LON-CAPA Next Generation Code Manual

LON-CAPA Consortium

May 27, 2014

Chapter 1

License


```
The LearningOnline Network with CAPA - LON-CAPA

Copyright (C) 2014 Michigan State University Board of Trustees

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
```

Chapter 2

Requirements

2.1 Interface

2.1.1 Fully accessible

All aspects of the interface need to be accessible to screen readers. They should not only function, but also explain themselves. For example, if a number of form fields are optically bundled on screen, they need a fieldset and hidden labels that screen readers can pick up. All interaction with the system needs to be possible by using only the keyboard, no functionality shall depend on only the mouse; if using mice input is convenient or desirable for seeing users, for example in a canvas element, at least an alternative viable textual input method must be provided.

2.1.2 Fully internationalized

The interface and all input elements need to be fully internationalized. This includes menus in helper applications, text direction (right to left), full unicode transparency, and number formats (e.g., comma versus decimal point).

2.1.3 System reacts in less than one second

Any user interaction needs to be acknowledged by the system in less than one second, and non-active interface elements disabled. If a user interaction requires more than one second of processing, progress indicators need to be provided.

2.1.4 Mobile support

Mobile devices, most notably iOS and Android, must be supported. Both instructors and students must be able to conduct everyday course business on mobile devices including smartphones. Some interface functions like MouseOver are not supported in some mobile operating systems, so alternatives must be provided.

2.1.5 No Java or Flash

The interface must not depend on Java or Flash. Instead, JavaScript/HTML5/CSS need to be used.

2.1.6 Cross-browser compatibility

The system needs to support Firefox, Safari, Opera, Chrome, and Internet Explorer. However, the minimum version of Internet Explorer to be considered is Version 11 (keeping in mind that development of the system will take some time, so IE will move on in the meantime).

2.1.7 No complete screen rebuilds

The screen should never completely rebuild. Instead the URL of the screen should always be the server itself and interactivity achieved through AJAX or iframes. Deep-linking and single signon should be enabled but bounce the session back to the normal full screen setup. We are not subservient to other systems.

2.1.8 Easy things easy, hard things possible

The product should be able to directly compete with commercial course management solutions, which means that the 80%-functionality must be as easy to accomplish as in commercial systems. Our heritage demands that we still make the 20% possible.

2.1.9 Preventing users from shooting themselves into the foot

A powerful system will have ample opportunity to configure things in non-sensical ways (for example, having an opening date after the due date, etc). The system must warn students and instructions about potential problems and offer direct ways to remedy the conflict - however, we need to be careful with "auto-correct" or being obnoxiously helpful ("Are you writing a letter?").

2.2 Assets

2.2.1 Backward compatibility

The new LON-CAPA must be backward-compatible to old LON-CAPA using conversion and clean-up processes. Given the size of the LON-CAPA resource pool, more than 99% of the conversion must be fully automatic. The remaining 1% must be identified by the conversion process, so we do not have surprises, and must be fixable. Dynamic metadata from the last decade must be converted to the new system, so we do not lose this usage information.

2.2.2 Printability

Assets have to be printable at different granularity (single, chapter, course) in high quality and compactly. This is essential for exam functionality.

2.3 Data

2.3.1 All changes take effect after at most 10 minutes

Any changes to user roles, course tables of contents, due dates, portfolios, etc, must take effect all across the network in at most 10 minutes, and this includes running sessions.

2.3.2 Separation

No content or interface handlers must directly touch local disks. All network functionality must be abstracted away beyond the "entity" level.

2.4 Network functionality

2.4.1 SSL

All communication between servers is at least 4096-bit encrypted. LON-CAPA issues server and client certificates. Servers need full reverse DNS.

2.4.2 Predictable storage

To observe privacy and export rules, any permanent user data (beyond temporary session information) needs to be stored in a predictable location. The domain concept of LON-CAPA facilitates this.

2.4.3 Authentication

It is the responsibility of the homeserver of a user to authenticate the user.

2.4.4 Run everywhere

All system functionality must be available on all servers, so that content, courses and users can be served from any machine in the cluster (subject to freely configurable limitations).

2.4.5 Servers

Servers need to be dedicated to LON-CAPA for security and privacy reasons. In the age of virtual machines, that should not be a problem.

2.5 Coding environment

2.5.1 Linux Distributions

We are developing on CentOS/RedHat Enterprise. We do not have the time to support other distributions ourselves, and will only accept them if some individual guarantees to be their longterm curator. In the age of virtual machines, it should not be a problem to run a VM with a particular distribution, particularly since the machines are completely dedicated anyway.

2.5.2 Open source

The software is licensed under the GNU General Public License Version 3 or higher, copyright Michigan State University Board of Trustees. Only software components that are compatible with this license shall be used.

Chapter 3

Network

3.1 Nodes

LON-CAPA is built as a network of servers, which can host sessions and store data. LON-CAPA can have several independent clusters of such servers, but it is expected that in real operation, there is essentially one production cluster. In principle, any server in a cluster can host sessions for any user in the cluster, and any server in the network should be able serve any asset in the system. In practice, limitations may be put on this to preserve privacy and copyright.

There are two classes of servers: access and library. The difference is that library servers permanently store data, and access servers to do. This means that library servers need to remain stable and have backup, which access servers can be added or removed on demand to absorb session loads.

3.2 Cluster Table

3.2.1 Configuration

Clusters are established by cluster tables, see below for an example

```
{ domains   : { 'msu'      :    { name    : 'Michigan State University',
                                  class   : 'university',
                                  locale  : 'en-us',
                                  timezone: 'America/Detroit' },
                'sfu'      :    { name    : 'Simon Fraser University',
                                  class   : 'university',
                                  locale  : 'en-ca',
                                  timezone: 'America/Vancouver' },
                'ostfalia' :    { name    : 'Ostfalia University of Applied Sciences',
                                  class   : 'university',
                                  locale  : 'de',
                                  timezone: 'Europe/Berlin' },
                'elps'     :    { name    : 'East Lansing Public Schools',
                                  class   : 'k12',
                                  locale  : 'en-us',
                                  timezone: 'America/Detroit' }
              },
  hosts     : { 'zaphod'   :    { address : 'zaphod.localdomain',
                                  default : 'msu',
                                  domains : { 'msu'      : { function : 'library' },
                                              'elps'     : { function : 'library' },
                                              'sfu'      : { function : 'access'  }
                                            }
                                },
                'marvin'   :    { address : 'marvin.localdomain',
                                  default : 'msu',
                                  domains : { 'msu'      : { function : 'library' },
                                              'elps'     : { function : 'access'  },
```

```
                                          'sfu'      : { function : 'access'  }
                                        }
                              },
             'arthur'   :    { address : 'arthur.localdomain',
                               default : 'sfu',
                               domains : { 'msu'      : { function : 'access'  },
                                           'elps'     : { function : 'access'  },
                                           'sfu'      : { function : 'library' }
                                         }
                              },
             'slarti'   :    { address : 'slartibartfast.localdomain',
                               default : 'ostfalia',
                               domains : { 'ostfalia' : { function : 'library' },
                                           'elps'     : { function : 'access'  },
                                           'sfu'      : { function : 'access'  }
                                         }
                              }
            }
}
```

Thus a particular server can serve more than one domain in different functions. Internally, servers are identified by the internal ID, e.g. "slarti" — this allows migrating nodes between hardware.

3.2.2 Homeservers

Each entity in the system has a so-called homeserver within its domain which holds the authoritative and permanent copy of their data.

3.2.3 Cluster manager

One server in any cluster is the cluster manager. This machine holds the authoritative copy of the cluster configuration table. Who is cluster manager is configured in

```
/home/loncapa/cluster/cluster_manager.conf
```

This need to be the full DNS name of the cluster manager, and essentially configures which cluster a server is a member of.

The cluster manager can trigger updating this table on other servers via

```
# On non-cluster manager, this triggers fetching the cluster table
#
<Location /fetch_cluster_table>
SetHandler perl-script
PerlHandler Apache::lc_init_cluster_table
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

The other server will then fetch the cluster table via the cluster manager's

```
# Cluster manager serves up authoritative cluster table
#
<Location /cluster_table>
SetHandler perl-script
PerlHandler Apache::lc_cluster_table
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

3.3 Connections

Connections between the servers are mediated via https.

3.3.1 Connection authentication

Connections are authenticated via a configuration in lc.conf:

```
<Location /connection_handle>
SetHandler perl-script
PerlHandler Apache::lc_connection_handle
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</Location>
```

3.3.2 Connection handling

Connections are handled by the module lc_connection_handle. Any routines that should be externally accessible need to be registered with the module, like so:

```
BEGIN {
    &Apache::lc_connection_handle::register('modify_namespace',undef,undef,undef,
                                            \&local_modify_namespace,'entity','domain','name','data');
    &Apache::lc_connection_handle::register('dump_namespace',undef,undef,undef,
                                            \&local_json_dump_namespace,'entity','domain','name');
}
```

The first argument is the command name, the following three arguments will be used for additional security, the next argument is the pointer to the subroutine that deals with this, and the remaining arguments are the named arguments of the function.

3.3.3 Local versus remote

Routines need to determine if the data is local or remote, depending on the homeserver of the entity, like so:

```
#
# Dump namespace from local data source
#
sub local_dump_namespace {
   return &Apache::lc_mongodb::dump_namespace(@_);
}

sub local_json_dump_namespace {
   return &Apache::lc_json_utils::perl_to_json(&local_dump_namespace(@_));
}


#
# Get the namespace from elsewhere
#
sub remote_dump_namespace {
   my ($host,$entity,$domain,$name)=@_;
   my ($code,$response)=&Apache::lc_dispatcher::command_dispatch($host,'dump_namespace',
                                 "{ entity : '$entity', domain : '$domain', name : '$name' }");
   if ($code eq HTTP_OK) {
      return &Apache::lc_json_utils::json_to_perl($response);
   } else {
      return undef;
   }
}
```

```
# Dump current namespace for an entity
# Call this one
#
sub dump_namespace {
   my ($entity,$domain,$name)=@_;
   if (&Apache::lc_entity_utils::we_are_homeserver($entity,$domain)) {
      return &local_dump_namespace($entity,$domain,$name);
   } else {
      return &remote_dump_namespace(
             &Apache::lc_entity_utils::homeserver($entity,$domain),$entity,$domain,$name);
   }
}
```

The named arguments are sent as JSON.

Most routines that register services are in /entities — other handlers should use the routines provided in the entity-handlers and *never* directly access the disk, the databases, or the network. The routines to be called are the ones without "local" or "remote."

3.4 Replication

Assets are replicated between servers. Any server in the network can serve any asset in the server under the same URL path. This is transparent, i.e., the same URL path can be requested from any server, regardless of whether or not that asset is already locally present. The server needs to locate the asset in the network and replicate it if needed.

Assets are versioned. Replication works by copying a certain version of an asset from its homeserver using

```
<LocationMatch "^/raw/">
SetHandler perl-script
PerlAccessHandler Apache::lc_raw_acc
SSLRequireSSL
SSLVerifyClient require
SSLVerifyDepth 2
</LocationMatch>
```

This separate address (as opposed to the normal "/res" address) is important, since we need the raw XML, not a rendered version.

The system independently caches the most recent version of a resource. Once this cache expires, a server again asks the homeserver of the asset what the most recent version is via the "current_version" command that is registered by lc_entity_urls. If this is not the version currently stored, a new version is fetched.

Chapter 4

Entities

4.1 Identification

Users, courses, and assets are entities in the system. Since the network has domains, an entity is identified by two components:

- An entity code (in the system internally often referred to as just "entity", e.g., "qLLTNbdEhQaxQ8AZyYp"
- The domain (e.g., "msu")

Thus, entities should *always* be refered to by both, qLLTNbdEhQaxQ8AZyYp:msu. Just specifying the entity code is not enough!

Entity codes are guaranteed to be unique within a domain, but not between domains.

Entities do not change. An individual, course, or assets always keeps its entity:domain.

4.2 Storage and Interface

Information in the system is always stored by entity:domain, not by usernames or course codes. However, entity:domain is not exposed on the interface level, here we talk in terms of usernames, personal ID numbers, or course codes.

4.3 Users

Usernames and personal ID numbers (PIDs) are properties of the user entity. They need to be unique within a domain, thus to fully specify a user entity in this fashion, one needs to specify the username *and* domain. Just specifying the username is not enought!

Usernames and PIDs can change. For example, username changes can happen due to live events or because the user wants a vanity ID. At most universities, usernames once given out are never recycled. Also, PIDs can change if a person changes from student to faculty/staff or vice versa. Once again, PIDs at most places are never recycled.

The system should thus assume:

- At any given point in time, users have one primary username and PID.
- The system needs to still support the old usernames and PIDs, as particularly during mid-semester changes, uploadable grading lists or other spreadsheets may still include the old username/PID

Each username or PID points to one and only one user entity. However, a user entity can have more than one supported username or PID.

4.4 Courses

Courses are treated very similarily to users. They have course IDs such as "phy231fs14", which might have. These are treated the same way as usernames or PIDs.

4.5 Assets

Instead of usernames, PIDs, or couse IDs, assets have URLs. One asset can have more than one URL, however, the URL of published assets looks like

```
/asset/n/3/msu/smith/...
```

where the first two components after "asset" designate the version, followed by the domain and the publishing author.

The above example explictly asks for version number 3 of the asset. Other ways are

```
/asset/as_of/2014-01-08_04:05:06/msu/smith/...
```

which asks for the version as-of a certain date. Finally,

```
/asset/-/-/msu/smith/...
```

asks for the most recent version.

If at all possible, relative URLs should be used, so version arguments like "as_of" get preserved to also get the right versions of dependent files.

Chapter 5

Storage

5.1 Databases

The system uses two databases: PostgreSQL and MongoDB.

- PostgreSQL is a traditional relational database which is used for predictable data that can reside in tables. These are often lookup-tables that need fast search
- MongoDB is a noSQL database which is used for flexible, structured data. Searches are rare and not performance critical

5.2 Caching

The system uses two caching mechanisms: in-memory and Memcached

- In-memory is occasionally used if a particular process needs to preserve variables, so they do not need to be reinitialized every time. Examples are database handles, etc.
- Memcached is the main caching mechanism, which caches data in memory across processes. The cache items have associated expiration times. The time to refresh these caches is distributed across processes and users.

5.3 File system

Assets are stored on the file system in /home/loncapa/res

5.4 Programmatic access

Databases should not be touched by any higher order handlers. While drivers are located in the /databases GIT directory, e.g., lc_memcached.pm lc_mongodb.pm lc_postgresql.pm, these should not be called. Instead, handlers should access the abstractions in the /entities GIT directory, and there only the routines that are not starting with “remote” or “local”.