

ECSA — Elearning Community Service Architecture

Copyright © 2011,2012,2013,2014 Heiko Bernlöhr (FreeIT.de)

This file is part of ECS.

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

ECS 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with ECS. If not, see <http://www.gnu.org/licenses/>.

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
7b6a2e6	2014-08-30	Added links to official cookie specification.	Heiko Bernloehr
73f70f9	2014-08-08	ecs_hash_url now marked as DEPRECATED.	Heiko Bernloehr
0e2c56b	2014-04-03	Broken link repaired.	Heiko Bernloehr
73f05e5	2014-02-18	Unicorn application server.	Heiko Bernloehr
c576ee8	2014-02-18	Some minor text changes.	Heiko Bernloehr
a4b86b1	2014-02-18	Added notice to use Debian Wheezy.	Heiko Bernloehr

Contents

1 Overview	1
1.1 Sample usage scenario	1
2 Participants	3
2.1 Basic functionalities and requirements	3
2.1.1 Technology / Architecture	3
2.1.2 Authentication	3
2.2 Authorization	3
2.3 Ressource extensions / alterations	3
2.4 Web interfaces	4
2.5 Communication procedures / scenarios	4
2.5.1 Direct participant to participant communication	4
3 ECS	7
3.1 HTTP Header	7
3.1.1 ECS specific headers	7
3.1.2 HTTP standard header	8
3.2 HTTP return codes	8
3.3 Addressing	8
3.3.1 Membership IDs	9
3.3.2 Community names and ids	9
3.3.3 Create a resource	9
3.3.4 Get a resource	9
3.4 Community selfrouting	9
3.5 Authentication	10
3.6 Anonymous participants	10
3.7 System resources	10
3.7.1 Events	10
3.7.2 Memberships	11
3.7.3 Auths	12
3.7.3.1 Courselinks	13

3.8	Application specific resources	14
3.8.1	Resource structure	14
3.8.2	Message resource	14
3.8.2.1	Subresource details	15
3.8.3	List resource	15
3.8.3.1	Subresource details	16
3.8.3.2	Querystrings	17
3.8.4	Queue resource	18
3.8.5	Postrouting	18
3.9	JSON-Schemas	18
3.10	Participant Cluster	18
3.10.1	Cluster building	19
3.10.2	Cluster broadcasting	19
4	Filter plugins	20
4.1	Template	20
5	CampusConnect	22
5.1	Scriptable REST commandline client	22
5.1.1	Monitoring resources	22
5.2	Simulating LSFproxy	22
5.2.1	Create	22
5.2.2	Delete	23
5.2.3	Update	23
5.2.4	Show Memberships	23
5.2.5	Show Usage	23
5.3	Resources	24
5.3.1	/campusconnect/courses	24
5.3.2	/campusconnect/organisation_units	24
5.3.3	/campusconnect/terms	24
5.3.4	/campusconnect/course_members	24
5.3.4.1	LSF-Proxy course_members representation	25
5.3.5	/campusconnect/directory_trees	25
5.3.6	/campusconnect/courselinks	25
5.3.7	/campusconnect/course_urls	26
5.4	LSF-Proxy	26
5.5	Redirect procedure/protocol when consuming courselinks	28
6	Interconnectivity	29
6.1	Necessary extensions to ECS	29
6.2	Interconnection procedure	29

7	Installation	31
7.1	Base installation on GNU/Linux Debian (Squeeze)	31
7.2	Base installation on GNU/Linux Debian Wheezy	33
7.3	NGINX HTTP-Proxy-Server	34
7.4	Unicorn application server	35
7.5	CampusConnect	35
7.6	ViP	35
8	Glossary	36
9	Index	37

Chapter 1

Overview

An ECSA is a service architecture for elearning based webservices. It provides mechanisms for communication and authorization between elearning systems among each other and management systems. This is implemented via a MOM.

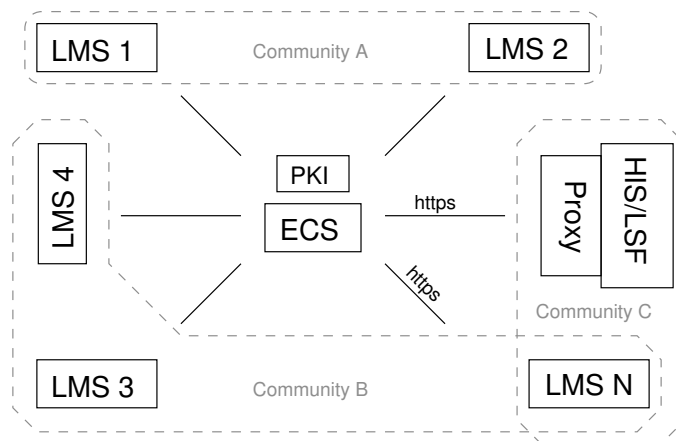


Figure 1.1: Components of an ECSA network.

An ECSA builds up of three primary components:

- The ECS (elearning community server) serves the core functionality of an ECSA network. It provides named message resources to allow communication between all participants.
- An ECC (elearning community client) is a participant in an ECSA network. It has to be registered at ECS and must be able to talk to the ECS as a REST based client. This participant normally has a native implementation of the ECS interface. Our favourite ECCs are LMSs (learning management systems).
- An ECP (elearning community proxy) represents a special kind of participant. It serves as a proxy for a none ECSA compliant system so that such a system is able to participate in an ECSA network without ever knowing about it.

1.1 Sample usage scenario

Suppose you have several LMSs (learning management systems) and want to share courses between them. You decide not to interchange the real courses but only course links which consist of some meta data of the appropriate course especially a link formed by an URL pointing to the real course so you can call it through the WWW e.g.:

http://ilias.freeit.de/goto.php?target=pg_26_43&client_id=ecs2

Now it's possible for each LMS to communicate the released courses by the resources provided from the ECS to an explicit LMS (point to point) or to a community of LMSs (point to multipoint).

Because of the uniform application interface — there are only GET, PUT, DELETE and POST operations — receiving participants can fetch messages through a GET on the resource URL or sending messages by a POST on the resource URL (with some additional query parameters or header variables to point to the appropriate receivers).

To illustrate this we use the simple ECC application `curl` to send a message from one participant to another:

```
curl -i -H 'X-EcsAuthId: pid01' \  
      -H 'X-EcsReceiverMemberships: mid02' \  
      -H 'Content-Type: application/json' \  
      -X POST \  
      -d '{  
          "name": "Mathematics II",  
          "url" : "http://ilias...?target=pg_26_43&client_id=ecs2",  
          ...  
      }' \  
      http://ecs.freeit.de/campusconnect/courselinks
```

In order to receive a message (in fifo mode) the receiving participant may call:

```
curl -i -H 'X-EcsAuthId: pid02' \  
      -H 'Accept: text/plain; application/json' \  
      -X GET \  
      http://ecs.freeit.de/campusconnect/courselinks/fifo
```

Of course, there are several ways to operate on a resource.

Chapter 2

Participants

A participant represents a legal client in an ECSA network.

2.1 Basic functionalities and requirements

2.1.1 Technology / Architecture

- has to communicate with the ECS as a **REST** client.
- **HTTP 1.1** as transport and application protocol
- provide persistent connection (keep-alive)
- provide SSL/TLS transport layer
- has to use **UTF-8**

2.1.2 Authentication

- **HTTP Basic auth**
- X.509 Certificates (SSL/TLS client authentication)

2.2 Authorization

A client should be able to use a simple "one touch token" authorization through the ECS `sys/auths` resource. This token could be used to accomplish delegated authorization for accessing resources on participants of a common ECSA network. E.g. in redirecting users clicking on course links or in [direct communicating of participants](#).

2.3 Ressource extensions / alterations

To make resource extensions and alteration possible the clients have to easily permit

- additional ressources
 - extensible data formats
 - Postels's Law (robustness principle): *Be conservative in what you send; be liberal in what you accept.*
-

- versioning through request and response header (content negotiation)
 - Accept: application/vnd.my-format.v1+json
 - Accept: application/vnd.my-format.v2+json

2.4 Web interfaces

- Interface for ECS configuration data

2.5 Communication procedures / scenarios

In order to take part in an ECSA network a participant has to communicate with the ECS and other participants in different ways.

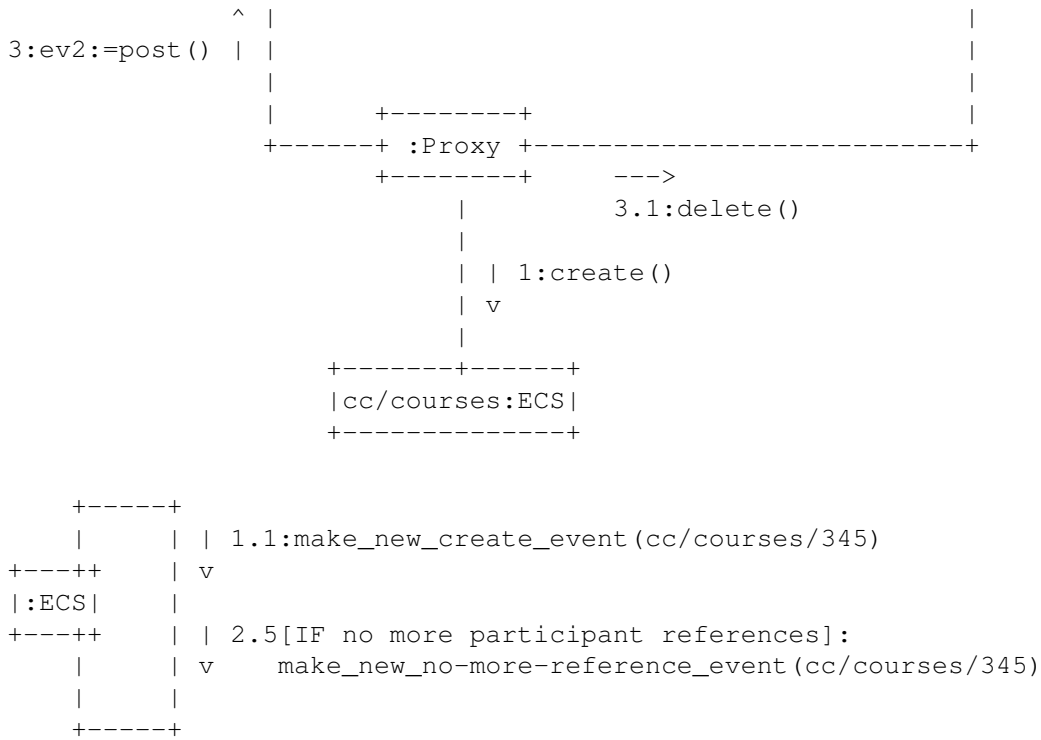
2.5.1 Direct participant to participant communication

This communication procedure takes place if a participant wants to get resource data directly from another participant. Normally all participants communicate only with the ECS and not face to face to each other. For example this could be necessary if you're sharing a central ECS with other organizations and you are not allowed to give away sensible data out of the control of your organization. Of course therefore the communicating participants have to be in the controlling area of your organization.

Assuming following situation: A Proxy (first participant) wants to send course data to a LMS (second participant). This message holds sensible data, which doesn't have to leave the controlling area of the organization by law. Both participants were controlled by the organization.

This procedure guarantee that the appropriate course data will remain on the proxy until the LMS has successfully fetched the data.





1

The proxy creates a new course representation on the ECS, which was addressed to the LMS. The proxy doesn't store the sensible data there, instead it stores an URL where the real data could be fetched.

1.1

The ECS makes a new create event on the event queue of the receiving LMS, storing the new generated resource URL `cc/courses/345` in it.

2

Then the LMS fetches (POST) this event message from its event resource (`sys/events/fifo`) of ECS, which gives it a new or updated coursedata URL on ECS. This would be `cc/courses/345` with `Content-Type:text/uri-list` (mime type see [rfc2483](#)).

2.1

Now the LMS takes this URL and fetches (GET) it from ECS (the LMS only fetches the message via a GET, so that the message will still be there). Only now the LMS gets the real resource URL to fetch the desired course data from the proxy. This url maybe an obscured url like `https://.../746354389534`

2.2

Next the LMS fetches (POST) a one touch token from the `sys/auths` resource of ECS in case the proxy use it for authorization against ECS.

2.3

Then the LMS gets (GET) the actual course data from the proxy URL provided by the received message in 2.1.

2.3.1

The proxy returns the coursedata to the LMS if the auth token (`auth.hash`) provided in 2.3 is still available at `sys/auths` resource on ECS.

2.4

When it will get back the course resource representation in 2.3 successfully, it deletes (DELETE) the message `cc/courses/345` received in 2.1 on ECS.

2.5

If there are no further references on `cc/courses/345` ECS makes a new no-more-reference event for `cc/courses/345` addressed to the proxy (original sender).

3

Proxy is fetching (POST) its events queue.

3.1

If the received event in 3 was a no-more-reference event for `cc/courses/345`, Proxy knows, that nobody further references `cc/courses/345`. This tells the Proxy, that every addressed participants have fetched `cc/courses/345` on ECS and hence all participants has fetched `https://.../746354389534`. If this resource was not configured persistent Proxy could delete `https://.../746354389534`.

Chapter 3

ECS

The elearning community server (ECS) is designed as a message oriented middleware (MOM) and is implemented as a **REST** conform application.

Because the ECS was born in an elearning context the following definition shows consideration of that. Nevertheless the ECS could be used in other areas of responsibility.

The ECS groups its participants in so called *communities*. Participants could address each other only if they share a community. Therefore they could address an explicit participant, a participant list or the whole members of the community (see ECS API for details).

All participants have to register at the ECS. Every registered participant has access to at least three system resources (`/sys/memberships`, `/sys/events`, `/sys/auths`) to get informed and take part at a ECSA network. To design/map your specific application communication you can create as many application resources you want.

3.1 HTTP Header

3.1.1 ECS specific headers

X-EcsAuthId

Has to be a valid participant id. In a standard ECS configuration this HTTP header will be attached by the authentication process running on the proxy server.

X-EcsReceiverCommunities

Has to be a valid community id/ids or community name/names. Addresses all participants joined the community/communities. You are able to note multiple communities, either by name or by id, spaced by comma. Only allowed by POST.

X-EcsReceiverMemberships

Has to be a valid membership id/ids. Addresses all listed memberships. You are able to note multiple memberships spaced by comma. Only allowed by POST.

X-EcsSender

Describes the sender of a message. If you GET a resource this header variable shows the sender membership id. Additionally the ECS sets the X-EcsReceiverCommunities variable to the community from which you have received the message. If the message reach you from several communities X-EcsSender show you a comma separated list of membership ids representing the appropriate membership id of the sender in these communities. In this case the X-EcsReceiverCommunities variable would also represent a comma seperated list of a corresponding community ids.

X-EcsQueryStrings

Used to provide querystrings.

3.1.2 HTTP standard header

Accept

Content-Types that are acceptable.

Content-Type

The mime type of the body of the request (used with POST and PUT requests).

If-None-Match

Allows a 304 Not Modified to be returned if content is unchanged.

Cookie

An HTTP cookie previously sent by the server with Set-Cookie ([Wikipedia](#), [RFC6265](#)).

Content-Type

The mime type of this content.

ETag

An identifier for a specific version of a resource.

Location

Used in redirection, or when a new resource has been created.

Set-Cookie

An HTTP cookie ([Wikipedia](#), [RFC6265](#)).

3.2 HTTP return codes

200

Successful GET.

201

Successful POST.

304

A *Not Modified* response on a conditional GET. This means the requested resource has not been changed.

404

Resource not available.

4xx

General client side error.

5xx

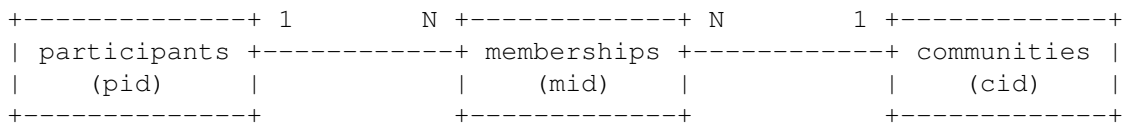
General server side errors.

3.3 Addressing

In order to communicate to each other you have to provide a unique address. These addresses can either be a so called membership id or a community id or community name.

3.3.1 Membership IDs

These are unique ids in the scope of an ECS. They establish a relationship between a participant and a community:



Therefore a participant can be associated to different communities. Every participant can inquire his membership ids by calling the [memberships resource](#).

3.3.2 Community names and ids

A community can be referenced by his community id (cid) or his community name. If you address a community you implicit address all members of the community. This applies also to the sender joining the receiver community if the sender has set his community_selfrouting flag (default off), otherwise the sender will be implicitly excluded from the receiver list. Every participant can inquire his communities memberships by calling the [memberships resource](#).

3.3.3 Create a resource

If you want to POST to a resource you have to provide either a `X-EcsReceiverMemberships` or `X-EcsReceiverCommunities` header or both together.

If you want to address a single membership or a dedicated number of memberships you have to set the `X-EcsReceiverMemberships` header. This header can have a list of values, e.g.

```
X-EcsReceiverMemberships: 3,6,47
```

If you want to address a community you have to set the `X-EcsReceiverCommunities` header. This header can have a list of values, e.g.

```
X-EcsReceiverCommunities: SWS,23,25
```

3.3.4 Get a resource

If you GET a resource then the ECS set the `X-EcsSender` and the `X-EcsReceiverCommunities` header to show you from whom and where your received message comes. If there is a list of `X-EcsReceiverCommunities` values than there is also a list of corresponding `X-EcsSender` values, i.e. the sending participant is member of multiple communities and addressed his message to multiple communities also, e.g.

```
X-EcsSender: 3,19
X-EcsReceiverCommunities: UnisBW,SUV
```

This means that this message is addressed to you through two communities (UnisBW, SUV) and the sender has the membership id 3 in UnisBW and 19 in SUV.

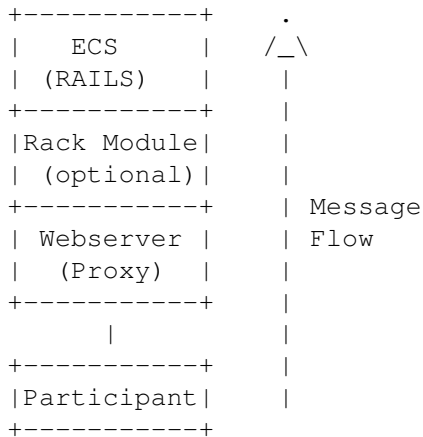
3.4 Community selfrouting

If community selfrouting is activated at the participant (administration area) you can decide if you also want to receive the message which you send to an appropriate community, i.e. you get an event notification (if events on this resource is activated) and you get it listed by its [list resource](#) and could access it through its [queue resource](#). Of course, as sender of the message you can always access it by its [message resource](#).

3.5 Authentication

All participants have to be authenticated in order to use ECS services. A participant is deemed to be authenticated if the `X-EcsAuthId` header is set and the ECS knows it. The real authentication take place in front of the ECS, normally at the Webserver. But this depends on configuration/installation of ECS:

Message flow through ECS application.



Currently supported authentication methods:

- Basic Auth
- X.509 certificates

3.6 Anonymous participants

The creation of a new anonymous participant automatically takes place by every call to an ECS resource if the calling participant didn't set `X-EcsAuthId` or `Cookie` header, by setting a `Set-Cookie` header in the response. On subsequent calls the participant has to provide this cookie in a `Cookie` header in order to be identified as the previously calling participant. Additionally those participants were automatically joined to the *public* community. Further their lifetime will be limited and all resources will be silently deleted after this lifetime becomes zero. With succesional accesses to ECS this lifetime will be refreshed. For general cookie handling see [Wikipedia](#) and [RFC6265](#). See also ECS curl [examples](#).

3.7 System resources

3.7.1 Events

Provides a general queue which accumulates the resource tasks: creation, deletion and renewal. Available representations are `application/json` and `application/xml`. It's recommended to use the events queue to supervise all your possible application specific resources. Further you only have to poll the events queue in order to supervise all your application specific resources and this further take down system load.

Remark: If you wisely decide to use the events queue to supervise your application specific resources you have to manage the validity of events queue yourself, i.e. you shouldn't additionally poll your application specific resources directly, because then you will get stale events in the events queue.

`/sys/events`

GET provides a list of events for the appropriate calling participant. Optionally the query string parameter *count* could be used to limit the amount of returned events.

/sys/events/fifo

GET provides an event (the oldest one) for the appropriate calling participant. Optionally the query string parameter *count* could be used to extend the amount of returned events. POST provides an event (the oldest one) for the appropriate calling participant and removes it from the events queue. Optionally the query string parameter *count* could be used to extend the amount of returned events.

Following a sample representation in JSON:

```
[
  {
    "status": "created",
    "ressource": "numlab/exercises/7"
  },
  {
    "status": "destroyed",
    "ressource": "numlab/exercises/3"
  }
]
```

3.7.2 Memberships

Provides information of the affiliation of the calling participant to the available communities. Available representations are `application/json` and `application/xml`.

/sys/memberships

GET provides a list of memberships for the appropriate calling participant. It implies all participants joining an appropriate community including the caller itself.

With the `itsyou` key the caller of the `/sys/memberships` resource will be informed which participant in the different communities is assigned to him.

Following a sample representation in JSON:

```
[
  {
    "community": {
      "name": "cc_courselinks",
      "description": "CampusConnect courselinks."
    },
    "participants": [
      {
        "name": "ILIAS-ECS Client 1",
        "itsyou": true,
        "org": {
          "name": "Leifos",
          "abbr": "LEI"
        },
        "mid": 1,
        "pid": 1,
        "description": "Development participant.",
        "dns": "n/a",
        "email": "meyer@leifos.com"
      },
      {
        "name": "FreeIT.de Testparticipant",
        "itsyou": false,
        "org": {
          "name": "FreeIT Softwaredevelopment.",
          "abbr": "FreeIT"
        }
      }
    ]
  }
]
```

```

    },
    "mid": 2,
    "pid": 4,
    "description": "A general test participant.",
    "dns": "n/a",
    "email": "Heiko.Bernloehr@FreeIT.de"
  },
  {
    "name": "ILIAS-ECS Client 2",
    "itsyou": false,
    "org": {
      "name": "Leifos",
      "abbr": "LEI"
    },
    "mid": 3,
    "pid": 7,
    "description": "",
    "dns": "n/a",
    "email": "meyer@leifos.com"
  }
]
}
]

```

3.7.3 Auths

This means authorization through one touch tokens. Provides a mechanism to grant each participant authorization to consume services from any service-providing-participant in an ECS network.

The interface is the same as for [application specific resources](#). If you want to create an authorization token, you have to provide at least a realm (authorization context) or a url (authorization context, DEPRECATED):

```
curl ... -X POST -d '{"realm":"authorization context string"}' https://.../sys/auths
```

and you will get back something like this:

```

{
  "hash": "5a944e72346e6e3102d32ccfecc18862d23e1dc0",
  "sov": "2011-03-08T23:25:27+01:00",
  "eov": "2011-03-08T23:26:27+01:00",
  "url": "authorization context string",
  "realm": "authorization context string",
  "abbr": "LEI",
  "pid": 35
}

```

hash

provides the authorization token (one touch token)

sov

stands for start of validation

eov

stands for end of validation

url

provides the authorization context (DEPRECATED)

realm

provides the authorization context (replaces url)

abbr

provides an abbreviation of the participant which has been created the authorization token (DEPRECATED, use pid as reference key in `sys/memberships` representation to get participant information)

pid

provides the participant id of the participant which has been created the authorization token

You're allowed to set the `sov` and/or `eov` to determine the validity period of the authorization token. If you do not, the validity period is set to one minute starting at current time.

The recommended way to fetch an authorization token when knowing the one touch hash:

```
curl .... -X DELETE https://.../sys/auths/<one touch hash>
```

This will return the auths representation (same structure/form as when creating; see above) and delete it server side. If the authorization token is outtimed, i.e. the current time is not between `sov` and `eov`, you will get back a return code 409 (conflict) and following descriptonal text in the body: *Authorization token outtimed.*

3.7.3.1 Courselinks

When using authorization tokens in a courselink context you are utilizing the `/sys/auths` resource additionally as a secure channel to provide a sha1 message digest of `Location-Header` values to the course hosting LMS for integrity. You have to store your message digest into the `realm` attribute, while calling for a authorization token.

The sha1 message digest should be calculated over the concatenation of the values of the courselink url and following [user redirect](#) query string parameters: `ecs_login`, `ecs_firstname`, `ecs_lastname`, `ecs_email`, `ecs_institution`, `ecs_uid_hash` or `ecs_uid`. Old systems use `ecs_uid_hash` and new systems use `ecs_uid`, because they only know this. As the value of both parameters should always be the same the sha1 digest will be of the same value too. Lets give a real example how to calculate such a digest:

```
courselink-url="https://ilias.uni-stuttgart.de/goto.php?target=crs_95034&client_id=
=USTGT"
ecs_login="DD12345"
ecs_firstname="Dagobert"
ecs_lastname="Duck"
ecs_email="Dagobert.Duck@cashstore-ag.com"
ecs_institution="Cashstore AG"
ecs_uid_hash="il_0_usr_13492"
ecs_uid="il_0_usr_13492"
```

Concatenate the values of the course-url and the appropriate query string parameters to one large string:

```
https://ilias.uni-stuttgart.de/goto.php?target=crs_95034&client_id=
USTGTDD12345DagobertDuckDagobert.Duck@cashstore-ag.comCashstore
AGil_0_usr_13492
```

And calculate the sha1 digest (Unix bash):

```
echo -n "https://ilias.uni ... il_0_usr_13492" | shasum -
=> e10f642b8fcc701a473296686edb0072063ea54f -
```

The same as Ruby snippet:

```
require 'digest/sha1'
Digest::SHA1.hexdigest 'https://ilias.uni ... il_0_usr_13492'
=> "e10f642b8fcc701a473296686edb0072063ea54f"
```

3.8 Application specific resources

All application specific resources have to be configured at ECS. There are three types of application specific resources:

1. messages
2. lists
3. queues

Generally resources are an abstract concept:

- clearly identifiable (in an HTTP context through URLs)
- have one or more representations (e.g. JSON, XML, text, ...)

According to resources it plays no role how a representation is produced. It could be done by returning a static file or running a complex server side application, that doesn't matter. Furthermore by looking at a resource you can't conclude how the representation has been made. An evaluation of a resource based on internal operations and circumstances, it is thus also negligible, and even be inadmissible.

3.8.1 Resource structure

```
/<projectnamespace>/<name>  
/<projectnamespace>/<name>/details  
/<projectnamespace>/<name>/<id>  
/<projectnamespace>/<name>/<id>/receivers  
/<projectnamespace>/<name>/<id>/details  
/<projectnamespace>/<name>/fifo  
/<projectnamespace>/<name>/lifo
```

3.8.2 Message resource

A message resource receives/saves messages for each participant. The participant can fetch (GET) his messages from the resource. A message resource could hold its messages enduringly (see Section 3.8.5), so new participants joining a community after a message has been sent to this community will also receive it.

GET

Returns message with status code 200.

DELETE

Deletes message and returns deleted resource representation with status code 200.

PUT

Renew message and returns with status code 200.

POST

Illegal call. Returns with status code 405 (Method Not Allowed).

Resource structure: `/<projectnamespace>/<name>/<id>`

3.8.2.1 Subresource details

You can ask for detailed (meta) information of a posted message. Only the original sender or a receiver can do that:

GET

Returns details about the requested message.

Resource structure: `/<projectnamespace>/<name>/<id>/details`

You will get back something like this:

```
{
  "receivers": [
    {
      "itsyou": false,
      "mid": 1,
      "cid": 2
      "pid": 19,
    },
    {
      "itsyou": false,
      "mid": 4,
      "cid": 3
      "pid": 29,
    }
  ],
  "senders": [
    {
      "mid": 5
    },
    {
      "mid": 7
    }
  ],
  "url": "courselinks/10",
  "content_type": "application/json"
  "owner": {
    "pid": 3,
    "itsyou": true
  }
}
```

The "receivers" and "senders" have corresponding arrays: The first array entry in "senders" has been addressed the first array entry of "receivers" and so on.

3.8.3 List resource

GET

Returns URI message list with status code 200. If there are no messages to list the HTTP body will be empty (Content-Length: 0). The Content-Type will be `text/uri-list`. The URI list will be represented by [relative references](#). URIs are specified in [RFC3986](#).

DELETE

Illegal call. Returns with status code 405 (Method Not Allowed).

PUT

Illegal call. Returns with status code 405 (Method Not Allowed).

POST

Creates new message, returns with status code 201 and a HTTP header `Location`: providing the new message URI.

Resource structure: `/<projectnamespace>/<name>`

3.8.3.1 Subresource details

Now it's possible to ask for detailed (meta) information of a list resource. All querystrings supported my normal list resources could be used. Only the original sender can do that:

GET

Returns details about all resource URIs listed.

Resource structure: `/<projectnamespace>/<name>/details`

You will get back something like this:

```
[
  {
    "senders": [ ],
    "receivers": [ ],
    "url": "courselinks/35",
    "content_type": "text/plain",
    "owner": {
      "pid": 3,
      "itsyou": true
    }
  },
  {
    "senders": [
      {
        "mid": 2
      }
    ],
    "receivers": [
      {
        "mid": 19,
        "cid": 2,
        "pid": 19,
        "itsyou": false
      }
    ],
    "url": "courselinks/36",
    "content_type": "text/plain",
    "owner": {
      "pid": 3,
      "itsyou": true
    }
  },
  {
    "senders": [
      {
        "mid": 2
      }
    ],
    "receivers": [
      {
```

```

      "mid": 19,
      "cid": 2,
      "pid": 19,
      "itsyou": false
    }
  ],
  "url": "courselinks/37",
  "content_type": "text/plain",
  "owner": {
    "pid": 3,
    "itsyou": true
  }
}
]
]

```

The first element of the returned array of the details list subresource probably needs some explanation. Both senders and receivers are empty lists. This means that the appropriate message isn't any more addressed to any participant. This further implies that all participants which had been addressed in the past have been received the message from their appropriate resource. But why was the message then not deleted? Because the resource has been configured to be "postrouted". If that has not been the case, ECS would have removed the message.

3.8.3.2 Querystrings

To affect the returned representation you could assign the following querystrings to X-EcsQueryStrings header variable:

receiver

It's possible to filter the returned index from a list resource to only those items to which the calling participant was formerly an addressed receiver (this is also the default, therefore it could be omitted):

```

curl .... -H 'X-EcsQueryStrings: receiver=true' -X GET https://server/< namespace>/<name>

```

sender

It's possible to filter the returned index from a list resource to only those items to which the calling participant is the original sender:

```

curl .... -H 'X-EcsQueryStrings: sender=true' -X GET https://server/<namespace <
>/<name>

```

all

It's possible to filter the returned index from a list resource to show all messages either as addressed receiver or as original sender:

```

curl .... -H 'X-EcsQueryStrings: all=true' -X GET https://server/<namespace>/< name>

```

Using the X-EcsQueryStrings header variable is the recommended way to use querystrings. If you have to assign multiple querystrings please delimit the querystrings by comma (,).

Of course you can also specify the querystring by appending it to the end of the resource url, e.g.

```

curl .... -X GET https://server/<namespace>/<name>?all=true

```

3.8.4 Queue resource

The queue resource is modelled as a subresource of a list resource and it can operate either in lifo (last in first out) or fifo (first in first out) mode.

GET

Returns last (lifo) or first (fifo) message with status code 200. If there are no more messages in queue you will get an empty message (`Content-Length: 0`) and also status code 200.

DELETE

Illegal call. Returns with status code 405 (Method Not Allowed).

POST

Returns last (lifo) or first (fifo) message with status code 200 and deletes it. If there are no more messages in queue you will get an empty message (`Content-Length: 0`) and also status code 200.

PUT

Illegal call. Returns with status code 405 (Method Not Allowed).

Resource structure: `/<projectnamespace>/<name>/fifo` or `/<projectnamespace>/<name>/lifo`

3.8.5 Postrouting

If a resource has set its postroute flag, then all new participants will get postrouted this resource e.g. if you have posted some messages to a community named testcommunity and later joins a new participant to this community, he will get postrouted the former posted messages.

3.9 JSON-Schemas

A json media type for describing the structure and meaning of json documents. It's **defined** as an Internet-Draft working document of the IETF. There is also a homepage where you can start to discover more over **JSON-Schemas**.

All resource representations must have a `Content-Type` header variable containing an optional parameter `profile` pointing to its describing schema. For a respond on a `/campusconnect/courses` request this could be:

```
Content-Type: application/json; \
  profile=http://repo.or.cz/w/ecs.git/blob_plain/ \
  e5cc81b2201ac24294d2ac3e732f9ddac954cc84:/ \
  campusconnect/schemas/cc_courses.schema.json
```

It's up to you to validate and check the received data against the provided schema or to decide if you are able to process the format just receiving. There is always a version id inbetween the `profile` URL representing the commit id of the git repository. For the last `Content-Type` example this was `e5cc81b2201ac24294d2ac3e732f9ddac954cc84`. You can always ask for the latest schema of an appropriate resource by using `HEAD` as the version id.

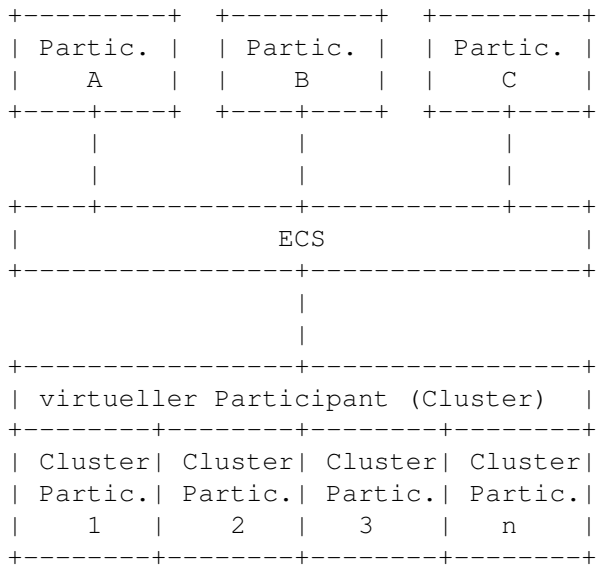
Of course you can use the schema of an appropriate resource for discovering the names and types of the data elements in order to match them dynamically to other internal meta data of your application.

3.10 Participant Cluster

The ECS is able to cluster participants. In the ECS network a cluster is seen as an ordinary participant.

3.10.1 Cluster building

First lets show the topology of a clustered ECS network:



1. The ECS registers a virtual participant. All cluster participants use this registration, i.e. the ECS doesn't know which cluster participant is communicating. This way you can scale your cluster easily by attaching another cluster participant also using the previously generated virtual participant registration. You don't have to make any further settings at ECS.
2. If you want to send a message to the cluster you only have to send it to the virtual participant. When all cluster participants compete against each other to get a message this would maybe the simplest resource access mode (message dispatching). Every cluster participant have to access the appropriate resource as a queue resource via DELETE method. This assures that every message could only be fetched by one cluster participant.

3.10.2 Cluster broadcasting

In order to explicitly communicate with a cluster participant we have to use a broadcasting mechanism. Every resource could be used as a broadcasting resource. It only depends on how the cluster participants access this resource. They have to do it like this:

1. Every cluster participant checks the broadcast resource as a queue resource with the idempotent GET method and decides by looking inside the message if this message is targeted to him. If it does belong to him he should compute and DELETE the message.
2. The ECS garbage collects the broadcast resource at a default time period.

Chapter 4

Filter plugins

Messages could be changed at runtime by so called filter plugins. These filters could be attached to 5 different queues and triggered by one of the actions hereafter. The filter queues were mapped to a special path under the filesystem:

1. Show filter. Triggered when calling a message/queue resource with GET. Filter path:
filter/<project-name-space>/<resource-name>/show/filter-name>
2. Index filter. Triggered when calling a list resource with GET. Filter path:
filter/<project-name-space>/<resource-name>/index/filter-name>
3. Create filter. Triggered when calling a list resource with POST. Filter path:
filter/<project-name-space>/<resource-name>/create/filter-name>
4. Update filter. Triggered when calling a message/queue resource with PUT. Filter path:
filter/<project-name-space>/<resource-name>/update/filter-name>
5. Delete filter. Triggered when calling a message/queue resource with DELETE. Filter path:
filter/<project-name-space>/<resource-name>/delete/filter-name>

You're able to create as many filters you want. They will be all queued/concatenated in lexical order:

```
unfiltered +-----+ +-----+ +-----+ filtered
----->| 1-fil |-->| 2-fil |...| n-fil |----->
message +-----+ +-----+ +-----+ message
```

If a filter was created and copied into the appropriate filesystem path, it would be automatically activated at runtime without additional configuration.

If there are any exceptions while reading (class loading) the filter the appropriate filter will be canceled and the processed message will be queued to the next one. If there are any exceptions while running the filter the appropriate filter will also be canceled and the processed message will also be queued to the next filter, but keep in mind, that all changes to the message prior to the occurred exception will remain. You are always working with the original message (no copy). There will be error messages in the logfile of the form: "ERROR Filter Error: ...".

4.1 Template

In the <filter-name> directory must be at least a file called "filter.rb" with following structure:

```
module Filter
  def self.start
    ...
  end
end
```

ECS will call "Filter.start". From there its on you :) ECS will also load any file with ".rb" extention under the directory <filter-name> into the namespace of "Filter".

The ECS core provides the constant `FILTER_API` as an API for accessing ECS messages:

FILTER_API.params

It's a hash to access the qreystings of message call:

```
http://ecs.rus.uni-stuttgart.de/numlab/exercises/23?properties=name, ↔
  description
...
elements = FILTER_API.params["elements"].split(",")
...
```

FILTER_API.record

This object provides access to the message body:

```
message = FILTER_API.record.body
```

Chapter 5

CampusConnect

5.1 Scriptable REST commandline client

As the ECSA is derived from an architecture style for distributed systems called REST, the major tool for developing in an ECSA context is a scriptable REST commandline client. An excellent choice would be `curl`.

5.1.1 Monitoring resources

Following code monitors the events resource every 5 seconds and shows/highlights the differences in the output:

```
watch -n 5 --differences=cumulative --no-title \  
'curl -i --cacert /path/to/freeit-root-ca.cert.pem \  
  --cert /path/to/participant.cert.pem \  
  --key /path/to/participant.key.pem \  
  --pass "secure pwd" \  
  -X GET https://ecsc.uni-stuttgart.de/ra/ecs-test/sys/events'
```

5.2 Simulating LSFproxy

To develop without a running LSFproxy you could simply provide static resources inside the ECSA network. At ECS you have to create the specific CampusConnect resources (`/campusconnect/courses`, `/campusconnect/directory_trees` and `/campusconnect/course_members`). The essential LSFproxy communication to ECS could be substituted with curl scripts/calls. In presence of a LSFproxy, its resource representations has to be retrieved indirectly through ECS (see Section 2.5.1). For each development party you should create a separate LSFproxy participant at ECS. For easier usage we provide a **shell script** simulating an LSFproxy. Please have a look at the top of the script and adjust the appropriate lines. For development we provide an ECS, LSFproxy certificates and LSFproxy data. Please contact us ([info\[at\]freeit\[dot\]de](mailto:info@freeit.de)). Of course you can drive all parts by your own and have to fill out the lines by your personal data.

5.2.1 Create

To POST new LSFproxy course to a community participant (e.g. a LMS) call the script like this:

```
./lsfproxy.sh -c -k 11 -u 1 create  
  
-c ... you want to work with courses  
-k ... you address membership 11  
-u ... static course data with DATA_URL1 (see top of LSFproxy.sh)  
create ... POST this message to ECS (URL of ECS see top of LSFproxy.sh)
```

For a new LSFproxy directory-tree:

```
./lsfproxy.sh -t -k 11 -u 3 create

-t ... you want to work with trees
-k ... you address membership 11
-u ... static tree data with DATA_URL3 (see top of LSFproxy.sh)
create ... POST this message to ECS (URL of ECS see top of LSFproxy.sh)
```

You always have to choose the appropriate DATA_URL (see top of LSFproxy.sh) that corresponds to the resource you want to operate on ECS. Feel free to download these static representations from the URLs assigned to the DATA_URL constants at top of the script and setup your own static LSFproxy data representations at your local web sever.

5.2.2 Delete

After you have created some messages, you can of course DELETE them (you are the owner). But first you have to distinguish the resource id on ECS. Either you have noticed the id at creation time looking at the provided Location header from ECS or you request first a resource listing and second request a resource representation from this id checking it is the right one before deleting it:

```
./lsfproxy.sh -c get
-c ... you want to work with courses
get ... get a list of the resource

./lsfproxy.sh -c -i 2445 delete
-c ... you want to work with courses
-i ... resource id
delete ... delete resource
```

5.2.3 Update

In order to update a resource call the script like this:

```
./lsfproxy.sh -c -i 2445 -k 11 -u 4 update
-c ... you want to work with courses
-i ... resource id
-k ... you address membership 11
-u ... static course data with DATA_URL4
update ... PUT this message to ECS
```

5.2.4 Show Memberships

You can also show your memberships by calling:

```
./lsfproxy.sh -s get
```

5.2.5 Show Usage

And last you can show a usage help text by calling the script with no parameters and options or with the `-h` option:

```
./lsfproxy.sh -h

Usage: lsfproxy.sh options <create|get|delete|update>
Options:
  -c ... courses
  -m ... course members
```

```
-t ... directory trees
-s ... memberships
-i <resource id>
-k <membership id>
-u <data url id>
-v ... verbose output
-h|? ... usage
```

Please remember, this script is only a help utility, not full-fledged application. So please adjust it to your needs.

5.3 Resources

All resource representations returned by the ECS have a `Content-Type` header. If this header has a value (mime type) of `text/uri-list` you have to process the returned URIs in order to get your resource of interest. Normally this would be just one URL which you should fetch (GET) again (see also Section 2.5.1).

5.3.1 /campusconnect/courses

[example](#), [schema](#)

- events generation is turned on
- persistent resource
- allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).

5.3.2 /campusconnect/organisation_units

[example](#), [schema](#)

- events generation is turned on
- persistent resource
- allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).

5.3.3 /campusconnect/terms

[example](#), [schema](#)

- events generation is turned on
- persistent resource
- allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).

5.3.4 /campusconnect/course_members

This resource representation is of `Content-Type:text/uri-list` and points to the real `course_members` representation on LSF-Proxy (see also Section 2.5.1). This is necessary because of sensible data not allowing to be transferred out of control of the organization.

- events generation is turned on
 - persistent resource
 - allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).
-

5.3.4.1 LSF-Proxy course_members representation

[example](#), [schema](#)

- allowed HTTP methods: GET

5.3.5 /campusconnect/directory_trees

```
{
  "rootID": <string>, ❶
  "directoryTreeTitle": <string>,
  "term": <string>,    //optional
  "nodes": [
    {
      "id": <string>,
      "title": <string>,
      "order": <integer>, //optional ❷
      "parent":
        {
          "id": <string>,
          "title": <string> //optional
        }
    },
    ...
  ]
}
```

- ❶ The root node itself constitutes a real node in the tree and therefore should be instantiated like every other node in the "nodes" array. Because it's the root node there are no sibling and parent nodes but only child nodes.
- ❷ If elements (nodes) are on the same level (plane), i.e. have the same parent, use this number to sort them in ascending order. Otherwise order them as they appear in the resource representation.

Schema definition.

- Events generation is turned on.
- Persistent resource.
- Allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).
- A PUT updates always whole trees.

5.3.6 /campusconnect/courselinks

[example](#), [schema](#)

- events generation is turned on
- persistent resource
- allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).

5.3.7 /campusconnect/course_urls

Is used to inform the Proxy-LSF about a course URL.

schema

- events generation is turned on
- persistent resource
- allowed HTTP methods: POST, GET, PUT, DELETE (see Section 3.8).

example:

```
{
  "cms_lecture_id": "lsfproxy.uni-stuttgart.de/443889",
  "ecs_course_url": "campusconnect/course/815",
  "lms_course_urls": [
    {
      "title": "Mathematics I - Group 1",
      "url": "http://dummy.lms.com/course/12345a"
    },
    {
      "title": "Mathematics I - Group 2",
      "url": "http://dummy.lms.com/course/12345b"
    },
    ...
  ]
}
```

- ❶ e.g. course id from LSF (campus management system)
- ❷ ECS resource URL to backreferencing course
- ❸ List of course URLs on learning management system (especially for parallel groups when established as separate courses).

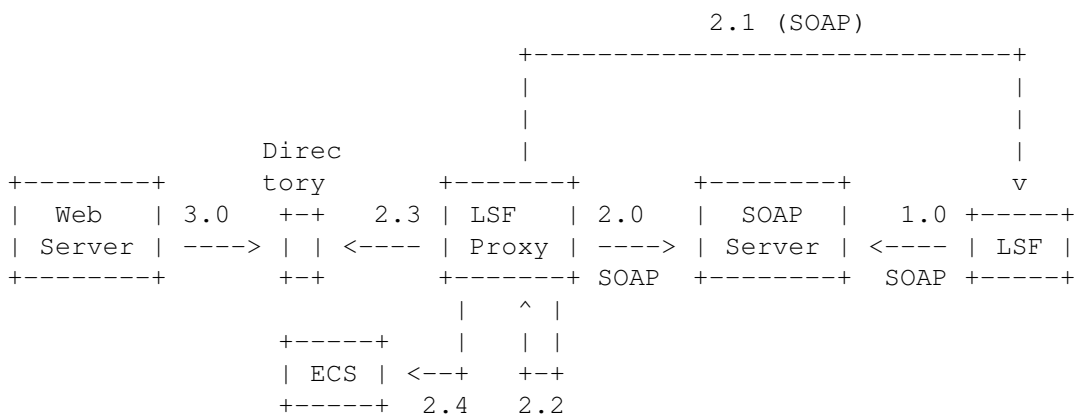
5.4 LSF-Proxy

It's responsible for connecting HIS/LSF into an ECSA network. Therefore the proxy get information from HIS/LSF and after preprocessing provides them in the ECSA network.

DESIGN, ARCHITECTURE AND TECHNOLOGY

- It's a Client-/Server Application implemented with "Ruby on Rails".
- **Apache** webserver and **Phusion Passenger** application server.
- **SQLite** database backend.
- Resource representations are stored on file system level.
- see Section 2.1.
- see Section 2.2.
- see Section 2.3.

LSF-Proxy to HIS-LSF communication:



1.0:changeEvent(Object.type,Object.id)

LSF sends a change event (notification) to our SOAP server triggered from his internal change recording. The SOAP server queues all notifications.

1.1:push(Object.type,Object.id)

The SOAP server sends the received notification data straight forward to a persistent, distributed queue.

2.0:Object:=pop()

LSF proxy pops (fifo) a notification from the the SOAP server (queuing all notifications).

2.1:LsfData:=getDataXML(Object.type,Object.id)

LSF proxy fetches (via SOAP call) data from LSF (referenced by the previously popped notification).

2.2:EcsData:=process(LsfData)

LSF proxy processes the fetched LSF data.

2.3:createResourceRepresentation(EcsData)

LSF proxy generates resource representations.

2.4:POST(ResourceURL)

LSF proxy inform ECS about the generated resource representations.

3.0:ResourceRepresentation:=GET(ResourceName)

A participant fetches a resource representation.

PROTOCOL OBJECT PUSHER (ENHANCED LSF PROTOCOL LISTENER)

- It's coupled at LSF recording ¹.
- Changes on LSF-objects through LSF-users triggers LSF-recording and hence a push info to subscribed LSF-Proxy by sending the appropriate LSF-ObjectIDs/ObjectTypes.

DB-INTERFACE

- The LSF-Proxy calls the LSF-DBInterface about the changed objects, process and provide them to the interested participants as resource representations via webservice.

In an ECSA network the LSF-Proxy takes the role of a participant. Therefore have a look at [participants](#). Especially for a [communication procedure between LSF-Proxy ↔ ECS ↔ LMS](#).

It's also worth to have a look at [LSF-Proxy simulating](#).

¹ in german: LSF-Protokollierung

5.5 Redirect procedure/protocol when consuming courselinks

When a user clicks a courselink on LMS_A he will be redirected to the course hosting platform LMS_B through a HTTP `Location-Header` representing the course URL. But the LMS_A appends some further query strings to the course URL:

ecs_hash

One touch token, e.g. `d862c2f0d6493d9a0dfc891896de1debb7cc0556`

ecs_hash_url

One touch token with following structure (DEPRECATED, use `ecs_hash`):

```

foo://example.com:8042/intermediate_path/sys/auths/hashid
  \_/      \_____/      \_/ \_____ /
  |         |           |         |
scheme     ECS         port      Path to one touch token
name      hostname
  |
http/https

```

As usual the port is optional and conforms to its well known port if omitted (`http`→80 and `https`→443). And here a real example: `https://ecsc.uni-stuttgart.de:443/ra/ecs/sys/auths/d862c2f0d6493d9a0dfc891896de1debb7cc0556`

ecs_login

Describes the login name of redirected user.

ecs_firstname

Describes the first name of redirected user.

ecs_lastname

Describes the last name of redirected user.

ecs_email

Describes the email address of redirected user.

ecs_institution

Describes the institution from where the redirected user comes from.

ecs_uid_hash

A unique user id (DEPRECATED, new name `ecs_uid`).

ecs_uid

A unique user id.

Before LMS_B provides the course to the caller several things happen:

- LMS_B checks `ecs_hash_url` query string against ECS (`sys/auths`). If the hash is invalid the caller will be rejected (see also Section 3.7.3) otherwise caller will be considered authorized. Further LMS_B checks a `sha1 message digest` provided by `realm` attribute from the authorization token.
- The remaining query strings will be used by LMS_B to create an appropriate login/session for the caller (for details how this have to be done in CampusConnect please have a look at the appropriate [use case](#)).

Chapter 6

Interconnectivity

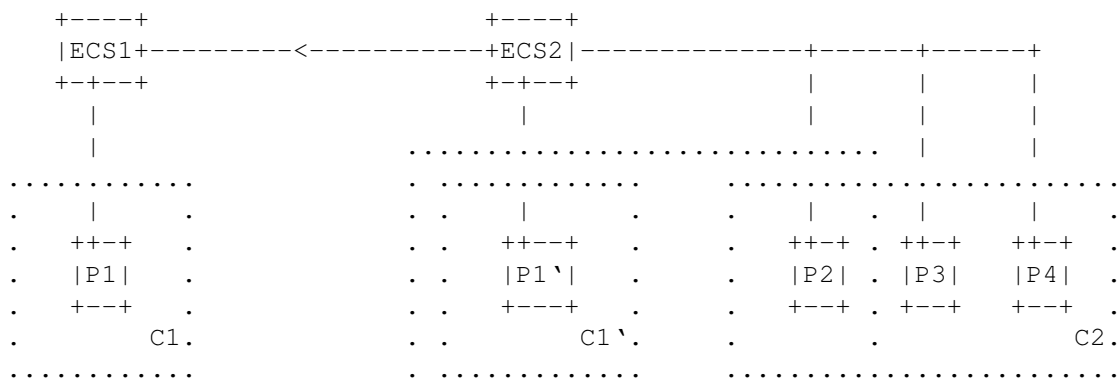
We want to interconnect several ECSs without any change to their joined participants. In other words this interconnection should be fully transparent to the appropriate participants. So there is no need for extra registration overhead to other ECSs. From the viewpoint of a participant there are just some more participants joining the communities of its primary ECS.

6.1 Necessary extensions to ECS

- In order to make use of event generation system the `sys/memberships` resource should be mapped to a normal application resource.
- Make resources model an application resource `sys/resources`.
- participants should get a new attribute `foreign` which marks them as members from another ECS.
- resources should get a new attribute `foreign` which marks them coming from another ECS.
- communities should get a new attribute `foreign` which marks them coming from another ECS.
- ECS must provide full participant functionality.

6.2 Interconnection procedure

- To bidirectional interconnect two ECSs they both have to be participants of each other i.e. they have to register themselves to each other.
- To unidirectional interconnect two ECSs only one of the ECS has to be registered at the other ECS. Then the registered ECS(2) provides its participants addressability to the participants of the registering ECS(1).



. .
. C3.
.....

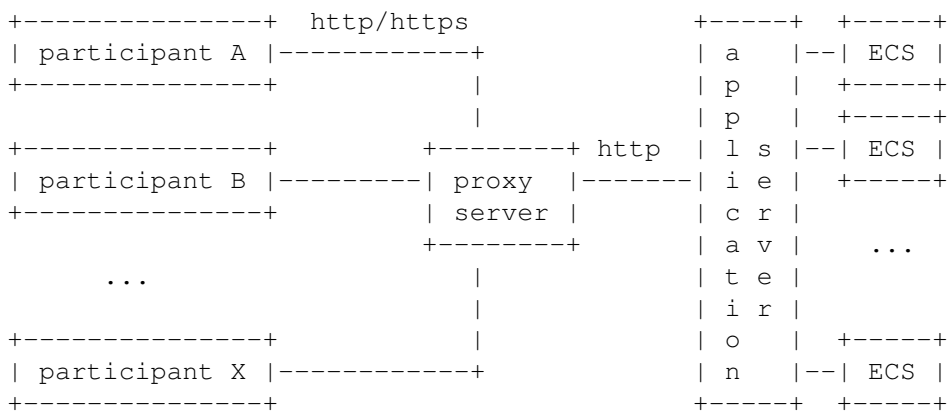
- As each normal participant, the ECS has to join the communities in interest. In opposite to a normal participant the ECS could not be seen by the other participants joining the same community hence they are not able to address an ECS directly but the ECS can determine them by reading its `sys/memberships` resource. Now the ECS is able to create the necessary remote participants and communities on his side and also join the new created remote participants to the right communities. This repeats whenever the ECS reads its `sys/memberships` resource. These remotely imported participants, communities and resources could not be altered in any way by the importing ECS.
 - By calling `sys/resources` the ECS is also able to import and create all the resources from the remote ECS.
 - Remotly added participants, communities and resources must not be imported by further ECSs. This prevents loops and retains full access determination of participants, communities and resources of the origin/owning ECS.
-

Chapter 7

Installation

As a matter of principle you should be able to install the ECS on all systems which are capable of running Ruby-Rails. Because of the differences between the systems I'm showing the installation process based on particular systems.

The ECS itself doesn't care about authentication of participants. This has to be done by a proxy server ([NGINX](#), Apache) in front of ECS, e.g. authentication through certificates or HTTP basic auth, As a result of such an authentication process the ECS will be informed through the X-EcsAuthId http header which identifies the calling participant uniquely.



Also the resource protection has to be done through proxy server (CRUD). Normally this means at least to protect /admin resource. Otherwise everybody could access these resources.

An excellent choice for the application server would be [Unicorn](#). Another good one will be [PHUSION Passenger](#).

7.1 Base installation on GNU/Linux Debian (Squeeze)



Important

You should use Debian Wheezy (stable Debian release) instead the old Debian Squeeze. The installation process should be nearly the same. But first have a look at [Debian Wheezy](#), because there are minor but important differences.

I'll show the exact installation procedure in reliance on a Debian Squeeze System. But of course you could use the actual [stable Wheezy](#) system as well. Just keep in mind, that you have to install the ruby1.8 package (version 1.8.7).

The base installation provides only a local running ECS. It's by no mean for using as a public or production installation. Therefore you additionally have to install a HTTP-/HTTP-Proxy Server and maybe a more production capable application server.

To obtain the ECS source code you have two choices:

1. Using git :

```
sudo aptitude install git
```

And clone the repository:

```
git clone http://repo.or.cz/r/ecs.git
```

After cloning you will notice the new application directory `ecs`.

2. Download an ECS snapshot (tar.gz or zip archive). Please browse to [ecs repository](#) and choose the newest stable snapshot available (normally tagged as "master"). After unpacking the archive you will notice the new application directory `ecs`.

All further operations have to take place inside this application directory:

```
cd ecs
```

You must install a ruby of version 1.8.7 (the latest 1.8.x version):

```
sudo aptitude install ruby
```

The following NEW packages will be installed:
libreadline5{a} libruby1.8{a} ruby ruby1.8{a}

You must install a rubygems of version 1.3.7 or later:

```
sudo aptitude install rubygems libsqlite3-dev
```

The following NEW packages will be installed:
binutils{a} build-essential{a} bzip2{a} cpp{a} cpp-4.4{a} dpkg-dev{a}
fakeroot{a} g++{a} g++-4.4{a} gcc{a} gcc-4.4{a} libalgorithm-diff-perl{a}
libalgorithm-diff-xs-perl{a} libalgorithm-merge-perl{a} libc-dev-bin{a}
libc6-dev{a} libdb4.7{a} libdpkg-perl{a} libgmp3c2{a} libgomp1{a}
libmpfr4{a} libstdc++6-4.4-dev{a} libtimedate-perl{a} linux-libc-dev{a}
make{a} manpages-dev{a} patch{a} perl{a} perl-modules{a} ruby1.8-dev{a}
rubygems rubygems1.8{a} libsqlite3-dev

Add the bin path of the local gem directory to the PATH environment variable:

```
export PATH=/home/your_user_name/.gem/ruby/1.8/bin:$PATH
```

Tip

Please consider to place the export command into an appropriate startup file (i.e. `.profile` or `.bashrc`).

You must install the bundler gem:

```
gem install bundler --user-install
```

You have to call:

```
bundle install --path=/home/your_user_name/.gem
```

Please standby until all further prerequisites are installed. Have an eye especially when it comes to installing the so called "native extensions". At any time during this installation process there should be no error message.

**Important**

The two previous commands should be called as a normal user (not root). This has the advantage, that all your gems (plugins) are installed locally (not system wide) under your user homedirectory.

Setting up the development database:

```
rake db:setup
```

Now running test cases. All should go well. Otherwise see error messages.

```
rake test
```

Now you can start the local WEBrick HTTP-Server:

```
./script/server
```

And point your Webbrowser to: <http://localhost:3000>. There you should see the ECS administration web interface.

Just for info:

```
gem list
```

```
*** LOCAL GEMS ***
```

```
actionmailer (2.3.14)
actionpack (2.3.14)
activerecord (2.3.14)
activeresource (2.3.14)
activesupport (2.3.14)
bundler (1.1.3)
columnize (0.3.6)
haml (3.1.4)
i18n (0.3.7)
json (1.5.3)
jsonpretty (1.1.0)
linecache (0.46)
rack (1.1.3)
rails (2.3.14)
rake (0.9.2.2)
rbx-require-relative (0.0.5)
rdoc (3.12)
ruby-debug (0.10.4)
ruby-debug-base (0.10.4)
simple-navigation (3.6.0)
sqlite3 (1.3.5)
```

7.2 Base installation on GNU/Linux Debian Wheezy

It should really going the same way as in [Debian Squeeze](#) just take care that you install `ruby1.8` instead of just the `ruby` package because in Wheezy the `ruby` package will be associated with the `ruby1.9.x` package.

7.3 NGINX HTTP-Proxy-Server

It's a fast, slim and reliable HTTP-Proxy-Server and the recommended one in conjunction with the unicorn application server. If you are running Debian Wheezy you're ready to use it from the standard Debian repository otherwise you have to compile nginx your own (you need a nginx version with lua enabled).

On Debian Wheezy just install nginx from your repository (nginx-extras is a lua enabled nginx version):

```
sudo apt-get install nginx-extras
```

If you want to authenticate your participants via X.509 certificates, then just make a new virtual server in `/etc/nginx/sites-available/` and link it into `/etc/nginx/sites-enabled/`:

```
upstream unicorn_ecs {
    server localhost:8080;
}

server {
    listen 443 ssl;
    server_name          ecs.localhost.local;
    ssl_protocols        SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers           HIGH:!aNULL:!MD5;
    ssl_session_cache    shared:TLSSL:16m;
    ssl_session_timeout  10m;
    ssl_certificate       /path/to/server-cert.pem; ❶
    ssl_certificate_key   /path/to/server-key.pem;
    ssl_client_certificate /path/to/ca-root-cert.pem
    ssl_verify_client on;
    keepalive_timeout    70;

    proxy_set_header    X-Forwarded-Proto https;
    proxy_set_header    Host $http_host;
    set $serial "nil";
    set $underscore "_";
    set $email "nil";
    if ($ssl_client_verify != "SUCCESS") {
        return 400 "client certificate failure\n";
    }
    set_by_lua $email "return string.match(ngx.var.ssl_client_s_dn, '.*emailAddress ←
    =(.*)')"; ❷
    proxy_set_header    X-EcsAuthId $ssl_client_serial$underscore$email; ❸

    location / {
        proxy_pass http://unicorn_ecs;
        allow all;
    }
    location /admin { ❹
        proxy_pass http://unicorn_ecs;
        if ($email != "fritz.lang@example.com") {
            return 401 "You're not allowed to access this resource.";
        }
    }
}
```

❶ When using a self signed CA you have to chain both certs:

```
cat server-cert.pem ca-root-cert.pem > chained-server-cert.pem
```


- 2 Parsing the email address out of the client subject dn.
- 3 Setting our X-EcsAuthId to form a unique participant authorization-id.
- 4 Resource protection. Of course you could use any access protection method provided by proxy server.

Tip

You can test if a certain nginx installation provides lua functionality. Just type `/usr/sbin/nginx -V` and search for the string `lua`.

On Squeeze there is no sufficient nginx in the standard repositories. Therefore I suggest to use Apache as the Proxy-Server or compile nginx with lua support your own.

7.4 Unicorn application server

To use this excellent ruby/rails application server you have to change into the ECS base/installation directory and call:

```
gem install unicorn
```

To start unicorn just call:

```
unicorn_rails -E <environment>
```

As `<environment>` just use `development` or `production`.

7.5 CampusConnect

To use the ECS in an CampusConnect context you have to create some [resources](#). Of course you can do that manually but it's much easier to use a special template. Change into the application directory (`ecs`) and call:

```
rake rails:template LOCATION=http://repo.or.cz/w/ecs_templates.git/blob_plain/HEAD:/ ↔  
ecs_campusconnect.rb
```

7.6 ViP

To use the ECS in a **ViP**¹ context you have to create some resources and modify the configuration file. Change into the application directory (`ecs`) and call:

```
rake rails:template LOCATION=http://repo.or.cz/w/ecs_templates.git/blob_plain/HEAD:/ecs_vip ↔  
.rb
```

¹ ViP - virtual programming lab

Chapter 8

Glossary

DRY

don't repeat yourself

CRUD

create, read, update and delete

ECC

elearning community client

ECP

elearning community proxy

ECS

elearning community server

ECSA

elearning community service architecture

IETF

internet engineering task force

LMS

learning management system

MOM

message orientated middleware

Chapter 9

Index

A

- anonymous participants, [10](#)
- anonymous-participants, [10](#)
- application resources, [14](#)
- architecture, [3](#)
- authentication, [3](#), [10](#)
- authorization, [3](#)
- auths resource, [12](#)

C

- CampusConnect, [22](#)
 - developing, [22](#)
 - installation, [35](#)
- communication, [4](#)

D

- Debian, [31](#), [33](#)
- developing, [22](#)

E

- ECS, [7](#)
 - anonymous participants, [10](#)
 - application resources, [14](#)
 - authentication, [10](#)
 - auths resource, [12](#)
 - events resource, [10](#)
 - memberships resource, [11](#)
 - selfrouting, [9](#)
 - system resources, [10](#)
- ECSA
 - overview, [1](#), [3](#)
- events resource, [10](#)

G

- git, [32](#)

I

- installation, [31](#), [35](#)
 - Debian, [31](#), [33](#)
 - git, [32](#)
 - NGINX, [34](#)
 - Squeeze, [31](#)
 - unicorn, [35](#)
 - Wheezy, [33](#)

L

- LSF, [26](#)
- LSF-Proxy, [26](#)

M

- memberships resource, [11](#)

N

- NGINX, [34](#)

O

- overview, [1](#), [3](#)

P

- participant
 - architecture, [3](#)
 - authentication, [3](#)
 - authorization, [3](#)
 - communication, [4](#)
 - resource extensions, [3](#)
 - technology, [3](#)
- proxy
 - LSF, [26](#)

R

- resources, [10](#), [14](#)
- resource extensions, [3](#)

S

- selfrouting, [9](#)
- Squeeze, [31](#)
- system resources, [10](#)

T

- technology, [3](#)

U

- unicorn, [35](#)

V

- ViP, [35](#)
 - installation, [35](#)

W

- Wheezy, [33](#)