Abstract:

*Detailed Specification & Design v3.0 D11.8*: Detailed specification of the KAMP Mk2 final system updated according to the system adjustments;
| **Keywords:** Detailed Specification |
| **Due Date:** 30/06/99 |
### Table of contents

1. **Introduction** .................................................. 5
   1.1. A general technical description of KAMP system .......................... 5
       1.1.1. How the system works ........................................... 5
           1.1.1.1. The client side ........................................... 6
           1.1.1.2. The server side ........................................... 7

2. **Database description** .......................................... 9
   2.1. Database structure ........................................... 9
   2.2. Description of the relationships ................................ 31

3. **Tools description** ............................................. 33
   1. Kampus system .................................................. 33
       1.1. Registration .................................................. 34
   1.2. Staff tools ................................................... 38
       1.2.1. The Course Designer ......................................... 40
           1.2.1.1. Asset designer ......................................... 42
           1.2.1.2. Activity designer ..................................... 46
           1.2.1.3. Module designer ....................................... 52
           1.2.1.4. Course designer ....................................... 59
           1.2.1.5. Copyright designer .................................... 66
       1.2.2. Course Administration ..................................... 70
           1.2.2.2. Group administration .................................. 76
           1.2.2.4. Results administration ................................ 85
           1.2.2.5. User administration .................................... 90
       1.2.3. The Learning Materials Base Manager ............................ 97
           1.2.3.1. Case Study Archive Manager ............................. 99
           1.2.3.2. Bibliography Manager ................................... 105
           1.2.3.3. Background Studies Manager ............................. 110
           1.2.3.4. Working Documents Manager ............................. 114
           1.2.3.5. Interviews Manager ..................................... 118
           1.2.3.6. Student Works Manager ................................ 122
           1.2.3.7. Software and Tools Archive Manager .................... 127
           1.2.3.8. Topic keywords Manager ................................ 132
       1.2.4. The Telematic Services Base Manager .......................... 137
           1.2.4.1. Mailing List Manager ................................... 138
           1.2.4.2. Discussion Groups Manager .............................. 143
           1.2.4.3. Chat Rooms Manager .................................... 146
   1.3. Student tools ................................................ 149
       1.3.1. Course delivery ............................................ 150
       1.3.2. Agenda ..................................................... 153
       1.3.3. Learning Materials Base .................................... 155
           1.3.3.1. Case studies browser .................................... 156
           1.3.3.2. Documents browser ..................................... 159
           1.3.3.3. Software and Tools browser ............................ 162
           1.3.3.4. Materials MAP .......................................... 164
           1.3.3.5. Topic Index - Glossary ................................ 165
           1.3.3.6. Search .................................................. 166
   1.4. Telematic Services Facilities .................................. 169
       1.4.1. Email ..................................................... 169
       1.4.2. File Transfer ................................................ 170
4. Installation procedures

4.1. General Description of KAMP architecture

4.1.1. Definition of the MASTER Site

4.1.2. Definition of a SLAVE Site

4.1.3. The Replication mechanism

4.1.3.1. Collection of the operations to replicate

4.1.3.2. Transfer of the list of operations

4.1.3.3. Update of slave databases

4.2. Site Installation HOW-TO

4.2.1. Preinstallation checks

4.2.2. Restore tar archive

4.2.3. Create KAMP users

4.2.4. Change configuration files

4.2.4.1. Changes in "/etc/aliases" file

4.2.5. Start-up/shutdown scripts

4.2.6. Services handled by inetd

4.2.7. New crontab entries
1. Introduction

This document is structured in four different chapters. The first one, the one you are reading, is an introduction that contains a general technical description of the system. The second chapter is related to the database design. It includes the commented list of all the tables and fields of the database and also the list of the relationships. The third chapter is the biggest and includes a detailed description of all the tools of the system since a technical point of view. The last, the forth chapter concerns to the installation procedures and includes a description of the replication mechanism and the installation process itself.

1.1. A general technical description of KAMP

The KAMP system is comprised of the following main software items:

- Postgres database for holding course and user records
- User interface based on HTML forms using JavaScript to improve
- Apache Web server. Interface between web server and database tools
- CGIs written mainly in Java (and C in communication service)
- Chat server and client created in Java
- Modified FTP server
- POP email server for KAMP users
- Email List server for handling email lists
- Discussion group software

All the software systems were developed for use on computers running Solaris, Linux, ...). Currently the KAMP sub-systems run on Silicon Graphics partner sites. The following diagram shows the access to the client user.

1.1.1. How the system works

In the current system the interface between the server and client becomes a HTML form-based interface. Basically, HTML forms are used to collect data in the client side, while JavaScript functions are used for interaction. In the old system, Java applets were used to connect to the database from the client. But problems with platform dependency and very low speed of Java applets made us to discard this solution.

On the other hand, in the server side, CGIs (Common Gateway) are used to get data from the web server. The most of those CGIs have been wrote...
1.0 and JavaPostgres95 v 0.2 package), and a part (telematic service in C.

The database stores both information about users and about courses. In this case, URLs and descriptions are stored, but not the content. You can see it in the diagram:

Current KAMP system

1.1.1.1. The client side

As explained previously, in the previous versions, Java applets were directly with the database, and the web server had a secondary role. An interesting solution that we think will be very common is presented, many problems concerning mainly to two aspects:
Many problems of platform dependency. Even when in principle platform independent, the AWT (Abstract Window Interface) API used to implement graphic user interfaces, has many lacks. An applet is usually seen in a different way depending on the operating system. Unstability problems were also an usual interface, especially in Macintosh computers.

Applets were quite complex and needed a lot of different code to be downloaded to execute it. The consequence is a downloading.

In that way, we thought that Java applets could only used in a closed environment, meaning a high-speed LAN with computers with similar characteristics appropriate for the open Internet.

This situation drove us to change the model in the client removing forms are used now in the clients. CGIs are used in the server to send and then create an on-the-fly HTML page which is sent to the client simple to allow a good interface. For instance in a Kamp tool when a student selects a course, and then a student of the course, we are not in the server when the user selects the course, but to do it with only a CGI every time that the user makes any simple action like selecting a field of a form.

Summing up, the CGI calls the database and creates a page with scripts in Javascript which are executed in the client.

1.1.1.2. The server side

The database is based on the Postgres SQL v6.0 package (new version). The package has been modified in order to provide a mechanism between the server site and client sites.

The Web service uses the Apache 1.2.6 package. The package has been ordered to provide a protection mechanism for the KAMP database. A module has been created in order to restrict access to KAMP pages and the authentication. This module allows different access rights according to the system (e.g. some pages are reachable for tutors, not for students).

A chat server has been implemented in Java. The client has been using a Java applet.

The File Transfer service which enables students to upload course assessment to the KAMP system is based on the Washington University package. The package has been modified in order to allow KAMP use. KAMP sites, using an authentication mechanism based on the KAMP database, users are directly connected to a particular directory and are only of the disk.
The Email service applies the system Sendmail package. The package was modified to allow messages to be stored in a mailbox on the KAMP server. This modification overcomes the problem of KAMP users not actually having accounts on the server, and allows users to store messages in a standard disk space.

The PostOffice service is based on the Qualcomm POP3 2.1.4 package modified in order to allow KAMP users with a mailbox on the KAMP server to access their messages. Enhancements allow the daemon to authenticate users against a database (other than the system authentication mechanism) and to allow users to store messages in a non-standard mailbox. These changes have been made because not all users are system users.

The Mailing list service uses the UNIX Listserv System version 6. The Internet News service applies the InterNet News Daemon 1.1.

The videoconference server has been removed from the system and NeXt software is used to allow this service.
2. Database description

2.1. Database structure

The database management system used for Kamp has been Postgres, concretly version 6.0. The same structure as the previous system has been kept, with only some minor differences.

Following the list of all the tables (alphabetically sorted) and the description of each field is presented.

**ACCESSRIGHT**

Not used in this version of Kamp

**ACTIVITY**

(List all the activities used in the KAMP)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity No:</td>
<td>An Activity is unique and can be accessed by its Activity number.</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by oid</td>
</tr>
<tr>
<td>Creation Date:</td>
<td>Creation Date and Hour.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification Date:</td>
<td>Date and Hour of the last modification.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion Date:</td>
<td>Logical deletion Date and Hour.</td>
<td>abstime</td>
<td></td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
</tr>
<tr>
<td>Name :</td>
<td>Name of the activity.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Description :</td>
<td>Description of the activity.</td>
<td>varchar</td>
<td>400</td>
<td>5 lines of 80 Char</td>
</tr>
<tr>
<td>Study Hours :</td>
<td>Number of study hours.</td>
<td>float4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration :</td>
<td>Duration of the activity in weeks.</td>
<td>float4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support Hours :</td>
<td>Number of support hours.</td>
<td>float4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity Shareability:</td>
<td>OK, if the activity can be shared by the other site.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value) Yes by default</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Length</td>
<td>Remark</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Assessment Criteria :</td>
<td>An assessment criteria for an activity. (See ASSESSMENT CRITERIA description).</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assessment Method :</td>
<td>An assessment method for an activity. (See ASSESSMENT METHOD description).</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assessment File</td>
<td></td>
<td>varchar</td>
<td>255</td>
<td>URL to the assessment file (if existing)</td>
</tr>
<tr>
<td>Teaching Method</td>
<td>An teaching method for an activity. (See TEACHING METHOD description).</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assessment Person</td>
<td>An assessment person for an activity. (See ROLE description).</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by table ASSESSUSER_COURSE</td>
</tr>
<tr>
<td>Support Person :</td>
<td>A support person for an activity. (See ROLE description).</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by table ASSESSUSER_COURSE</td>
</tr>
<tr>
<td>Credit Value:</td>
<td>Credit value for the activity</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AssetUsed:</td>
<td>A module can have no asset</td>
<td>bool</td>
<td></td>
<td>‘0’ Asset used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity OK:</td>
<td>OK, if the acivity is finished and can be used.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value)</td>
</tr>
<tr>
<td>Site_Owner</td>
<td>The university that owns the activity</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Creator</td>
<td>The creator of the activity</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>URL</td>
<td>URL of the contents of the activity</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ACTIVITY TELEM**

Not used in this version of Kamp

**ASSESSUSER_COURSE**

(Used to store the tutors of an activity of a course, for one group of students)
<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset No :</td>
<td>An Asset is unique and can be accessed by its Asset number.</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by oid</td>
</tr>
<tr>
<td>Creation Date:</td>
<td>Creation Date and Hour.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification Date:</td>
<td>Date and Hour of the last modification.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion Date:</td>
<td>Logical deletion Date and Hour.</td>
<td>abstime</td>
<td></td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
</tr>
<tr>
<td>Title :</td>
<td>Title of the asset.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Description of the asset.</td>
<td>varchar</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Owner:</td>
<td>Owner of the asset.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Copyright holder:</td>
<td></td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Extent:</td>
<td>Extent in minutes of the asset.</td>
<td>Int4</td>
<td></td>
<td>Duration in Minutes</td>
</tr>
<tr>
<td>Asset OK:</td>
<td>OK, if the asset is finished and can be used.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value)</td>
</tr>
<tr>
<td>Asset Shareability:</td>
<td>OK, if the asset can be shared by the other site.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value)</td>
</tr>
<tr>
<td>Asset FileName:</td>
<td>The Physical name of the Asset</td>
<td>varchar</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Thumbnail:</td>
<td>Thumbnail</td>
<td>varchar</td>
<td>255</td>
<td>Not used</td>
</tr>
<tr>
<td>Asset Type:</td>
<td>The type of the asset (</td>
<td>oid</td>
<td></td>
<td>Pointer towards Asset Type</td>
</tr>
</tbody>
</table>
### Detailed Specification & Design v3.0 D11.8

#### ASSET ACTIVITY

( Link between asset and activity tables)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity_No:</td>
<td>The activity oid</td>
<td>oid</td>
<td></td>
<td>Pointer to activity table</td>
</tr>
<tr>
<td>Asset_No:</td>
<td>The asset oid</td>
<td>oid</td>
<td></td>
<td>Pointer to asset table</td>
</tr>
</tbody>
</table>

#### ASSET TYPE

( List the different type of assets: Video, HTML, ....)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset Type. No:</td>
<td>An Asset Type is unique and can be accessed by its Asset Type number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the asset type.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

#### BACKSTUDY

( List all Background Studies in the Document Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>A Background Study is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Title:</td>
<td>Description of the</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
### BIBLIOGRAPHY

(List all publications in the Document Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>A Publication is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td>Counter</td>
<td></td>
</tr>
<tr>
<td>Reference:</td>
<td>Reference of the publication (if it exists)</td>
<td>varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Title:</td>
<td>Description of the publication.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Author:</td>
<td>Name of the author of the publication.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Year:</td>
<td>Year of publishing.</td>
<td>Integer</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Place:</td>
<td>Place of publishing.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Publisher:</td>
<td>Publisher.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

### CASE STUDY

(List all Studies in the Case Studies Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study No:</td>
<td>A Case Study is unique and can be accessed by its Study number.</td>
<td>Oid</td>
<td>Counter</td>
<td></td>
</tr>
<tr>
<td>SiteNo:</td>
<td>The site which hosts the case study.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Title:</td>
<td>Description of the case study.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Length</td>
<td>Remark</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>Category No:</td>
<td>A category is unique and can be accessed by its Category number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Name:</td>
<td>Name of the category.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

**CATEGORY**

(List the different type category of software and tools: editor tools, net tools,...)
(List of the mailing lists hosted in a site)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room No:</td>
<td>A Chat Room is unique and can be accessed by its Room number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Title:</td>
<td>The title of the chat room.</td>
<td>Varchar</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Owner:</td>
<td>The administrator of the chat room.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password:</td>
<td>Password for the administration of the list.</td>
<td>Varchar</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Motd:</td>
<td>The welcome message (Message Of The Day)</td>
<td>Varchar</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

**COPYRIGHT**

(Net the copyright of a course, module, activity of asset)

<table>
<thead>
<tr>
<th>Nom atribut</th>
<th>Descripció</th>
<th>Tipus</th>
<th>Longitud</th>
<th>Observacions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright_no</td>
<td>Unique identifier of every copyright</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by oid</td>
</tr>
<tr>
<td>Course_no</td>
<td>Course to set the copyright</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Course.</td>
</tr>
<tr>
<td>Module_no</td>
<td>Module to set the copyright</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Module.</td>
</tr>
<tr>
<td>Activity_no</td>
<td>Activity to set the copyright</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Activity.</td>
</tr>
<tr>
<td>Asset_no</td>
<td>Asset to set the copyright</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Asset.</td>
</tr>
<tr>
<td>Name</td>
<td>Name and surname of a person to contact</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Surname</td>
<td>Not used</td>
<td>varchar</td>
<td>50</td>
<td>Not used</td>
</tr>
<tr>
<td>Email</td>
<td>Email of the contact person</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>URL of contact</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Role_no</td>
<td>Role of the creator</td>
<td>Oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Title</td>
<td>Owner of the copyright of the course, module, activity or asset</td>
<td>Varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Length</td>
<td>Remark</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>--------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>University_No</td>
<td>University that owns the copyright</td>
<td>Oid</td>
<td></td>
<td>Pointer to Site table</td>
</tr>
<tr>
<td>Description</td>
<td>Not used</td>
<td>Varchar</td>
<td>150</td>
<td>Not used</td>
</tr>
<tr>
<td>Creation Date:</td>
<td>Creation Date and Hour.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification Date:</td>
<td>Date and Hour of the last modification.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion Date:</td>
<td>Logical deletion Date and Hour.</td>
<td>abstime</td>
<td></td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
</tr>
</tbody>
</table>

**COREQ MODULE**

Not used in this version of Kamp

**COURSE**

(List all the course available for KAMP)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course No:</td>
<td>A course is unique and can be accessed by its course number.</td>
<td>oid</td>
<td></td>
<td>Not used: replaced by oid</td>
</tr>
<tr>
<td>Creation Date:</td>
<td>Creation Date and Hour.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification Date:</td>
<td>Date and Hour of the last modification.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion Date:</td>
<td>Logical deletion Date and Hour.</td>
<td>abstime</td>
<td></td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
</tr>
<tr>
<td>Title:</td>
<td>Title of the course.</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Description of the course.</td>
<td>varchar</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>CourseOk</td>
<td>If the course is ready to be used</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CourseShareability</td>
<td>If the course can be used by other universities</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site_Owner</td>
<td>The university that owns the course</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Creator</td>
<td>The creator of the course</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>FirstModule_No</td>
<td>Number of the first</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
</tbody>
</table>
### Module

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>URL of the description of the course</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Director</td>
<td>Director of the course</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>MailDirector</td>
<td>The mail of the course director</td>
<td>varchar</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

### Discussion Group

(List of discussion group hosted in the site)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>A Discussion group is unique and can be accessed by its Name.</td>
<td>Varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td>The administrator of the discussion group.</td>
<td>varchar</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Moderated</td>
<td>A Discussion group can have free access or can be moderated by the administrator.</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Description of the discussion group.</td>
<td>varchar</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

### Document

(List of all document in the Document Archive of the Learning Materials Base)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>A Document is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
</tbody>
</table>

### Document Topic

(This table is used for the relation between documents and topic keywords)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentNo:</td>
<td>Unique identifier of the document.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**HAS CHANGED**

Not used in this version of Kamp

**INTERVIEW**

(List of all interviews in the Document Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>An Interview is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td>Counter</td>
<td></td>
</tr>
<tr>
<td>Reference:</td>
<td>Reference to the document containing the interview.</td>
<td>Varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>Name of the interviewee.</td>
<td>Varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Role:</td>
<td>Role of the interviewee.</td>
<td>Varchar</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

**KEYWORD**

(List of all topic keywords for the Learning Materials Base)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword No:</td>
<td>A keyword is unique and can be accessed by its Keyword number.</td>
<td>Oid</td>
<td>Counter</td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>Value of the keyword.</td>
<td>Varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>TopicNo:</td>
<td>A keyword is associated to one topic section</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Described:</td>
<td>The keyword has a description associated.</td>
<td>Bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Similar:</td>
<td>The keyword is similar to another topic keyword.</td>
<td>Bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealKeyword:</td>
<td>The keyword which is similar to.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glossary:</td>
<td>Description of the keyword.</td>
<td>Varchar</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>
**LOCAL COURSE**

(This table is used to replicate all the information of courses to the other sites. It stores the necessary information of courses, modules and activities and their relationships)

<table>
<thead>
<tr>
<th>Nom atribut</th>
<th>Descripció</th>
<th>Tipus</th>
<th>Longitud</th>
<th>Observacions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group_No</td>
<td>The group</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Course_no</td>
<td>The oid of the course</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Course.</td>
</tr>
<tr>
<td>Parent_no</td>
<td>The oid of the parent, that means, the course if this is a module or the module if this is an activity</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Module or Course.</td>
</tr>
<tr>
<td>This_no</td>
<td>The oid of the course, module or activity</td>
<td>oid</td>
<td></td>
<td>Pointer to the table Activity, Module of Course.</td>
</tr>
<tr>
<td>LType</td>
<td>Level or type of this row</td>
<td>char</td>
<td>1</td>
<td>Different value: C: Course M: Module A: Activity</td>
</tr>
<tr>
<td>Title</td>
<td>Name or title of the course, module or activity</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The description of the course or activity or the summary of the module</td>
<td>varchar</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>URL of the material of the activity or the description of the module or course</td>
<td>varchar</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>AssessmentFile</td>
<td>The file that contains the exercises to be done by the students (only if the level is activity)</td>
<td>varchar</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Xxx_No</td>
<td>Not used</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>CreationDate:</td>
<td>Creation Date and Hour of the course, module or activity</td>
<td>abstime</td>
<td></td>
<td>Used to set the sort the activities of a module and the modules of a course</td>
</tr>
<tr>
<td>MailDirector:</td>
<td>The mail of the course</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**MAILING LIST**
(List of the mailing lists hosted in a site)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailingList No:</td>
<td>A Mailing List is unique and can be accessed by its Mailing List number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Name:</td>
<td>The name of the mailing list.</td>
<td>Varchar</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Owner:</td>
<td>Administrator of the mailing list.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password:</td>
<td>Password for the administration of the list.</td>
<td>Varchar</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Comment:</td>
<td>A brief description of the mailing list goals.</td>
<td>Varchar</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

**MAILING LIST SUBSCRIPTION**
(This table is used for the relation of subscription between KAMP users and mailing list)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListNo:</td>
<td>Unique identifier of the mailing list.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserNo:</td>
<td>Unique identifier of the KAMP user.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MODULE**
(List all the modules of the KAMP)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAMP Code:</td>
<td>A module is unique and can be accessed by its KAMP Code.</td>
<td>varchar</td>
<td>15</td>
<td>Not used: replaced by oid</td>
</tr>
<tr>
<td>Internal Code:</td>
<td>A code for the module (according to each)</td>
<td>varchar</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Length</td>
<td>Remark</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
<td>--------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>University</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation Date:</td>
<td>Creation Date and Hour.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modification Date:</td>
<td>Date and Hour of the last modification.</td>
<td>abstime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deletion Date:</td>
<td>Logical deletion Date and Hour.</td>
<td>abstime</td>
<td></td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
</tr>
<tr>
<td>Title:</td>
<td>Title of the module.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Study mode:</td>
<td>study mode (Full time, Part time)</td>
<td>oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit Value:</td>
<td>Credit value for the module</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cats Rating:</td>
<td>Credit Accumulation and Transfer System</td>
<td>varchar</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Module leader:</td>
<td>Name of the module leader.</td>
<td>varchar</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Summary:</td>
<td>Summary of the module.</td>
<td>varchar</td>
<td>800</td>
<td>10 lines of 80 Char</td>
</tr>
<tr>
<td>Module OK:</td>
<td>OK, if the module is finished and can be used.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value)</td>
</tr>
<tr>
<td>Module Shareability:</td>
<td>OK, if the module can be shared by the other site.</td>
<td>bool</td>
<td></td>
<td>(Yes/No value)</td>
</tr>
<tr>
<td>Ref. Materials:</td>
<td>Description of the referencing materials for the module.</td>
<td>varchar</td>
<td>2000</td>
<td>20 lines of 100 Char</td>
</tr>
<tr>
<td>Outcomes:</td>
<td>A list of Learning Outcomes for the module.</td>
<td>varchar</td>
<td>2000</td>
<td>20 lines of 100 Char</td>
</tr>
<tr>
<td>Site_Owner</td>
<td>The university that owns the module</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>Creator</td>
<td>The creator of the module</td>
<td>oid</td>
<td></td>
<td>Not used</td>
</tr>
<tr>
<td>URL</td>
<td>URL of the description of the module</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MODULE ACTIVITY**

(Link between module and activity tables)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity_No</td>
<td>The oid of the activity</td>
<td>oid</td>
<td></td>
<td>Pointer to activity table</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Length</td>
<td>Remark</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
<td>------</td>
<td>--------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Module_No</td>
<td>The oid of the module</td>
<td>oid</td>
<td></td>
<td>Pointer to module table</td>
</tr>
<tr>
<td>Course_No</td>
<td>The oid of the course</td>
<td>oid</td>
<td></td>
<td>Pointer to course table</td>
</tr>
</tbody>
</table>

**MODULE COURSE**

(Link between module and course table)

**MODULE GROUP**

Not used in this version of Kamp

**PLATFORM**

(List the different type of Platform to be used in the KAMP system)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform No:</td>
<td>A Platform is unique and can be accessed by its Platform number.</td>
<td>oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Title:</td>
<td>Name of the Platform.</td>
<td>varchar</td>
<td>50</td>
<td>Predefined list (Windows 3.1, Windows 95, Windows (3.1 + 95) ..., MAC OS 7.5, ...)</td>
</tr>
</tbody>
</table>

**PREREQ MODULE**

Not used in this version of Kamp

**RESOURCE**

Not used in this version of Kamp

**RESOURCE TYPE**

Not used in this version of Kamp

**RESULTS**
(Used to store the results of each student and every activity and module of his course)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>User_no</td>
<td>The student owner of this result</td>
<td>oid</td>
<td></td>
<td>Pointer to the User table. The only users that can be related are students</td>
</tr>
<tr>
<td>Activity_no</td>
<td>The activity</td>
<td>oid</td>
<td></td>
<td>Pointer to the Activity table.</td>
</tr>
<tr>
<td>Module_no</td>
<td>The module</td>
<td>oid</td>
<td></td>
<td>Pointer to the Module table.</td>
</tr>
</tbody>
</table>
| State      | State of the module or activity                  | Int4 |        | For module: 1: validated 1: downloaded 2: uploaded 3: assessed 4: auto-assessed  
|            |                                                   |      |        | If no row for the module or activity the state is not done               |
| DownloadDate| Date and time when the student downloaded the activity | abstime |      |                                                                         |
| UploadDate | Date and time when the student uploaded the results of the exercises into the assessment file | abstime |      |                                                                         |
| AssessedDate| Date and time when the activity is assessed       | abstime |      |                                                                         |
| Mark       | Mark for the activity                            | varchar | 10   |                                                                         |
| RType      | Used to know if the result is related to an activity or a module. | char  | 1     | Two different values: A: Activity M: Module                               |

**ROLE**

( List the different Roles in the KAMP system)
<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role No:</td>
<td>A Role is unique and can be accessed by its Role number.</td>
<td>oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Title:</td>
<td>Name of the Role.</td>
<td>varchar</td>
<td>50</td>
<td>Predefined list of the eight roles '&lt;1': Author '&lt;2': Course Director '&lt;3': Tutor '&lt;4': Student '&lt;5': Visitor '&lt;6': Administrator (Support Staff) '&lt;7': Specialist '&lt;8': Partner</td>
</tr>
</tbody>
</table>

**ROLE TELEMATICS**

Not used in this version of Kamp

**SITE**

( List the different sites in the KAMP system)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site No:</td>
<td>A site is unique and can be accessed by its site number.</td>
<td>oid</td>
<td>5</td>
<td>The 5 first letters of the Site Name.</td>
</tr>
<tr>
<td>Name:</td>
<td>Full name of the Site.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Address:</td>
<td>Internet Address of the Site</td>
<td>varchar</td>
<td>120</td>
<td>The address can be a numerical address (IP address) as well as a symbolic name (host name plus domain name).</td>
</tr>
<tr>
<td>E-mail:</td>
<td>E-mail address of the site information service</td>
<td>varchar</td>
<td>35</td>
<td>The address is: info@domain_name</td>
</tr>
<tr>
<td>ISDN:</td>
<td>Phone number of ISDN line</td>
<td>varchar</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PSTN:</td>
<td>Phone number of PSTN line</td>
<td>varchar</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
### Software Archive

(List sofware and tools in the Software Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software No:</td>
<td>A Software is unique and can be accessed by its Software number.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category No:</td>
<td>A software is inserted in a category.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Description of the software.</td>
<td>varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Reference:</td>
<td>Reference of the software or tool.</td>
<td>varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>Name of the software.</td>
<td>varchar</td>
<td>128</td>
<td></td>
</tr>
</tbody>
</table>

### SOFTWARE

Not used in this version of Kamp

### SOFTWARE TOPIC

(This table is used for the relation between software and tools and topic keywords)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software No:</td>
<td>Unique identifier of the software or tool.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword No:</td>
<td>Unique identifier of the topic keyword.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### STUD_GROUP_MOD

Not used in this version of Kamp
**STUDENTWORK**
(List student works in the Document Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>A Student Work is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Reference:</td>
<td>Reference to the document containing the student work.</td>
<td>Varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Description of the student work.</td>
<td>Varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Author:</td>
<td>Name of the author of the student work.</td>
<td>Varchar</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

**STUDENT GROUP**
(List the different Student group in the KAMP system)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group_No:</td>
<td>A student group is unique and can be accessed by his code.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>GroupTitle</td>
<td>Title of the student group</td>
<td>varchar</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>EmailList:</td>
<td>The address of the mailing list</td>
<td>varchar</td>
<td>35</td>
<td>The address is: group_name@domain_name group_name is a part of the Group Title</td>
</tr>
<tr>
<td>Course_No</td>
<td>The course of the group</td>
<td>Oid</td>
<td></td>
<td>Pointer to the table Course</td>
</tr>
<tr>
<td>Agenda</td>
<td>The agenda of the group</td>
<td>varchar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**STUDY MODE**
(List the different Study mode for the student)
### STUDY MODE

*A student mode is unique and can be accessed by his number.*

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudyMode No:</td>
<td>A student mode is unique and can be accessed by his number.</td>
<td>Char</td>
<td>1</td>
<td>‘1’ for Full-time study mode ‘2’ for Part Time study mode</td>
</tr>
<tr>
<td>Title</td>
<td>Title of the study mode</td>
<td>varchar</td>
<td>9</td>
<td>List with (Full-time and Part-time)</td>
</tr>
</tbody>
</table>

### STUDY_TOPIC

(This table is used for the relation between case studies and topic keywords)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudyNo:</td>
<td>Unique identifier of the case study.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KeywordNo:</td>
<td>Unique identifier of the topic keyword.</td>
<td>Oid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TELEM SERVICES

Not used in this version of Kamp

### TOOL

Not used in this version of Kamp

### TOPIC

(List the topic sections in the Learning Materials Base)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic No:</td>
<td>A Topic Section is unique and can be accessed by his number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Name:</td>
<td>Name of the topic section.</td>
<td>Varchar</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

### USER

(List the different users in the KAMP system; this list includes the students)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User No:</strong></td>
<td>A user is unique and can be accessed by its user number.</td>
<td>oid</td>
<td>5</td>
<td>Not used: replaced by the oid (the login is also used as unique identifier)</td>
</tr>
<tr>
<td><strong>Login:</strong></td>
<td>A user is unique and can be accessed by his login.</td>
<td>varchar</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td><strong>Creation Date:</strong></td>
<td>Creation Date and Hour.</td>
<td>absdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modification Date:</strong></td>
<td>Date and Hour of the last modification.</td>
<td>absdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Deletion Date:</strong></td>
<td>Logical deletion Date and Hour.</td>
<td>absdate</td>
<td>The physical deletion from the Database can be made after a certain time (to be defined)</td>
<td></td>
</tr>
<tr>
<td><strong>Name:</strong></td>
<td>Name of the user.</td>
<td>varchar</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td><strong>Surname:</strong></td>
<td>Surname of the user.</td>
<td>varchar</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td><strong>Birthdate:</strong></td>
<td>Birthdate of the user.</td>
<td>absdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Gender:</strong></td>
<td>Gender of the user</td>
<td>Char</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Address:</strong></td>
<td>Postal address of the user</td>
<td>Varchar</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td><strong>Telephone:</strong></td>
<td>Telephone number of the user</td>
<td>Varchar</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><strong>Fax:</strong></td>
<td>Fax number of the user</td>
<td>Varchar</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><strong>Email:</strong></td>
<td>E-Mail of the user.</td>
<td>varchar</td>
<td>35</td>
<td>The address is: login_name@domain_name</td>
</tr>
<tr>
<td><strong>Password:</strong></td>
<td>Password of the user.</td>
<td>varchar</td>
<td>8</td>
<td>Main skills</td>
</tr>
<tr>
<td><strong>Resume:</strong></td>
<td>Resume</td>
<td>varchar</td>
<td>800</td>
<td>10 lines of 80 char</td>
</tr>
<tr>
<td><strong>Site:</strong></td>
<td>The User University</td>
<td>oid</td>
<td>Pointer to Site table</td>
<td></td>
</tr>
<tr>
<td><strong>Platform_1:</strong></td>
<td>First Home platform</td>
<td>oid</td>
<td>Pointer to Platform table</td>
<td></td>
</tr>
<tr>
<td><strong>Platform_2:</strong></td>
<td>Second Home platform</td>
<td>oid</td>
<td>Pointer to Platform table</td>
<td></td>
</tr>
<tr>
<td><strong>Trainee:</strong></td>
<td>Differs trainee from student</td>
<td>bool</td>
<td>Yes/No value, Only for Students</td>
<td></td>
</tr>
<tr>
<td><strong>Student Group:</strong></td>
<td>The Student Group (only for student)</td>
<td>oid</td>
<td>Pointer to Student Group Table</td>
<td></td>
</tr>
<tr>
<td><strong>CourseNo:</strong></td>
<td>The course that the user has subscribed to (only for student)</td>
<td>oid</td>
<td>Pointer to Course</td>
<td></td>
</tr>
<tr>
<td><strong>Role1:</strong></td>
<td>The first role of the user</td>
<td>oid</td>
<td>A user can have until 4 different roles. Pointer to Role</td>
<td></td>
</tr>
<tr>
<td><strong>Role2:</strong></td>
<td>The second role of the user</td>
<td>oid</td>
<td>A user can have until 4 different roles. Pointer to Role</td>
<td></td>
</tr>
</tbody>
</table>
Role3 : The third role of the user oid A user can have until 4 different roles. Pointer to Role

Role4 : The fourth role of the user oid A user can have until 4 different roles. Pointer to Role

LastAccess Date and time of the last access to the system Abstime Used to know the users presents in the system

LastIP IP of the user in his last access Varchar 24 Used to know the users presents in the system

**USER ACTIVITY**
Not used in this version of Kamp

**USER MODULE**
Not used in this version of Kamp

**VIDEOCONFERENCE**
Not used in this version of Kamp

**WORKING**
(List of all working documents in the Document Archive)

<table>
<thead>
<tr>
<th>Field NAME</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document No:</td>
<td>A Working Document is unique and can be accessed by its Document number.</td>
<td>Oid</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>Reference:</td>
<td>Reference to the document containing the working document.</td>
<td>Varchar</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Description of the working document.</td>
<td>Varchar</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Purpose:</td>
<td>Purpose of the working document.</td>
<td>Varchar</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Application:</td>
<td>Name of the application needed to load the working document.</td>
<td>Varchar</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
2.2. Description of the relationships

One course is composed by one or more modules. Each module can be part of zero or more courses.
Each module has only zero or one study_mode. One study_mode is used by zero or more modules.
One module is composed by one or more activities. Each activity can be part of zero or more modules.
One user (student) belongs to only one student_group. One student_group is formed by zero or more users (students).
One user (student) is subscribed to only one site. One site has zero or more users (students).
Each user has one or more (until four) roles. Each role can be associated to zero or more users.
Each user (student) is subscribed only to one course. Each course can be studied by zero or more users (student).
Each user can has one or two platforms. Each platform can be used by zero or more users.
Each user perhaps has not got any results, or perhaps has many (one for each module and activity). Each result is only of one user.
Each results is only for one module. Each module can have zero or more (one for every student of the course of this module) results.
Each results is only for one activity. Each activity can have zero or more (one for every student of the course of this activity) results.
These last two relationships were excludent, that means that one result of one student is only for one module or one activity, but not for both.
Each student_group belongs only to one course. Each course can have zero or more student_groups.
Each activity can be composed by one or more assets. Each asset can be part of zero or more activities.
Each item of the Learning Materials Base (casestudy, document, soft) is associated to zero or more topic keywords. Each topic keyword is associated to zero or more items of the Learning Materials Base (casestudy, document, soft).
Each keyword belongs to one topic section. A topic section can have zero or more keywords.
Each software (soft) belongs to one software category. Each software category can have zero or more software (soft).
Each user is subscribed to zero or more mailing_list. Each mailing_list can have one or more users subscribed.
Each **mailing_list** is administrated by one **user**. A **user** can manage zero or more **mailing_lists**.

Each **discussiongroup** is administrated by one **user**. A **user** can manage zero or more **discussiongroup**.

Each **chatroom** is administrated by one **user**. A **user** can manage zero or more **chatroom**.
3. Tools description

In this chapter of the deliverable we show the technical description of all the tools of the system. For every tool this information is presented:

- Description of the form
- Data-flow diagram
- Process specifications
- Cgi description (or Java class description)

1. Kampus system

The system is divided in four sections:

- 1.1.- Registration: this is the public part of the system that allows people to register to a Kampus course.
- 1.2.- Staff tools: here there are the tools for the staff, that are course designer, administrative tools, learning material base manager and telematic services manager.
- 1.3.- Student tools: these are the tools available by the student: the course delivery, the agenda and the learning material base.
- 1.4.- Telematic services: this part is common for students and staff and that is the reason why it appears as a different section. The other three sections have a different home page, but not this, that appears both in the student and in the staff home.

Data-flow diagram
1.1. Registration

The aim of this tool is to help students to register in the Kamp system. This on-line tool will provide the student a way of filling in a registration form to the system. The student has to subscribe one course (he can browse a description of each one).

**Description of the form**

This form contains the following information:

- **Name:** Student fills-in his name
- **Surname:** Student fills-in his surname
- **Birth date:** Student fills-in his birth date
- **Student E-Mail address:** Student fills-in his e-mail address. If he has no e-mail address, the university will give him one.
- **Student password:** Student fills-in his password to access the Kamp system.
Student or Trainee:  
*Student chooses if he wants to follow a defined course (like Master) or he wants to study in part-time mode.*

Course:  
*Student chooses, in a list, the course he wants to follow.*

Resume:  
*Student fills in his former studies or professional experiences (only for trainees)*

When the student completes his registration form, information will be transferred to the Kamp system and an automatic mail will be sent to the director of the course to inform him of new registration. This information is temporarily stored on the Kamp database until the course director or administrator gives validation or negation, using the user administration tool.

**Data-flow diagram**

![Data-flow diagram](image)

**Process specification**

**REGISTER**

*Begin*

  *Connect to the local database*

  *If connection successful then*

    *Show the register form with the available courses to the user*
(data comes from the tables LOCAL_COURSE and PLATFORM)
if user wants to register then
    read user’s data
    create the user with the temporal role in the USER table
    send an e-mail to the director of the course the user wants to register to
    show an “operation successful “ message to the user
end if
close the database connection
end if
end

NOTE: The site assigned to the user is the site where the user is connected. The site identification number comes from the SITE table.

**Java class description**

The student registration tool is implemented into the Java class **Register.class**. This is a description of the methods of this class:

**main**

public static void main(String args[])
    Open the connection with the local database, depending on the user request makes
the corresponding action and finally close the connection with the local database.

**readValues**

public static void readValues(UserObject tmp)
    Read the values from the form, using the class cgi
    Parameters:
        tmp - The object where the information of the user is stored

**initRegister**

public static boolean initRegister()
    Retrive the name of all the courses and platforms
    Returns: True if no errors have occured

**createUser**

public static void createUser()
    Create a new temporal user

**sendEmailToDirector**

public static void sendEmailToDirector()
Send an e-mail (with the user data) to the director of the course that the user wants to register to

**makeHtmlPage**

public static void makeHtmlPage()

Make the HTML page to show to the user if there were not any error

**makeHtmlPage2**

public static void makeHtmlPage2()

  Make the HTML page to show to the user the operation successful message
1.2. Staff tools

This is the set of tools available for the staff (excluding telematic tools, in part 1.4.), that comprise the tools for the roles:

- Course director
- Author
- Tutor
- Administrator

It is divided in four sections:

- 1.2.1. Course designer: used to design a course and its structure
- 1.2.2. Course administration: manages the administrative aspects of the course
- 1.2.3. Learning material base manager: manages the learning material base
- 1.2.4. Telematic services manager: manages the telematic services

Data-flow diagram
## 1.2 STAFF TOOLS

### 1.2.1 COURSE DESIGNER
Design a course and its structure. Includes the copyright designer.

### 1.2.2 COURSE ADMINISTRATION
Manages the administrative aspects of a course.

### 1.2.3 LEARNING MATERIAL BASE MANAGER
Manages the learning material base.

### 1.2.4 TELEMA TIC SERVICES MANAGER
Manages the telemedical services.
1.2.1. The Course Designer

The Course Designer is the set of tools used to create the courses materials. A design of a course follows a hierarchy. The base of this is the asset. An activity is composed by one or more assets, while a module is composed by one or more activities and finally a course by one or more modules. A different tool is used to manage every of these levels. The aim of this part is to describe the main functionality of every level of the course management tool.

There are some more tools related to the management of the courses. On one hand, copyright designer that allows to set copyright information at any level (course, module, activity or asset). On the other hand, there are tools for browsing the materials: course browsing, module browsing and activity browsing that allow the author to browse the contents.

**Data-flow diagram**
Management of the process to create a course.

1.2.1.1 ASSETS_DES
Create, remove, update or query an asset

1.2.1.2 ACTIVITY_DES
Create, remove, update or query an activity

1.2.1.3 MODULE_DES
Create, remove, update or query a module

1.2.1.4 COURSE_DES
Create, remove, update or query a course

Copyright

Copyright information of a course, module or activity or asset
1.2.1 Asset designer

The Assets (learning materials) are the smallest entities of the system. They describe the kind of media being used (text, video, etc...) and the software needed, their physical location, to build an activity.

**Form description**

These are the fields of the form of the asset designer tool:

<table>
<thead>
<tr>
<th>Assets available:</th>
<th>list of all the assets in the system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>title of the asset</td>
</tr>
<tr>
<td>Description:</td>
<td>summary of the asset</td>
</tr>
<tr>
<td>URL:</td>
<td>URL of the asset</td>
</tr>
<tr>
<td>Extent time:</td>
<td>extent in minutes of the asset</td>
</tr>
<tr>
<td>Asset type:</td>
<td>the type of the asset</td>
</tr>
<tr>
<td>Asset owner:</td>
<td>owner of the asset</td>
</tr>
<tr>
<td>Asset shareability:</td>
<td>defines if the asset can be shared</td>
</tr>
<tr>
<td>Asset state:</td>
<td>defines if the asset is ready to be used or not</td>
</tr>
</tbody>
</table>

**Data-flow diagram**
Process specifications

ASSETDES

Begin

  Connect to the master database

  If connection successful then
    Read the user request
    If request = assets list then
      Call ASSET_LIST function
    End if
    If request = asset query then
      Call ASSET_QUERY function
    End if
    If request = asset creation then
      Call ASSET_CREATION function
  End if
End if
If request = asset remove then
    Call ASSET_REMOVAL function
End if
If request = asset modification then
    Call ASSET_MODIFICATION function
End if
Close the master database connection
End if
End

ASSET LIST
Begin
    Read the name of all assets from the ASSET table
    Read the name of all sites from the SITE table
    Show the result to the user
End

ASSET_QUERY
Begin
    Read the asset identifier
    Read the asset’s data requested from the ASSET and ASSET_ACTIVITY tables
    Show the result to the user
End

ASSET_REMOVAL
Begin
    Read the asset identifier
    Remove the asset from the ASSET table
    Call ASSET_LIST function
End

ASSET_MODIFICATION
Begin
    Read the asset’s data and the asset identifier
    Update the data into the ASSET table
    Call ASSET_QUERY function
End

ASSET_CREATION
Begin
    Read the asset’s data
Save the data into the ASSET table
Call ASSET_QUERY function
End

Java class description

The asset designer is implemented into the Java class `asset.class`. This is a description of the methods of this class:

main

```java
public static void main(String args[])
    Open the connection with the remote DB, depending on the user request
    makes the corresponding action and finally close the connection
```

initAsset

```java
public static boolean initAsset()
    Get the list of all the assets from the table asset
    Returns: Returns True if no errors
```

readValues

```java
public static void readValues()
    Read the values from the form, using the class cgi
```

createAsset

```java
public static boolean createAsset()
    Insert the asset into the table asset
    Returns: Returns True if no errors
```

updateAsset

```java
public static void updateAsset()
    Update the asset into the table asset
```

removeAsset

```java
public static void removeAsset()
    Delete the asset from the table asset
```

makeHtmlPage

```java
public static void makeHtmlPage()
    Make the HTML page to show to the user if there were not any error
```
1.2.1 Activity designer

The activities are composed of assets. The users have to describe in this tool the list of assets being used, the way the activity will be assessed and the teaching method.

In order to build an activity, the user must define:

• the main characteristics of the activity (Name, Description, Owner, Duration, Study Hours, ....)
• the list of assets which compose the activity.
• the assessment criteria, the assessment method and the assessment person.
• the teaching method being used.

The tutors that will give support and assessment have to be specified in the course administration tool.

Form description

These are the fields of the form of the activity designer tool:

- Activities available: list of all the activities in the system
- Name: name of the activity
- Description: summary of the activity
- URL: URL of the activity
- Study hours: number of the study hours
- Support hours: number of support hours
- Duration: duration of the activity in weeks
- Assessment criteria: an assessment criteria for the activity
- Assessment method: an assessment method for the activity
- Credit value: credit value of the activity
- Assessment file: URL of the assessment file
- Assets available: list of all the assets in the system
- Selected assets: specifies the different assets composing the activity
- Activity shareability: defines if the activity can be shared
- Activity state: defines if the activity is ready to be used or not

Data-flow diagram
1.2.1.2 ACTIVITY_DES
Activity designer: Create, remove, update query an activity

---

Process specifications

ACTIVITY_DES
Begin
  Connect to the master database
  If connection successful then
    Read the user request
    If request = activities list then
      Call ACT_LIST function
    End if
    If request = activity query then
      Query the data of a concrete activity
      If request = activity identifier then
        Remove one activity
      End if
      Update the data of an activity
  End if
End if
Call ACT_QUERY function
End if
If request = activity creation then
    Call ACT_CREATION function
End if
If request = activity remove then
    Call ACT_REMOVAL function
End if
If request = activity modification then
    Call ACT_MODIFICATION function
End if
Close the **master** database connection
End if
End

**ACT_LIST**
Begin
    Read the name of all activities from the ACTIVITY table
    Read the name of all assets from the ASSET table
    Show the result to the user
End

**ACT_QUERY**
Begin
    Read the activity identifier
    Read the activity’s data requested from the ACTIVITY and ASSET_ACTIVITY tables
    Show the result to the user
End

**ACT_REMOVAL**
Begin
    Read the activity identifier
    If the activity is part of one or more courses (LOCAL_COURSE query) then
        Show error message (you can’t remove an activity used in other courses)
    Else
        Remove the asset from the ASSET table
        Call ASSET_LIST function
    End if
End

**ACT_MODIFICATION**
Begin
    Read the activity’s data and the activity identifier
    Update the data into the ACTIVITY table
    If the modification involves some of the following fields (name, description, URL, assesmentFile) then
        save this modifications in the LOCAL_COURSE table
    End if
    If the modification involves activity’s assets then
        Save the modification in the ASSET_ACTIVITY table
    End if
    Call ACT_QUERY function
End

ACT_CREATION
Begin
    Read the activity’s data
    Save the data into the ACTIVITY table
    If exists assets associated to the activity then
        Save the asset’s data into the ASSET_ACTIVITY table
    End if
    Call ASSET_QUERY function
End

Java class description

The asset designer is implemented into the Java class activity.class. This class has the following methods:

main
    public static void main(String args[])
        Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

loadActivity
    public static Object loadActivity(Long ActNo)
        Gets all the information of the selected activity from the table activity
        Parameters:
            ActNo - selected activity oid
        Returns: If no errors, it returns an activity object with the data of the selected activity

readValues
    public static void readValues()
Read the values from the form, using the class cgi

**checkChangesLocalCourse**

```java
public static void checkChangesLocalCourse()
```

Check if LocalCourse has to be updated. This happens only if any of the fields name, description, URL or Assessment file has changed

**readAssets**

```java
public static void readAssets()
```

Read the assets of the activity from the table asset_activity If the list changes, calls to saveAssetChosen to store the changes

**saveAssetChosen**

```java
public static boolean saveAssetChosen(Long ActivityNo)
```

Store the assets of the activity into the table asset_activity

**Parameters:**

- ActivityNo - activity oid

**Returns:** Returns True if no errors

**initAct**

```java
public static boolean initAct()
```

Get the list of all the activities from the table activity

**Returns:** Returns True if no errors

**initAsset**

```java
public static boolean initAsset()
```

Get the list of all the assets from the table asset

**Returns:** Returns True if no errors

**retrieveSelAsset**

```java
public static boolean retrieveSelAsset(Long ActivityNo)
```

Get the list of all the selected assets from the table asset_activity

**Parameters:**

- ActivityNo - oid of the activity

**Returns:** Returns True if no errors

**makePageNotAllowed**

```java
public static void makePageNotAllowed()
```

Make the HTML page to show to the user if there were some error

**makeHtmlPage**

```java
public static void makeHtmlPage()
```
Make the HTML page to show to the user if there were not any error
1.2.1.3. Module designer

The Modules are composed of activities. They will describe the type of the module, the list of activities being used and the list of learning Outcomes used.

In order to build a module, the user must define:
- the main characteristics of the module (Title, Description, Summary, Module leader, Credit value, Internal code, ...)
- the learning outcomes for this module.
- the list of activities which composed the module.

**Form description**

These are the fields of the form of the module designer tool:
- Modules available: list of all the modules in the system
- Title: title of the module
- Internal code: a code for the module (according to each university)
- Modules summary: summary of the module
- Learning outcomes: a list of learning outcomes for the module
- Description (URL): URL of the description of the module
- Study mode: type of study mode (full time, part time, ...)
- Credit value: credit value for the module
- Cats rating: credit accumulation and transfer system
- Module leader: name of the module leader
- Activities available: list of all the activities in the system
- Selected activities: specifies the different activities composing the module
- Module shareability: defines if the module can be shared
- Module state: defines if the module is ready to be used or not

**Data-flow diagram**
Process specifications

MODULE_DES
Begin
  Connect to the master database
  If connection successful then
    Read the user request
If request = modules list then
   Call MOD_LIST function
End if
If request = module query then
   Call MOD_QUERY function
End if
If request = module creation then
   Call MOD_CREATION function
End if
If request = module remove then
   Call MOD_REMOVAL function
End if
If request = module modification then
   Call MOD_MODIFICATION function
End if
Close the master database connection
End if
End

MOD_LIST
Begin
   Read the name of all modules from the MODULE table
   Read the name of all activities from the ACTIVITY table
   Show the result to the user
End

MOD_QUERY
Begin
   Read the module identifier
   Read the module’s data requested from the MODULE and
   MODULE_ACTIVITY tables
   Show the result to the user
End

MOD_REMOVAL
Begin
   Read the module identifier
   If the module is part of one or more courses (LOCAL_COURSE query) then
      Show error message (you can’t remove a module used in other courses)
   Else
      Remove the module from the MODULE table
      Call MOD_LIST function
Detailed Specification & Design v3.0 D11.8

MOD_MODIFICATION
Begin
  Read the module’s data and the module identifier
  If the modification involves module’s activities and the module is part of one or
  more courses (LOCAL_COURSE query) then
    Show error message (you can’t modify the activities of a module that is used in
    others courses)
  End if
  Update the data into the MODULE table
  If the modification involves some of the following fields (title, description,
  URL)
    then
      Save these modifications in the LOCAL_COURSE table
    End if
  If the modification involves module’s activities then
    Save the modification in the MODULE_ACTIVITY table
  End if
  Call MOD_QUERY function
End

MODULE_CREATION
Begin
  Read the module’s data
  Save the data into the MODULE table
  If exists activities associated to the module then
    Save the module’s data into the MODULE_ACTIVITY table
  End if
  Call MOD_QUERY function
End

MODULE_BROWSE
Begin
  Read from the form the selected activities
  Connect to the master database
  If connection successful then
    For each activity
      Read the activity data from the table ACTIVITY
    End loop
    Show the result to the user
  End if
End
End if
Close the database connection
End

Java class description

The module designer is implemented into the Java class module.class, excepting the module browsing which is implemented into browse_module.class. This is a description of the methods of both classes:

module.class

main
  public static void main(String args[])
  Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

initModule
  public static void initModule(Long moduleKey)
  Get the information concerning to modules from the table module. If any module selected, gets all its information (loadModuleFromDataBase). Otherwise, gets the list of all the modules (loadListModuleFromDataBase)
  Parameters:
  ModuleKey - selected module oid (if there is)

readValues
  public static void readValues(ModuleObject moduleTmp)
  Read the values from the form, using the class cgi
  Parameters:
  moduleTmp - An object where the information of the module is stored

loadListModuleFromDataBase
  public static void loadListModuleFromDataBase()
  Get the list of all the modules from the table module

loadModuleFromDataBase
  public static Object loadModuleFromDataBase(Long ModuleKey)
  Get all the information of the selected module from the table module
  Parameters:
  ModuleKey - selected module oid
Returns: If no errors, it returns a module object with the data of the selected module

**loadListActivity**

**public static void loadListActivity()**

- Get the list of all the activities from the table activity

**loadListActivityChosen**

**public static void loadListActivityChosen(Long ModulNo)**

- Get the list of all the activities of the module from the table module_activity

**Parameters:**

- ModulNo - module oid

**saveActivityChosen**

**public static boolean saveActivityChosen(Long ModuleNo)**

- Store the activities of the module into the table module_activity

**Parameters:**

- ModuleNo - module oid

**Returns:** Returns True if no errors

**createModule**

**public static boolean createModule()**

- Insert the module into the table module

**Returns:** Returns True if no errors

**removeModule**

**public static void removeModule()**

- Delete the module from the table module

**updateModule**

**public static void updateModule()**

- Update the module into the table module

**checkChangesLocalCourse**

**public static void checkChangesLocalCourse(ModuleObject moduleTemp)**

- Check if LocalCourse has to be updated. This happens only if any of the fields title, summary or URL has changed

**Parameters:**

- moduleTemp - the module object that stores the data of the selected module

**makeHtmlPage**

**public static void makeHtmlPage()**

- Make the HTML page to show to the user if there were not any error
makePageNotAllowed

public static void makePageNotAllowed()

Make the HTML page to show to the user if there were some error

browse_module.class

main

public static void main(String args[])

Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

initActivity

public static boolean initActivity(Long ActSelNo)

Get the name, description and assessment file of the activity from the table activity
Parameters:

ActSelNo - oid of the activity
Returns: True if no errors

makeHtmlPage

public static void makeHtmlPage()

Make the HTML page to show to the user if there were not any error

makeHtmlPageError

public static void makeHtmlPageError()

Make the HTML page to show to the user if there were some error
1.2.1.4. Course designer

A Course is composed of a set of modules. It describes the way modules will be managed. In order to build a course, the user must define:

- the main characteristics of the course (Title, Description, ...).
- the list of modules which composed the course.

**Form description**

These are the fields of the form of the module designer tool:

- **Courses available:** list of all the courses in the system
- **Name:** name of the course
- **Description:** description of the course
- **Information (URL):** URL of the page with the information about the course
- **Course director mail:** specifies the mail of the director of the course
- **Modules available:** list of all the modules in the system
- **Selected modules:** specifies the different modules composing the course
- **Course shareability:** defines if the course can be shared
- **Course state:** defines if the course is ready to be used or not

**Data-flow diagram**
Process descriptions

COURSE_DES

Begin

  Connect to the master database
  If connection successful then
    Read the user request
    If request = courses list then
      Call COURSE_LIST function
End if
If request = course query then
    Call COURSE_QUERY function
End if
If request = course creation then
    Call COURSE_CREATION function
End if
If request = course removal then
    Call COURSE_REMOVAL function
End if
If request = course modification then
    Call COURSE_MODIFICATION function
End if
Close the master database connection
End if
End

COURSE LIST
Begin
    Read the name of all courses from the COURSE table
    Read the name of all modules from the MODULE table
    Show the result to the user
End

COURSE_QUERY
Begin
    Read the course identifier
    Read the course’s data requested from the COURSE and MODULE_COURSE tables
    Show the result to the user
End

MODULE_REMOVAL
Begin
    Read the course identifier
    Remove the course from the COURSE and LOCAL_COURSE table
    Call COURSE_LIST function
End

COURSE_MODIFICATION
Begin
    Read the course’s data and the course identifier

Update the data into the COURSE table
If the modification involves some of the following fields (title, description, URL, mailDirector) then
    Save these modifications in the LOCAL_COURSE table
End if
If the modification involves course’s modules then
    Save the modification in the MODULE_COURSE table
End if
Call COURSE_QUERY function
End

COURSE_CREATION
Begin
    Read the course’s data
    Save the data into the COURSE table
    If exists modules associated to the course then
        Save the course’s data into the MODULE_COURSE table
    End if
    Save the data into the LOCAL_COURSE table
    Call COURSE_QUERY function
End

COURSE_BROWSE
Begin
    Connect to the master database
    If connection successful then
        For each module of the course loop
            Read the module data from the table LOCAL_COURSE
            For each activity of the module loop
                Read the activity data from the table LOCAL_COURSE
            End loop
        End loop
        Show the result to the user
    End if
    Close the database connection
End

Java class description
The course designer is implemented into the Java class `course.class`, excepting the course browsing which is into `browse.class`. This is a description of the methods of both classes:

**course.class**

**main**
```java
public static void main(String args[])
    Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection
```

**loadCourse**
```java
public static Object loadCourse(Long CourseNo)
    Gets all the information of the selected course from the table course
    Parameters:
        CourseNo - selected course oid
    Returns: If no errors, it returns a course object with the data of the selected course
```

**readValues**
```java
public static void readValues()
    Read the values from the form, using the class cgi
```

**checkChangesLocalCourse**
```java
public static void checkChangesLocalCourse()
    Check if LocalCourse has to be updated. This happens only if any of the fields title, description, URL or course director mail has changed
```

**readMods**
```java
public static void readMods()
    Read the modules of the course from the table module_course If the list changes, calls to saveModChosen to store the changes
```

**saveModChosen**
```java
public static boolean saveModChosen(Long CourseNo)
    Store the modules of the course into the table module_course
    Parameters:
        CourseNo - course oid
    Returns: Returns True if no errors
```

**initCourse**
```java
public static boolean initCourse()
```
Get the list of all the courses from the table course
Returns: Returns True if no errors

**initMod**

public static boolean initMod()
Get the list of all the modules from the table module
Returns: Returns True if no errors

**retrieveSelMod**

public static boolean retrieveSelMod(Long CourseNo)
Get the list of all the selected modules from the table module_course
Parameters:
CourseNo - oid of the course
Returns: Returns True if no errors

**makeHtmlPage**

public static void makeHtmlPage()
Make the HTML page to show to the user if there were not any error

**browse.class**

**main**

public static void main(String args[])
Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

**initModules**

public static boolean initModules(Long Course)
Get all the modules of the course from the table localcourse
Parameters:
CourseNo - oid of the course we are browsing
Returns: True if no errors

**initActivities**

public static boolean initActivities(Long Course, Long Module)
Get all the activities of one course and module from the table localcourse
Parameters:
CourseNo - oid of the course we are browsing
ModuleNo - oid of one module
Returns: True if no errors
makeHtmlPage
public static void makeHtmlPage()
    Make the HTML page to show to the user if there were not any error

makeHtmlPageNoAgenda
public static void makeHtmlPageNoAgenda()
    Make the HTML page to show to the user if there were some error
1.2.1.5. Copyright redesigne

The aim of this tool is to manage the creation, the modification or the deletion of the copyright of course, a module, an activity or an asset. Only one copyright can be associated with a course, a module, an activity or an asset.

**Form description**

These are the fields of the form of the module designer tool:
- **Courses available:** list of all the courses in the system
- **Copyright owner:** the owner of the copyright
- **Site:** the site of the owner
- **Contact information Person:** full name of a contact person
- **Contact information Person Email:** e-mail for contact information
- **Contact information Person URL:** URL for contact information

**Data-flow diagram**
Process specifications

COPYRIGHT_DES

Begin

Connect to the master database

If connection successful then

Read the user request

Call COPYRIGHT_QUERY function

If request = copyright creation then

Call COPYRIGHT_CREATION function

End if

If request = copyright removal then

Read the user request

Call COPYRIGHT_REMOVAL function

End if

Print the copyright information of the course, module, activity or asset

End if

End
Call COPYRIGHT_REMOVAL function
End if
If request = copyright modification then
    Call COPYRIGHT_MODIFICATION function
End if
Close the master database connection
End if
End

COPYRIGHT_QUERY
Begin
    Read the course, module, activity or asset identifier
    Read the course/module/activity/asset copyright information requested from the
    COPYRIGHT tables
    Show the result to the user
End

COPYRIGHT_REMOVAL
Begin
    Read the course, module, activity or asset identifier
    Remove the copyright row from the COPYRIGHT table
    Call COPYRIGHT_QUERY function
End

COPYRIGHT_MODIFICATION
Begin
    Read the course/module/activity/asset copyright information and the
    course/module/activity/asset identifier
    Update the data into the COPYRIGHT table
    Call COPYRIGHT_QUERY function
End

COPYRIGHT_CREATION
Begin
    Read the course/module/activity/asset copyright information
    Save the data into the COPYRIGHT table
    Call COPYRIGHT_QUERY function
End

Java class description
The copyright designer is implemented into the Java class `copyright.class`. This is a description of the methods of this class:

**main**

```java
public static void main(String args[])
{
    Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection
}
```

**initCopyright**

```java
public static boolean initCopyright()
{
    Get the copyright of the selected item (course, module, activity or asset) from the table copyright
    Returns: True if no errors
}
```

**initSite**

```java
public static boolean initSite()
{
    Get the list of all the sites from the table site
    Returns: True if no errors
}
```

**makeHtmlPage**

```java
public static void makeHtmlPage()
{
    Make the HTML page to show to the user if there were not any error
}
```
1.2.2. Course Administration

The course administration is a set of five tools used to manage users and the support of the courses. More concretely, there are five tools integrated in the course administration:

- management of users
- management of groups of students (group administration)
- management of tutors for assessment and support (course administration)
- management of results of the students
- management of sites

Data-flow diagram
1.2.2.1. Course administration

This tool is created to assign tutors to activities for every group of students. In that way, a group of students has two tutors for every student, one for assessment and another for support.

**Forms description**

The tool is split into two different windows. The first window allows the user to select a course and then a group of students and an activity, while in the second window s/he can assign both tutors to the selected group and activity.

First window:
- Courses: lists of all the courses in the system
- Groups: lists of all the groups of the selected course
- Activities: lists of all the activities of the selected course

Second window:
- Assessment by: it allows to select the tutor for the assessment from the list of all the tutors of the system
- Supported by: it allows to select the tutor for the supporting from the list of all the tutors of the system

**Data-flow diagram**
**Process specifications**

**COURSE_ADM**
Begin
Connect to the local database
If connection successful then
    Read the name and activities of every course from the table LOCAL_COURSE
    Read the name of the student groups from the table STUDENT_GROUP
    Show results to the user
End if
Close the database connection
End

**TUTOR_ADM**
Begin
Connect to the **local** database
If connection successful then
   If user request = query then
      Call QUERY_TUTOR function
   End if
   If user request = assign then
      Call ASSIGN_TUTOR function
   End if
End if
Close the database connection
End

**QUERY_TUTOR**
Begin
   Read group identifier
   Read the name of assessment and support tutors from the table
**ASSESSUSER_COURSE**
End

**ASSIGN_TUTOR**
Begin
   Read group identifier
   Read group identifier, assessment tutor identifier and supporting tutor identifier
   Assign these values in the table **ASSESSUSER_COURSE**
   Call QUERY_TUTOR function
End

**Java classes description**

The course administration is implemented into two Java classes, one for the first window, **myCourse.class**, and one for the second, **tutors.class**. This is a description of the methods of these two classes:

**myCourse.class:**

**main**
public static void main(String args[])
   Open the connection with the local database, depending on the user request makes the corresponding action and finally close the connection with the local database.

**initADMCourse**
public static boolean initADMCourse()
Retrieve the name of all the courses from the database
Returns: True if no errors have occurred

**retrieveGroup**
public static void retrieveGroup(Long courseNo)
- Retrieve all the groups of the given course from the database
- Parameters:
  - courseNo - The course identifier

**retrieveActivities**
public static void retrieveActivities(Long courseNo)
- Retrieve all the activities of the given course from the database
- Parameters:
  - courseNo - The course identifier

**makeHtmlPage**
public static void makeHtmlPage()
- Make the HTML page to show to the user if there were not any error

**tutors.class:**

**main**
public static void main(java.lang.String[] args)
- Open the connection with the local DB, depending on the user request makes the corresponding action and finally close the connection

**readValues**
public static void readValues()
- Read the values from the form, using the class cgi

**initTutors**
public static boolean initTutors()
- Read all the tutors and the assess for the given groupNo and courseNo

**getTutors**
public static void getTutors()
- Retrieve tutor for assessment and tutor for support for the given activity

**processForm**
public static void processForm()
Read the values of the new tutors using the class cgi and calls to removeTutors and addTutors if those had changed

**removeTutors**
- public static void removeTutors()
  - Remove the tutors assigned to an activity, course and group from the table asessuser_course

**addTutors**
- public static void addTutors()
  - Insert the tutors assigned to an activity, course and group into the table asessuser_course

**makeHtmlPage**
- public static void makeHtmlPage()
  - Make the HTML page to show to the user if there were not any error
1.2.2.2. Group administration

This tool is created to allow the creation groups of students that are subscribed to the same course. Note that when a student is created using the user tool, s/he has to be assigned to one group. Every group has a different mailing list as well as an agenda, where all the remarkable related to the course actions are shown, e.g. when the results of an activity have to be uploaded, when a chat session will take place, ...

**Forms description**

Courses : list of all the courses in the system
Groups : list of all the groups of the selected course. Clicking on the button “Students”, the students of the group is shown (assigned with the user administration tool)
Title : name of the selected group
Group Email : e-mail of the selected group. Clicking on the button “Mailing Lists Administration” this tool is opened in a new window
Agenda (URL) : agenda for the selected group. Clicking on the button “View”, the agenda is shown in a new window

**Data-flow diagram**
**Process specifications**

**GROUP ADMINISTRATION**

Begin

Connect to the **local** database

If connection successful then

Read the user request

If request = courses and groups list then

Call GROUP_LIST

End if

If request = group creation then

Call GROUP_CREATION function

End if

If request = group remove then

Call GROUP_REMOVAL function

End if
If request = group modification then
    Call GROUP_MODIFICATION function
End if
End if
Close the database connection
End

GROUP_LIST
Begin
    Read all the course names from the table LOCAL_COURSE
    Read all the group data for every course from the table STUDENT_GROUP
    Show results to the user
End

GROUP_CREATION
Begin
    Read new group data
    Create the group into the table STUDENT_GROUP
    Call GROUP_LIST function
End

GROUP_REMOVAL
Begin
    Read group identifier to delete
    Remove group from the table STUDENT_GROUP
    Call GROUP_LIST function
End

GROUP_MODIFICATION
Begin
    Read group data
    Save updated data into STUDENT_GROUP table
    Call GROUP_LIST function
End

STUDENT ADMINISTRATION
Begin
    Connect to the local database
    If connection successful then
        Read group identifier
        Read the name of every student of the group from the table USER
        Show results to the user
    End
End
Java class description

The group administration is implemented into the Java class `ADMGroup.class`. This is a description of the methods of this class:

**main**

```java
public static void main(String args[])
```

Open the connection with the local database, depending on the user request makes the corresponding action and finally close the connection with the local database.

**readValues**

```java
public static void readValues()
```

Read the values from the form, using the class cgi

**initADMGroup**

```java
public static boolean initADMGroup()
```

Retrive the name of all the courses from the database

Returns: True if no errors have occured

**retrieveGroup**

```java
public static void retrieveGroup(Long courseNo)
```

Retrive all the groups of the given course from the database

Parameters: courseNo - The course identifier

**createGroup**

```java
public static Boolean createGroup()
```

Create a new group

Returns: True if no errors have occured

**modifyGroup**

```java
public static Boolean modifyGroup()
```

Modify the data of a group from the database

Returns: True if no errors have occured

**removeGroup**

```java
public static Boolean removeGroup()
```

Remove a group from the database

Returns: True if no errors have occured
makeHtmlPage
public static void makeHtmlPage()
    Make the HTML page to show to the user if there were not any error
1.2.2.3. Site

The main functionality of this tool is to maintain general and technical information concerning a KAMP site.

**Forms description**

These are the fields of the form of the site administration tool:

- **Sites:** List of all the sites in the KAMP system
- **Site Name:** Name of the selected site
- **Address:** Postal address of the site
- **Email:** e-mail of the site or the responsible
- **PSTN number:** Phone number of the site or the responsible
- **ISDN number:** ISDN number of the site or the responsible
- **Site Logo URL:** URL of the logo of the site

**Data-flow diagram**
### Process specifications

ADM_SITE

Begin

Connect to the local database

If connection successful then

Read the user request

If request = site list then

Call SITE_LIST

End if

If request = site creation then

Call SITE_CREATION function

End if

If request = site remove then

Call SITE_REMOVAL function
End if
If request = site modification then
    Call SITE_MODIFICATION function
End if
End if
Close the database connection
End

SITE_LIST
Begin
    Read all the data from all the sites at the table LOCAL_COURSE
    Show results to the user
End

SITE_CREATION
Begin
    Read new site data
    Create the site into the table SITE
    Call SITE_LIST function
End

SITE_REMOVAL
Begin
    Read site identifier to delete
    Remove site from the table SITE
    Call SITE_LIST function
End

SITE_MODIFICATION
Begin
    Read site data
    Save updated data into SITE table
    Call SITE_LIST function
End

Java class description

The site administration is implemented into the Java class ADMSite.class. This is a description of the methods of this class:

main
public static void main(String args[])  
Open the connection with the master database, depending on the user request makes the corresponding action (create, modify, remove) and finally close the connection with the local database.

readValues  
public static void readValues()  
Read the values from the form, using the class cgi

initADMSite  
public static boolean initADMSite()  
Retrive all the sites from the database  
Returns: True if no errors have occured

makeHtmlPage  
public static void makeHtmlPage()  
Make the HTML page to show to the user if there were not any error
1.2.2.4. Results administration

The aim of this tool is for a tutor, to supervise the works that have been done by a student, and validate modules and activities.

**Forms description**

This tool is used to administrate the progress of the students. It is split into two windows. In the first one, the user can select one student of one course, while in the second window s/he can view the works of the student, validate a module and assign a mark to an activity.

First window:
- **Courses:** lists of all the courses in the system
- **Students:** lists of all the students of the selected course

Second window:
- **Button “View work”:** allows the user to view the works of the student
- **Modules:** list of all the modules of the activity
- **State:** state of the module for this student. The state can be validated or not validated. This field is not editable, but using the button “Validate” s/he can validate the module.
- **Activities:** lists of all the activities of the module
- **State:** state of the activity for this student.
- **Mark:** mark of the activity for this student. This field is not editable. To change the mark, the user has to fill in the field “assign new mark” and click on the button “assign”

**Data-flow diagram**
Process specification

ADM_RESULT
Begin
Connect to the local database
If connection successful then
    Read the name of all the courses from the table LOCAL_COURSE
    Read all the students of every course from the table USER
    Show results to the user
End if
Close the database connection
End

RESULTS QUERY
Begin
Connect to the local database
If connection successful then
    Read course and student
    Read all the modules of the course from the table LOCAL_COURSE
End if

For every module do
    Read the state from the table RESULTS
    Read all the activities of the module from the table
LOCAL_COURSE
    For every activity do
        Read the state and the mark of the activity from the table
End for
End for
Show the results to the user
End if
Close the database connection
End

VIEW WORK
Begin
    Read the login of student
    Show the directory of the student (/STUDENTSPACE/login)
End

CHANGE MODULE STATE
Begin
    Read module and user identifiers
    Check in the table RESULTS if there is a row with the user and module
    If there is then
        Nothing to do
    Else
        Create a new row into the RESULTS table
    End if
    Call RESULTS QUERY function
End

CHANGE ACTIVITY MARK
Begin
    Read module, activity and user identifiers
    Check in the table RESULTS if there is a row with the user and module
    If there is then
        Show “Operation not allowed” message
    Else
        If state = auto-assessed or state = uploaded
            Update mark and state into the RESULTS table
        Else
End
Show “Operation not allowed” message
End if
End if
Call RESULTS QUERY function

Java classes description

The results administration is implemented into two Java classes, one for the first window, results.class, and one for the second, validate.class. This is a description of the methods of these two classes:

results.class:

main
public static void main(String args[])
   Open the connection with the local DB, depending on the user request makes the corresponding action and finally close the connection

initADMResults
public static boolean initADMResults()
   Get the list of all the courses from the table localcourse
   Returns: True if no errors

retrieveStudent
public static boolean retrieveStudent(Long CourseNo)
   Get the list of all the students of the selected course from the table student
   Parameters: CourseNo - oid of the selected course
   Returns: True if no errors

makeHtmlPage
public static void makeHtmlPage()
   Make the HTML page to show to the user if there were not any error

validate.class:

main
public static void main(String args[])
   Open the connection with the local DB, depending on the user request makes the corresponding action and finally close the connection
**initValidate**

public static boolean initValidate()

Get the list of all the modules of the selected course from the localcourse table and all the state of each one from the table results

Returns: True if no errors

**retrieveAct**

public static boolean retrieveAct(Long ModNo)

Get the list of all the activities of the selected module from the localcourse table

Parameters:

- ModuleNo - oid of the selected module

Returns: True if no errors

**validateMod**

public static boolean validateMod(Long userNo, Long moduleNo)

Validate the selected module, inserting a new row into the table results

Parameters:

- userNo - oid of the student
- moduleNo - oid of the selected module

Returns: True if no errors

**newMark**

public static boolean newMark(Long userNo, Long moduleNo, Long activityNo, String mark, String Download, String Upload, int State)

Assign a mark to the activity, inserting or updating a row into the table results

Parameters:

- userNo - oid of the student
- moduleNo - oid of the selected module
- activityNo - oid of the selected activity
- mark - the mark to be assigned to the activity
- Download - the date when the activity was downloaded
- Upload - the date when the activity was uploaded
- State - the current state of the activity. It can be: downloaded (1); uploaded (2); assessed (3); auto-assessed (4); not done (if it is not into the table results)

Returns: True if no errors

**makeHtmlPage**

public static void makeHtmlPage()

Make the HTML page to show to the user if there were not any error
1.2.2.5. User administration

The main functionality of this tool is the management of users. It allows to create new users of any role, and concretely students. In this case, the student has to be subscribed to a course and a group of students. A user can be also removed from the system and finally updated (his data). A user is given a login and password to allow him to access the system. If the user is a student, a mail is sent to him with the address of the student home page and the login and password.

**Forms description**

This tool is split into two different windows. At the first window, the administrator can select an existing user or create a new one. At a second window, s/he can fill in the data of a user.

First window:
- **Type of user:** List of all the different categories of user. These are: temporary, author, course director, tutor, student, visitor, administrator, specialist and partner.
- **Users:** List of all the users presents in the system. Login and name of the user are shown.

Second window:
- **Name:** user’s name
- **Surnamee:** user’s surname
- **Birth Date:** user’s birth date
- **Gender:** user’s gender
- **User Login:** user’s login (the user has been given a login and a password to allow him/her to access the system)
- **User Password:** user’s password
- **E-mail address:** user’s e-mail
- **Address:** user’s postal adress
- **Telephone:** user’s telephone
- **Fax:** user’s fax
- **Main skills:** user’s main skills
- **Roles:** list of all the possible roles or categories of users. A user have until four roles
- **Courses:** if the user is a student, the course that the student is subscribed to
- **Groups:** if the user is a student, the group that the student is subscribed to
**Data-flow diagram**

1.2.25 ADM_USER

**Process specifications**

USER_ADMINISTRATION

Begin

Connect to the **local** database

If connection successful then

Read the user request
If request = user list then
    Call USER_LIST function
End if
If request = user query then
    Call USER_QUERY function
End if
If request = user creation then
    Call USER_CREATION function
End if
If request = user removal then
    Call USER_REMOVAL function
End if
If request = user modification then
    Call USER_MODIFICATION function
End if
Close the local database connection
End if
End

USER LIST
Begin
    Read the name of all users from the USER table
    Read the name of all roles from the ROLE table
    Show the result to the user
End

USER_QUERY
Begin
    Read the user identifier
    Read the user’s data requested from the USER table
    Show the result to the user
End

USER_REMOVAL
Begin
    Read the user identifier
    Remove the user from the USER table
End

USER_MODIFICATION
Begin
    Read the user’s data and the user identifier
Update the data into the USER table
End

USER_CREATION
Begin
   Read the user’s data
   Save the data into the USER table
End

Java classes description

The user administration is implemented into three Java classes, one for the first window, ADMUser.class, and two for the second, depending if the action is a creation, ADMcreate_user.class, or modification, ADMedit_user.class.

This is a description of the methods of these three classes:

ADMUser.class:

main
   public static void main(java.lang.String[] args)
      Open the connection with the local DB, depending on the user request makes the corresponding action (create, modify, remove) and finally close the connection with the local database.

checkLogin
   public static int checkLogin()
      Check if the user exists in the system

readValuesUser
   public static void readValuesUser()
      Read the values from the form, using the class cgi

initADMUser
   public static boolean initADMUser()
      Make the HTML page to show to the user if there were not any error
      Returns: true if no errors

ADMcreate_user.class:

main
public static void main(java.lang.String[] args)

  Open the connection with the local DB, depending on the user request makes the corresponding action (create, modify, remove) and finally close the connection with the local database.

**checkLogin**

  public static int checkLogin()

  Check if the user exists in the system

**sendRegistrationEmail**

  public static void sendRegistrationEmail(UserObject user)

  Send an e-mail to the user telling him/her that s/he is a new user of the system, the role, the username and the password

  Parameters:
  
  user - is the user object

**readValuesUser**

  public static boolean readValuesUser()

  Read the values from the form, using the class cgi

  Returns: true if no errors

**createUserDirectory**

  public static void createUserDirectory(java.lang.String dirName)

  Create a new directory

  Parameters:

  dirName - is the name of the new directory

**retrieveGroup**

  public static void retrieveGroup(java.lang.Long courseNo)

  Retrive all the groups of the given course from the database

  Parameters:

  courseNo - - The course identifier

**initcreateADMUser**

  public static boolean initcreateADMUser()

  Make the HTML page to show to the user if there were not any error

  Returns: true if no errors

**endcreateADMUser**

  public static boolean endcreateADMUser()

  make the HTML page to show to the user the creation successful message
endnocreateADMUser

public static boolean endnocreateADMUser()

Make the HTML page to show to the user if there were some error

ADMedit_user.class:

main

public static void main(java.lang.String[] args)

Open the connection with the local DB, depending on the user request makes the corresponding action (create, modify, remove) and finally close the connection with the local database.

checkLogin

public static int checkLogin()

Check if the user exists in the system

sendRegistrationEmail

public static void sendRegistrationEmail(UserObject user)

Send an e-mail to the user telling him/her that s/he is a new user of the system, the role, the username and the password

Parameters:

user - is the user object

readValuesUser

public static boolean readValuesUser(UserObject tempUser)

Read the values from the form, using the class cgi

Returns: true if no errors

createUserDirectory

public static void createUserDirectory(java.lang.String dirName)

Create a new directory

Parameters:

dirName - is the name of the new directory

retrieveGroup

public static void retrieveGroup(java.lang.Long courseNo)

Retrive all the groups of the given course from the database

Parameters:

courseNo - - The course identifier

initeditADMUser
public static boolean initeditADMUser(UserObject user)
   Make the HTML page to show to the user if there were not any error
   Returns: true if no errors

endeditADMUser
   public static boolean endeditADMUser()
      make the HTML page to show to the user the creation successful message

endnoeditADMUser
   public static boolean endnoeditADMUser(String UserNo)
      Make the HTML page to show to the user if there were some error
1.2.3. The Learning Materials Base Manager

The Learning Materials Base Manager is the set of tools used to manage the Materials Base. The Learning Materials Base is composed by three archives (Case Studies, Documents, Software and Tools). Each item can be linked to zero or more topic keywords, which compose the glossary of the Learning Materials Base. A different tool is used to manage different areas. The aim of this part is to describe the main functionality of every level of the Materials Base Management tool.

*Data-flow diagram*
1.2.3 LEARNING MATERIALS BASE MANAGER

Management of the process to populate and modify the Learning Materials Base.

1.2.3.1 STUDY_MAN

Create, remove, update or query a case study

1.2.3.2-6 DOC_MAN

Create, remove, update or query a document

1.2.3.7 SOFT_MAN

Create, remove, update or query a software

1.2.3.8 KEY_MAN

Create, remove, update or query a topic keyword

- 98 -
1.2.3.1. Case Archive Manager

Case Studies are the first kind of item contained in the Materials Base. They describe the solution of a problem. Each case study is divided into seven textual sections, one demonstration and one interview to staff which solve the study. The system provides a page with the list of all case studies stored in the archive. The user can choose to add a new case study or to update an existent study. In both cases, the system provides a form containing information regarding the chosen study.

**Form description**

These are the fields of the form of the case study manager:

- **Title**: title of the case study
- **Site**: site which hosts the case study
- **Summary**: reference (URL) of the summary
- **Concept**: reference (URL) of the concept section
- **Design**: reference (URL) of the design section
- **Production**: reference (URL) of the production section
- **Publishing**: reference (URL) of the publishing section
- **Management**: reference (URL) of the management section
- **Marketing**: reference (URL) of the marketing section

**Data-flow diagram**
Process specifications

CASE_MAN

Begin

Connect to the master database

If connection successful then

Call CASE_LIST function

If request = case creation then

Call CASE_CREATION function

End if

If request = case query then

Call CASE_QUERY function

End if

Close the master database connection

End if
CASE_LIST
Begin
  Read the title and the study identifier of all case studies from the CASESTUDY table
  Show the result to the user
End

CASE_QUERY
Begin
  Read the study identifier
  Read the study’s data from the CASE_STUDY and the SITE tables
  Show the result to the user
  If request = case removal then
    Call CASE_REMOVAL function
  End if
  If request = case modification then
    Call CASE_MODIFICATION function
  End if
  If request = topic list then
    Call TOPIC_LIST function
  End if
End

CASE_REMOVAL
Begin
  Read the study identifier
  Remove relations between topics and the study from STUDY_TOPIC table
  Remove the study from the CASE_STUDY table
End

CASE_MODIFICATION
Begin
  Read the study’s data and the study identifier
  Update the data into the CASE_STUDY table
End

CASE_CREATION
Begin
  Read the study’s data
  Save the data into the CASE_STUDY table
TOPIC_LIST
Begin
   Read the topic keywords from KEYWORD table.
   Show the result to user.
   If request = topic modification then
      Call TOPIC_MODIFICATION function
   End if
End

TOPIC_MODIFICATION
Begin
   Read the list of keyword identifiers and the study identifier
   Insert and remove relations between keywords and study in the STUDY_TOPIC table
End

Cgi description

The Case Studies Archive Manager is implemented by the C program casestudy.cgi. The program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

main
main(int argc, char* argv)
   Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

maincase
void maincase(PGconn* conn)
   Get the list of all the case studies from the table casestudy

infocase
void infocase(PGconn* conn)
   Show the data of the case study, or the form for a new case study

insertcase
void insertcase(PGconn* conn)
   Insert the case study into the table casestudy.
**deletecase**

```c
void deletecase(PGconn* conn)
```

Delete the case study from the table asset

**changepcase**

```c
void changecase(PGconn* conn)
```

Update the case study into the table case study

**linkcase**

```c
void linkcase(PGconn* conn)
```

Shows the list of topic keywords

**makelinkcase**

```c
void makelinkcase(PGconn* conn)
```

Add and remove relations between the case study and topic keywords

**header**

```c
void header(char* helpSection)
```

Start the creation of an HTML page to show results to user

**footer**

```c
void header()
```

Finish the creation of the HTML page for the user
Document Archive Manager

Documents are the second kind of item contained in the Materials Base. The archive is divided into several sections. Each section contains a particular type of document: assets, publications, background studies, working documents, interviews, student works. Assets are already managed by the “Asset Designer”, therefore they are not included in this part of the system. For each section, the system allows the same actions. The main difference is the content of the form which holds information about the document.
1.2.3.2. Bibliography Manager

The system provides a page with the list of all publications stored in the archive. The user can choose to add a new document or to update an existent publication. In both cases, the system provides a form containing information regarding the chosen publication.
Form description

These are the fields of the form of the bibliography manager.

Title:  title of the publication
Author: author of the publication
Year of publishing:  year of publishing
Place of publishing:  place of publishing
Publisher:  name of the publisher
ISBN code:  ISBN code associated to the publication

Data-flow diagram
Process specifications

PUBL_MAN
Begin
  Connect to the master database
  If connection successful then
    Call PUBL_LIST function
    Show the result to the user
    If request = document creation then
      Call PUBL_CREATION function
    End if
  If request = document query then
    Call PUBL_QUERY function
  End if
  Close the master database connection
End if
End

PUBL_LIST
Begin
  Read the title, the ISBN Code and the document identifier of all publications
  from the BIBLIOGRAPHY table
  Show the result to the user
End

PUBL_QUERY
Begin
  Read the document identifier
  Read the publication’s data from the BIBLIOGRAPHY table
  Show the result to the user
  If request = document removal then
    Call PUBL_REMOVAL function
  End if
  If request = document modification then
    Call PUBL_MODIFICATION function
  End if
  If request = topic list then
    Call TOPIC_LIST function
  End if
End

PUBL_REMOVAL
Begin
    Read the document identifier
    Remove relations between topics and the document from DOCUMENT_TOPIC table
    Remove the publication from the BIBLIOGRAPHY table
End

PUBL_MODIFICATION
Begin
    Read the publication’s data and the document identifier
    Update the data into the BIBLIOGRAPHY table
End

PUBL_CREATION
Begin
    Read the publication’s data
    Save the data into the BIBLIOGRAPHY table
End

TOPIC_LIST
Begin
    Read the topic keywords from KEYWORD table.
    Show the result to user.
    If request = topic modification then
        Call TOPIC_MODIFICATION function
    End if
End

TOPIC_MODIFICATION
Begin
    Read the keyword identifier and the document identifier
    Insert and remove relations between keywords and the publication in the DOCUMENT_TOPIC table
End

Cgi description

The Bibliography Manager is implemented by the C program bibliography.cgi. Each program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

    main
main(int argc, char* argv)
   Open the connection with the remote DB, depending on the user request
   makes the corresponding action and finally close the connection

mainbiblio
void mainbiblio(PGconn* conn)
   Get the list of all the publications from the table bibliography

infobiblio
void infobiblio (PGconn* conn)
   Show the data of the publication, or the form for a new publication

insertbiblio
void insertbiblio (PGconn* conn)
   Insert the publication into the table bibliography

deletebiblio
void deletebiblio (PGconn* conn)
   Delete the publication from the table bibliography

changebiblio
void changebiblio (PGconn* conn)
   Update the publication into the table bibliography

linkbiblio
void linkbiblio (PGconn* conn)
   Shows the list of topic keywords

makelinkbiblio
void makelinkbiblio (PGconn* conn)
   Add and remove relations between the publication and topic keywords

header
void header(char* helpSection)
   Start the creation of an HTML page to show results to user

footer
void footer()
   Finish the creation of the HTML page for the user
1.2.3.3. Background Studies Manager

The system provides a page with the list of all background studies stored in the archive. The user can choose to add a new document or to update an existent background study. In both cases, the system provides a form containing information regarding the chosen background study.

**Form description**

These are the fields of the form of the bibliography manager.

- **Title:** title of the background study
- **Author:** author of the background study
- **Year:** year of the background study
- **Reference:** reference (URL) of the background study

**Data-flow diagram**

```
- 12.3.3 BACK_MAN
  | Data of background study
  | Request
  | 12.3.3 BACK_LST
  | List of available background studies
  | Document Identifier
  | 12.3.3 BACK_REMOVE
  | Remove a background study
  | Document Identifier
  | 12.3.3 BACK_CREATION
  | Create a new background study
  | Data of background study
  | 12.3.3 BACK_QUERY
  | Query the data of a background study
  | Data of background study
  | 12.3.3 BACK_MODIFY
  | Update the data of a background study
  | Data of background study
  | 12.3.3 TOPI
  | List of available keywords
  | Document Identifier
  | 12.3.3 TOPI_MODIFY
  | Add and remove a relation between a background study and a topic keyword
```

- 110 -
**Process specifications**

BACK_MAN

Begin

  Connect to the **master** database

  If connection successful then

    Call BACK_LIST function
    Show the result to the user

    If request = document creation then
      Call BACK_CREATION function
    End if

    If request = document query then
      Call BACK_QUERY function
    End if

  Close the **master** database connection

End if

End

BACK_LIST

Begin

  Read the title and the document identifier of all background studies from the BACKSTUDY table
  Show the result to the user

End

BACK_QUERY

Begin

  Read the document identifier
  Read the study’s data from the BACKSTUDY table
  Show the result to the user

  If request = document removal then
    Call BACK_REMOVAL function
  End if

  If request = document modification then
    Call BACK_MODIFICATION function
  End if

  If request = topic list then
    Call TOPIC_LIST function

End if
End

BACK_REMOVAL
Begin
  Read the document identifier
  Remove relations between topics and the document from DOCUMENT_TOPIC table
  Remove the study from the BACKSTUDY table
End

BACK_MODIFICATION
Begin
  Read the study’s data and the document identifier
  Update the data into the BACKSTUDY table
End

BACK_CREATION
Begin
  Read the study’s data
  Save the data into the BACKSTUDY table
End

TOPIC_LIST
Begin
  Read the topic keywords from KEYWORD table.
  Show the result to user.
  If request = topic modification then
    Call TOPIC_MODIFICATION function
  End if
End

TOPIC_MODIFICATION
Begin
  Read the keyword identifier and the document identifier
  Insert and remove relations between keywords and the background study in the DOCUMENT_TOPIC table
End

Cgi description
The Background Studies Manager is implemented by the C program backstudy.cgi. Each program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

**main**

```c
main(int argc, char* argv)
```

Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

**mainback**

```c
void mainback (PGconn* conn)
```

Get the list of all the background studies from the table backstudy

**infoback**

```c
void infoback (PGconn* conn)
```

Show the data of the background study, or the form for a new background study

**insertback**

```c
void insertback (PGconn* conn)
```

Insert the background study into the table backstudy.

**deleteback**

```c
void deleteback (PGconn* conn)
```

Delete the background study from the table backstudy

**changeback**

```c
void changeback (PGconn* conn)
```

Update the background study into the table backstudy

**linkback**

```c
void linkback (PGconn* conn)
```

Shows the list of topic keywords

**makelinkback**

```c
void makelinkback(PGconn* conn)
```

Add and remove relations between the background study and topic keywords

**header**

```c
void header(char* helpSection)
```

Start the creation of an HTML page to show results to user
void header()
    Finish the creation of the HTML page for the user

1.2.3.4. Working Documents Manager

The system provides a page with the list of all working documents stored in the archive. The user can choose to add a new document or to update an existent working document. In both cases, the system provides a form containing information regarding the chosen working document.

Form description

These are the fields of the form of the bibliography manager.

<table>
<thead>
<tr>
<th>Description</th>
<th>name of the working document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>description of the purpose of the sample document</td>
</tr>
<tr>
<td>Application</td>
<td>application needed to see the document</td>
</tr>
<tr>
<td>Reference</td>
<td>reference (URL) of the working document</td>
</tr>
</tbody>
</table>

Data-flow diagram

**Process specifications**

**WORK_MAN**

**Begin**

- Connect to the **master** database
  
  If connection successful then
  
  - Call WORK_LIST function
  
  Show the result to the user
  
  If request = document creation then
  
  - Call WORK_CREATION function

  End if

  If request = document query then
  
  - Call WORK_QUERY function

  End if

- Close the **master** database connection
WORK_LIST
Begin
   Read the description and the document identifier of all working documents from the WORKDOCUMENT table
   Show the result to the user
End

WORK_QUERY
Begin
   Read the document identifier
   Read the publication’s data from the WORKDOCUMENT table
   Show the result to the user
   If request = document removal then
      Call WORK_REMOVAL function
   End if
   If request = document modification then
      Call WORK_MODIFICATION function
   End if
   If request = topic list then
      Call TOPIC_LIST function
   End if
End

WORK_REMOVAL
Begin
   Read the document identifier
   Remove relations between topics and the document from DOCUMENT_TOPIC table
   Remove the working document from the WORKDOCUMENT table
End

WORK_MODIFICATION
Begin
   Read the working document’s data and the document identifier
   Update the data into the WORKDOCUMENT table
End

WORK_CREATION
Begin
Read the working document’s data
Save the data into the INTERVIEW table
End

TOPIC_LIST
Begin
Read the topic keywords from KEYWORD table.
Show the result to user.
If request = topic modification then
    Call TOPIC_MODIFICATION function
End if
End

TOPIC_MODIFICATION
Begin
Read the keyword identifier and the document identifier
Insert and remove relations between keywords and the working document in the DOCUMENT_TOPIC table
End

Cgi description

The Working Documents Manager is implemented by the C program workingdocs.cgi. Each program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

main

main(int argc, char* argv)
Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

maindoc

void maindoc (PGconn* conn)
Get the list of all the working documents from the table working

infodoc

void infodoc (PGconn* conn)
Show the data of the working document, or the form for a new working document

insertdoc

void insertdoc (PGconn* conn)
Insert the working document into the table working.

**deletedoc**

```c
void deletedoc (PGconn* conn)
```

Delete the working document from the table working

**changedoc**

```c
void changedoc (PGconn* conn)
```

Update the working document into the table working

**linkdoc**

```c
void linkdoc (PGconn* conn)
```

Shows the list of topic keywords

**makelinkdoc**

```c
void makelinkdoc(PGconn* conn)
```

Add and remove relations between the working document and topic keywords

**header**

```c
void header(char* helpSection)
```

Start the creation of an HTML page to show results to user

**footer**

```c
void header()
```

Finish the creation of the HTML page for the user

### 1.2.3.5. Interviews Manager

The system provides a page with the list of all interviews stored in the archive. The user can choose to add a new document or to update an existent interview. In both cases, the system provides a form containing information regarding the chosen interview.

**Form description**

These are the fields of the form of the bibliography manager.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name :</td>
<td>name of the interviewee</td>
</tr>
<tr>
<td>Role :</td>
<td>role of the interviewee</td>
</tr>
<tr>
<td>Reference :</td>
<td>reference (URL) of the interview</td>
</tr>
</tbody>
</table>

**Data-flow diagram**
### Process specifications

**INTER_MAN**

**Begin**

- Connect to the **master** database
- If connection successful then:
  - Call **INTER_LIST** function
  - Show the result to the user
- If request = case creation then:
  - Call **INTER_CREATION** function
- End if
- If request = case query then:
  - Call **INTER_QUERY** function
- End if
Close the **master** database connection
End if
End

INTER_LIST
Begin
Read the name of the interviewee and the document identifier of all case studies from the INTERVIEW table
Show the result to the user
End

INTER_QUERY
Begin
Read the document identifier
Read the interview’s data from the INTERVIEW table
Show the result to the user
If request = document removal then
    Call INTER_REMOVAL function
End if
If request = document modification then
    Call INTER_MODIFICATION function
End if
If request = topic list then
    Call TOPIC_LIST function
End if
End

INTER_REMOVAL
Begin
Read the document identifier
Remove relations between topics and the document from DOCUMENT_TOPIC table
Remove the publication from the INTERVIEW table
End

INTER_MODIFICATION
Begin
Read the document’s data and the document identifier
Update the data into the INTERVIEW table
End

INTER_CREATION
Begin
  Read the document’s data
  Save the data into the INTERVIEW table
End

TOPIC_LIST
Begin
  Read the topic keywords from KEYWORD table.
  Show the result to user.
  If request = topic modification then
    Call TOPIC_MODIFICATION function
  End if
End

TOPIC_MODIFICATION
Begin
  Read the keyword identifier and the document identifier
  Insert and remove relations between keywords and the interview in the
  DOCUMENT_TOPIC table
End

Cgi description

The Interview Manager is implemented by the C program interview.cgi. Each program
can be invoked with different names. Each name provides a different action on the
database. This is a description of the function used in this program:

main
main(int argc, char* argv)
  Open the connection with the remote DB, depending on the user request
  makes the corresponding action and finally close the connection

maininterview
void maininterview(PGconn* conn)
  Get the list of all the interviews from the table interview

infointerview
void infointerview (PGconn* conn)
  Show the data of the interview, or the form for a new interview

insertinterview
void insertinterview (PGconn* conn)
Insert the interview into the table interview.

**deleteinterview**
void deleteinterview (PGconn* conn)
Delete the interview from the table interview

**changeinterview**
void changeinterview (PGconn* conn)
Update the interview into the table interview

**linkinterview**
void linkinterview (PGconn* conn)
Shows the list of topic keywords

**makelinkinterview**
void makelinkinterview (PGconn* conn)
Add and remove relations between the interview and topic keywords

**header**
void header(char* helpSection)
Start the creation of an HTML page to show results to user

**footer**
void header()
Finish the creation of the HTML page for the user

### 1.2.3.6. Students Manager

The system provides a page with the list of all student works stored in the archive. The user can choice to add a new document or to update an existent student work. In both cases, the system provides a form containing information regarding the chosen student work.

**Form description**

- Description : *description of the work*
- Author : *name of the student which provided the document*
- Reference : *reference (URL) of the student work*

**Data-flow diagram**
**Process specifications**

**STUD_MAN**

Begin

Connect to the **master** database

If connection successful then

Call STUD_LIST function

Show the result to the user

If request = document creation then

Call STUD_CREATION function

End if

If request = document query then

Call STUD_QUERY function

End if
Close the **master** database connection

End if

End

STUD_LIST

Begin

Read the description and the document identifier of all student works from the STUDENTWORKS table

Show the result to the user

End

STUD_QUERY

Begin

Read the document identifier

Read the student work’s data from the STUDENTWORKS table

Show the result to the user

If request = document removal then

Call STUD_REMOVAL function

End if

If request = document modification then

Call STUD_MODIFICATION function

End if

If request = topic list then

Call TOPIC_LIST function

End if

End

STUD_REMOVAL

Begin

Read the document identifier

Remove relations between topics and the document from DOCUMENT_TOPIC table

Remove the publication from the STUDENTWORKS table

End

STUD_MODIFICATION

Begin

Read the student work’s data and the document identifier

Update the data into the STUDENTWORKS table

End

STUD_CREATION
Begin
  Read the student work’s data
  Save the data into the STUDENTWORKS table
End

TOPIC_LIST
Begin
  Read the topic keywords from KEYWORD table.
  Show the result to user.
  If request = topic modification then
    Call TOPIC_MODIFICATION function
  End if
End

TOPIC_MODIFICATION
Begin
  Read the keyword identifier and the document identifier
  Insert and remove relations between keywords and the student work in the DOCUMENT_TOPIC table
End

Cgi description

The Student Works Manager is implemented by the C program studentworks.cgi. Each program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

main
  main(int argc, char* argv)
    Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

mainwork
  void mainwork (PGconn* conn)
    Get the list of all the student works from the table studentwork

infowork
  void infowork (PGconn* conn)
    Show the data of the student work, or the form for a new student work
**insertwork**
void insertwork (PGconn* conn)
  Insert the student work into the table studentwork.

**deletework**
void deletework (PGconn* conn)
  Delete the student work from the table studentwork

**changework**
void changework (PGconn* conn)
  Update the student work into the table studentwork

**linkwork**
void linkwork (PGconn* conn)
  Shows the list of topic keywords

**makelinkwork**
void makelinkwork(PGconn* conn)
  Add and remove relations between the student work and topic keywords

**header**
void header(char* helpSection)
  Start the creation of an HTML page to show results to user

**footer**
void header()
  Finish the creation of the HTML page for the user
1.2.3 Software and Tools Archive Manager

The Software and Tools Archive completes the Learning Materials Base. This archive contains references to useful software and tools for the KAMP system. Items of this archive are divided in homogenous sections (categories). The system provides a page with the list of all software and tools stored in the archive. The user can choose to add a new software or to modify an existent software. In both cases, the system provides a form containing information regarding the chosen software.

Form description

These are the fields of the form of the case study manager:

- Name: title of the case study
- Description: site which hosts the case study
- Category: reference (URL) of the summary
- Reference: reference (URL) of the summary

Data-flow diagram
Process specifications

SOFT_MAN
Begin
  Connect to the master database
  If connection successful then
    Call SOFT_LIST function
    Show the result to the user
    If request = soft creation then
      Call SOFT_CREATION function
    End if
    If request = soft query then
      Call SOFT_QUERY function
    End if
  End if
Close the master database connection
End if
End

SOFT_LIST
Begin
Read the description and the software identifier of all software s from the SOFT table
Show the result to the user
End

SOFT_QUERY
Begin
Read the software identifier
Read the software’s data from the SOFT table
Show the result to the user
If request = soft removal then
Call SOFT_REMOVAL function
End if
If request = soft modification then
Call SOFT_MODIFICATION function
End if
If request = topic list then
Call TOPIC_LIST function
End if
End

SOFT_REMOVAL
Begin
Read the software identifier
Remove relations between topics and the study from SOFTWARE_TOPIC table
Remove the study from the SOFT table
End

SOFT_MODIFICATION
Begin
Read the software’s data and the software identifier
Update the data into the SOFT table
End

SOFT_CREATION
Begin
Read the software’s data
Save the data into the SOFT table
End

TOPIC_LIST
Begin
Read the topic keywords from KEYWORD table.
Show the result to user.
If request = topic modification then
    Call TOPIC_MODIFICATION function
End if
End

TOPIC_MODIFICATION
Begin
Read the keyword identifier and the document identifier
Insert and remove relations between keywords and the software in the
DOCUMENT_TOPIC table
End

Cgi description

The Software and Tools Archive Manager is implemented by the C program
software.cgi. The program can be invoked with different names. Each name provides a
different action on the database. This is a description of the function used in this
program:

main
main(int argc, char* argv)
    Open the connection with the remote DB, depending on the user request
    makes the corresponding action and finally close the connection

mainsoft
void mainsoft (PGconn* conn)
    Get the list of all the softwares from the table soft

infosoft
void infosoft (PGconn* conn)
    Show the data of the software, or the form for a new software

insertsoft
void insertsoft (PGconn* conn)
Insert the software into the table soft.

**deletesoft**

```c
void deletesoft (PGconn* conn)
```
Delete the software from the table soft

**changesoft**

```c
void changesoft (PGconn* conn)
```
Update the software into the table soft

**linksoft**

```c
void linksoft (PGconn* conn)
```
Shows the list of topic keywords

**makelinksoft**

```c
void makelinksoft(PGconn* conn)
```
Add and remove relations between the software and topic keywords

**header**

```c
void header(char* helpSection)
```
Start the creation of an HTML page to show results to user

**footer**

```c
void header()
```
Finish the creation of the HTML page for the user
1.2.3.8. Topic Keywords Manager

The Topic Keywords Manager is a tool which allows to handle topic keywords for the Learning Materials Base. Topic keywords can be used as search facilities, and provides the glossary of the Materials Base. Items of this archive are divided in homogenous topic section.

The system provides a page with the list of all keywords stored in the base. The user can choice to add a new keyword or to modify an existent keyword. In both cases, the system provides a form containing information regarding the chosen keyword.

**Form description**

These are the fields of the form of the topic keyword manager:

- **Keyword**: value of the keyword
- **Topic**: topic section which contains the keyword
- **Description**: a checkbox which indicates that the keyword has a description
  the description of the keyword
- **Similar**: a checkbox which indicates that is similar to another keyword
  the keyword which is similar to

**Data-flow diagram**
**Process specifications**

**KEY_MAN**

**Begin**

Connect to the **master** database

If connection successful then

Call KEY_LIST function

Show the result to the user

If request = keyword creation then

Call KEY_CREATION function

End if

If request = keyword query then

Call KEY_QUERY function

End if
Close the **master** database connection

End if

End

KEY_LIST
Begin
    Read the keyword and the keyword identifier of all keywords from the KEYWORD table
    Show the result to the user
End

KEY_QUERY
Begin
    Read the keyword identifier
    Read the s keyword’s data from the KEYWORD table
    Show the result to the user
    If request = keyword removal then
        Call KEY_REMOVAL function
    End if
    If request = keyword modification then
        Call KEY_MODIFICATION function
    End if
End

KEY_REMOVAL
Begin
    Read the keyword identifier
    Remove relations between the keyword and studies from STUDY_TOPIC table
    Remove relations between the keyword and documents from DOCUMENT_TOPIC table
    Remove relations between the keyword and softwares from SOFTWARE_TOPIC table
    Remove the keyword from the KEYWORD table
End

KEY_MODIFICATION
Begin
    Read the s keyword’s data and the keyword identifier
    Update the data into the KEYWORD table
End

KEY_CREATION
Begin
   Read the keyword’s data
   Save the data into the KEYWORD table
End

Cgi description

The Topic Keywords Manager is implemented by the C program keywords.cgi. The program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

main
   main(int argc, char* argv)
   Open the connection with the remote DB, depending on the user request makes the corresponding action and finally close the connection

mainkey
   void mainkey (PGconn* conn)
   Get the list of all the keywords from the table keyword

infokey
   void infokey (PGconn* conn)
   Show the data of the keyword, or the form for a new keyword

insertkey
   void insertkey (PGconn* conn)
   Insert the keyword into the table keyword.

deletekey
   void deletekey (PGconn* conn)
   Delete the keyword from the table keyword

changekey
   void changekey (PGconn* conn)
   Update the keyword into the table keyword

header
   void header(char* helpSection)
   Start the creation of an HTML page to show results to user

footer
void header()
    Finish the creation of the HTML page for the user
1.2.4. The Telematic Services Base Manager

The Telematic Services Manager is the set of tools used to manage the configuration files of some telematics services. Services that can be configured are mailing list, discussion groups and chat rooms. The aim of this part is to describe the main functionality of every level of the course management tool.

Data-flow diagram
1.2.4.1. Mailing Manager

The system provides the opportunity to create a new mailing list, to remove an existing
mailing list and to administrate a mailing list. When the user create a mailing list, he
provides a password, which must be verified for any other operation on the list.
The user can add and remove users from the mailing list with the administration
facilities.

**Forms description**

These are the fields of the form to create a mailing list:

- **Name**: name of the list
- **Administrator**: user which administrates the list
- **Password**: password used to authenticate the administrator
- **Max users**: maximum number of users of the list
- **Description**: description of the list
- **Student group**: student group associated to the list, if any

These are the fields of the form to delete a mailing list:

- **Name of the list**: name of the list
- **Password**: password of the administrator

These are the fields of the form to manage a mailing list:

- **Name**: name of the user to subscribe or unsubscribe

These are the fields of the form to subscribe a user to a mailing list:

- **Name**: name of the user to subscribe

**Data-flow diagram**
Process specifications

LIST_MAN

Begin

Connect to the local database

If connection successful then

  If request = list creation then
    Call LIST_CREATION function
  End if

  If request = list removal then
    Call LIST_REMOVAL function
  End if

  If request = list query then
    Call LIST_QUERY function
  End if

End if
Close the **local** database connection

End if

End

LIST_QUERY
Begin
  Read the name, the administrator and the description from the MAILINGLIST table
  Show the result to the user
End

LIST_USERS
Begin
  Read the mailing list identifier
  Read the users subscribed from the MAILING_LIST_SUBSCRIPTION and the USER tables
  Show the result to the user
  If request = subscribe user then
    Call LIST_SUBSCRIPTION function
  End if
  If request = unsubscribe user then
    Call LIST_UNSUBSCRIPTION function
  End if
End

LIST_REMOVAL
Begin
  Read the mailing list identifier
  Remove relations between mailing list and users from MAILING_LIST_SUBSCRIPTION table
  Remove the study from the MAILING_LIST table
End

LIST_CREATION
Begin
  Read the mailing list’s data
  Save the data into the MAILING_LIST table
End

LIST_SUBSCRIPTION
Begin
  Read the user identifier and the mailing list identifier
Insert the relation between user and mailing list into the MAILING_LIST_SUBSCRIPTION table
End

LIST_UNSUBSCRIPTION
Begin
Begin
 Read the user identifier and the mailing list identifier
 Delete the relation between user and mailing list from the MAILING_LIST_SUBSCRIPTION table
End

**Cgi description**

The Mailing List Manager is implemented by the C program mailinglist.cgi. The program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

**main**

code={
main(int argc, char* argv)
 Open the connection with the local DB, depending on the user request makes the corresponding action and finally close the connection

**addlist**

code={
void addlist()
 Show the form to add a mailing list

**deletelist**

code={
void deletelist()
 Show the form to delete a mailing list

**newlist**

code={
void newlist()
 Create a new mailing list

**rmlist**

code={
void rmlist()
 Remove a mailing list

**admlist**

code={
void admlist()
Show the list of mailing lists

**newuser**

void newuser()

Show the list of users in order to subscribe them to a mailing list

**listsubscribers**

void listsubscribers()

Show the list of people subscribed to a mailing list

**subscribe**

void subscribe()

Subscribe one or more users to a mailing list

**unsubscribe**

void unsubscribe()

Unsubscribe one or more users from a mailing list

**headerlist**

void headerlist(char* helpSection)

Start the creation of an HTML page to show results to user

**footer**

void header()

Finish the creation of the HTML page for the user
1.2.4.2. Discussion Groups Manager

The system provides the opportunity to create a new discussion group, to remove an existing discussion group and to retrieve the list of all discussion groups created in that site. The system prevents unauthorised users to remove discussion groups.

**Forms description**

These are the fields of the form to create a discussion group:

- **Name**: name of the discussion group
- **Administrator**: user which administrates the group
- **Description**: description of the group
- **Moderated**: articles are filtered by the administrator

These are the fields of the form to delete a discussion group:

- **Name**: name of the group

**Data-flow diagram**

```
<table>
<thead>
<tr>
<th>DISC_MAN</th>
<th>Discussion Groups Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Create, move, update or query a discussion group</td>
</tr>
</tbody>
</table>

1.2.4.2.1 DISC_LIST
Query the data of a discussion group

1.2.4.2.2 DISC_REMOVAL
Remove a discussion group

1.2.4.2.3 DISC_CREATION
Create a new discussion group

Data of a new discussion group
```

**Process specifications**

DISC_MAN
Begin
  Connect to the **local** database
  If connection successful then
    If request = list creation then
      Call DISC_CREATION function
    End if
    If request = list removal then
      Call DISC_REMOVAL function
    End if
    If request = list query then
      Call DISC_QUERY function
    End if
    Close the **local** database connection
  End if
End

DISC_QUERY
Begin
  Read the data of all discussion groups from the DISCUSSION_GROUP table
  Show the result to the user
End

DISC_REMOVAL
Begin
  Read the discussion group identifier
  Remove the discussion group from the DISCUSSION_GROUP table
End

DISC_CREATION
Begin
  Read the discussion group’s data
  Save the data into the DISCUSSION_GROUP table
End

**Cgi description**

The Discussion Groups Manager is implemented by the C program discussiongroup.cgi. The program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:

**main**
main(int argc, char* argv)
  Open the connection with the local DB, depending on the user request makes
  the corresponding action and finally close the connection

addgroup
  void addgroup()
  Show the form to add a discussion group

deletegroup
  void deletegroup()
  Show the form to delete a discussion group

newgroup
  void newgroup()
  Create a new discussion group

rmgroup
  void rmgroup()
  Remove a discussion group

listgroup
  void listgroup()
  Show the list of discussion groups

headergroup
  void headergroup(char* helpSection)
  Start the creation of an HTML page to show results to user

footer
  void header()
  Finish the creation of the HTML page for the user
1.2.4.3. Chat Rooms Manage

The system provides the opportunity to create a new chat room, to remove an existing chat room and to retrieve the list of all chat rooms created in that site. The system prevents unauthorised users to remove chat rooms.

Forms description

These are the fields of the form to create a chat room:

- **Name**: name of the chat room
- **Administrator**: user which administrates the room
- **Welcome message**: message that will be displayed when a user enters the room
- **Password**: password needed by users to enter the chat room

These are the fields of the form to delete a discussion group:

- **Name**: name of the chat room

Data-flow diagram

![Diagram](image)

Process specifications
CHAT_MAN
Begin
    Connect to the local database
    If connection successful then
        If request = list creation then
            Call CHAT_CREATION function
        End if
        If request = list removal then
            Call CHAT_REMOVAL function
        End if
        If request = list query then
            Call CHAT_QUERY function
        End if
        Close the local database connection
    End if
End

CHAT_QUERY
Begin
    Read the data of all chat rooms from the CHAT_ROOM table
    Show the result to the user
End

CHAT_REMOVAL
Begin
    Read the chat room identifier
    Remove the chat room from the CHAT_ROOM table
End

CHAT_CREATION
Begin
    Read the chat room’s data
    Save the data into the CHAT_ROOM table
End

Cgi description

The Chat Rooms Manager is implemented by the C program chatroom.cgi. The program can be invoked with different names. Each name provides a different action on the database. This is a description of the function used in this program:
main
main(int argc, char* argv)
  Open the connection with the local DB, depending on the user request makes
  the corresponding action and finally close the connection

addchat
void addchat ()
  Show the form to add a chat room

deletechat
void deletechat ()
  Show the form to delete a chat room

newchat
void newchat ()
  Create a new chat room

rmchat
void rmchat ()
  Remove a chat room

listchat
void listchat ()
  Show the list of chat rooms

headerchat
void headerchat(char* helpSection)
  Start the creation of an HTML page to show results to user

footer
void header()
  Finish the creation of the HTML page for the user
1.3. Student tools

This is the set of tools available for the student (excluding telematic tools, in part 1.4.). It is divided in four sections:

- 1.3.1. Delivery: shows all the information of the course
- 1.3.2. Agenda: shows the of the course
- 1.3.3. Material base browsing: shows the learning material base

Data-flow diagram

```
1.3 STUDENT TOOLS

1.3.1 DELIVERY
Course delivery: shows all the information of the course to the student

1.3.2 AGENDA
Shows the agenda of the group and course of the student

1.3.3 MATERIALS BASE BROWSING
Shows the learning material base
```
1.3.1. Course delivery

The course delivery is a tool used to show the courses to the student. Here students can see the list of the modules of the course, with their summary, description and information concerning to the copyright. For every module, there is a list of its activities. And for each activity you can see its description, information concerning to copyright and if the activity has assessment file. This assessment file, if exists, has to be downloaded and contains instructions to make some work concerning to the activity. Once the work the student has to upload the results.

**Data-flow diagram**

```
1.3.1 DELIVERY

CourseDelivery: shows all the information of the course to the student

- StudentLogin
- List of modules and activities
- Results
- Copyright

List of modules and activities
```

**Process specifications**

DELIVERY
Begin
  Connect to the local database
  If connection successful then
    Read the course identifier from the table USER
    Read the course copyright information from the table COPYRIGHT
    For each module of the course loop
      Read the module data from the table LOCAL_COURSE
      Read the module copyright information from the table COPYRIGHT
      For each activity of the module loop
        Read the activity data from the table LOCAL_COURSE
        Read the state of the activity from the table RESULTS
        Read the activity copyright information from the table COPYRIGHT
    End loop
  End loop
  Show the result to the user
End if
Close the database connection
End

Java class description

The course delivery is implemented into the Java class delivery.class. This is a description of the methods of this class:

main
public static void main(String args[])
  Open the connection with the remote DB and retrieve the course content and state for the user

initDelivery
public static boolean initDelivery()
  Get the user course
  Returns: True if no errors

initModules
public static boolean initModules(Long Course)
  Get the modules of the course
  Parameters:
    course - oid of the course we are browsing
  Returns: True if no errors
initActivities
public static boolean initActivities(Long Course, Long Module)
    Get the activities of the module and course
    Parameters:
        Course - oid of the course we are browsing
        Module - oid of the module we are browsing
    Returns: True if no errors

retrieveState
public static Integer retrieveState(Long User, Long Activity)
    Get the state of the activity for the user
    Parameters:
        User - oid of the user
        Activity - oid of the activity we are browsing
    Returns: the state of the activity represented by an integer

copyright
public static void copyright(int level, Long oid)
    Get the copyright information for the given Course, module or activity
    Parameters:
        oid - oid of the course, module or activity we want to get the copyright information

makeHtmlPage
public static void makeHtmlPage()
    Make the HTML page to show to the user if there were not any error

makeHtmlPageNoAgenda
public static void makeHtmlPageNoAgenda()
    Make the HTML page to show to the user if there were some error
1.3.2. Agenda

The agenda tool shows the agenda of the course and group of the student. In that way, every pair of course and group of students has one agenda, that contains all the actions to be done related to the course, like a chat session, the upload of an activity.

Data-flow diagram

Process specification

AGENDA
Begin
  Connect to the local database
  If connection successful then
    Read the Login of the student
    Read the group identifier of the student from the table USER
Read the agenda URL from the table STUDENT_GROUP
Open a window with the URL
End if
Close the database connection
End

Java class description

main
public static void main(String args[])
  Open the connection with the local DB, gets the user identifier and retrieve agenda of the user group.

initAgenda
public static boolean initAgenda()
  Retrieve the agenda of the user's group
  Returns: True if no errors

makeHtmlPage
public static void makeHtmlPage()
  Make the HTML page to show the agenda to the user

makeHtmlPageNoAgenda
public static void makeHtmlPageNoAgenda()
  Make the HTML page to show to the user if there were not any agenda
1.3.3. Learning Materials Base

The Learning Materials Base tool let user browse the materials base. This tool allows to visit the three archives, following links due to topic keywords. This tool also offers some browsing utility, like a map of the site and a keyword-based search functionality.

Data-flow diagram

![Data-flow diagram of Learning Materials Base tool](image-url)
1.3.3.1. Case studies browser

This tool allows to browse the content of the Case Studies Archive. The user can enter a case study and visit all sections in the right order, or he can select a pop up index and select the section he prefer to visit.

Data-flow diagram

```
CASE_BROWSE
  CASE_INDEX  CASE_QUERY
              CASE_DATA
                SHOW_SECTION
                  SHOW_CASE
```

Process specification

CASE_BROWSE
Begin
  Connect to the local database
  If connection successful then
Call CASE_LIST function
If request = query case
    Call CASE_QUERY function
End if
If request = summary browse
    Call SUMMARY_BROWSE function
End if
End if
Close the database connection
End

CASE_LIST
Begin
    Read the Study Identifier
    Read Title and Study Identifier from the table STUDY
    Show the result to the user
End

CASE_QUERY
Begin
    Read the Study Identifier
    Read study’s data from the table STUDY
    Show the result to the user
    If request = case section then
        Call SHOW_SECTION function
    End if
    If request = case index then
        Call CASE_INDEX function
    End if
    If request = whole case then
        Call CASE_PAGE function
    End if
End

SUMMARY_BROWSE
Begin
    Read Title, Summary reference and Study Identifier from the table STUDY
    Read summary sections using URL and merge them in a unique document
    Show the document to the user
    If request = case query then
        Call CASE_QUERY function
    End if
CASE_INDEX
Begin
  Read all section references from the table STUDY
  Show the result in a new window
  If request = case section then
      Call SHOW_SECTION function
  End if
End

SHOW_SECTION
Begin
  Read study Identifier and section
  Read URL for section from the table STUDY
  Show the document associated to the URL in the main page
End

SHOW_CASE
Begin
  Read all textual section references from the table STUDY
  Read sections using URL and merge them in a unique document
  Show the document to the user
End

CGI description
main
void main(int argc, char* argv[])
  Open the connection with the local DB, depending on the user request make the corresponding action and finally close the connection

allcases
void allcases()
  Show the list of all case studies in the database

maincase
void maincase()
  Show the list of all sections in a case study in the main window

caseindex
void caseindex()
    Show the list of all sections in a case study in a separate window

casestudypage
void casestudypage(char* id, int sect)
    Retrieve the URL of the section sect for the case study
    Show an HTML document containing the section

gentirecase
void getentirecase()
    Retrieve the URL of the all sections for a case study
    Show an HTML document containing all the sections

getsummaries
void getsummaries()
    Retrieve the URL of the summary section of all case studies
    Show an HTML document containing all the sections

header
void header(char* title, int section, char* identifier)
    Start the creation of an HTML page to show results to the user

footer
void footer()
    Finish the creation of the HTML page

1.3.3.2. Documents browse

This tool allows to browse the content of the Documents Archive. Each section provides a different layout of the data of the document.

Data-flow diagram
### Process specification

**DOCUMENT_BROWSE**

Begin

- Connect to the **local** database
- If connection successful then
  - Call **DOC_LIST** function
    - If request = query section
      - Call **SHOW_SECTION** function
    - End if
  - End if
- Close the database connection

End

**DOC_LIST**

Begin

- Show the document archive sections to the user

End

**SHOW_SECTION**
Begin
    Read the section
    Read documents data from the table associated to the section
    Show the result to the user
    If request = get document
        Show the document in a separate window
    End if
End

CGI description

main
void main(int argc, char* argv[])
    Open the connection with the remote DB, depending on the user request make the corresponding action and finally close the connection

mainpage
void mainpage()
    Show the list of all sections of the document archive

biblio
void biblio(PGconn* pg)
    Show the list of all elements in the Bibliography section

backstudy
void backstudy(PGconn* pg)
    Show the list of all elements in the Background studies section

workingdocument
void workingdocument(PGconn* pg)
    Show the list of all elements in the Working Documents section

interview
void interview(PGconn* pg)
    Show the list of all elements in the Interviews section

reference
void reference(PGconn* pg)
    Show the list of all assets

studentwork
void biblio(PGconn* pg)
   Show the list of all elements in the Student Works section

**header**
void header(char* title, int section)
   Start the creation of an HTML page to show results to the user

**footer**
void footer()
   Finish the creation of the HTML page

1.3.3.3. **Software browse**

This tool allows to browse the content of the Software and Tools Archive. The user can select the reference to the software and the system will try to get it.

**Data-flow diagram**

```
  1.3.3.3 SOFTWARE_BROWSE
   Learning Materials Base: Browse the Software and Tools Archive
      1.3.3.1 SOFT_LIST
         List the available categories
      1.3.3.2 SHOW_CATEG.
         Query the data of a software category
      category   sof
      Show all items in the category
```

**Process specification**

SOFTWARE_BROWSE
Begin
    Connect to the local database
    If connection successful then
        Call SOFT_LIST function
        If request = query section
            Call SHOW_CATEGORY function
        End if
    End if
    Close the database connection
End

SOFT_LIST
Begin
    Show the document archive sections to the user
End

SHOW_CATEGORY
Begin
    Read the section
    Read documents data from the table associated to the section
    Show the result to the user
    If request = get software
        Show the document in a separate window
    End if
End

**CGI description**

**main**
void main(int argc, char* argv[])
    Open the connection with the local DB, depending on the user request retrieve and show the list of categories or the list of items in a particular category. Finally close the connection

**header**
void header(char* title, int section)
    Start the creation of an HTML page to show results to the user

**footer**
void footer()
    Finish the creation of the HTML page
1.3.3.4. Materials Map

This tool allows to have a schema of the Materials Base. The map contains references to:

- the main page of each archive
- the main page of each case study
- the main page of each document section
- the main page of each software category
- the main page of each utility tool

The system will show the map in a separate window.

Data-flow diagram

```
  1.3.3.4 Learning Materials Base: Create the Map of the base
     |
     | 1.3.3.4.1 MAP_QUERY
     | Query the data of case studies and software category
     |  
     |   case study
     |   category
     |  
     | Show all items in the category

Process specification

MAP_QUERY
Begin
  Connect to the local database
  If connection successful then
    Read case studies titles from the table CASESTUDY
    Read category names from the table CATEGORY
    Show the result to user
    If request = move to
```
Show the chosen archive or document
End if
End if
Close the database connection
End

**CGI description**

This tool uses functions provided by the Case Studies Browser and the Software Browser.

**1.3.3.5. Topic Index - Glossary**

This tool allows to know the topic keywords associated to the Materials Base. Keywords are arranged into topic sections. This tool provides also the complete glossary of the Materials Base.

**Data-flow diagram**

![Data-flow diagram]

**Process specification**

KEY_LIST
Begin
   Connect to the **local** database

If connection successful then
  If request = topic list
    Read topic section name from the table TOPIC
    Show results to the user
  If request = show topic
    Read topic Identifier
    Read keyword name and topic identifier from the table KEYWORD
    Show results to the user
  End if
  End if
  If request = show glossary
    Read keyword’s data from the table KEYWORD
    Show the result to the user
  End if
End if
Close the database connection
End

**CGI description**

**main**

`void main(int argc, char* argv[])`

Open the connection with the local DB, depending on the user request retrieve and show the list of topic categories, the list of keywords in a particular category or the glossary of the base. Finally close the connection

**header**

`void header(int section)`

Start the creation of an HTML page to show results to the user

**footer**

`void footer()`

Finish the creation of the HTML page

**1.3.3.6. Search**

This tool allows to search elements in the Materials Base which are related to a particular keyword. The user must specify which archives must be searched.
Data-flow diagram

```
1.3.36 
MATERIALS SEARCH

Learning Materials Base: Search in the materials base

keyword

archives

SEARCH

Query the relation data of the keyword for the archives

Show elements associated to the keyword

study_topic

document_topic

software_topic

End
```

Process specification

SEARCH

Begin

Connect to the local database

If connection successful then

  Read keyword and archives

  Find items associated to the keyword in the chosen database

  Show results to the user

End if

Close the database connection

End

CGI description
main
void main(int argc, char* argv[])
    Open the connection with the local DB, depending on the user request retrieve and show items of the Materials Base which are related with the keyword used for the search process. Finally close the connection

header
void header(int section)
    Start the creation of an HTML page to show results to the user

footer
void footer()
    Finish the creation of the HTML page
1.4. Telematic Services Facilities

The system provides to user some basic telematic services. Most of the services are implemented by the client platform itself. However the system offers a software layer, which has been developed in order to facilitate the interaction with KAMP resources.

Basic telematic services are:

- email (personal messages as well as mailing lists)
- file transfer
- discussion groups
- chat rooms

For each service the system provides some facilities in order to simplify its usage.

1.4.1. Email

The system allows users to send easily messages to other users and to mailing list already subscribed. Moreover, the system tries to keep track of user presence, and people that are thought to be connected to the system are marked in a particular way. The system offers two facilities regarding email. The first is reserved only to support emails (i.e. site administrators, tutors, ..), while the second is general purpose.

Data-flow diagram

```
1.4.1 Email Service: Find out all email addresses to which the user can write.

User Identifier | EMAIL | Show allowed addresses
               | All   |                
               | EMAIL |                
               | user  |                
               | Mailing_list |              
               | Mailing_list_subscription |              
```

Process specification
EMAIL
Begin
   Connect to the **local** database
   If connection successful then
      If request = support
         Read Name, Email from USER table for support users
      Else
         Read Name, Email from USER table
      End if
      Read Name from MAILING_LIST table where subscribed relation is true
      End if
   Show results to the user
End if
Close the database connection
End

**CGI description**

**main**
void main(int argc, char* argv[])
   Open the connection with the local DB. Search all needed addresses, according to user request (support or full). Finally close the connection

**1.4.2. File Transfer**

The system allows users to connect to an FTP server within the KAMP system or outside it. The connection to KAMP site is direct. Users which want to connect to an external site must provide some information. Once connected the user can upload and download files using the client application.

**Forms description**

These are the fields of the form to connect to a site:

   Name : *name of the KAMP site*

These are the fields of the form to connect to an external site:

   Site address : *name or IP address of the site*
File path:  

path of the file to retrieve or the directory to connect to

Anonymous login:  

login as anonymous user

User login:  

login to access the server

Data-flow diagram

1.4.2  

FILE TRANSFER SERVICE  

1.4.2.1  

FTP  

List the KAMP sites

1.4.2.2  

FTP_EXT  

Create a URL for the chosen FTP site

Data to connect to an external site

Show the home page of the KAMP FTP site

Show the home page of the FTP site

Process specification

FTP

Begin

If request = KAMP site

Connect to the chosen site

Else

Call FTP_EXT function

End if

End

FTP_EXT

Begin

Read Site Name, Path, Login

Create the URL

Connect to the chosen site

End

Javascript description

validate
void validate()
  Validate data in the form and create a complete URL based on the data entered.

1.4.3. Discussion Groups

The system allows users to connect to the discussion group hosted in their site.

**Data-flow diagram**

![Data-flow diagram](#)

**Process specification**

GROUP
Begin
  Connect to the local database
  If connection successful then
    Read Name, Description from DISCUSSION_GROUP table
    Show results to the user
  End if
  Close the database connection
End

**CGI description**

**main**

```c
void main(int argc, char* argv[])
```
Open the connection with the local DB, search all discussion group names, and close the connection.

### 1.4.4. Chat Rooms

The system allows users to connect to a chat server of the site. The server is configured to host several chat rooms, and the user can choose one of them. The access is protected by a password. This service is based on a client application written in Java.

#### Applet interface description

These are the fields of the interface:

- **Recipients**: list of all people which joined the room
- **Chat board**: list of all messages sent to the room
- **Message**: message to send to other users
- **Send**: send button
- **Disconnect**: disconnect button

#### Data-flow diagram

1. Connect to the chat server
2. Send and connect to one of them.

#### Process specification

**CHAT**

Begin

Connect to the chat server
If connection successful then
   Select a chat room
   Send the password for that chat room
   While status = chat session opened
      Call CHAT_BODY function
   End if
   Close the database connection
End

CHAT_BODY
Begin
   If request = show message
      Show sender, message to the user
   End if
   If request = select recipient
      Select the recipient for the next message
   End if
   If request = send message
      Send message to the selected recipient
   End if
   If message = close session
      Disconnect from chat server
      status = chat session closed
   End if
End

Java description

init
   public void init()
      Verify user identity, connect to the chat server.

start
   public void start()
      Show any buffered message, execute any buffered command.

stop
   public void stop()
      Start buffering messages and commands

addConference
public void addConference(String id, String conference)
    Add a new conference to the conference list.

joinConference
public void joinConference()
    Choice the conference to join.

addUser
public void addUser(String id, String name, boolean forced)
    Add a user to the recipients list.

deleteUser
public void deleteUser(String id, boolean forced)
    Remove a user from the recipient list.

addText
public void addText(String message)
    Add the message to the Chat Board.

action
public void action(Event evt, Object arg)
    Connect server, disconnect server, select recipient or send message.

4. Installation procedures

4.1. General Description of KAMP architecture

The KAMP architecture is defined as a multiple Web server architecture. It contains a
MASTER site and 4 SLAVES sites.

Each Site will contain those elements:

- Web Server
- Database Server (Postgres Server) running for responding to SQL requests
  through the KAMP application.
- Telematic Servers:
• Email Server
• News Server
• Ftp Server
• Ircd Server

• The KAMP Database managed by the KAMP Database server
• The KAMP Application.

4.1.1. Definition of the MASTER Site

The **Master site** is the site where Common references between all KAMP Universities are stored. Those common references are:

- Courses, modules, activities, assets and copyright descriptions
- The information of the learning material base

Those references are modified in the MASTER site through the KAMP Application by users of all the Universities and the Replication Mechanism is in charge of the replication of all those data to all the other SLAVE Site. The replication is made from the MASTER to the SLAVES site during each night (if needed). Due to this mechanism, the data stored or modified during the day are not available in the Slaves site during the day. A user connected to a slave site has to wait for the day after to see the changes in his own site.

4.1.2. Definition of a SLAVE Site

A **Slave site** is a site where non Common references between all KAMP Universities are stored. Those non common references are:

- Users (staff and students)
- Groups of students
- Local Courses (copy of the main information of the courses and its hierarchy)
- Results of the students

In a Slave site, users are managing their own data with some centralised informations (which are created and modified in the master site) and are used the day after (replication during the night) for local purposes.

4.1.3. The Replication mechanism
The duplication of information amongst KAMP sites will be based on a replication mechanism. In the system there will be one major site which will act as master server for the shared information, while other sites will be configured as slave sites. All the changes that involve shared information will be performed on the master site database. If an operation is completed successfully, it will be replicated on each slave site database.

The replication of information will be based on the replication of the operations on data, rather than a replication of data themselves. The mechanism is divided into three phases:

1. the collection of the operations to replicate
2. the transfer of these information
3. the update of slave databases

The mechanism will be implemented applying a patch to the database management system of the master site and running automatically some scripts on each site.

4.1.3.1. Collection of the operations to replicate

The first phase is performed by the database management system of the master site, which will store in a file all the operations affecting shared information which has been completed successfully. In this phase the file containing the successful operation will be copied in a temporary file, in order to let the system store new updates to the database.

4.1.3.2. Transfer of the list of operations

The second phase is performed by the master site, which try to transfer the temporary file to the slave sites. The transfer is base on the remote copy Unix command, which allows to know if the file has been safely transferred or not. In order to solve any problem due to temporary failures in the connection amongst sites, the master site will try several times to transfer the file to the slave sites.
At the end of this phase the administrator of the master site will receive a notification about the state of the transfer to each slave site. The temporary file with the operation will be stored in a reserved directory, in order to maintain a copy of all the operations to duplicate.

4.1.3.3. Update of slave databases

- 177 -
The third phase is performed by the database management system of slave sites, which will execute all the operations written in the file transferred. During this part, the system can perform successfully the duplication, or can fail it; a third case is that the transfer of the updates file failed, and no operation is performed. At the end of this phase, the administrator of each slave site will receive a notification about the state of the duplication on his site.

The duplication will be performed once a day, in order to grant a quite fast replication of information.
4.2. Site Installation HOW-TO

This procedure can be used for an installation from the scratch of the system, or for an already installed any component of the system, but it is not working, you want to replace it. If you have already installed a part of the system, which is running, but you want to preserve it, please contact me (m.latini@cpr.it).

You can choose to merge the KAMP system with your SGI system, or you can create links to the service and keep the KAMP system in a partition on a separate machine.

The components of the system are:

- Apache web server (http daemon)
- Postgres DBMS
- Innd server (news daemon)
- Unix Listserver (mailing list daemon)
- WU ftpd server (ftp daemon)

The system contains three main directories:

- local contains the services and some data
- spool contains high volatile information (mailboxes and news repositories)
- etc contains some start/stop scripts

In the first part of the installation you will restore the directories "local" and "spool", and you will adjust links for the integration of the KAMP system within the SGI file system.

After the installation of the tar archive you should do other little operations:

- create four new users
- modify configuration files
- change the /etc/aliases file
- add some start-up/shutdown scripts
- add some crontab commands

4.2.1. Preinstallation checks

First of all, log as superuser (root) and check that the following user IDs are available:

- 1119
- 1120
- 1121
- 1122

These user IDs will be used for KAMP system users: please change the user ID which have one of these IDs.

Add to the /etc/group file the following two lines:

- kamp::103:
- news::104:
Please change the groupId to other groups, which have one of these IDs. Create the group nogroup, groupId of 60002 if it doesn't exist in /etc/group.

4.2.2. Restore tar archive

Choose the location which will hold the system on your file system (e.g., this installation procedure we'll refer this location with the $PATHINSTALL)

1. Create the directory $PATHINSTALL and go there.
2. Download the following files at this URL http://kamp2.uaib.es/download
   user “kampus” password “1kamping”
   KAMPsystem.tar.gz
   autoconfig.tar.gz

3. Uncompress the file KAMPsystem.tar.gz with the command gunzip and untar file with options ‘-xvf’, at the directory $PATHINSTALL

   Gunzip KAMPsystem.tar.gz
   Tar -xvf KAMPsystem.tar

Now you should see some directories which are not owned by Unix users (you user Id not the user name of the owner).

4. Construct links to integrate KAMP system in the SGI file system.

   cd /usr/local
   ln -s $PATHINSTALL/local/etc /usr/local/etc
   ln -s $PATHINSTALL/local/java /usr/local/java
   ln -s $PATHINSTALL/local/man /usr/local/man
   ln -s $PATHINSTALL/local/news /usr/local/news
   ln -s $PATHINSTALL/local/pgsql /usr/local/pgsql
   ln -s $PATHINSTALL/local/server /usr/local/listserver
   ln -s $PATHINSTALL/local/server /usr/local/server
   cd /var/spool
   ln -s $PATHINSTALL/spool/news news

If you have any problem to create the links mentioned above, please conta-

5. Check the exisitance of /usr/local/etc/ftpd/dev/zero

This file is a device needed for receiving the list of files in a directory (tar should maintain links, but should ignore devices) it must be created.

   cd /usr/local/etc/ftpd/dev
   mknod zero c 37 0
   chown root:sys zero
   chmod 666 zero
4.2.3. Create KAMP users

The KAMP system need four Unix users:

- postgres, which manages the Postgres DBMS
- news, which manages news daemon
- server, which manages listserver
- webadmin, which manages the Apache Daemon and the Chat Server

Now you must create these users. Please refer the table below in order to needed for the creation.

<table>
<thead>
<tr>
<th>User</th>
<th>Home directory</th>
<th>UserId</th>
<th>GroupId</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgres</td>
<td>/usr/local/pgsql</td>
<td>1119</td>
<td>103</td>
</tr>
<tr>
<td>news</td>
<td>/usr/local/news</td>
<td>1120</td>
<td>104</td>
</tr>
<tr>
<td>server</td>
<td>/usr/local/listserver</td>
<td>1121</td>
<td>103</td>
</tr>
<tr>
<td>webadmin</td>
<td>/usr/local/etc/httpd</td>
<td>1122</td>
<td>103</td>
</tr>
</tbody>
</table>

4.2.4. Change configuration files

There are some configuration file to modify.

/usr/local/etc/kampconfig contains the address of the local database for the manage local users and the username which should be used for all KAMP users activ:

/usr/local/etc/httpd/conf contains the configuration files of the Apache server. You should modify information regarding the port, the server name administrator email.

You should also modify the variables LOCAL_HOST_DATABASE, LOCAL_HOST_HTTP_SERVER, REMOTE_HOST_DATABASE and REMOTE_HOST_HTTP_SERVER (note that if you use an port different than 80 for server you should write it in the HOST_HTTP_SERVER variables, but not in HOST_DATABASE).

srm.conf: you should not modify it.
access.conf: you should modify information regarding database address. If you plan to use a virtual server for the KAMP site please contact me.

/usr/local/etc/httpd/htdocs/.params contains the address of the local database (you should change it), the name of the database, the username used to access the database password used to get html documents from the sites

/usr/local/server/config the configuration of the mailing list server. You should also remove regarding lists from the file /usr/local/server/owners (keep only the first entry).

/usr/local/news/inn.conf contains some information about sites. You should change
4.2.4.1. Changes in “/etc/aliases” file

The aliases file must be changed in order to create aliases for users email list.
The modification will be inserted after the following line:

# end of common aliases--do not remove this line

First of all you must insert an alias for the news daemon:

usenet:news

The next step is to insert the following line

# KAMP aliases

After this line the system will automatically place users aliases. It is very important not to modify its content.
Now you can insert a blank line, followed by two lines:

# Program aliases
listservers: "|/usr/local/server/catmail -r -f"

As stated before the commented line is used by tools which manage automatic updates. You must not modify it.
The last line contains the alias for the mailing list server. Aliases for other services can be added similarly.
You must also create the file ‘/etc/.LOCK_ALIASES’, used to control multiple users access to the aliases file.

4.2.5. Start-up/shutdown scripts

Move the file autoconfig.tar.gz in the / directory. Uncompress the file with gunzip and untar it with options ‘-xvf’.

    gunzip autoconfig.tar.gz
tar -xvf autoconfig.tar

This procedure will unpack the archive which holds the scripts for the automatic start-up/shutdown of services. The archive contains three kinds of file: actual scripts and system links.

Actual scripts are batch procedures which allow to start and to stop a daemon service with the following scripts:
apache, which handles http daemon
chat, which handles chat daemon
listserver, which handles mailing list daemon
news, which handles news daemon
postgres, which handles the database manager
Control files contain only one word ("on" or "off"). Each script can be enabled or disabled changing the content of its control file.

System links allow the system to call automatically the correspondent actuator.

### 4.2.6. Services handled by inetd

You must add two entry in the "/etc/inetd.conf" files in order to handle file mail download request.
You should comment the line for the ftp service (the line starting with "ftp:
following line:

```bash
ftp  stream  tcp  nowait  root /usr/local/etc/ftpd/bin/ftpd
ftpd -l
```

### 4.2.7. New crontab entries

You must add the following crontab commands.

**User postgres**

```bash
# vacuum database
30 4 * * * /usr/local/pgsql/bin/psql -c "vacuum" -o vacuum.log -d KAMP
# Replications
0 5 * * * /usr/local/pgsql/bin/dupprepare
15,30,45 5 * * * /usr/local/pgsql/bin/dupscript
0 6 * * * /usr/local/pgsql/bin/masterdupcheck
```

**User news**

```bash
5 6 * * * /usr/local/news/bin/news.daily
0,15,30,45 * * * * /usr/local/news/bin/nntpsend
```