@{$to_check}) {
            if ($item eq 'facstaffID') {
                if ($id =~ /\z\d{8}$/i) {
                    $resultshash->{$item} = 1;
                } else {
                    $resultshash->{$item} = 0;
                }
            } elsif ($item eq 'studentID') {
                if ($id =~ /\^a\d{8}$/i) {
                    $resultshash->{$item} = 1;
                } else {
                    $resultshash->{$item} = 0;
                }
            }
        }
        $outcome = 'ok';
    }
    return $outcome;
}

```

### selfcreate\_check

```

sub selfcreate_check {
    my ($dom,$selfcreatename,$to_check,$resultshash) = @_;
    my $outcome;
    if (ref($to_check) eq 'ARRAY') {
        foreach my $item (@{$to_check}) {
            if ($item eq 'msuemail') {
                if ($selfcreatename =~ /\w{2,8}@msu\.edu$/)
                {
                    $resultshash->{$item} = 1;
                } else {
                    $resultshash->{$item} = 0;
                }
            }
        }
        $outcome = 'ok';
    }
    return $outcome;
}

```

## 4.4 Institutional Directory Information

Two subroutines exist in `localenroll.pm` to provide a connection between institutional directory data (e.g., user information from LDAP) and LON-CAPA. The first is `get_userinfo()` which can operate in two modes.: (a) it can be used to provide first name, last name, e-mail address, student/employee ID etc., for a specified username, e.g., for a new user being created in LON-CAPA, and (b) it can be used to retrieve user information for multiple users from an institutional directory searches when (for example) a course coordinator is adding a new user directly to a course. At MSU the routine which actually queries institutional data sources is itself called by `get_userinfo()`. This was done so that the same underlying routine can also be used by the second of the two subroutines: `allusers_info()` which is called by `Autoupdate.pl` (a script which can be run periodically to reconcile user information in LON-CAPA with institutional directory data for all users).

### **get\_userinfo**

Four required arguments and additional optional arguments

Two modes of operation:

1. Retrieves institutional data for a single user either by username, if `$uname` is included as second argument,  
or by ID if `$id` is included as a third argument. Either second or third arguments must be provided; seventh, eighth and ninth args will be undefined.
2. Retrieves institutional user data from search of an institutional directory based on a search. seventh and eighth args are required; ninth is optional. second and third will be undefined.

Arguments:

1. \$dom - domain
2. \$uname - username of user
3. \$id - student/faculty ID of user
4. \$instusers - reference to hash which will contain info for user as key = value; keys will be one or all of: lastname, firstname, middlename, generation, id, inststatus - institutional status (e.g., faculty,staff,student).  
Values are all scalars except inststatus, which is an array.
5. \$instids - reference to hash which will contain ID numbers - keys will be unique IDs (student or faculty/staff ID)  
values will be either: scalar (username) or an array if a single ID matches multiple usernames.
6. \$types - optional reference to array which contains institutional types to check.
7. \$srchby - optional if \$uname or \$id defined, otherwise required.  
Allowed values include: 1. lastfirst, 2. last, 3. uname corresponding to searches by 1. lastname,firstname; 2. lastname; 3. username
8. \$srchterm - optional if \$uname or \$id defined, otherwise required - String to search for.
9. \$srchtype - optional. Allowed values: contains, begins (defaults to exact match otherwise).

Returns 'ok' if no error occurred. Side effects - populates the \$instusers and \$instids refs to hashes with information for specified username, or specified id, if fifth argument provided, from all available, or specified (e.g., faculty only) institutional datafeeds, if sixth argument provided.

At MSU six separate MS-SQL database tables are queried, with each table corresponding to a specific institutional type. A routine is called to connect to the database. and the actual queries are handled by a separate routine - *query\_user\_tables()*.

```
sub get_userinfo {
    my ($dom,$uname,$id,$instusers,$instids,$types,
        $srchby,$srchterm,$srchtype) = @_;
    my $outcome;
        my @srchtables;
    my %tables = (
        Faculty => 'FACULTY_VU',
        Staff => 'STAFF_VU',
        Student => 'STUDENT',
        Assistant => 'ASSISTANT',
        StaffAff => 'AFFILIATE',
        StuAff => 'STUDENT_AFFILIATE'
```

```

);
my ($dbh,$dbflag) = &connect_DB('HR');
foreach my $type (@{$types}) {
    if (exists($tables{$type})) {
        push(@srchtables,$tables{$type});
    }
}
if (@srchtables == 0) {
    foreach my $type (keys(%tables)) {
        push(@srchtables,$tables{$type});
    }
}
if ($srchby eq '' && $srchterm eq '') {
    if ($uname ne '') {
        $srchby = 'uname';
        $srchterm = $uname;
    } elsif ($id ne '') {
        $srchby = 'id';
        $srchterm = $id;
    }
}
if ($srchterm ne '') {
    $outcome = &query_user_tables($dbflag,$dbh,\@srchtables,$instusers,$instids,
        $srchby,$srchterm,$srchtype,$types);
}
if ($dbflag) {
    &disconnect_DB($dbh);
}
return $outcome;
}

```

Although `query_user_tables()` is not a subroutine included as a stub in the standard `localenroll.pm`, it is included below to show how the database queries are implemented at MSU.

```

sub query_user_tables {
    my ($dbflag,$dbh,$srchtables,$instusers,$instids,$srchby,$srchterm,$srchtype,
        $condition,$ldapfilter) = @_;
    my ($outcome,$condition,%multipids,$ldapfilter);
    if ($srchby eq 'uname') {
        if ($srchterm =~ /\w{2,8}$/) {
            if ($srchtype eq 'contains') {
                $condition = "WHERE MSUNetID LIKE '%$srchterm%'";
                $ldapfilter = '(uid=*. $srchterm.*)';
            } elsif ($srchtype eq 'begins') {

```

```

        $condition = "WHERE MSUNetID LIKE '$srchterm%'";
        $ldapfilter = '(uid='. $srchterm. '*)';
    } else {
        $condition = "WHERE MSUNetID = '$srchterm'";
        $ldapfilter = '(uid='. $srchterm. ')';
    }
}
} elseif ($srchby eq 'lastname') {
    if ($srchterm =~ /[A-Za-z\-\.\'\s]+/) {
        if ($srchtype eq 'contains') {
            if ($dbflag) {
                my $quoted_last = $dbh->quote('%'. $srchterm. '%');
                $condition = "WHERE LastName LIKE $quoted_last";
            }
            $ldapfilter = '(sn=*'. $srchterm. '*)';
        } elseif ($srchtype eq 'begins') {
            if ($dbflag) {
                my $quoted_last = $dbh->quote($srchterm. '%');
                $condition = "WHERE LastName LIKE $quoted_last";
            }
            $ldapfilter = '(sn='. $srchterm. '*)';
        } else {
            if ($dbflag) {
                my $quoted_last = $dbh->quote($srchterm);
                $condition = "WHERE LastName = $quoted_last";
            }
            $ldapfilter = '(sn='. $srchterm. ')';
        }
    }
}
} elseif ($srchby eq 'lastfirst') {
    my ($srchlast, $srchfirst) = split(/,/, $srchterm);
    $srchlast =~ s/\s+$/;
    $srchfirst =~ s/^\s+//;
    if (($srchlast =~ /[A-Za-z\-\.\'\s]+/) && ($srchfirst
    =~ /[A-Za-z\-\.\'\s]+/)) {
        my ($quoted_first, $quoted_last);
        if ($srchtype eq 'contains') {
            if ($dbflag) {
                $quoted_last = $dbh->quote('%'. $srchlast. '%');
                $quoted_first = $dbh->quote('%'. $srchfirst. '%');
                $condition = "WHERE ( LastName LIKE $quoted_last
                AND FirstName LIKE $quoted_first )";
            }
            $ldapfilter = '(&(sn='. $srchlast. '*)(givenName='. $srchfirst. '*))';
        } else {
            foreach my $stable (@{$srchtables}) {

```



```

        next if ($srchby && $condition eq '');
        my $statement = "SELECT MSUNetID,Pid,FirstName,LastName,Person_Type
FROM $table $condition";
        my $sth = $dbh->prepare("$statement");
        $sth->execute();
        while ( my($uname,$pid,$first,$last,$type)
= $sth->fetchrow_array ) {
            $pid=lc($pid);
            if (ref($instusers->{$uname}) eq 'HASH')
            {
                if (ref($instusers->{$uname}){'instst
if ($dbflag) {
                    $quoted_last = $dbh->quote($srchterm);
                    $quoted_first = $dbh->quote($srchterm);
                    $condition = "WHERE ( LastName = $quoted_last
AND FirstName = $quoted_first )";
                }
                $ldapfilter = '(&(sn='.$srchlast.')(givenName='.$srchfirst.'))';
            }
        }
    } elsif ($srchby eq 'id') {
        if ($dbflag) {
            if ($srchterm =~ /^[AZ]\d{8}$/) {
                $condition = "WHERE Pid = '$srchterm'";
            }
        }
    }
}
if ($dbflag) {
    foreach my $table (@{$srchtables}) {
        next if ($srchby && $condition eq '');
        my $statement = "SELECT MSUNetID,Pid,FirstName,LastName,Person_Type
FROM $table $condition";
        my $sth = $dbh->prepare("$statement");
        $sth->execute();
        while ( my($uname,$pid,$first,$last,$type)
= $sth->fetchrow_array ) {
            $pid=lc($pid);
            if (ref($instusers->{$uname}) eq 'HASH')
            {
                if (ref($instusers->{$uname}){'inststatus'})
eq 'ARRAY') {
                    if (!grep(/^$type$/,@{$instusers->{$uname}{'inststatus'}}))
                    {
                        push(@{$instusers->{$uname}{'inststatus'}},$type);
                    }
                }
            }
        }
    }
}

```

```

        if ($pid ne $instusers->{$uname}{'id'})
        {
            if ($instusers->{$uname}{'id'} =~ /^A\d{8}$/)
            {
                if ($pid =~ /^A\d{8}$/) {
                    if (ref($multipids{$uname}) eq 'ARRAY')
                    {
                        if (!grep(/^$pid$/,@{$multipids{$uname}}))
                        {
                            push(@{$multipids{$uname}},$pid);
                        }
                    } else {
                        @{$multipids{$uname}} = ($instusers->{$uname}{'id'},$pid)
                    }
                    $instusers->{$uname}{'id'} = $pid;
                }
            } elseif ($instusers->{$uname}{'id'} =~
/^Z\d{8}$/) {
                if ($pid =~ /^Z\d{8}$/) {
                    if (ref($multipids{$uname}) eq 'ARRAY')
                    {
                        if (!grep(/^$pid$/,@{$multipids{$uname}}))
                        {
                            push(@{$multipids{$uname}},$pid);
                        }
                    } else {
                        @{$multipids{$uname}} = ($instusers->{$uname}{'id'},$pid)
                    }
                } elseif ($pid =~ /^A\d{8}$/) {
                    $instusers->{$uname}{'id'} = $pid;
                }
            }
        }
    } else {
        $instusers->{$uname} = {
            firstname => $first,
            lastname => $last,
            id => $pid,
            permanentemail => $uname.'@msu.edu',

            inststatus => [$type],
        };
    }
    if (defined($instids->{$pid})) {
        if (ref($instids->{$pid}) eq 'ARRAY') {
            if (!grep(/^$uname$/,@{$instids->{$pid}}))
            {

```

```

        push(@{$instids->{$pid}}, $uname);
    }
    } elsif ($instids->{$pid} ne $uname) {
        @{$instids->{$pid}} = ($instids->{$pid}, $uname);
    }
    } else {
        $instids->{$pid} = $uname;
    }
    }
    $outcome = 'ok';
}
}
if ($ldapfilter ne '') {
    my $ldapres = &ldap_search($ldapfilter, $instusers, $types);
    if (!$dbflag) {
        $outcome = $ldapres;
    }
}
return $outcome;
}

```

At MSU, a search of the LDAP directory is used to supplement SQL queries of Faculty, Staff and Student database tables, because there are no student/employee IDs available from MSU's LDAP service. The LDAP search is used to retrieve information about users who have MSUNetIDs (i.e., official usernames from MSU), but are not currently affiliated with any of the institutional user types, so are absent from the six SQL database tables.

```

sub ldap_search {
    my ($ldapfilter, $instusers, $types) = @_;
    my $outcome;
    my $ldap = Net::LDAP->new( 'ldap.msu.edu' );
    if ($ldap) {
        $ldap->bind;
        my $mesg = $ldap->search(
            base => "dc=msu, dc=edu",
            filter => $ldapfilter,
            attrs => ['sn', 'givenName', 'title', 'uid', 'mail', 'employeeType'],
        );
        if ($mesg->code) {
            $ldap->unbind;
            return;
        } else {
            $outcome = 'ok';
        }
        foreach my $entry ($mesg->entries) {

```

```

    my $uname = $entry->get_value('uid');
    next if ($uname eq '');
    my $first = $entry->get_value('givenName');
    my $last = $entry->get_value('sn');
    my $email = $entry->get_value('mail');
    my $type;
    if (($entry->get_value('employeeType') eq 'Faculty')
|| ($entry->get_value('employeeType') eq 'Staff'))
    {
        $type = $entry->get_value('employeeType');
    } elsif ($entry->get_value('title') eq 'Student')
    {
        $type = $entry->get_value('title');
    }
    if (ref($types) eq 'ARRAY') {
        if (@{$types} > 0) {
            if (($type ne '') && !(grep(/^$type$/,@{$types})))

                next if (!grep(/^default$/,@{$types}));
            }
            next if (($type eq '') && (!grep(/^default$/,@{$types})));
        }
    }
    if (ref($instusers->{$uname}) eq 'HASH') {
        if (ref($instusers->{$uname}{'inststatus'})
eq 'ARRAY') {
            if (!grep(/^$type$/,@{$instusers->{$uname}{'inststatus'}}))
            {
                push(@{$instusers->{$uname}{'inststatus'}},$type);
            }
        }
    } else {
        $instusers->{$uname} = {
            firstname => $first,
            lastname => $last,
            id => '',
            permanentemail => $email,
            inststatus => [$type],
        };
    }
    $ldap->unbind;
}
return $outcome;
}

```

**allusers\_info**

Three arguments are required:

1. \$dom - domain
2. \$instusers - reference to hash which will contain hashes, where keys will be usernames and value will be a hash of user information.

Keys in the inner hash will be some or all of: lastname, firstname, middlename, generation, id, inststatus - institutional status (e.g., faculty,staff,student)

Values are all scalars except inststatus, which is an array.

3. \$instids - reference to hash which will contain ID numbers. keys will be unique IDs (student or faculty/staff ID).

Values will be either: scalar (username) or an array if a single ID matches multiple usernames.

Returns 'ok' if no error occurred.

Side effects - populates the \$instusers and \$instids refs to hashes with information for all users from all available institutional datafeeds.

In the MSU case, six SQL database tables are queried via the *query\_user\_tables()* routine described above.

```

sub allusers_info {
    my ($dom,$instusers,$instids) = @_;
    my $outcome;
    my ($dbh,$dbflag) = &connect_DB('HR');
    if ($dbflag) {
        my @srchtables = ('FACULTY_VU','STAFF_VU','STUDENT','AFFILIATE','ASSISTANT');
        &query_user_tables($dbflag,$dbh,\@srchtables,$instusers,$instids);
        $outcome = 'ok';
        &disconnect_DB($dbh);
    }
    return $outcome;
}

```

**4.5 Search Filters for Official Course Categories**

Courses in a domain can be self-cataloging if assigned an institutional code. For this to work two routines need to be customized in *localenroll.pm* - *instcode\_format()* and *instcode\_defaults()*. The first of these is used to split institutional course codes into their constituent parts, and populate some perl data structures with these data, which LON-CAPA can use to generate linked select boxes which users can use to create filters to apply when searching the “official” course catalog. The second routine constructs a regular expression used when searching for courses based on the filter chosen by the user, which will contain fragments of an institutional code.

**instcode\_format**

Six arguments are required:

1. domain (\$dom)
2. reference to hash of institutional course IDs (\$instcodes)
3. reference to hash of codes (\$codes)
4. reference to array of titles (\$codetitles), e.g., @{\$codetitles} = ("year","semester","department","number");
5. reference to hash of abbreviations used in categories, (\$cat\_titles) e.g.,

```
%{ $$cat_titles{'Semester'}} = (
    fs =_ 'Fall',
    ss =_ 'Spring',
    us =_ 'Summer');
```

6. reference to hash of arrays specifying sort order used in category titles (\$cat\_order), e.g., @{\$\$cat\_order{'Semester'}} = ('ss','us','fs');

The routine returns 'ok' if no errors occurred.

At MSU, "fs03nop590," is an example of an institutional course code; including the following entry in the instcodes hash passed in by reference - \$\$instcodes{'43551dedcd43febmsull'} = 'fs03nop590' would cause the \$codes perl data structure to be populated.

fs03nop590 would be split as follows:

```
$$codes{'year'} = '2003'
$$codes{'semester'} = 'Fall'
$$codes{'department'} = 'nop'
$$codes{'number'} = '590'
```

The routine used at MSU is as follows:

```
sub instcode_format {
    my ($dom,$instcodes,$codes,$codetitles,$cat_titles,$cat_order)
    = @_;
    @{$codetitles} = ("Year","Semester","Department","Number");
    %{$$cat_titles{'Semester'}} = (
        fs => 'Fall',
        ss => 'Spring',
        us => 'Summer'
    );
    @{$$cat_order{'Semester'}} = ('ss','us','fs');
    foreach my $cid (keys %{$instcodes}) {
        if ($$instcodes{$cid} =~ m/^( [suf]s )(\d{2})(\w{2,3})(\d{3,4}\w?)$/ )
        {
            $$codes{$cid}{'Semester'} = $1;
            $$codes{$cid}{'Department'} = $3;
            $$codes{$cid}{'Number'} = $4;
            my $year = $2;
            my $numyear = $year;
            $numyear =~ s/^0//;
        }
    }
}
```

```

        $$codes{$cid}{'Year'} = $year;
        unless (defined($$cat_titles{'Year'}{$year}))
        {
            $$cat_titles{'Year'}{$year} = 2000 + $numyear;
        }
    }
}

my $outcome = 'ok';
return $outcome;
}

```

### instcode\_defaults

Three arguments are required:

1. domain (\$dom)
2. reference to hash which will contain default regular expression matches for different components of an institutional course code (\$defaults)
3. reference to array which will contain order of component parts used in institutional code. (\$code\_order)

The routine returns 'ok' if no errors occurred.

At MSU, the regular expression fragments used mirror those included in the regular expression used in instcode\_format() to split an institutional course code into its component parts.

```

sub instcode_defaults {
    my ($dom,$defaults,$code_order) = @_;
    %{$defaults} = (
        'Year' => '\d{2}',
        'Semester' => '^[sfu]s',
        'Department' => '\w{2,3}',
        'Number' => '\d{3,4}\w?',
    );
    @{$code_order} = ('Semester','Year','Department','Number');
    return 'ok';
}

```