LON-CAPA Next Generation Code Manual

LON-CAPA Consortium

May 28, 2014

Contents

Chapter 1

License

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 Authorization

Authorization in the system is based on roles. A user can have several (time-limited) roles, each of which provides a certain set of privileges within a particular realm.

2.4.5 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).

2.4.6 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. We will not accept short-life-cycle distributions, as only long-life-cycle distibution guarantee the required security and stability. 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 and if ready-to-run images are provided.

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.

2.5.3 Porting from LON-CAPA 2.x

The project will not succeed in a timely fashion if we do not port code from LON-CAPA 2.x. Eventually, program code will need to be cleaned up and moved over in order to keep timelines. This will mean that sometimes we need to sacrifice using the "latest-greatest" technology in order to facilitate portability of code or even just algorithms.

Chapter 3

URL Translation

3.1 Accessing handlers

Handlers are defined in lc.conf using Location and LocationMatch statements. For example

```
<Location /menu>
SetHandler perl-script
PerlAccessHandler Apache::lc_auth_optional
PerlHandler Apache::lc_ui_menu
</Location>
```

defines how the URL /menu is handled. During the Apache request cycle, first the PerlAccessHandler is called. There are two versions,

- lc_auth_acc, which requires a session and loads the session environment - otherwise, an error handler usually points to the login handler
- lc_auth_optional, which only loads a session environment if a session is active

The statement then defines lc_ui_menu as the response handler. Additional error handlers can be called, for example to route back to the login screen.

Some handlers also have cleanup handlers, which may do the actual work, for example

```
<Location /async>
SetHandler perl-script
PerlAccessHandler Apache::lc_auth_acc
PerlHandler Apache::lc_ui_async
PerlCleanupHandler Apache::lc_ui_async::main_actions
</Location>
```

This handler can reply to the browser immediately and then do the actual work in the cleanup phase. This is most useful in connection with asynchronous AJAX requests.

3.2 Accessing assets

Access to assets is more complicated. They are caught in the global translation handler

```
PerlTransHandler Apache::lc_trans
```

which (at some point) looks/looked like this:

```
sub handler {
    my $r = shift;
# We care about assets
    if ($r->uri=~/^\/asset\//) {
# First check if we can even find this
       my $filepath=&Apache::lc_entity_urls::url_to_filepath($r->uri);
       unless ($filepath) {
```

```
# Nope, this does not exist anywhere
          return HTTP_NOT_FOUND;
       }
# Is this locally present?
       unless (-e $filepath) {
# Nope, we don't have it yet, let's try to get it
          unless (&Apache::lc_entity_urls::replicate($r->uri)) {
# Wow, something went wrong, not sure why we can't get it
             &logwarning("Failed to replicate ".$r->uri);
             return HTTP_SERVICE_UNAVAILABLE;
          }
       }
# Bend the filepath to point to the asset entity
       $r->filename($filepath);
       return OK;
    } elsif ($r->uri=~/^\/raw\//) {
       $r->filename(&Apache::lc_entity_urls::raw_to_filepath($r->uri));
       return OK;
    }
# None of our business, no need to translate URL
    return DECLINED;
}
```

This deals with URLs of the type /asset and /raw, and does the translation of URL-paths to the asset entity:domain, as well as versioning and replication. By the time this handler is done, the asset is either present or we throw a NOT_FOUND.

3.3 System Pages

System pages such as the dashboard or the preferences are actually HTML-pages, which include LON-CAPA XML. They use the same XML parser as for example homework pages. An example is lc_preferences.html, the user preferences screen:

```
<html>
<head>
<title>Preferences</title>
<script src="/scripts/lc_preferences.js"></script>
</head>
<body>
<h1><localize>Preferences</localize></h1>
<p>
<span class="lcstandard"><localize>You can modify your user preferences below.</localize></span>
<span class="lcerror"><localize>A problem occured, please try again later.</localize></span>
<span class="lcproblem"><localize>A problem occurred while saving your preferences.</localize></span>
<span class="lcsuccess"><localize>Your preferences were saved.</localize></span>
 
</p>
<lcform id="preferencesform">
<lcformtable>
<lcformtableinput type="language" description="Language" id="language" default="user" />
<lcformtableinput type="timezone" description="Timezone" id="timezone" default="user" />
</lcformtable>
<lcformtrigger id="storebutton" description="Store" />
</lcform>
</body>
</html>
```

Chapter 4

Network

4.1 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.

4.2 Cluster Table

4.2.1 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.

4.2.2 Homeservers

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

4.2.3 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>
```

4.3 Connections

Connections between the servers are mediated via https.

4.3.1 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>
```

The certificates need to be located in /home/loncapa/certs/.

4.3.2 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.

4.3.3 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."

4.4 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 5

Entities

5.1 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.

5.2 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.

5.3 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.

5.4 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.

Communities are special kinds of courses which do not have grade book, and which have different associated roles. Internally, they are distinguish by the "type" in the course profile being "community" instead of "regular."

Courses can have sections and groups.

- Sections usually correspond to educational venues such as laboratories or recitations. A student can only be in one section at a time.

- Groups are more like teams, working together. A student can be in more than one group at a time

Internally the two are distinguished by "type" in the course/community profile.

5.5 Assets

Assets are HTML pages, problems, images, movies, etc, which are stored in user portfolios. 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.

5.5.1 Profiles and metadata

Both users and courses have profiles, which store basic information like their full name or title, or any configuration preferences. Assets have metadata, whcih is used for various cataloging purposes and populated with both static and dynamic metadata. These are stored in MongoDB..

Chapter 6

Authorization

6.1 Roles

Authorization is handled via roles. Each role has a certain set of privileges, which are defined within their realms through roles.json. Here's an excerpt:

```
{
"superuser"          : { "realm"  : "system",
                         "system" : {
                                      "view_role"       : "1",
                                      "modify_role"     : {
                                                            "domain_coordinator" : "1"
                                                          }
                                    }
                       },
"domain_coordinator" : { "realm"  : "domain",
                         "domain" : {
                                      "view_role"       : "1",
                                      "modify_role"     : {
                                                            "course_coordinator" : "1"
                                                          },
                                      "modity_auth"     : "1"
                                    }
                       },
"course_coordinator" : { "realm"  : "regular",
                         "course" : {
                                      "view_role"       : "1",
                                      "modify_role"     : {
                                                            "course_coordinator" : "1",
                                                            "instructor"         : "1",
                                                            "teaching_assistant" : "1",
                                                            "student"            : "1"
                                                          },
                                      "modify_settings": "1",
                                      "edit_toc"        : "1",
                                      "access_content" : {
                                                            "open"               : "1",
                                                            "closed"             : "1",
                                                            "hidden"             : "1"
                                                          },
                                      "modify_grade"    : "1",
                                      "notify"          : "1"
                                    }
                       },
"instructor"         : { "realm"  : "regular",
```

```
                        "course" : {
                                    "access_content" : {
                                                        "open"              : "1",
                                                        "closed"            : "1",
                                                        "hidden"            : "1"
                                                       }
                                   },
                        "section": {
                                    "view_role"      : "1",
                                    "modify_grade"   : "1",
                                    "notify"         : "1"
                                   }
                       },
"teaching_assistant" : { "realm"  : "regular",
                        "course" : {
                                    "access_content" : {
                                                        "open"              : "1"
                                                       }
                                   },
                        "section": {
                                    "view_role"      : "1",
                                    "modify_grade"   : "1",
                                    "notify"         : "1"
                                   }
                       },
"student"            : { "realm"  : "regular",
                        "course" : {
                                    "access_content" : {
                                                        "open"              : "1"
                                                       }
                                   }
                       },
"community_organizer": { "realm"  : "community"
                       },
"member"             : { "realm"  : "community"
                       },
"author"             : { "realm"  : "user"
                       },
"co_author"          : { "realm"  : "user"
                       }
}
```

Chapter 7

Storage

7.1 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.

Data only sits on the homeserver of the user or course.

7.1.1 PostgreSQL tables

The tables in PostgreSQL are

URL table

The table is used to look up the entity code for a certain URL. More than one URL can point to the same asset. The domain is not stored, since it is already encoded in the URL.

```
#
# Make the URLS table
#
create table urls
(url text primary key not null,
entity text not null)
```

User table

This table is used to look up the entity code of usernames. More than one username can point to the same entity code. Both entity code and username would be in the same domain.

```
#
# Make the user lookup table
# Get the entity for a username
#
create table userlookup
(username text not null,
domain text not null,
entity text not null,
primary key (username,domain))
```

PID table

Same as user table for PIDs.

```
#
# Make the pid lookup table
# Get the entity for a PID
#
create table pidlookup
(pid text not null,
domain text not null,
entity text not null,
primary key (pid,domain))
```

Course ID table

Same as username and PID tables for course IDs.

```
#
# Make the courseID lookup table
# Get the entity for a courseID
#
create table courselookup
(courseid text not null,
domain text not null,
entity text not null,
primary key (courseid,domain))
```

Homeserver table

Table used to look up the homeserver of an entity:domain.

```
#
# Make the homeserver lookup table
#
create table homeserverlookup
(entity text not null,
domain text not null,
homeserver text not null,
primary key (entity,domain))
```

Roles

This is for looking up who has certain roles. It is not the authoritative version.

```
#
# The role table
# These are the roles on this server
# The primary cluster server (and only the primary cluster server)
# also has the system and domain-wide roles
# This is for lookup, the actual roles are with the users
#
create table rolelist
(roleentity text,
roledomain text,
rolesection text,
userentity text not null,
userdomain text not null,
role text not null,
startdate timestamp,
enddate timestamp,
manualenrollentity text,
manualenrolldomain text,
primary key (roleentity,roledomain,rolesection,userentity,userdomain,role))
```

Assessment table

This has the standardized data about assessments in courses, which is used for gradebooks. More structured data to be called up when the history of a problem is desired, or if it is to be brought up on the screen, resides in MongoDB.

```
#
# This is the big table of course assessments on the homeserver of the courses
# Authoritative
#
create table assessments
(courseentity text not null,
coursedomain text not null,
userentity text not null,
userdomain text not null,
resourceid text not null,
partid text not null,
scoretype text,
score text,
totaltries text,
countedtries text,
status text,
responsedetailsjson text,
primary key (courseentity,coursedomain,userentity,userdomain,resourceid,partid))
```

7.2 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.

7.3 File system

Assets are stored on the file system in /home/loncapa/res. The authoritative copy of assets is on the author's homeserver (see section 4.2.2 on page 12), but they are copied through replication (see section 4.4 on page 14) to other servers.

The filesystem uses the entity code as part of the location. For example,

`/home/loncapa/res/msu/M/4/O/W/M4OWjvPTQCIE6o8RTMJ_3`

is version 3 of the asset with entity code M4OWjvPTQCIE6o8RTMJ in domain msu.

7.4 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".

Chapter 8

XML Parser

8.1 Invocation

The XML parser is at the heart of much of LON-CAPA's operation. As opposed to the original LON-CAPA, it does not distinguish between problems and non-problems, and it also serves system pages. It is called via lc.conf:

```
<LocationMatch "\.(xml|html|htm|xhtml|xhtm)$">
SetHandler perl-script
PerlHandler Apache::lc_asset_xml
</LocationMatch>
```

The main handler is thus lc_asset_xml. All other modules need to register their tags with this parser.

The XML parser actually since the early days of LON-CAPA used an HTML parser,

```
use HTML::TokeParser();
```

so it is more robust against trying to make sense of malformed XML, however, no malformed document should be stored. As this is the same parser mechanism as is used in LON-CAPA 2.x, porting of the extensive homework functionality will be facilitated. Documentation for this module is online.

8.2 Tag types

Tags are defined via subroutines. The name of these subroutines has three parts: start or end, tagname, and target. For example, here is lc_xml_localize, which defines the localize-tag:

```
package Apache::lc_xml_localize;

use strict;
use Apache::lc_ui_localize;

our @ISA = qw(Exporter);

# Export all tags that this module defines in the list below
our @EXPORT = qw(start_localize_html start_localize_tex);

sub start_localize_html {
   my ($p,$safe,$stack,$token)=@_;
   my $text=$p->get_text('/localize');
   $p->get_token;
   pop(@{$stack->{'tags'}});
   return &mt($text,
              split(/\s*\,\s*/,&Apache::lc_asset_safeeval::texteval($safe,$token->[2]->{'parameters'})));
}

sub start_localize_tex {
   my ($p,$safe,$stack,$token)=@_;
```

```
   return &mt($p->get_text('/localize'),
               split(/\s*\,\s*/,&Apache::lc_asset_safeeval::texteval($safe,$token->[2]->{'parameters'})));
}

1;
__END__
```

Target html is for web, target tex for printing. Defined tags need to be exported. In lc_asset_xml, lc_xml_localize needs to be used (without parenthesis).

If a tag is not defined, it is just printed to the screen on the web and ignored in print.

8.3 Stacks and safeeval

The stack is used to store the arguments of tags, so they can be retrieved later or inside of nested tags. The safeeval-space is a Perl safespace in which calculations and storage can happen (same as in the old LON-CAPA).

Chapter 9

Internationalization

9.1 Encoding

Unicode UTF-8 is used throughout. All assets, program code, translation files, stored data, etc, is stored in utf-8 in order to avoid encoding problems down the road.

9.2 Maketext

LON-CAPA uses Maketext for internationalization, both inside of documents and called programatically. For each language, a translation file needs to be written. The format is simple:

```
package Apache::lc_localize::de;
use base qw(Apache::lc_localize);
use utf8;

%Lexicon=('_AUTO' => 1,
'language_code'      => 'de',
'language_direction' => 'ltr',
'language_description' => 'Deutsch',
'date_locale'  => '$weekday, $day. $month $year, $twentyfour:$minutes:$seconds Uhr',
'date_short_locale' => '$day.$month.$year',
'date_months'  => 'Jan.,Feb.,März,April,Mai,Juni,Juli,Aug.,Sep.,Okt.,Nov.,Dez.',
'date_days'    => 'So.,Mo.,Di.,Mi.,Do.,Fr.,Sa.',
'date_am' => 'vormittags',
'date_pm' => 'nachmittags',
'date_format' => '24',

'superuser' => 'Superuser',
'domain_coordinator' => 'Domänenkoordinator',
'course_coordinator' => 'Kurskoordinator',
'instructor' => 'Dozent',
'teaching_assistant' => 'Tutor',
'student' => 'Studierender',
'community_organizer' => 'Gemeinschaftsorganisator',
'member' => 'Mitglied',
'author' => 'Autor',
'co_author' => 'Co-Autor',

'Modify Selected Entries' => 'Ausgewählte Einträge ändern',

'Add New Entry' => 'Neuen Eintrag hinzufügen',

"Showing [_1] to [_2] of [_3] entries" => "Einträge [_1] bis [_2] von [_3] insgesamt",

...
```

```
);

1;
__END__
```

The files need to be utf-8-encoded, and we need to make sure that Perl knows this by specifying "use utf8."

Some configuration entries such as date_locale are mandatory. Where translations are missing, English is used. The [_*n*] are placeholders for variables.

9.3 Invocation in documents

In documents, phrases can be translated using the <localize>-tag, e.g.

```
<span class="lcsuccess"><localize>Your preferences were saved.</localize></span>
```

Placeholders can be defined using the parameter argument, e.g., <localize parameters="42,'Fred'">. This tag is defined in module lc_xml_localize.

9.4 Invocation from Perl

Internationalization is available from Perl via the %mt()-function. The first argument is the text, any following arguments come from the remaining arguments, e.g.,

```
&mt("Showing [_1] to [_2] of [_3] entries",$start,$finish,$total)
```

9.5 Special cases

Some Javascript utilities expect a configuration file. This can be generated dynamically like for example in lc_ui_datatable_i14n, which is invoked by lc.conf:

```
sub handler {
# Get request object
   my $r = shift;
   $r->content_type('application/json; charset=utf-8');
   my $items={
    "sEmptyTable"    => &mt("No data available in table"),
    "sInfo"          => &mt("Showing [_1] to [_2] of [_3] entries",'_START_','_END_','_TOTAL_'),
    "sInfoEmpty"     => &mt("Showing 0 to 0 of 0 entries"),
    "sInfoFiltered"  => &mt("(filtered from [_1] total entries)",'_MAX_'),
    "sInfoPostFix"   => "",
    "sInfoThousands" => &mt(","),
    "sLengthMenu"    => &mt("Show [_1] entries",'_MENU_'),
    "sLoadingRecords"=> &mt("Loading..."),
    "sProcessing"    => &mt("Processing..."),
    "sSearch"        => &mt("Search:"),
    "sZeroRecords"   => &mt("No matching records found"),
    "oPaginate" => {
        "sFirst"    => &mt("First"),
        "sLast"     => &mt("Last"),
        "sNext"     => &mt("Next"),
        "sPrevious" => &mt("Previous")
                   },
    "oAria" => {
        "sSortAscending"  => &mt(": activate to sort column ascending"),
        "sSortDescending" => &mt(": activate to sort column descending")
               }
   };
   $r->print(&Apache::lc_json_utils::perl_to_json($items));
   return OK;
}
```

Chapter 10

Glossary

Asset An HTML page, homework problem, image, movie, etc.

Author A user who has author privileges, i.e., who can publish assets

Cluster Group of machines linked into a network governed by a cluster table

Course An entity with members and a table of contents. Used as a general term to also include communities

Community Special kind of "course" without a gradebook, usually used for collaboration

Domain Logical segmentation of a LON-CAPA cluster, usually by institution

Entity Code Part of the unique identifier of assets, users, and courses, the other part being the domain

Group A special kind of section, mostly for teamwork. Students can be in more than one section at a time

Homeserver Node in a cluster which holds the authoritative and permanent copy of a user's or course's data and assets

Portfolio The space where users store their assets

Realm The extend of privileges, i.e., system-wide, domain-wide, course-wide, section-wide, or user-wide

Role Users can have a number of (time-limited) roles which each grant a certain set of privileges within a particular realm

Section Sub-group of a course, for example a lab section. A student can only be in one section at a time

Index