Empweb Manual
Version 0.2

August 2009

Contents

Introduction

1- What is EmpWeb, who developed it (is it part of ABCD or complementary to ABCD?)
2- How does it work (broad outline)
3- What is the underlying technology
4- How does EmpWeb deal with loans transactions and how is this implemented ?
5- For what kind of libraries is it recommended

Requirements before installing EmpWeb v0.9

Before installing EmpWeb you must have the following software installed:

- MySql. version mysql-essential-5.1.35-win32 or later.
- ABCD version ABCD_full_20090702 or later.

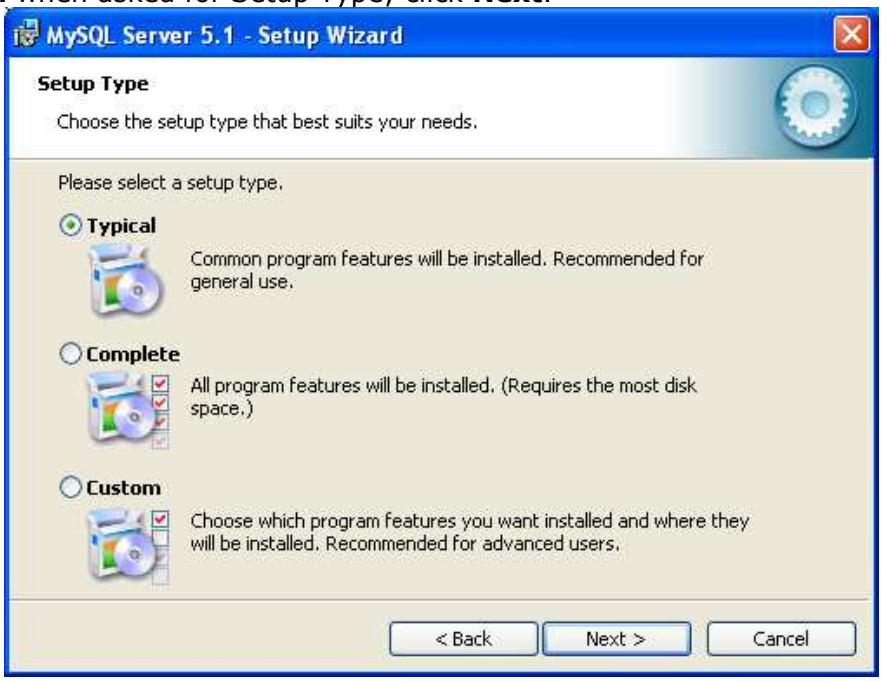
These programs interact with each other and are necessary for the correct functioning of EmpWeb.

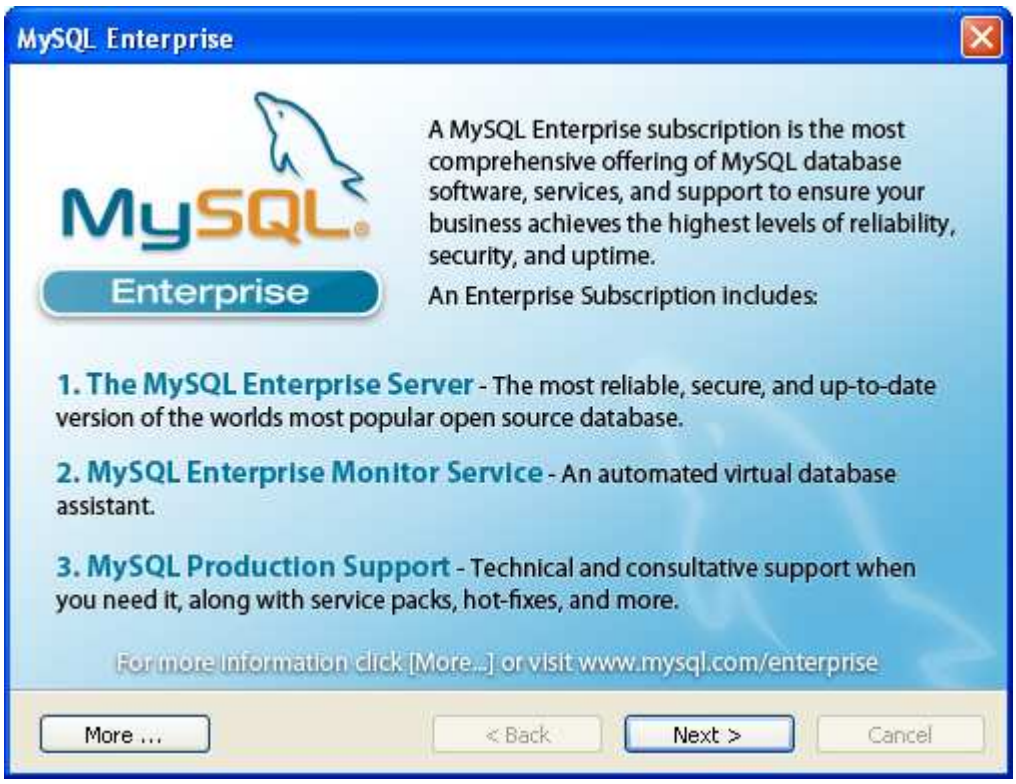
If you already have MySql and ABCD installed, jump to the section Installing EmpWeb.
If you already have MySql and need to install ABCD, jump to the section Installing ABCD.

Warning: Before installing any of these software packages, be sure to make the necessary backups of databases and personal files.

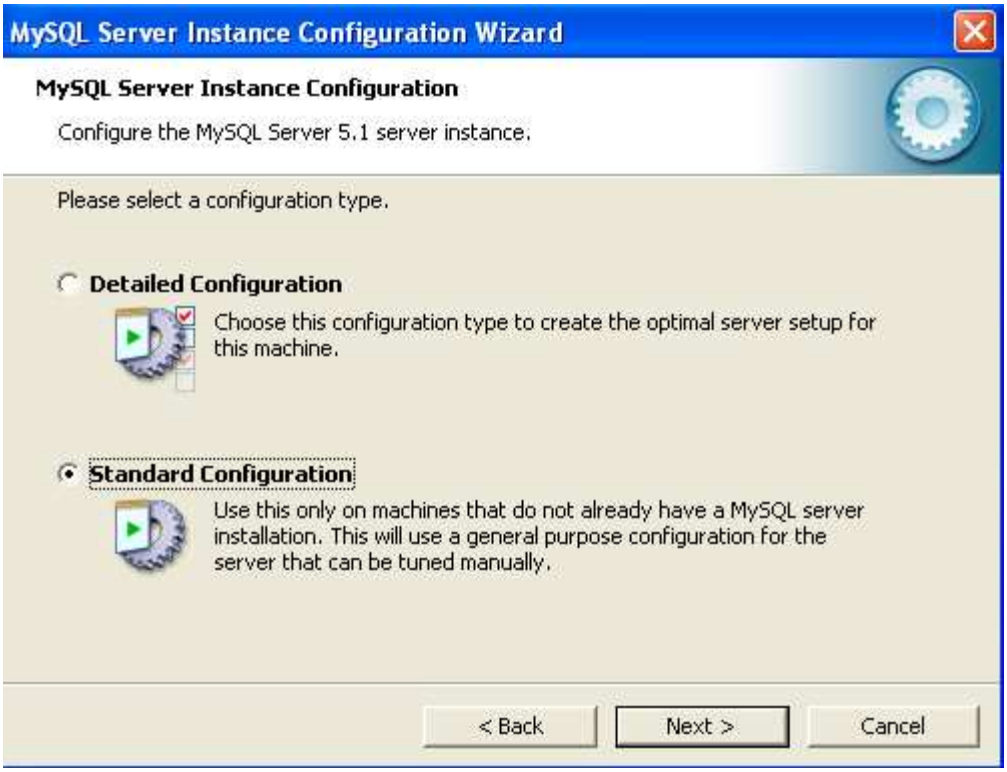
Installing MySql

Download the MySql installer **mysql-essential-5.1.35-win32.msi** from your favourite downloading site and run it to start the step by step installation wizard.
Select **Typical** when asked for Setup Type; click **Next**.





The installation process takes a few minutes, after which you are asked to select configuration details.
In the configuration wizard, select **Standard Configuration**; click **Next**.



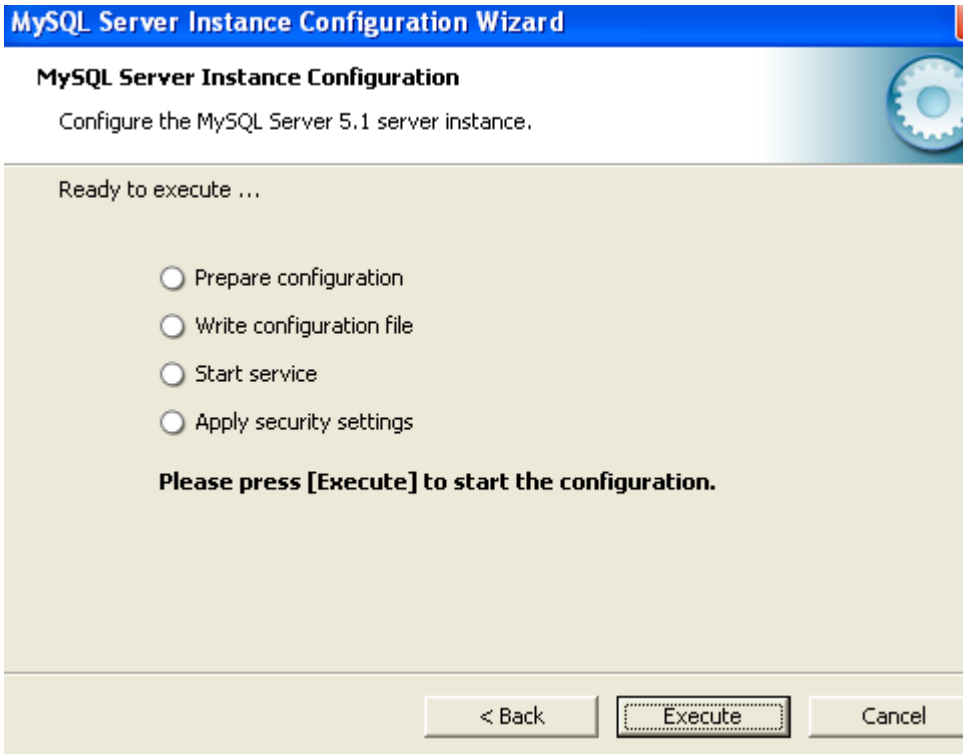
In the next window, select **Install AS Windows Service;** in **Service Name** select **MySql**; select also **Include Bin directory in PATH** (it is not needed by EmpWeb, but is essential for troubleshooting MySql later). Click **Next**.

The next step requires a **root password**.
If this is the first time you install MySql on this computer, the **Current root password** field (marked in red on the figure) will not appear.

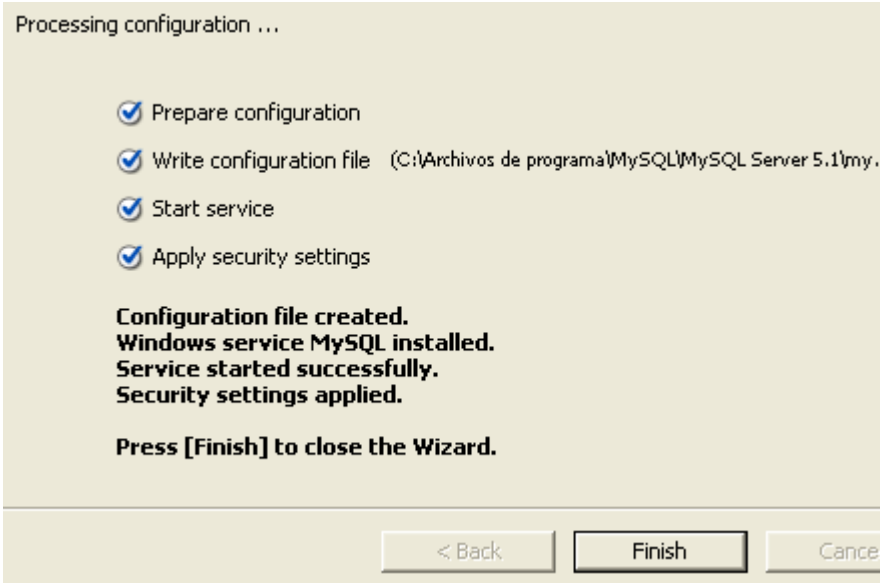
If you have had a previous MySql version installed, the **Current root password** window will appear and you must enter the corresponding root password.
In the **New root password** field you should enter '**empweb',** and repeat the same password in t**he Confirm:** field**.**

After entering the empweb password and clicking Next, there is nothing more to configure in this setup process. The EmpWeb installation package has **root/empweb** pre-defined as the MySql connection account.

Next you will see a screen like the one below, showing that the configuration will take place. Click **Execute**.



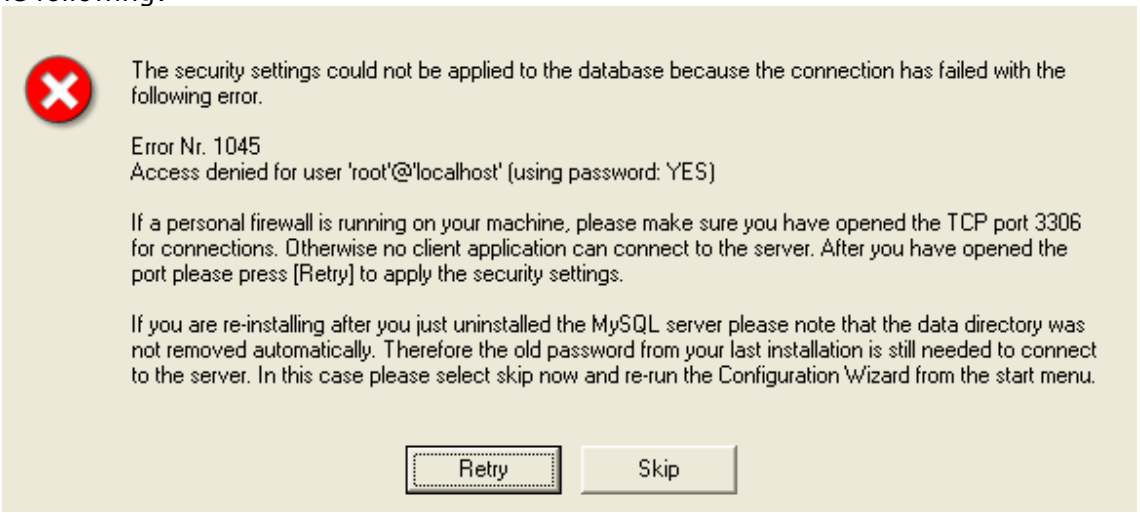
If the configuration runs without any error, a screen like the one below will appear, informing that everything has been successful. Click **Finish** to complete the installation process.



After installing and configuring MySql successfully, you should continue with the installation of ABCD and finally with EmpWeb.

Warning:

If you have had a previous MySql installation on the current computer, the setup process will probably not finish successfully, and you will get error messages. The most common message is the following:



This message says that the Current root password is not the correct one. In this case, click **Skip** and go to the MySql Troubleshooting section for help.

MySql Troubleshooting

Problems during MySql setup

The most common problem has to do with the previous existence of MySql installations on the same computer, because the old root/password remains stored after uninstalling the product.

If you have previously uninstalled MySql from the Control Panel / Add remove programs, most probably the old MySql Service is still active and conflicts with the new installation.

ErrorCode 0 – Error Number 0 - Problem 0.

The installer returns the null error code when attempting to start the MySql service and another version of the service is already running.
You can solve the problem by stopping the MySql Service (from the Start Menu - Control Panel - Administrative Tools – Services. Find MySql in the list of services, doubleclick and select STOP) or by rebooting the computer to stop the old service and start the new one.

ErrorCode 1045 – Error Number 1045 - Problem 1045.

Another frequent problem on machines working as MySql servers, is that the current operator does not know the root/password of the previous installation. The installer will always ask for the old root password if it detects that there has been a previous installation, and there is no way to skip this step.
In this case, we recommend the following procedure for a successful MySql installation:

1- Uninstall any MySql versions from the computer.
2- Delete the C:\Program files\MySql folder
3- Install the new MySql version

You will probably still get the Error 1045 message. Press **Skip**.

4- Stop the MySql Service from the Start Menu - Control Panel - Administrative Tools – Services. Find MySql in the list of services, doubleclick and press STOP.
5- Start a command console: from Start – Run. Write **cmd** and press **Enter.**
6- In the command console window, move to the MySql program folder by writing: **cd C:\Program Files\MySQL\MySQL Server 5.1\bin (or any other folder where MySQL was installed)** and pressing **Enter**.
7- Create a textfile called **mysql-init.txt** (from the console with the command: **edit mysql-init.txt**).
8- Copy the following lines into the mysql-init.txt file:
UPDATE mysql.user SET Password=PASSWORD('empweb') WHERE User='root';
FLUSH PRIVILEGES;
9- Save and close the file. Go back to the command console.
10- Run the following command:
`mysqld --init-file=mysql-init.txt --console`
11- You will receive a message indicating that MySql has been initiated and the root/password has been changed.
12- Close the command console.

Restart the MySql Service from Start - Control Panel - Administrative Tools – Services.
Search for MySql in the list of services, double click and press Start.

More information about resetting MySql passwords can be found at:
http://dev.mysql.com/doc/refman/5.0/en/resetting-permissions.html

Installing ABCD

ABCD is the library management software suite in which EmpWeb and the MySql transactions database will be working as 'Advanced Loans' module.

ABCD itself comes with a relatively simple loans management module that uses standard ISIS databases for user registration and loan transactions. However, for heavy duty lending environments such as universities or large public libraries with many concurrent transactions and the need for detailed borrower profiles, EmpWeb is a more robust and developed solution.
After installing ABCD, when you run the EmpWeb installer the setup process will overwrite some of the ABCD configuration files so as to merge functionally with the suite.
For the correct installation of the ABCD suite, please check the ABCD Manual.

Installing Empweb

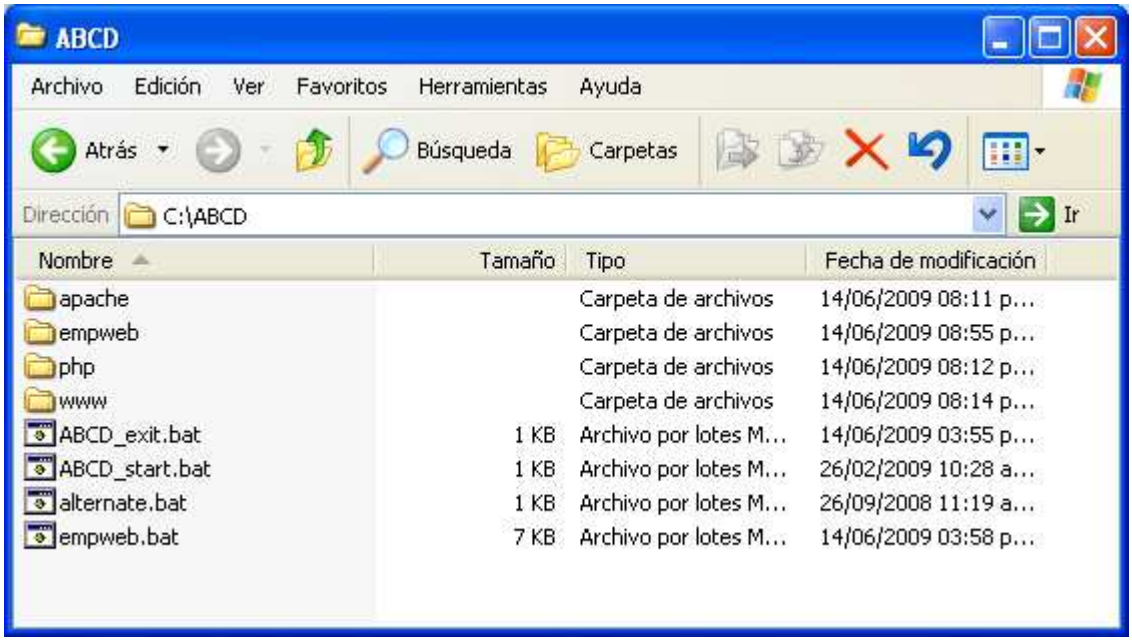
Before proceeding with the installation of EmpWeb, be sure you have MySql and ABCD correctly installed and functioning. These programs will interact and are necessary for EmpWeb to work correctly.

- **MySql. Version mysql-essential-5.1.35-win32** or later, configured according to the instructions given in the Install MySql section of this Manual.
- **ABCD. Version ABCD_full_20090702** or later. Please check the ABCD Manual for installation details.

Warning:
If you have a previous MySql installation on your computer, you will have to create a new transactions database in MySql called **transa**, and assign to it a username and passwod according to the EmpWeb configuration. See the section on Manual configuration of EmpWeb under MySql, below for more details.

In order to install EmpWeb, you should first download and unzip the installation file (EMPWEB_20090813.zip or later) into a temporary folder and then copy the resulting folder tree to the root level at C:. ABCD and EmpWeb share the same folder structure and should be merged.
Click Accept/YES when prompted to overwrite existing files and folders. EmpWeb will overwrite some configuration files in apache, php and ABCD, and will affect MARC and users databases, so backing up these is advised.

The main ABCD folder should have the following structure after the EmpWeb files have been copied correctly.

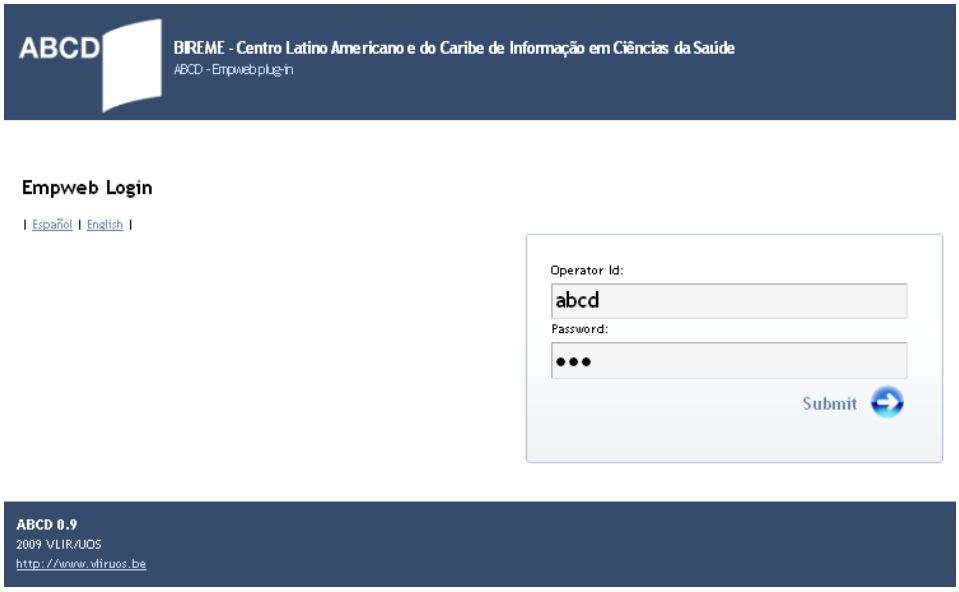


When you activate ABCD_start.bat, two command console windows will open, one for starting Apache and another for the Jetty_Java_server needed by EmpWeb. After a few minutes, the Apache console window closes and the Java_server remains open. Do not close this window while using EmpWeb, or it will not work properly.

In order to close the applications, run ABCD_exit.bat. This batch file stops the services initiated by Apache and closes the Java_server console window.

Initiating EmpWeb.

After activating ABCD_start, enter the URL http://localhost:9090/empweb/ in your browser.
After a few seconds the main login page of EmpWeb will open. Remember to add the final slash to the address, or EmpWeb will not work properly.



Enter the same administrator data as for ABCD (abcd/adm) and press **Send/Submit**.

A second login page will be displayed, where you should select a library. This step is necessary for users with access privileges to more than one library.

Select the library you will work in and press Send/Submit. The EmpWeb application will now open.

Main configuration of EmpWeb.

After accessing with login, the main Transactions page will open, and you must perform a very important step before being able to use EmpWeb for the first time, i.e. initialising the transactions database.

At the bottom of the page just above the footer (see red rectangle in the illustration), you will find a status list with information about the transactions related to the current library. This information shows data from the transactions database and should be empty because no transaction has yet taken place. Every item on the list should show a status value of 0 (null) at its right, but the first

time we open the Transactions page only the item names are shown because the transactions database has not been initialised yet.

In order to initialise the transactions database, go to the Administration menu, select the Databases (Bases) sub-menu and a initialisation Confirmation/Warning window will appear. Confirm the operation by clicking **YES**. If the process is successful, you will receive a confirmation message.

Back at the main Transactions page, you will now see that a 0 (null) figure has appeared at the right of each Status item, confirming that the transa database has been activated.

Note: This is proof that the connection between EmpWeb and MySql is correct.

Database initialisation problems.

As mentioned before, if you had a previous MySql installation on your computer, the old root/password and other configuration data remains in the system even if you delete the Program Files\MySql folder completely, and may affect the functioning of EmpWeb, including the initialisation of the transa database.

The solution to this problem is to open a console, invoke mysql and manually drop the database:

1- Open a cmd window.
2- Execute the command **mysql –uroot –p** (press Enter)
3- *** insert the password *** (type the password 'empweb') (press Enter)
4- mysql> DROP DATABASE transa; (press Enter)
5- mysql> quit; (press Enter)

After doing this, you will be able to initialise the mysql database from EmpWeb.

Manual configuration of EmpWeb under MySql.

It is not recommended to operate in production mode with the root/empweb user setting. You should change to another username/password combination in a final production environment.
In order to configure the root user account manually, you have to edit the file /ABCD/empweb/engine/WEB-INF/conf/engineconf.xml. Use a text editor like Notepad for this. Locate line 150 and from there on you will find the following content:

```
<base name="TRANSA" type="transa">
<uri>jdbc:mysql://localhost/transa</uri>
<user>root</user>
<password>empweb</password>
<schema>ew15db-schema.sql</schema>    <!-- It's a resource inside ew15db.jar -->
<backupDir>C:/ABCD/empweb/db</backupDir>
<poolSettings>
  <driverClassName>com.mysql.jdbc.Driver</driverClassName>
  <minPoolSize>3</minPoolSize>
  <maxPoolSize>20</maxPoolSize>
  <initialPoolSize>3</initialPoolSize>
  <acquireIncrement>2</acquireIncrement>
  <idleConnectionTestPeriod>30</idleConnectionTestPeriod>
  <testConnectionOnCheckin>false</testConnectionOnCheckin>
  <automaticTestTable>ew_test_table</automaticTestTable>
  <maxIdleTime>30</maxIdleTime>
</poolSettings>
<collation></collation>
</base>
```

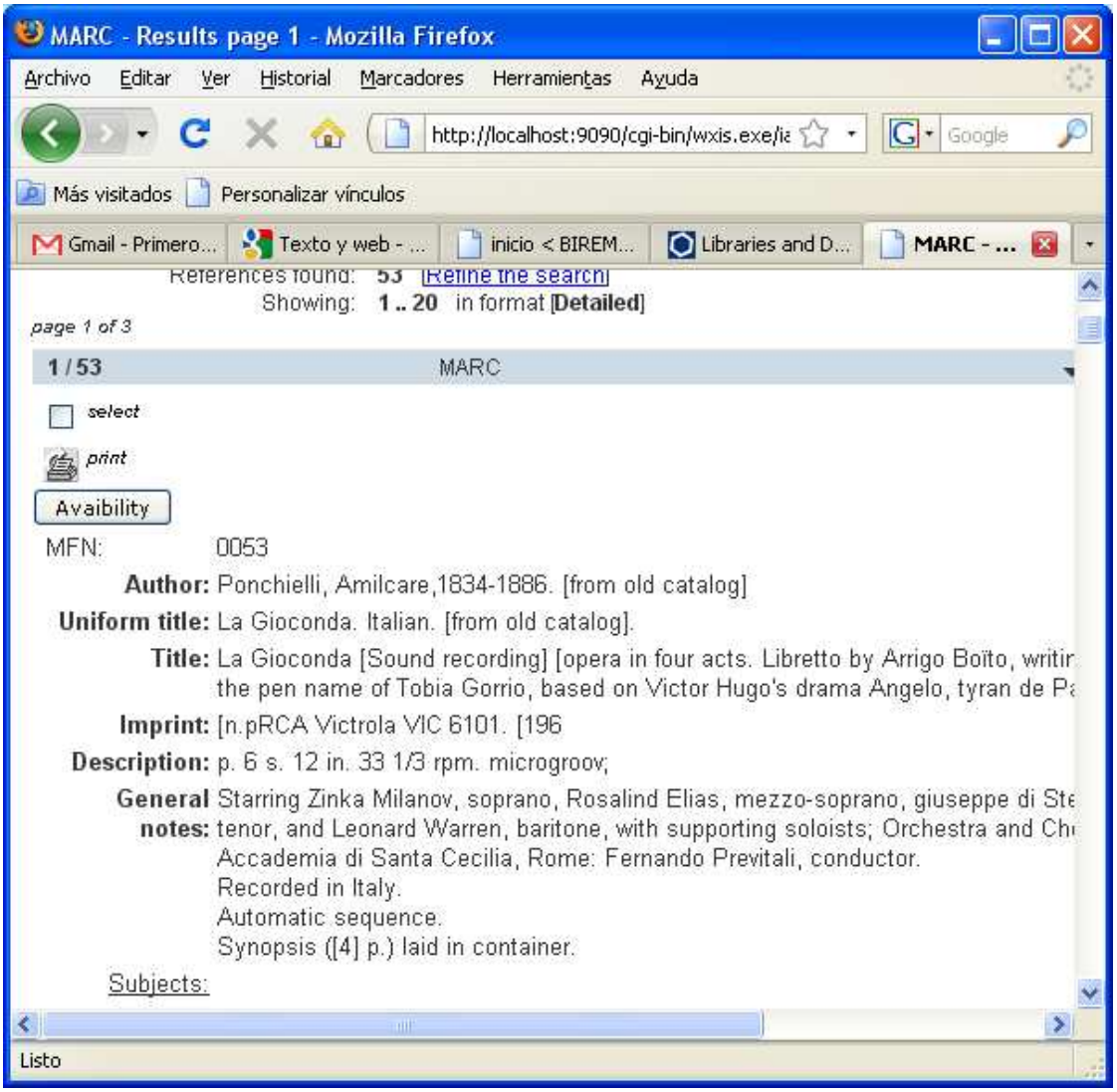
Change <user> and <password> and save the file. For the changes to have effect, you must shut down ABCD with ABCD_exit.bat, and then restart using ABCD_start.bat

Transactions

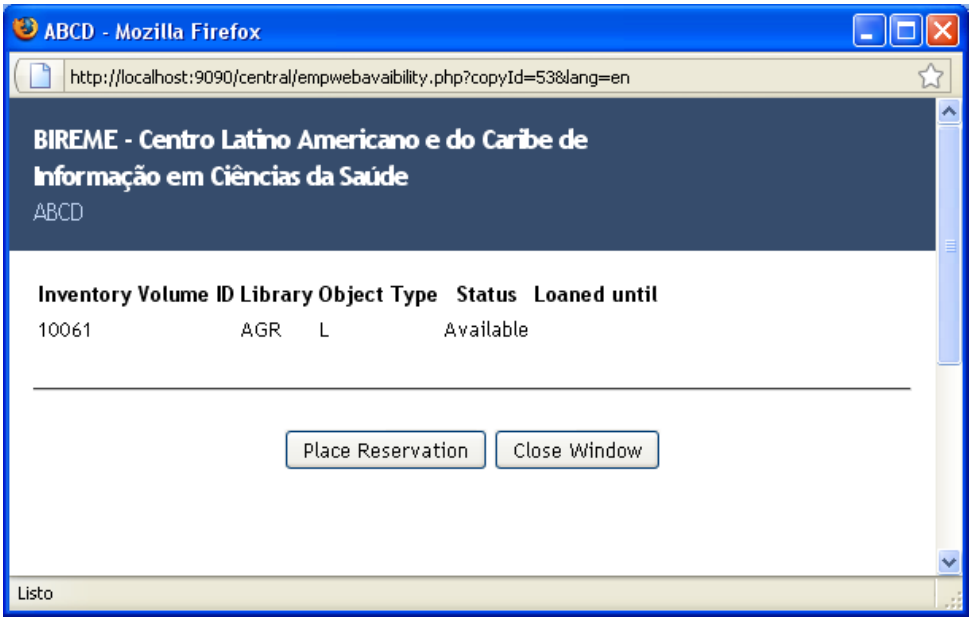
The EmpWeb installation package is pre-defined to do lending transactions with the MARC bibliographic database.
When you go to MySite (http://localhost:9090/site) and access the MARC database directly, if you do a successful search an Availability button will appear at the left of each item registered as a loan object. The association is done with the Control Number field as key.

Click the Availability button and a window opens displaying the available copies of the selected record.

In the current EmpWeb demo installation loanobjects items have been defined for a number of records. You can access these doing a generic search ($) and the following screen will appear:

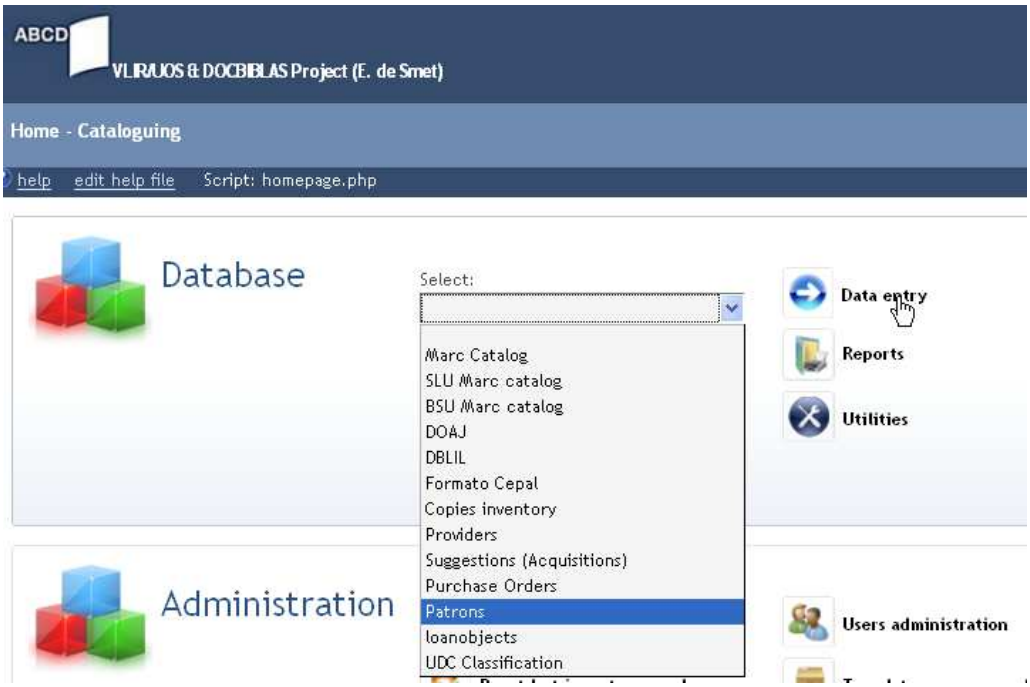


Click the Availability button and the existing copies are shown.



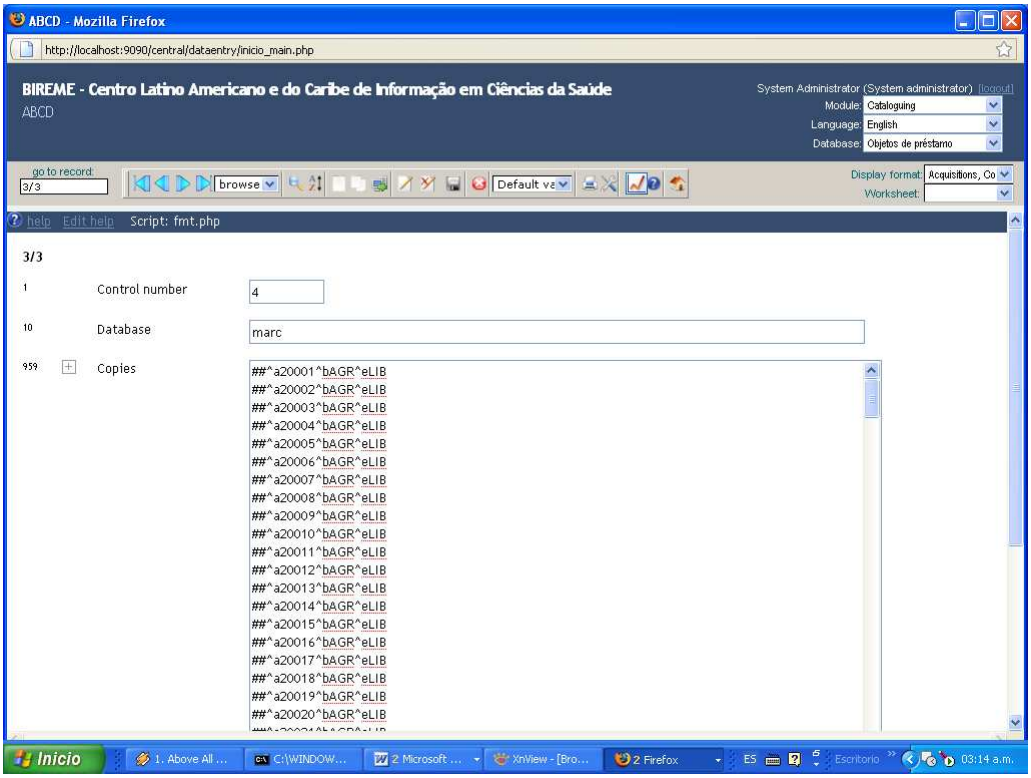
As you can see, it is possible to make a reservation of this item clicking the Place Reservation button. This will open the MySite page to execute the reservation. The program will ask for login/password to access the Users database.

To have access to the Users (or patrons) database you must go to the main ABCD page (localhost:9090/) and login as a user with Loans Administrator profile. In the database you can Edit/Create/Modify users applying the necessary permissions to access MySite.



The loanobjects database was included in this distro to be shown as one of the editable databases in ABCD for testing purposes only. Remember that loanobjects is an internal database administered by ABCD and is not supposed to be directly edited by the final user.

By editing the records you can load data in the different fields. Field 001 is the ID-unique identifier for the record; field 10 is the database name (MARC in our case) and the repeatable field 959 contains the unique identifier for each loanable copy.


ABCD - Mozilla Firefox
http://localhost:9090/central/dataentry/inicio_main.php
BIREME - Centro Latino Americano e do Caribe de Informação em Ciências da Saúde
ABCD
System Administrator (System administrator) [logout]
Module: Cataloguing
Language: English
Database: Objetos de préstamo
go to record: 3/3
browse
Default va
Display format: Acquisitions, Co
Worksheet:
help Edit help Script: fmt.php
3/3
1 Control number 4
10 Database marc
959 Copies
##^a20001^bAGR^eLIB
##^a20002^bAGR^eLIB
##^a20003^bAGR^eLIB
##^a20004^bAGR^eLIB
##^a20005^bAGR^eLIB
##^a20006^bAGR^eLIB
##^a20007^bAGR^eLIB
##^a20008^bAGR^eLIB
##^a20009^bAGR^eLIB
##^a20010^bAGR^eLIB
##^a20011^bAGR^eLIB
##^a20012^bAGR^eLIB
##^a20013^bAGR^eLIB
##^a20014^bAGR^eLIB
##^a20015^bAGR^eLIB
##^a20016^bAGR^eLIB
##^a20017^bAGR^eLIB
##^a20018^bAGR^eLIB
##^a20019^bAGR^eLIB
##^a20020^bAGR^eLIB
Inicio
1. Above All ...
C:\WINDOW...
2 Microsoft ...
XnView - [Bro...
2 Firefox
ES
Escritorio
03:14 a.m.

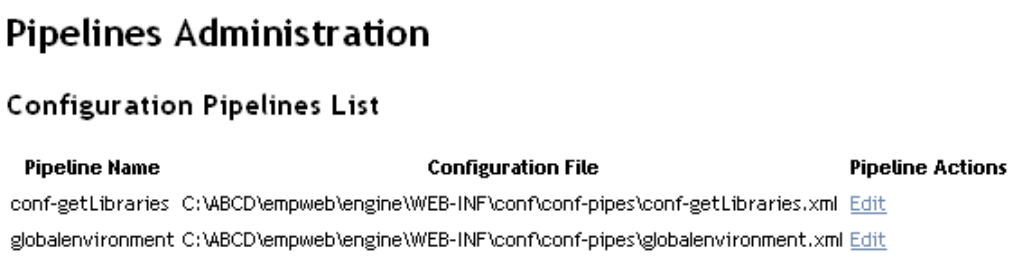
Instructions for the configuration of libraries (v0.85)

The steps to configure libraries in EmpWeb are the following:

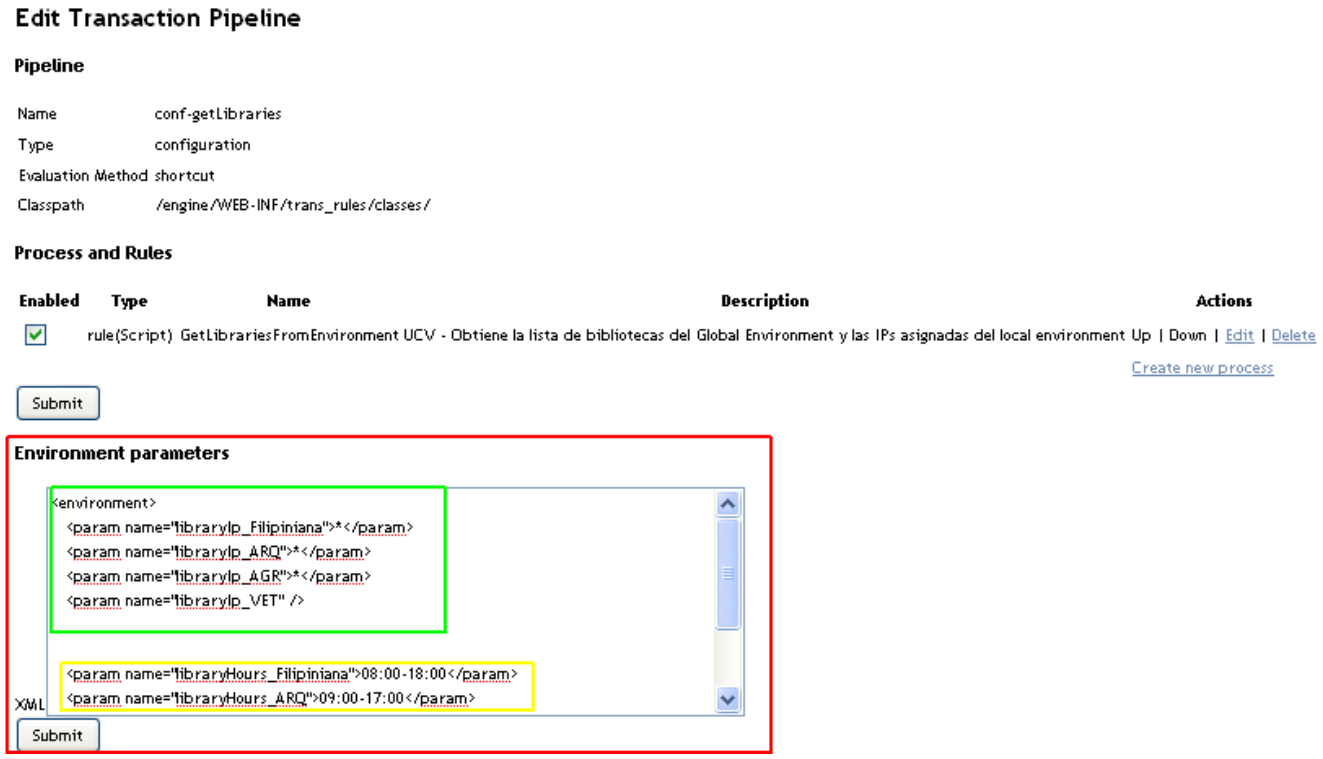
1. Change the library definition in the **conf-getLibraries pipeline**.
2. Change global parameters of the library in the **globalenvironment pipeline**.
3. Assign user rights.

1. Change the library definition in conf-getLibraries pipeline

After starting the EmpWeb application, select Administration -> Pipelines and a screen appears with the title Transaction Pipelines Administration (similar to the figure below) where you will see the first two options.



To make changes, clic on Edit and the following window appears.



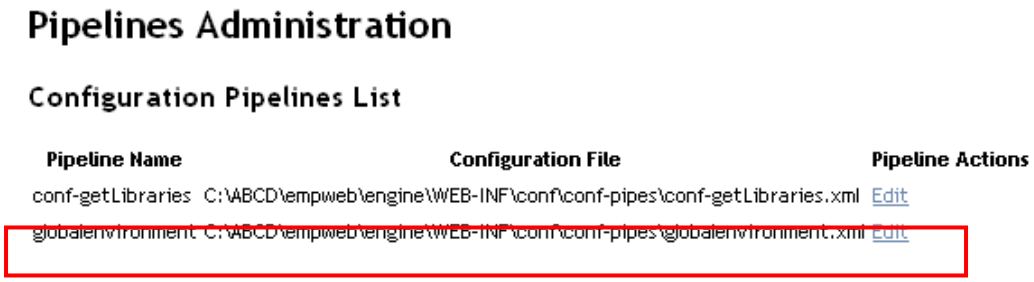
The XML parameters to be changed are indicated in the big red rectangle.
In this example we can see that the libraries ING, ARQ, AGR y VET are configured.

In the first list of parameters (green rectangle) the names of the libraries are defined. Here you can enter the IP addresses (like 192.168.0.123) or IP masks of the libraries.

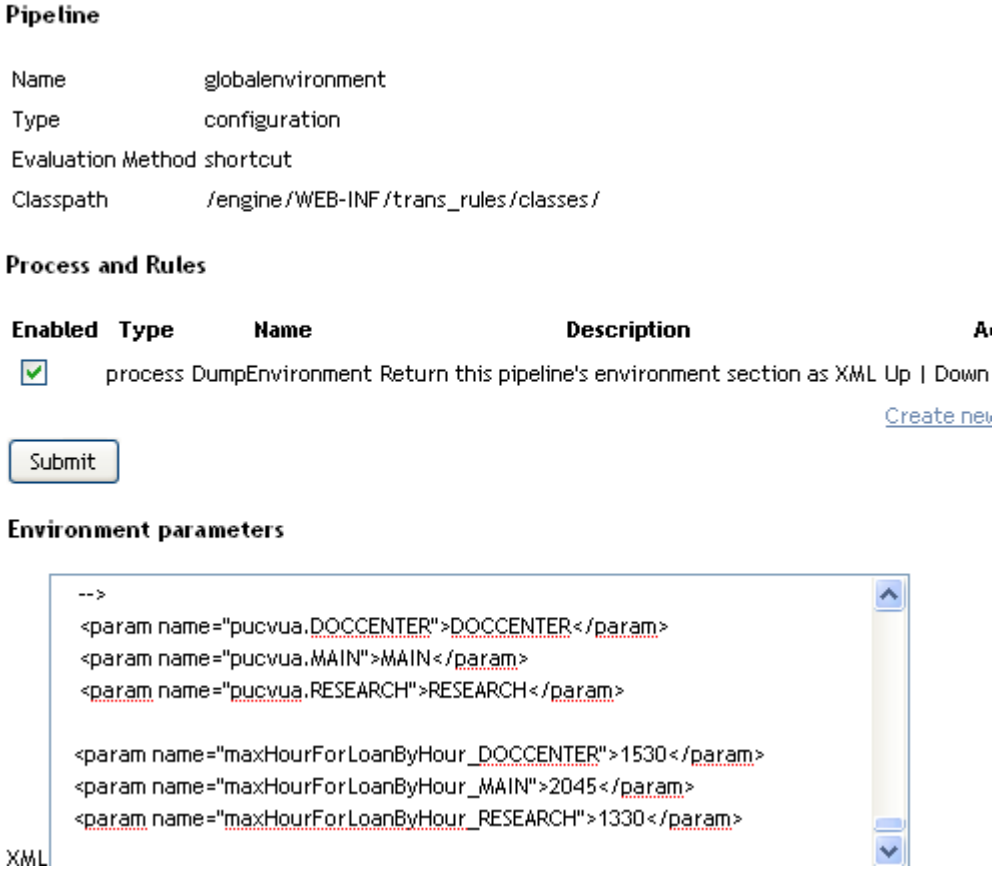
The second group of parameters (yellow rectangle) establishes the operating hours of each library defined in the previous parameter.

2. Changing global parameters of the library in the globalenvironment pipeline.

By editing the parameters defined in the globalenvironment pipeline (red rectangle in the next image) we can change several parameters affecting EmpWeb globally.



Click on Edit and the following screen appears with two groups of parameters in XML language.



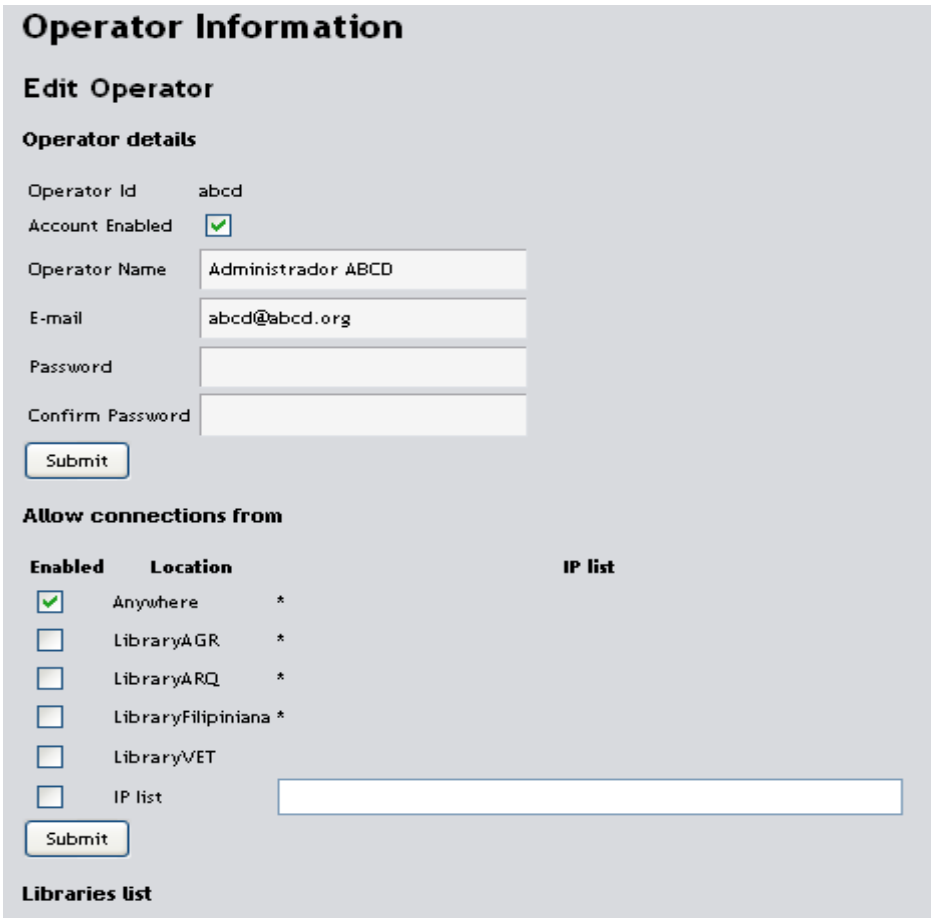

```
<param name="pucvua.DOCCENTER">DOCCENTER</param>
<param name="pucvua.MAIN">MAIN</param>
<param name="pucvua.RESEARCH">RESEARCH</param>

<param name="maxHourForLoanByHour_DOCCENTER">1530</param>
<param name="maxHourForLoanByHour_MAIN">2045</param>
<param name="maxHourForLoanByHour_RESEARCH">1330</param>
```


The first group defines the name of the library as it will appear on the web interface. The second group defines the lending hours for each library in the system.

3.Assign user rights to the system.

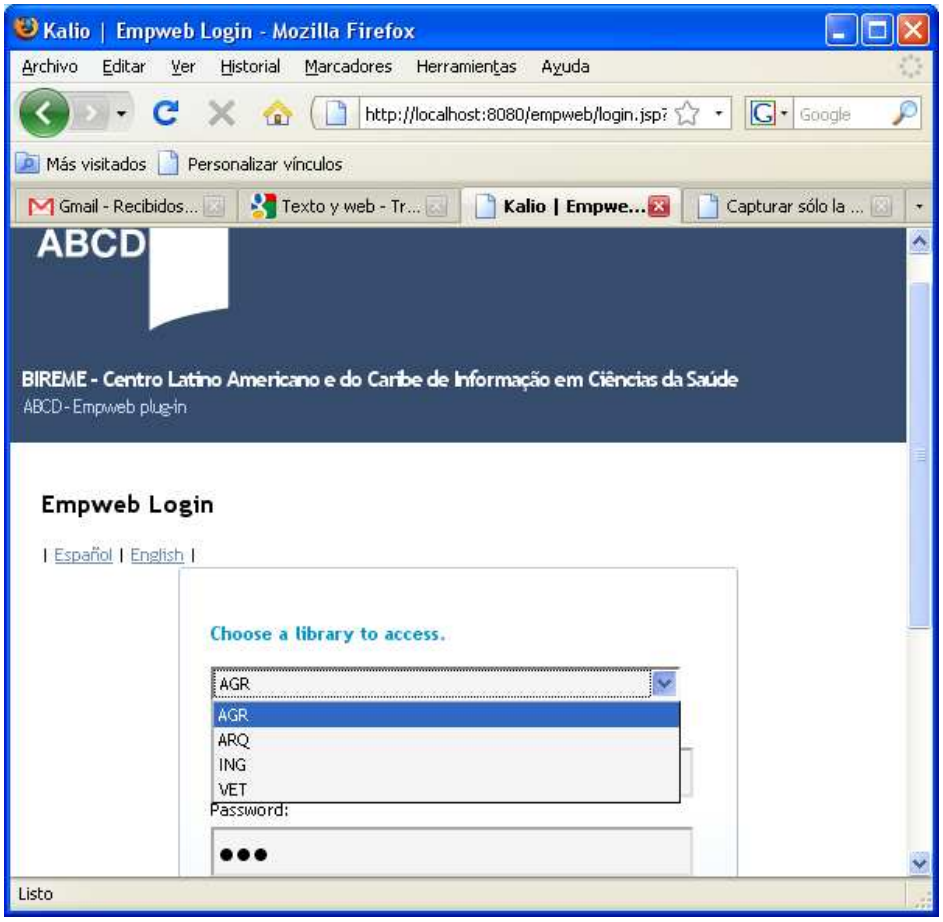
After defining the libraries which populate the system in EmpWeb, you must establish the necessary permissions for the users to access EmpWeb. The permissions will have effect as of the next session.
Go to the EmpWeb Administration menu - > Operators. Select the "super user" with access to all applications – in this case the user **abcd** who has full administrator privileges.
From this access you can create new users and assign the corresponding rights. It is also possible to edit, modify or delete existing users, as shown in the next image.

By editing the super user **abcd**, you gain access to the library definitions and can assign the new access permits to each.

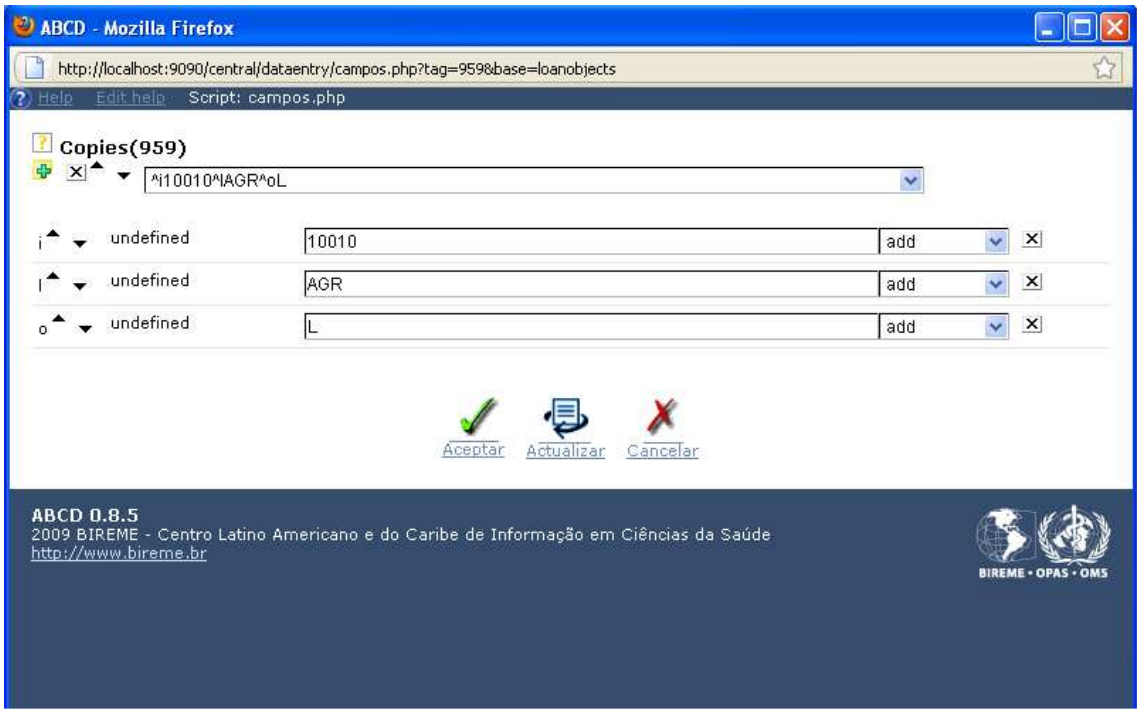


Warning: Remember to save the changes clicking on the Submit button for each group, or the new settings will not have effect.

Once you have assigned the new access permissions to the libraries, you must logout from the system (without closing EmpWeb).
Login again, and now you will see the list of available libraries assigned to the current user/password.



Important: Remember that in order to make loan transactions (loans/returns/reservations) in the new libraries, you must also define the **loanobjects** properties for each library.



In the example above, you will see the loanobjects data as defined in the subfields ^i, ^l and ^d of field Copies(959) in the Loanobjects database, for the object 10010.

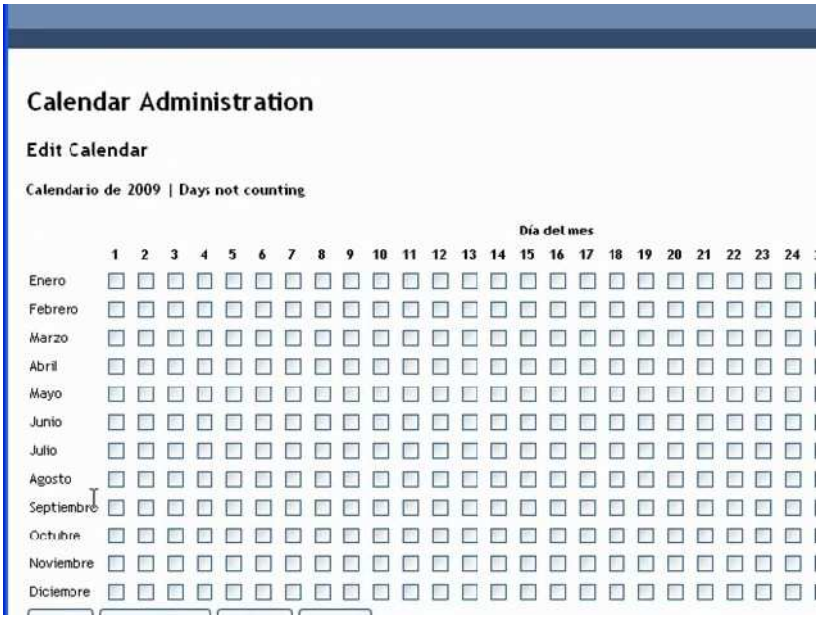
Administration of calendars in EmpWeb.

The administration of calendars for the loan transactions and returns in EmpWeb is global for the whole system and all libraries, so we can define all the calendars for each year. For this we need to enter into the option Administration / Calendars.

When editing a calendar, a frame is presented in which the rows correspond to months and the columns to the days.

Ticking the boxes for each day of the month we can define the non-working days to be taken into account by the system.

By default in this distribution of EmpWeb 15pm is configured to be the limit for returns at the date due; if preferred this time-table can be adjusted from the same menu Administration, Pipelines.



Returns processed from 15p.m. will be considered as late and subject to sanctions (fines or suspension) according to the user profile.

In the case that a non-working day had to be inserted in the calendar, all late returns on that date will be automatically transferred to the next working day.

Creation of new operators.

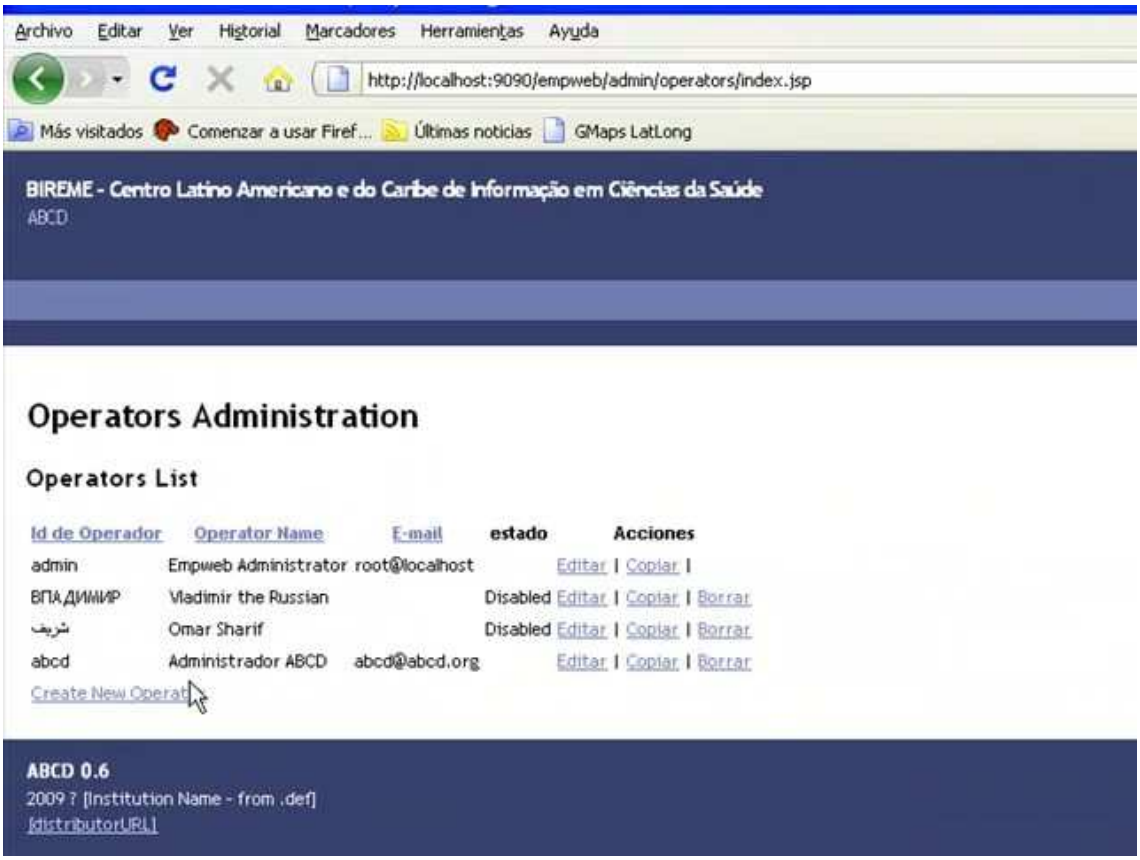
For this we enter EmpWeb with the address **localhost:8080/empweb/;** remember to include the ending slash as this is necessary to enter.

It is crucial to analyze correclty the operations assigned to each EmpWeb operator in order to assign the necessary permissions which give access to the different databases and operations.

After entering with the same login data as used for ABCD (abcd/adm) a second window is presented for selection of the library within which will be worked.

Warning: to create or modify operators one has to take into account the necessary permissions for such operations; in this case the user abcd/adm has the adequate permissions.

From the function administration, we can access the menu 'operators' which lists the operators already defined and offers the functions for editing/copying/deleting existing users on top of the creation of new operators. Such a list is shown in the picture below.



Creatin a new operator

When creating a new operator, the first thing requested is the operator's ID to be used for login into EmpWeb; subsequently a window will be presented with several sections.

These sections are :

1- information on the operator;

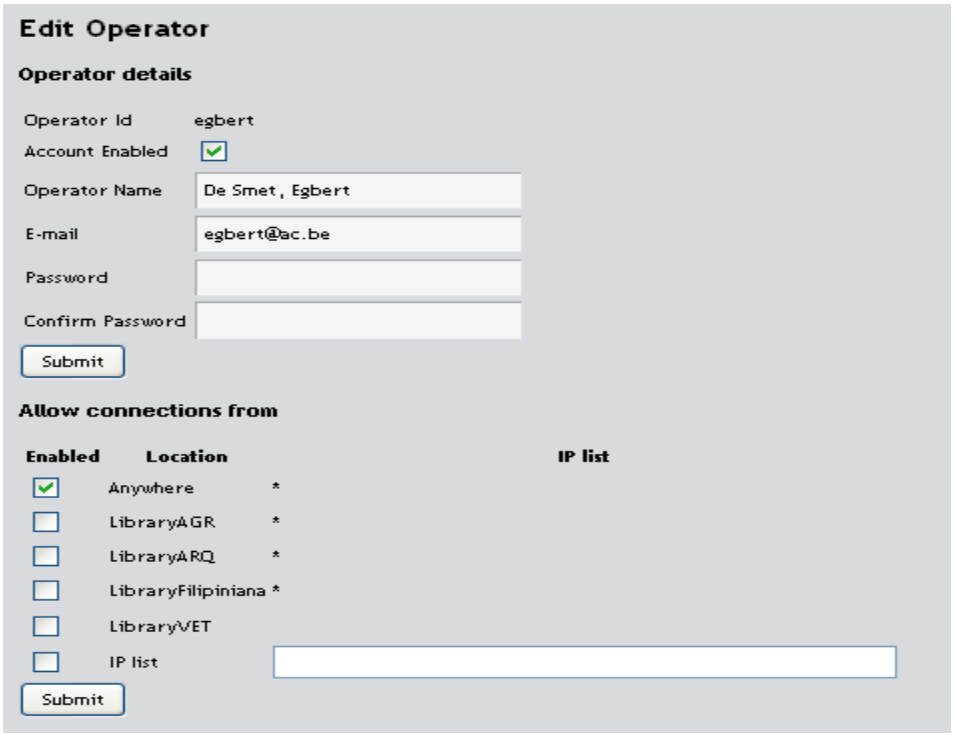
2- IP from where access will be granted;

3- lilbraries which can be accessed and their operating hours,

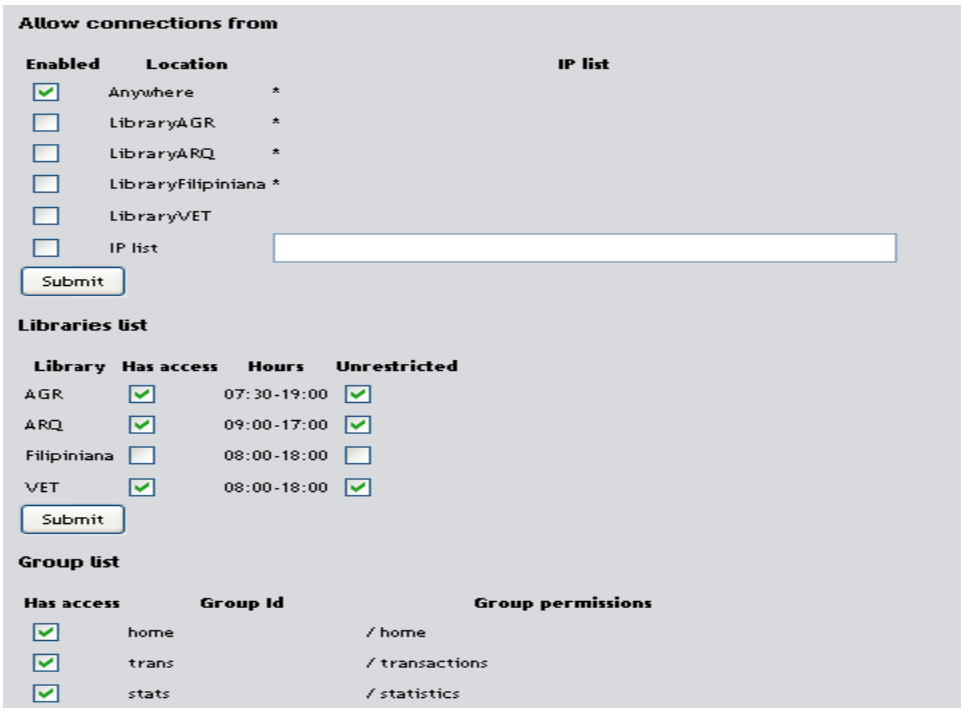
4- links to operations and permissions;

5- and finally the properties or permissions to search in the user and loanobjects databases.

Each section has a button SUBMIT to be used each time permissions in the section have been defined.



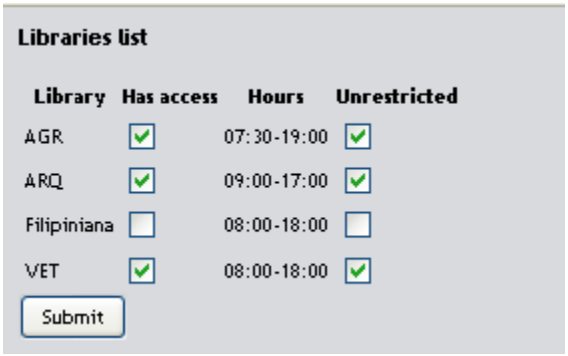
After the creatino of an operator has been confirmed, we still need to enter the details of the operator account, the IP's authorized to enter from and the assigned permissions.



Defining the IP's from which the operator can access, adds additional security to the one based on the login data (login/password) since the user/operator will only be allowed to enter from this specific library or IP. Also it is possible to enter an IP from which the operator can access in the List of IP Addresses or to enter more addresses separated by a semi-colon (;).

The access 'Anywhere' means allow access from any computer on the internet.

In the following section we need to define on which library an operatos has rights to operate and whether operations are confined by the operating hours table or not.

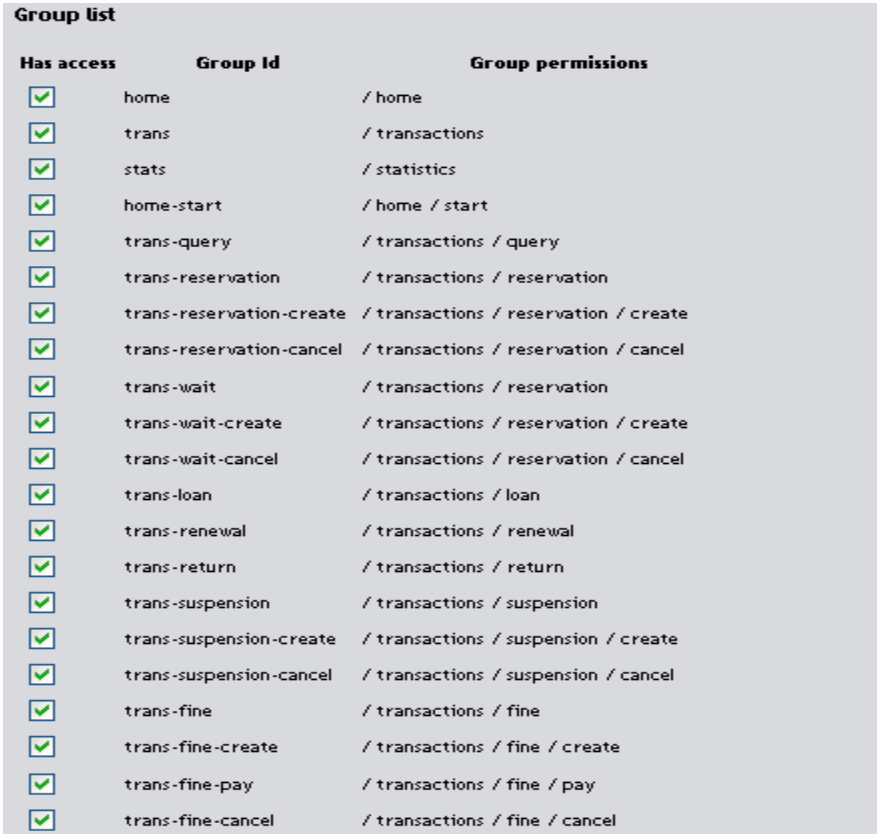


Membership of the operator groups and permissions assignment

In the next section we have to indicate which are the actions for which the operator has permission. The action 'enter Home' always needs to be permitted.

In the following picture the assigned permissions for the operator are shown. Transactions, Return Home; within transactions one can only enter the reservations menu and from there new reservations can be created and returns can be processed.

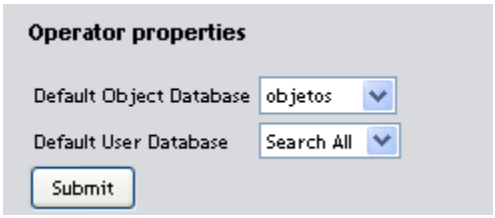
The window with permissions should be shown as illutrated below; don't forget to SUBMIT for the selected options to be saved.



Group list

Has access	Group Id	Group permissions
☑	home	/ home
☑	trans	/ transactions
☑	stats	/ statistics
☑	home-start	/ home / start
☑	trans-query	/ transactions / query
☑	trans-reservation	/ transactions / reservation
☑	trans-reservation-create	/ transactions / reservation / create
☑	trans-reservation-cancel	/ transactions / reservation / cancel
☑	trans-wait	/ transactions / reservation
☑	trans-wait-create	/ transactions / reservation / create
☑	trans-wait-cancel	/ transactions / reservation / cancel
☑	trans-loan	/ transactions / loan
☑	trans-renewal	/ transactions / renewal
☑	trans-return	/ transactions / return
☑	trans-suspension	/ transactions / suspension
☑	trans-suspension-create	/ transactions / suspension / create
☑	trans-suspension-cancel	/ transactions / suspension / cancel
☑	trans-fine	/ transactions / fine
☑	trans-fine-create	/ transactions / fine / create
☑	trans-fine-pay	/ transactions / fine / pay
☑	trans-fine-cancel	/ transactions / fine / cancel

In the last section of the permissions window (see next picture) the database(s) of loanobjects and users which can be administered are to be indicated. At the time of realizing a transaction access to one or all databases will be needed.



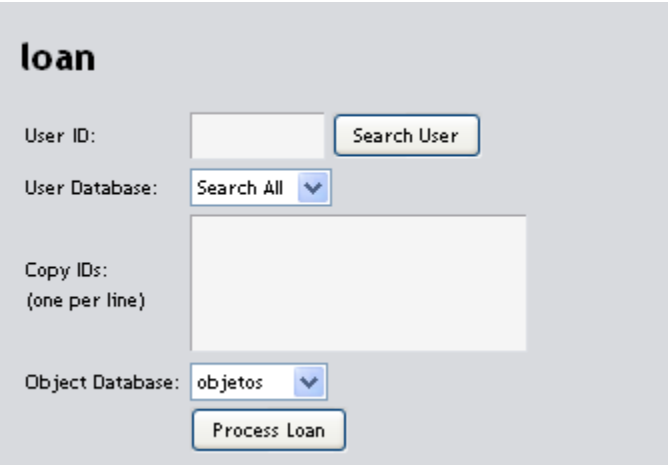
Finally, when an operator tries to enter a transaction for which the necessary authorisation has not been given, EmpWeb will show an error message informing that the operator is not properly authorized.

Linking the databases (ABCD-EmpWeb).

In the following part we will consider the following aspects :

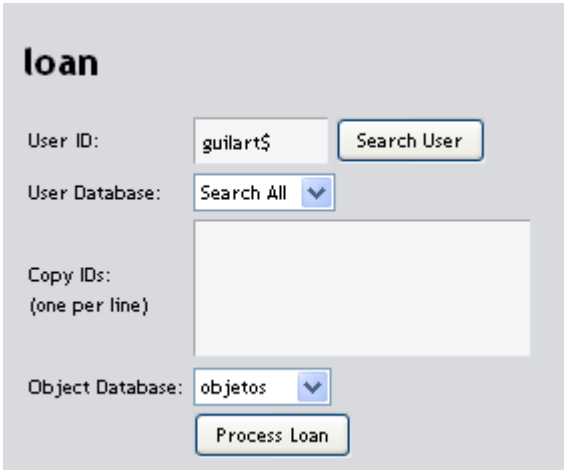
- linking users of ABCD with EmpWeb;
- create new users to be automatically included in EmpWeb;
- and how to consult the potential loanobjects (objects included in the loans system).

The user-databases are shared by all libraries in EmpWeb; when having entered as a system administrator it will be possible to consult the existing users in the databases. For this we enter the Loans option and a window like the following one will be shown :



The system manages transactions based on the user ID, so that if the user ID is known (e.g. By scanning the barcode on the ID) we enter it into the dedicated field and there is no need to perform a search. Such could be done by typing or reading by barcode reader – or any other similar device.

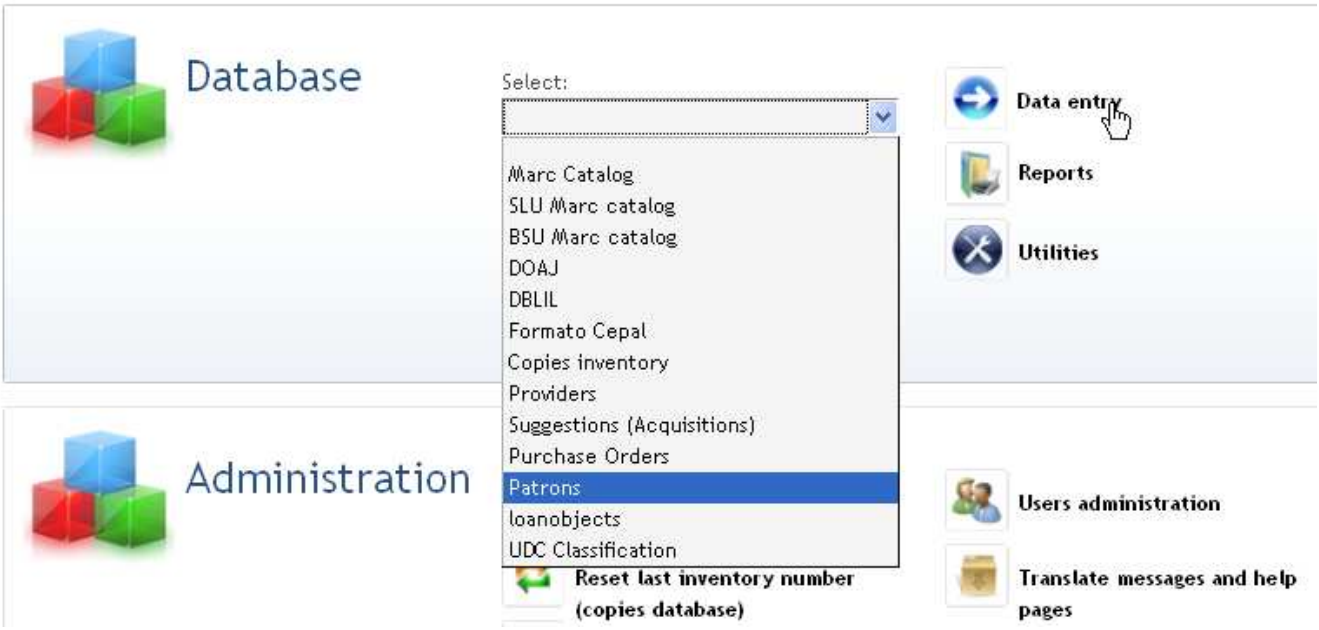
When the user ID is not known, a search is to be done using the ISIS-search technology; for this we enter the first characters followed by the ISIS-truncation ($)m, e.g. 'guilart$', and pressing the button 'search' will locate the user(s) who fit the search expression.



Any record fitting the search expression will be displayed; the correct entry will be selected by clicking on the user ID field and this then will be passed to the loans form for further processing.

Create new user.

For creation of a new user from ABCD we need to enter the Database Administration module (ABCD Central) and select the users database (a demo database is included with the EmpWeb distribution).



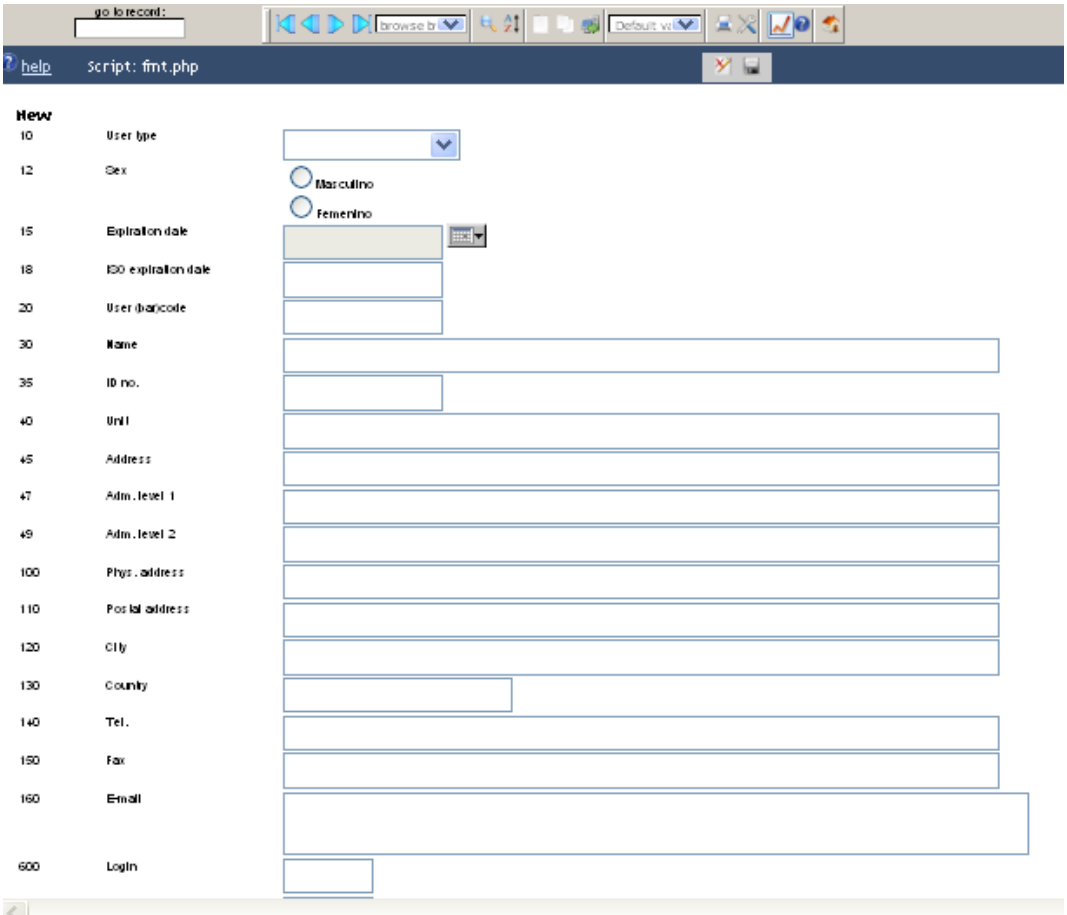
The option Data Entry allows to enter a new user in the database with the following fields :

- User type (based on the list of user types defined in Usuarios.TAB in the folder DEF of the user-database)
- sex male/female,
- expiry date (refers to the date until which the user will be considered active);
- expiry date in ISO date format (automatically created by the system)
- user code;
- Names;
- ID-card;
- Affiliation
- campus/brach
- Adm. Level 1
- Adm. Level 2
- Physical Address
- Mail Address
- City
- Country

- Telephone
- Fax
- E-mail
- Login to access Mysite
- Password to access Mysite
- Picture 4x4

In this distribution of EmpWeb the user_ID is extracted from field 10 from the user records or a SQL table called user_type which is linked e.g. from MySQL.

It is possible to configure the link to the users database from any field or combination of fields where the data can be extracted.



Editing the user record also allows uploading a picture to be included in the user-record.

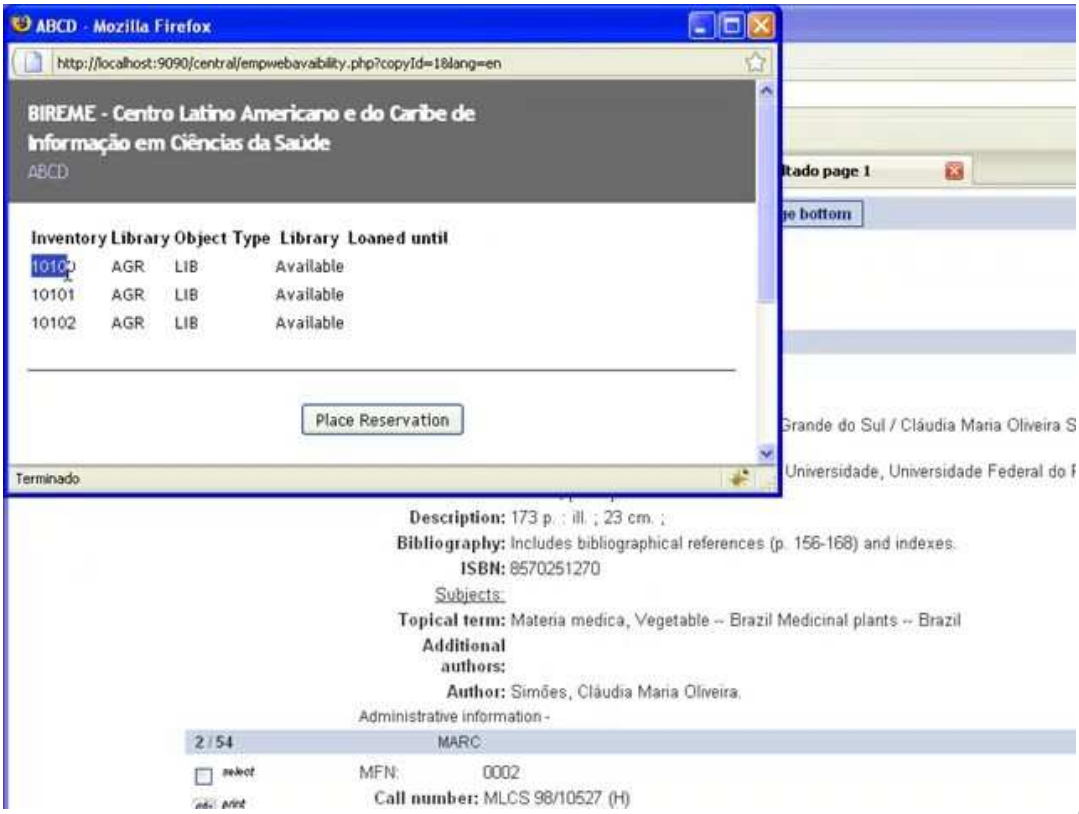
When we access user data from EmpWeb, clicking on the ID field it can be seen that the complete profile includes the picture plus the active loans and other information.

When we proceed with saving the record, by clicking on the 'diskette' icon at the bottom of the form, the user will immediately be activated and can loan materials from the library to which he is linked.

Loanobjects and the loanobjects database

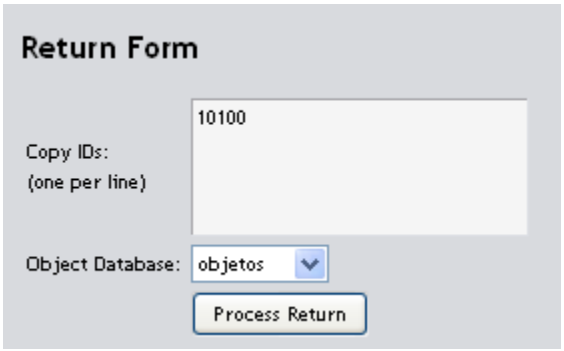
The loanobjects in this EmpWeb distribution are linked to the MARC bibliographic database and when we view the records we can know whether there are copies available by clicknig the button which is shown at the left of each record.

This action of consulting item availability, will allow making reservations; at this stage the user will have to enter his/her login data for the reservation registration.



This database LOANOBJECTS has a field 'Control Number' which links the record with the bibliographic database (MARC); also there is a field to identify the name of the bibliographic database (as there can be many in one Loans system) and finally there is a repeatable field in which the data of each copy are entered into the respective subfields for ID, library, shelf, part and volume no. and object type (subfields i, l, b, v and t).

To make a return transaction is simple : we enter the Transactions module and put the ID of the copy to return; when clicking the button 'process return' soe information about the success (or not) of the return transaction will be given and this will immediately be reflected in the item status of the library and the information of the MySite fuction.



Definition of profiles by user type and object type.

To define the treatment which EmpWeb has to apply to each user type and loanobjects type, profiles are used.

From EmpWeb we access Administration – policies; a window is shown, similar to the one shown below and the active policy is shown. We always will find one active policy which is the one actually being used in EmpWeb, e.g. the default one coming with the installation package.

A policy involves a set of profiles and each profile is a combination of user type with object type.

When editing the current policy, we will see that in reality it contains one profile only which is applied to each object type and each user type for the time being.

When we edit a profile, by clicking on the link 'Edit objects category' (as seen in the previous illustration), we can see all the parameters applied in each transaction.

Components of a profile.

A profile is based on four columns : the name of the parameter, the value of the paramter, the description and the transactions affected by this parameter.

Profile Information

Edit Profile

Profile Details

User Class *
Object Category *
Date 3/16/09 4:29 PM
Policy Name default (20090127_0A6C20)

Limits

Name	Value	Description	Used in transaction
createFineIfLate	false	(Boolean) Enables creation of fine when an object is returned late.	return
createFineIfReservationConfirmedIsCancelled	true	(Boolean) Create a fine if a confirmed reserve is cancelled.	
createFineIfReservationExpired	false	(Boolean) Create a fine if the user has an expired reservation	
createSuspensionIfLate	true	(Boolean) Enables creation of a suspension when an object is returned late.	return
createSuspensionIfReservationExpired	false	(Boolean) Create a suspension if the user has an expired reservation	cancelwait
expirationDays	3	Number of days to wait for a user to pick up a reservation (before expiring)	return | wait
fineAmountIfConfirmedReserveIsCancelled	5	Fine amount for a cancelled confirmed reservation.	
fineAmountIfExpiredNormal	1	Fine amount (multiplier) for an expired reservation when the object has no reservations	
fineAmountIfExpiredReserved	2	Fine amount (multiplier) for an expired reservation when other users have reservations for this object	
fineAmountNormal	20	Fine amount (multiplier) when an object is returned late but has no reservations or waits.	return
fineAmountReserved	30	Fine amount (multiplier) when an object is returned late and it has reservations or waits by other users.	return

Illustration_005. Editing the components of a profile

In the profile currently being edited, applied to all user types and object types, we can observer the following based on the four columns :

- no fine created when a return is late (False)
- fine created when a user cancels a confirmed reservation (True)
- no fine created when a user has a reservation which expires (False)
- suspension created when an item is returned late (True)
- no suspension is created when a user has a reservation which expires (False)
- the maximum number of days to wait for a user to collect a reserved item is put at 3
- fine amount for a confirmed reservation not taken (5)
- (multiplicator of) the fine when a reservation expires and the item has no other reservations(1)
- fine amount when a reservation expires and other users have reserved the same item (2)

Summarizing this section defines all parameters and limits applied on each of the transactions.

For example, the number of days for a loan for any object and user type is put at 5.

Creation of a new profile

When we need to define a new profile of a policy, we need to enter Administration – Policies, Edit policy (in this case Active Default, Edit, Create New Profile) and in there a new profile can be created.

E.g. when we want to create a new profile for the user type 'coordinators', we must identify the user category as 'coordinators' and the object type (e.g. CDROM).

In the window we must define each of the parameters and limits until all values are defined with TRUE or FALSE or a number.

E.g. :

- no fine created when an item is returned late or no fines used at all (False)
- suspension created when an object is returned late (True)
- no suspenion is created when a user has a reservation expired (False)
- number of days during which the user can collect the reserved item is put at 3.
- Fine amount (multiplicator) when an object is returned late without other reservations : 0.
- Fine amount (multiplicator) when an object is returned late and other reservations are noted : 0
- The number of days the loan lasts for this object (which can be absolute days or library days, as defined in the transaction). In this case put at 15.
- Total maximum fine a user can accumulate without losing the possibility to loan. Here it is 0 since no fine system is used.

- Maximum number of fines a user can have without losing the right ot loan. Left at 0 here.
- Maximum number of fines a user can have without losing the right to make reservations : 0
- Maximum number of loans allowe when user has fine : 0
- If the user has late returns how many new loans will be allowed ? Leave at 0
- Max. Number of loans allowed when user has suspension. Left at 0 implies that when suspended a user cannot have loans anymore.
- Maximum number of loans of the same object category, put at 2, meaning only two CDROMS can be taken at the same time
- Maximum no. of loans for the same volume/part item, put here at 1
- Maximum no. of loans this user can have in total, here put at 7
- Maximum no. of renewals for this user, put at 3, meaning that 3 renewals are allowed, but from MySite only 2 will be allowed.
- Maximum number of reservations if the user has fines, so in this case 0 since no fine system is used here.
- Maximum number of reservations if the user has late returns : no reservations are allowed with delayed returns
- Maximum no. of reservations if the user has suspension running, 0 means that the a suspended user cannot take any reservation
- Maximum no. of reservations for objects of this category. Same as the value for loans and same for reservations of the same type of objects
- Maximum no. of reservations for the same record, same case of 1 for laons would be the same case applied for volume/part. [???]
- Maximum no. of reservations allowed in total, in this case put at 5
- Suspension days when a reservation expires, in this case we do not apply suspension days, so 0 whether there are reservations or not
- When an item is returned late without having reservations, 4 days of suspension will be applied.
- When an object with reservations is returned late, 5 days of suspension will be applied.

Profile Information

Edit Profile

Profile Details

User Class *
Object Category *
Date 3/16/09 4:29 PM
Policy Name default (20090127_0A6C20)

Limits

Name	Value	Description	Used in transaction
createFineIfLate	false	(Boolean) Enables creation of fine when an object is returned late.	return
createFineIfReservationConfirmedIsCancelled	true	(Boolean) Create a fine if a confirmed reserve is cancelled.	
createFineIfReservationExpired	false	(Boolean) Create a fine if the user has an expired reservation	
createSuspensionIfLate	true	(Boolean) Enables creation of a suspension when an object is returned late.	return
createSuspensionIfReservationExpired	false	(Boolean) Create a suspension if the user has an expired reservation	cancelwait
expirationDays	3	Number of days to wait for a user to pick up a reservation (before expiring)	return | wait
fineAmountIfConfirmedReserveIsCancelled	5	Fine amount for a cancelled confirmed reservation.	
fineAmountIfExpiredNormal	1	Fine amount (multiplier) for an expired reservation when the object has no reservations	
fineAmountIfExpiredReserved	2	Fine amount (multiplier) for an expired reservation when other users have reservations for this object	
fineAmountNormal	20	Fine amount (multiplier) when an object is returned late but has no reservations or waits.	return
fineAmountReserved	30	Fine amount (multiplier) when an object is returned late and it has reservations or waits by other users.	return

Once all parameters have been filled in, click on the SUBMIT button to create a new profile with its own behaviour.

Always remember that we can see the profiles matrix as a list in which it can be verified which policy the created profile applies to each user and object category.

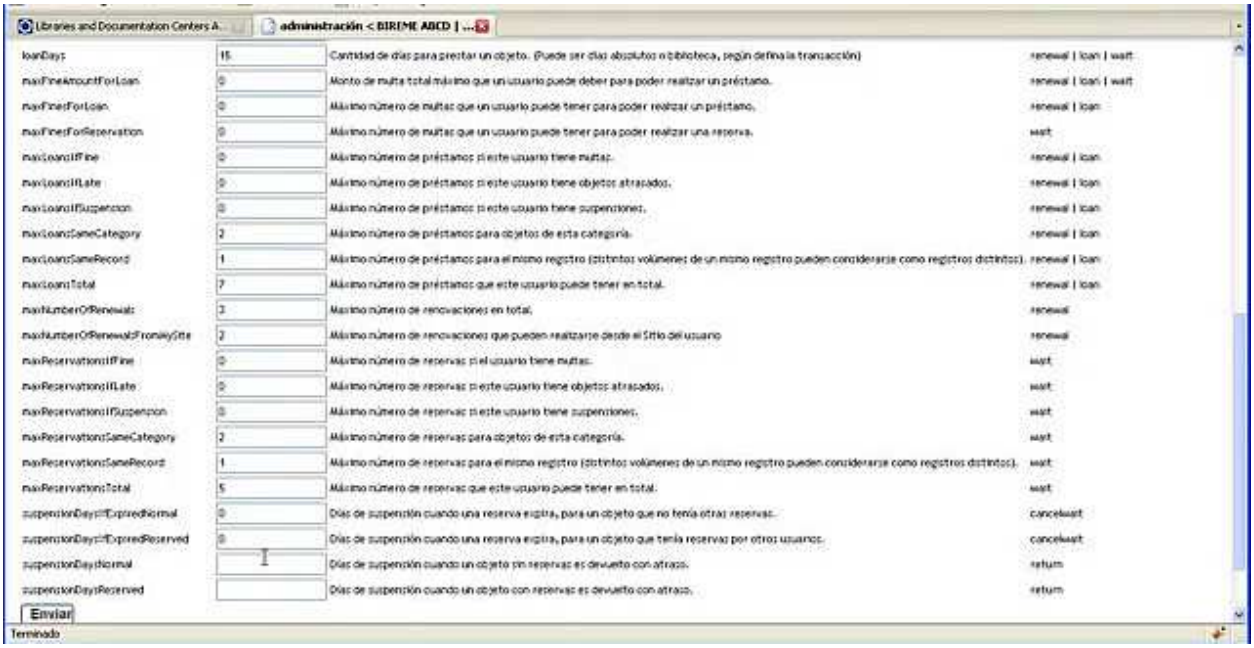
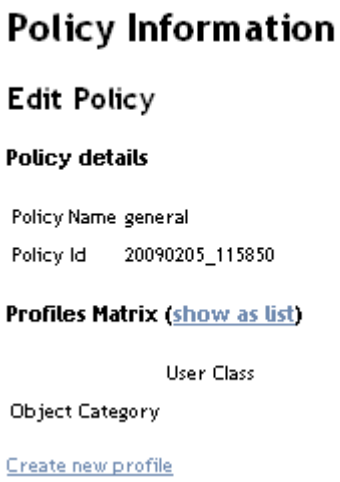


Illustration_009. Parameters of the profile Coordinators

This new profile is applied for the object type CDROM for the user type coordinators. We can see in the illustration_009b that a profiles is created for the Coordinators cateogry and the object categories CDR.



Application of a profile.

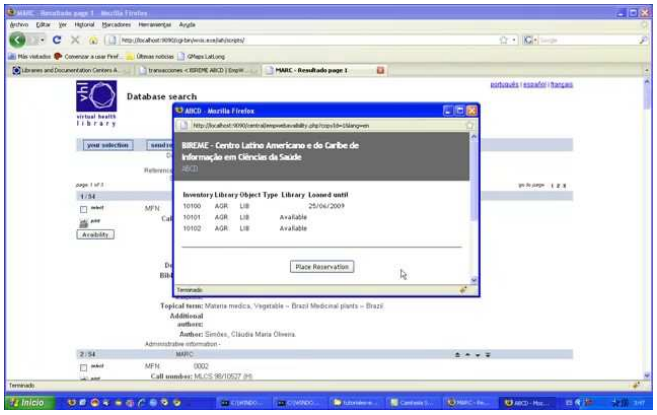
Once a profile has been edited clicking on SUBMIT will apply the profile on the new transactions. Remember that there should be loanobjects available for loaning and also a user category corresponding to the profile.

Reservations with EmpWeb.

Reservations can be made fron 2 access points : MySite or as another transaction from the operator's EmpWeb interface.

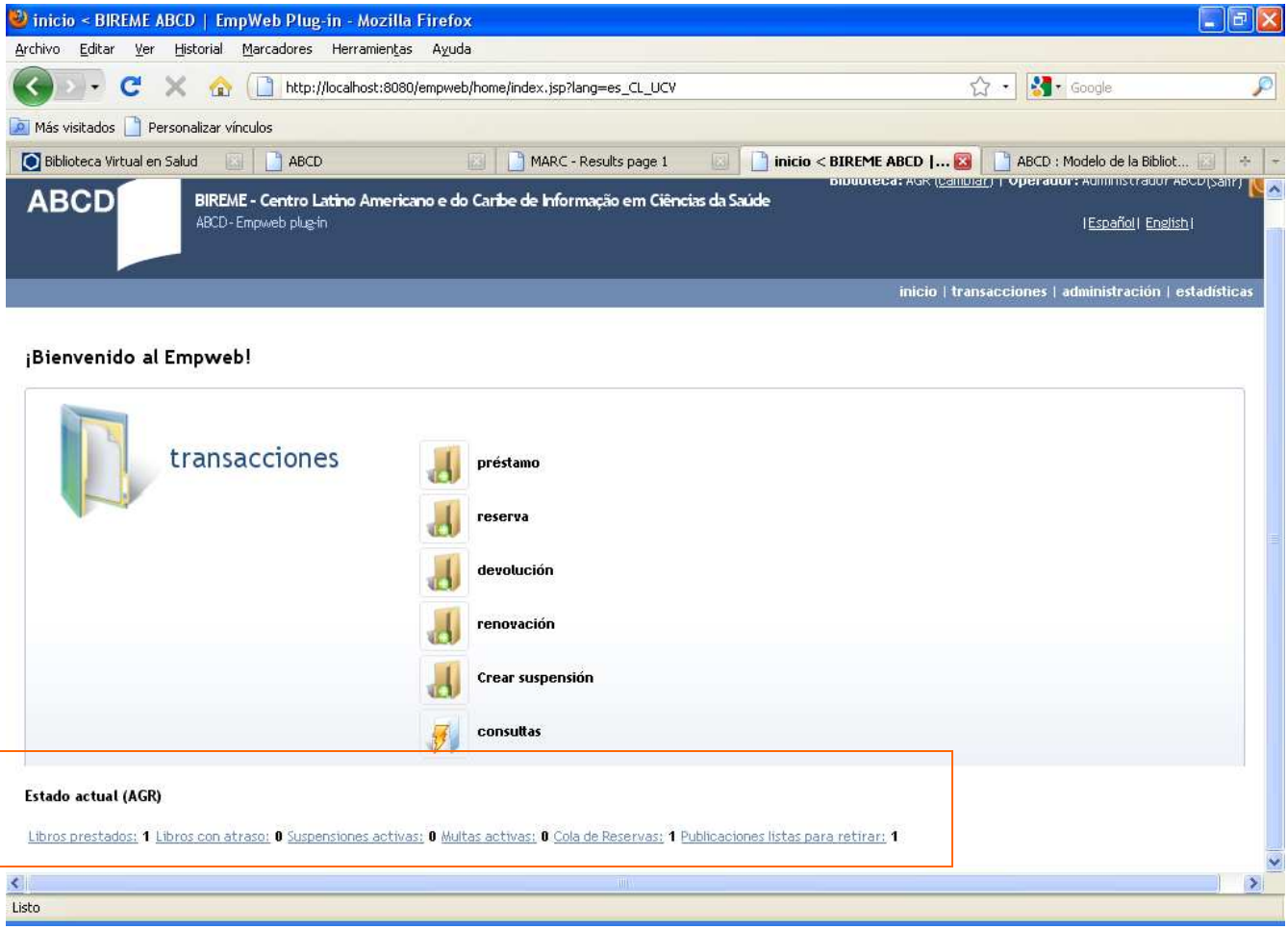
Consulting the availability of items for a selected loanobject, one can reserve it from MySite; the system will allow to make the reservation when the requested title does not have copies available and in addition the user complies with the user-profile defined conditions (e.g. not being suspended).

When making a reservation if copies are available, the system does not allow to make this reservation but will indicate that an available copy has to be collected from the library.



When we go to MySite we can see that the status now indicates that the reservation has been done today and which title has been reserved.

Now we can record a return transaction of an item with reservation, this information will be reflected immediately in the waiting list of reservations of the principal EmpWeb window



When we try to realize a loan on this item which has a reservation by another user, EmpWeb informs us that the number of running loans do not suffice for the confirmed reservations in order to allow this transaction. So an reserved item will not be issued on loan.

Automatically in the MySite page of the user the availability of the reserved item will be reflected and also it give information on the number of days it will remain reserved to be collected. This number of days to be available for collection by the user will be taken from the EmpWeb policy.

At the time of creating a loans transaction or canceling a reservation, the information in the main EmpWeb page will be updated with the actual status of the transactions.

Anex I . Pipelines and Groovy.

1. The concept of Pipelines.

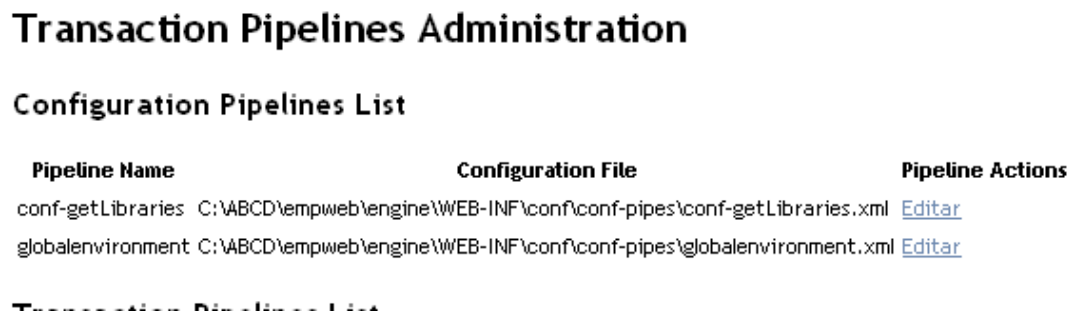
All operations realized by EmpWeb correspond to a pipeline.

We define a **pipeline** as a set of ordered processes, sequential or simultaneous, in which the output of one process is the input of the following one.

Groovy is a non-declarative programming language, with a syntax very similar to JAVA and with lots of programming facilities. The specifications of the Groovy language can be consulted at: http://groovy.codehaus.org/

The pipelines with which EmpWeb works in this distribution package of EmpWeb can be divided in the following groups :

Global pipelines. (Configuration Pipelines List)



These pipelines define values used both in the graphical interface and in other pipelines.

Here we find global values like e.g.

- The time-tables of opening hours of each library,
- the type of material registered for per-hour loans,
- the maximum time allowed for per hour loans,

- etc.

Transaction Pipelines (Transaction Pipelines List)

Transaction Pipelines List

Pipeline Name	Configuration File	Pipeline Actions
cancelwait	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\cancelwait.xml	Editar
cancelfine	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\cancelfine.xml	Editar
suspension	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\suspension.xml	Editar
fine	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\fine.xml	Editar
renewal	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\renewal.xml	Editar
loan	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\loan.xml	Editar
cancelsuspension	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\cancelsuspension.xml	Editar
return	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\return.xml	Editar
wait	C:\ABCD\empweb\engine\WEB-INF\conf\trans-pipes\wait.xml	Editar

These pipelines define the transactions which the main server of EmpWeb manages and of which the pipelines will be called by the corresponding processes of the graphical interface.

Reports pipelines. (Statistics Pipelines List)

Statistics Pipelines List

Pipeline Name	Configuration File	Pipeline Actions
stat-status-loans	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-loans.xml	Editar
stat-trans-by-ids	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-trans-by-ids.xml	Editar
stat-status-fines	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-fines.xml	Editar
stat-status-lateLoans	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-lateLoans.xml	Editar
stat-status-suspensions	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-suspensions.xml	Editar
stat-record-availability	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-record-availability.xml	Editar
stat-hist-user	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-hist-user.xml	Editar
stat-hist-loans	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-hist-loans.xml	Editar
stat-hist-fines	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-hist-fines.xml	Editar
stat-status-waits-assigned	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-waits-assigned.xml	Editar
stat-status-waits	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-waits.xml	Editar
stat-status-counts	C:\ABCD\empweb\engine\WEB-INF\conf\stat-pipes\stat-status-counts.xml	Editar

These pipelines are used at the moment of presenting a report. Such report will give an account of the requested information where this can be presented in the formats defined by the processes of the pipeline.

2. Detall of a pipeline.

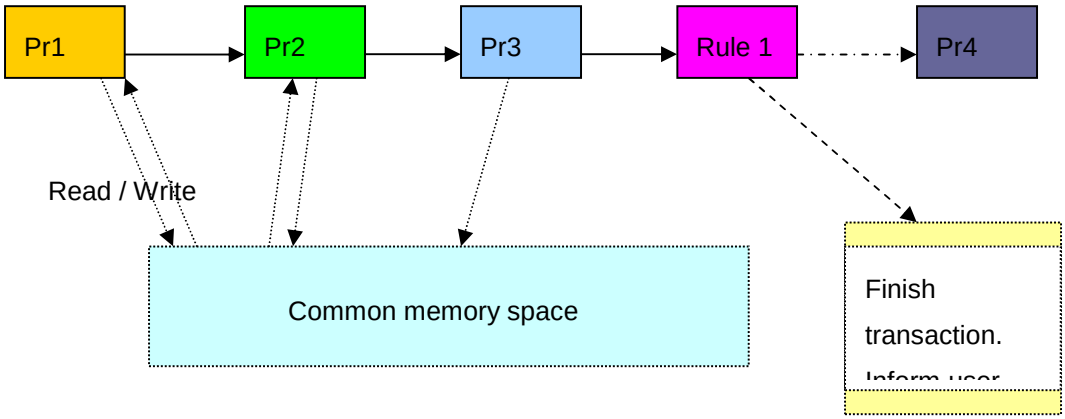
A pipeline is a set of processes, which are executed in a consecutive or simultaneous way, ordered and sharing data. Let us consider some of these :

Enabled	Type	Nombre	Descripción	Acciones
☑	rule	GetUser	Get User DOM from (userId, userDb)	Up | Down | Editar | Borrar
☑	rule	ExtractUserClass	Extract the user class from the user XML and store it in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	GetObject	Get Object DOM (mods) from (copyId/recordId, objectDb)	Up | Down | Editar | Borrar
☑	rule	ExtractObjectCategroy	Extract the object category from the object XML and store it in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	GetProfile	Gets a Profile for the userClass and objectCategory stored in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	AdjustProfileValues	Adjusts some of the Profile's values to the calculated ones.	Up | Down | Editar | Borrar
☑	process	PublishTimestampAdjustments	Publica al TC horas de devolucion, de expiracion de reserva, de inicio de reserva, y excepciones	Up | Down | Editar | Borrar
☑	rule	Lock	Logical lock of UserStatus and ObjectStatus	Up | Down | Editar | Borrar
☑	rule	GetUserStatus		Up | Down | Editar | Borrar
☑	rule	GetObjectStatus		Up | Down | Editar | Borrar
☑	rule	GetExistingWaits	Finds an existing Reservation for the user that matches the object we are lending.	Up | Down | Editar | Borrar
☑	rule(Script)	PucvObjetoEsDeBiblioteca	Verifica si el objeto pertenece a la biblioteca donde se estra realizando la transaccion.	Up | Down | Editar | Borrar
☑	rule(Script)	objectAlreadyLent	Checks whether the object is already lent.	Up | Down | Editar | Borrar
☑	rule(Script)	CheckValidityDateIsNotNull	Check null values in User cards	Up | Down | Editar | Borrar
☑	rule	HasFineOrSuspension	Verifies if the user has fine or suspension.	Up | Down | Editar | Borrar
☑	rule(Script)	LoanIfLate	Can we loan to a late user?	Up | Down | Editar | Borrar
☑	rule(Script)	UserQuantities		Up | Down | Editar | Borrar
☑	rule(Script)	checkAvailability	Check if this loan is not interferring the reservation queue	Up | Down | Editar | Borrar
☑	process(Script)	obtainLibrariesInformation		Up | Down | Editar | Borrar
☑	process	CreateLoan	Creates a Loan object in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	ValidateAvailabilitv	Validates avaibilitv to making the loan	Up | Down | Editar | Borrar

Since based on the pipeline concept the entry of a process is the output of a previous process, exceptions can occur: e.g. This will be the case with the first entry which could be based on data entered by the user. In the case of EmpWeb, an 'space' of shared objects is used from which the processes gather and request their values to shape the transaction as such.

There are two types of processes in EmpWeb, **rules** y **proper processes**.

1. The rules use the "common space for information exchange" to check specific values and the rule can return, in function of these values and the negociated rule, "true" o "false" as a result, indicating with this result if the pipeline should continue or stop.
2. The proper processes interact with the common space of memory to store values with two possible objectives : either to finish the complete transaction or to hand over values to subsequent rules in order to test negotiating rules.



3. Objects accessible during transaction realisation. Common memory space.

It is important to underline that in the common memory space it is possible to get and store information which defines a couple (identification, value). This way the identifiers were previously defined and it is based upon these couples (identification, value) that the rules and processes can interact.

One of the common memory spaces most used by EmpWeb is the one named TransactionContext. In the definition of EmpWeb this space space can gather or store values at the moment of running a transaction.

The concept of TransactionContext corresponds to a hash, which puts a key and developes a value from this key. This value can be linear like a String or an Empweb Object.

The following table summarizes the keys and their meaning for storage in the TransactionContext.

Clave	**Tipo de valor**
LOCAL_ENV	Environment of local parameters for the process. List of values.
GLOBAL_ENV	Environment of global parameters for the process. List of values.
USER_DOM	DOM of the user. Object
OBJECT_DOM	DOM del objeto. Objeto.
USER_ID	ID of the current user. String.
USER_DB	Database of the current user. String.
COPY_ID	Identification of the current copy. String.
VOLUME_ID	Volume of the current copy. String.
RECORD_ID	Identification of the current record. String.
OBJECT_DB	Current database of loanobjects. String.
OPERATOR_ID	Identification of the operator who manages the transaction. String.
DESK_OR_WS	Only for reservations. Whether the transaction comes from EmpWeb or from mySite
OBJECT_LOCATION	Identification of the library of the current copy
OBS	Observations entered by the user in the form. Only applicable in transactions like suspensione or fines.
FINE_AMOUNT	Current fine amount. Real value.
PAID_AMOUNT	Amount paid for a fine as entered by the operator. Real value.
DAYS_SUSPENDED	
USER_CLASS	User type on which the transaction will be run.
OBJECT_CATEGORY	Object type on which the transaction will be run.
USER_STATUS	
USER_CLASS	User type for the current transaction. String.
OBJECT_STATUS	

PROFILE	Contains the object of the current transaction profile class. Defines the user category and object type for the current transaction.

4. Detailed analysis of some basic classes of EmpWeb when running a transaction.

In quite some pipelines, we will find the following processes invoked as initial processes.

It should be noted that these classes have been compiled in JAVA and it is not possible to modify then without recompiling the application.

The comments are taken from the javadoc documentation.

GetUser

Clase: net.kalio.empWeb.engine.rules.GetUser

It gets the userCollection XML from the user database and does some validity checks.

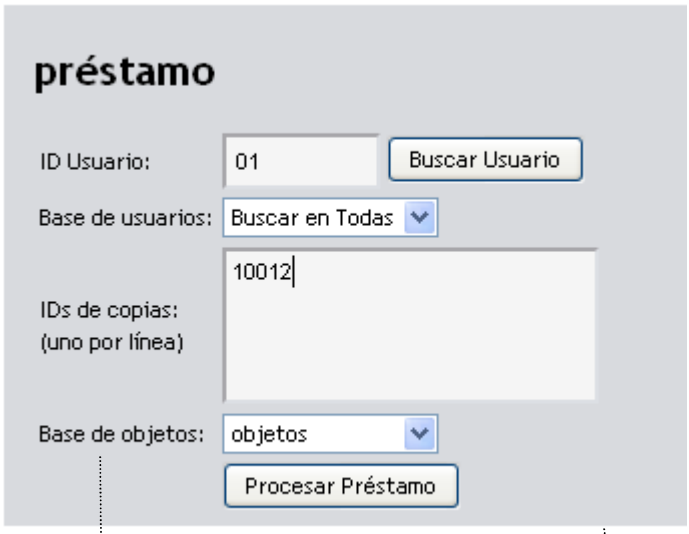
The Process expects to find the userId and userDb in the TransactionContext under TransactionContext.USER_ID ("userId") and TransactionContext.USER_DB ("userDb") keys, respectively.
The userCollection DOM object will be stored in the TransactionContext with setUserDOM.

If the userCollection does not contain at least one user, the Process fails. If it contains more than one, usually only the first one will be used by the following Processes.

The Process accepts a boolean parameter checkValidity (default: true). If the parameter is "true" or missing then the user expirationDate and state (active/inactive) will be checked and the Process may fail with an appropriate ProcessResult message.

(For the meaning of expirationDate and state, read the http://kalio.net/empWeb/schema/users/v1 schema)

Scheme of operations that this class runs:



TransactionContext.USER_ID=01

TransactionContext.USER_DB=*

TransactionContext.COPY_ID=10012

Webservice DB

Parameters

of the process

Calling GetUser

TransactionContext.USER_ID=01

TransactionContext.USER_DB=isis

TransactionContext.COPY_ID=10012

TransactionContext.OBJECT_DB=objetos

TransactionContext.OPERATOR_ID=1

Next process in the pipeline

Description: It gets a MODS modsCollection from the database representing the object MODS .
The database is specified in the TransactionContext's objectDb.
The object may be specified by copyId or recordId, depending on the mode in which this class is used.
The mode is specified with a "mode" parameter in the params of the transaction pipeline XML. The value of "mode" can be "copyId" or "recordId".

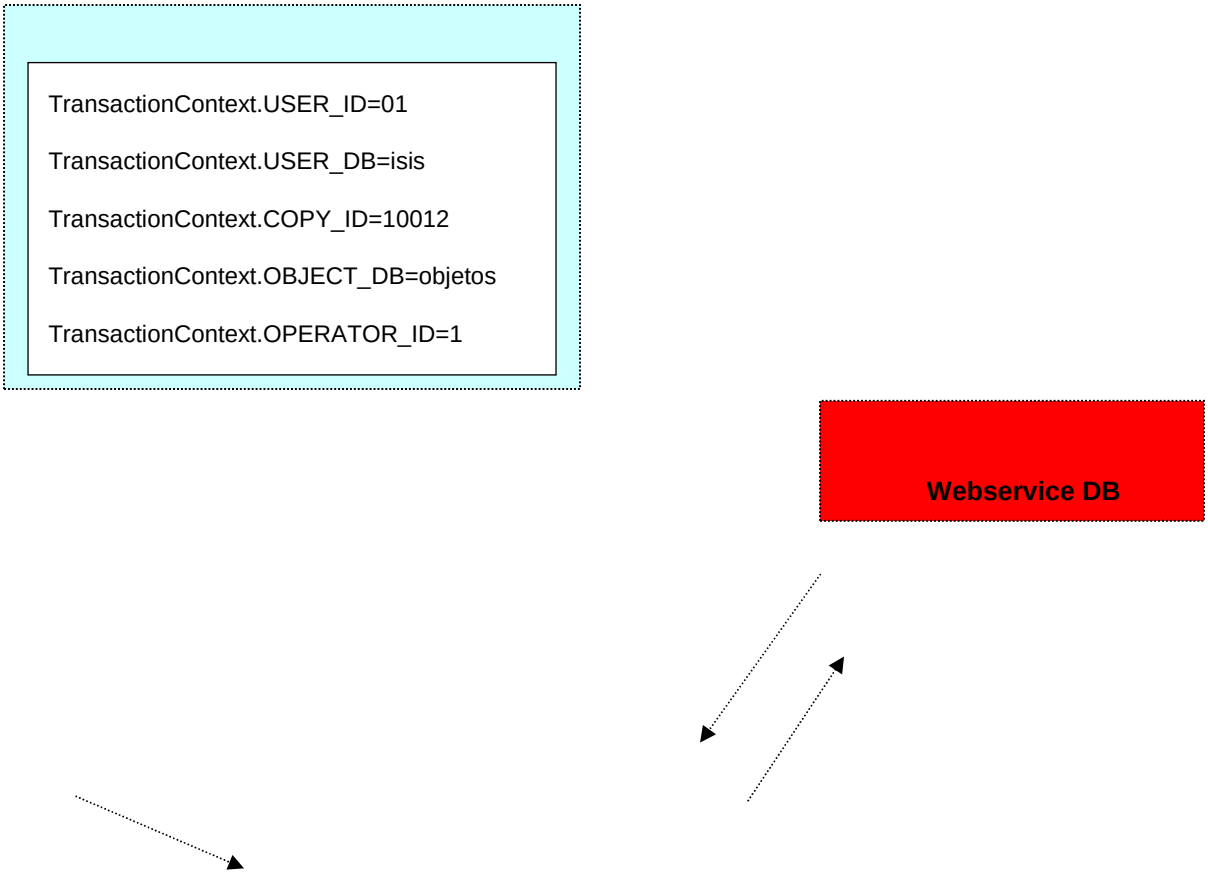
For "copyId" (the default behavior), it performs a searchObjectsById on the objectDB web service. For "recordId" it will perform a searchObjects with a recordId element.

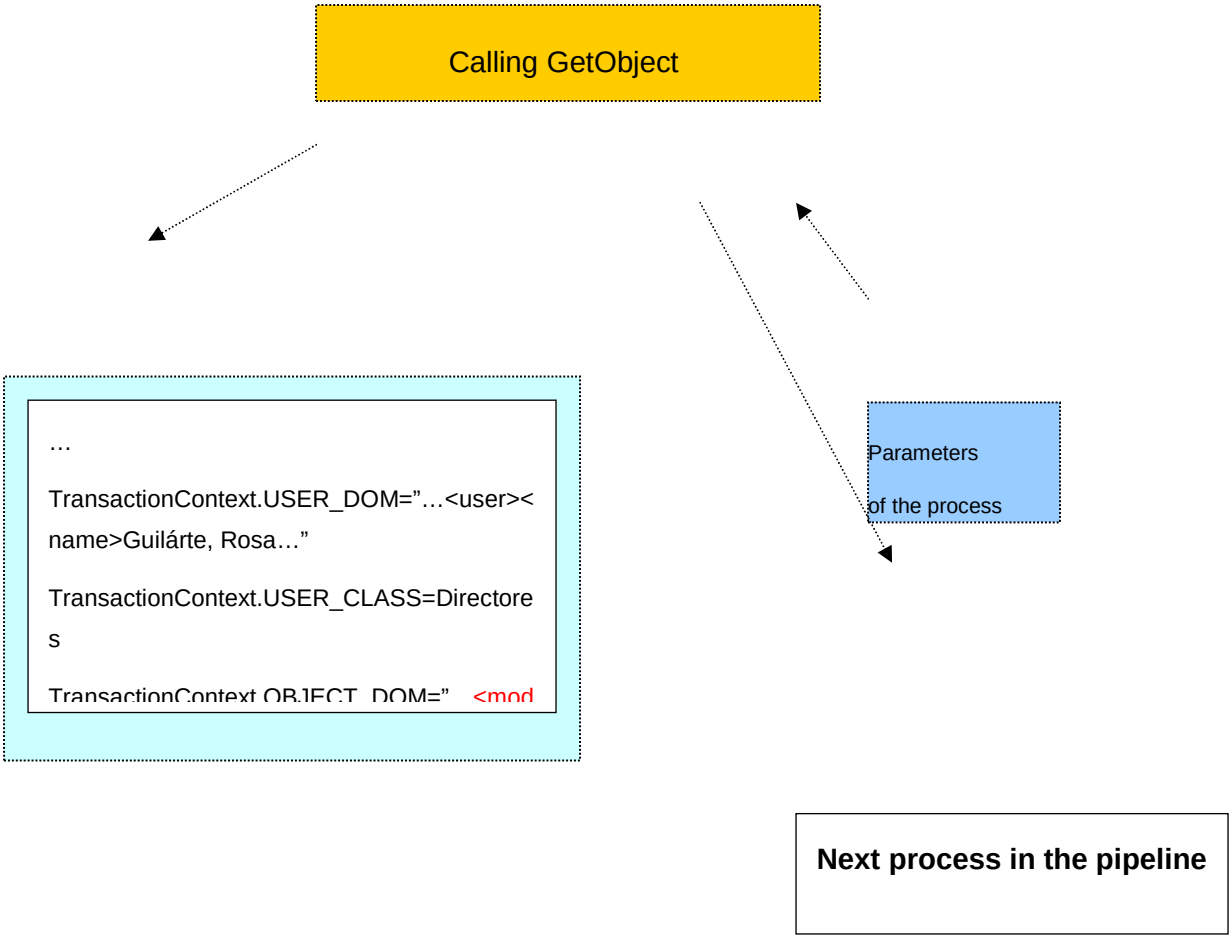
The MODS modsCollection DOM will be stored in the TransactionContext with setObjectDOM.

For convenience, in the case of "copyId" mode, the mods:recordInfo/mods:recordIdentifier will be stored in the TransactionContext under the "recordId" key.

The modsCollection will usually contain just one mods element. It MAY contain more than one but most transactions will use only the first mods. If modsCollection is empty, this Process will fail.

If the mode is "copyId", you can use the boolean extractExtraCopyInfo parameter (default: false) which will extract the copyLocation and volumeId (if exists) and store them under TransactionContext.OBJECT_LOCATION and TransactionContext.VOLUME_ID respectively (if they do not exist already in the TransactionContext).





ExtractObjectCategory

Description: Extracts the objectCategory from the object's EmpWeb holdingsInfo and stores the value in the TransactionContext under the well-known key "objectCategory".

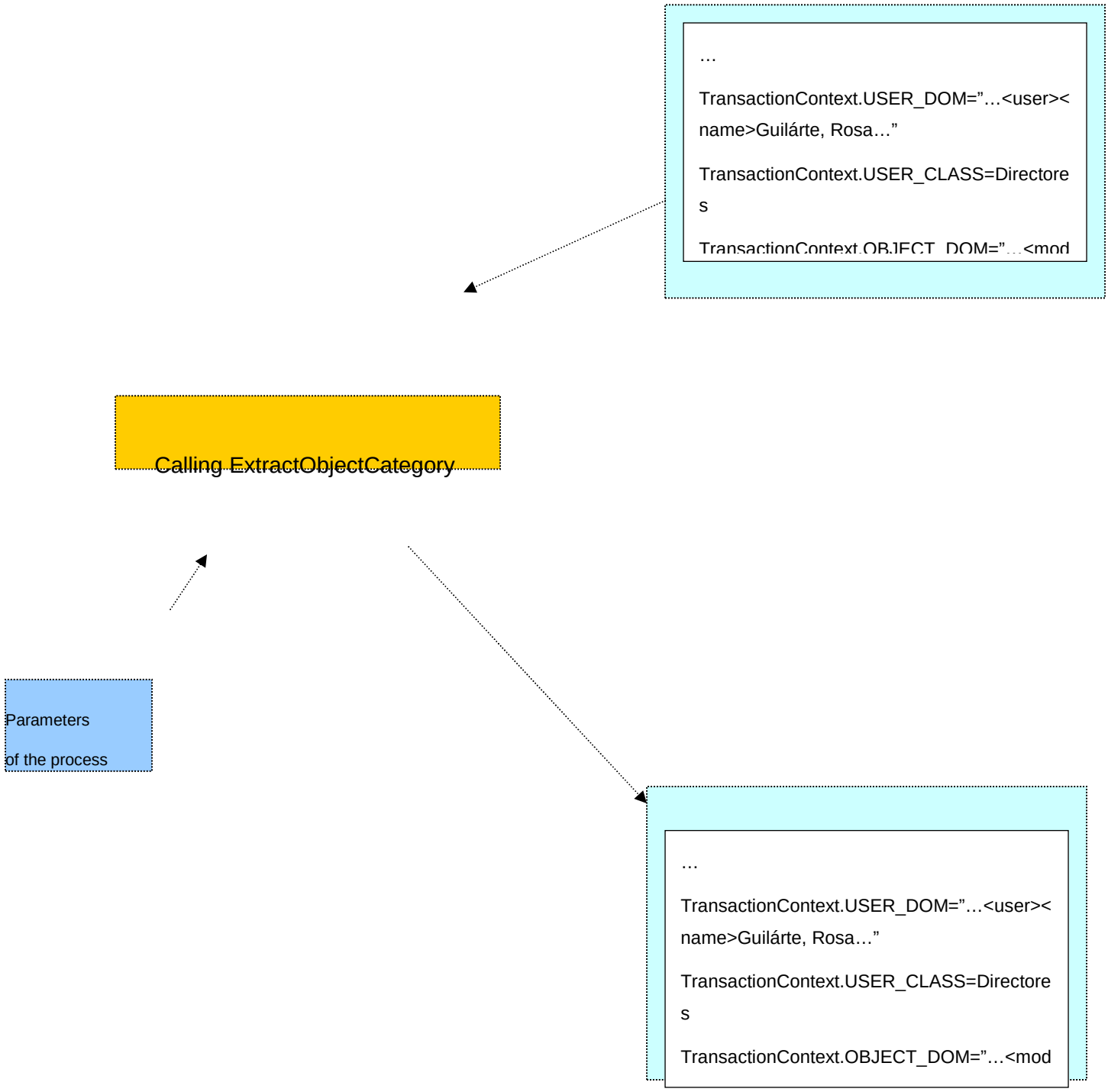
This Process accepts three parameters:

- mode: It can have the value "copyId" or the value "recordId" (defaults to "copyId")
- useDefault: it's a boolean and defaults to false
- dontConsider: it's a list of object categories separated by comma or semicolon. These object categories should be ignored when extracting an object's objectCategory

The algorithm works as follows: First, check whether the "mode" matches what is found in the TransactionContext. If the needed copyId or recordId isn't found, fail with an error message.

Then:

```
if (mode is recordId) then
  if (exists mods:mods for this recordId) then
    objectCategory:= "first objectCategory found in mods:mods that is not in the dontCondiser list"
  else
    Error: no_object_for_recordid
  end if
else // mode copyId by default
  objectCategory:= "the objectCategory of the given copyId"
end if

// if objectCategory still null and useDefault is false
if (objectCategory == null and not useDefault) then
  Error: record_with_no_object_category or copy_with_no_object_category
        according to mode
end if

store the objectCategory in the TransactionContext
```


…
TransactionContext.USER_DOM="…<user><name>Guilárte, Rosa…"
TransactionContext.USER_CLASS=Directores
TransactionContext.OBJECT_DOM="…<mod
Calling ExtractObjectCategory
Parameters
of the process
…
TransactionContext.USER_DOM="…<user><name>Guilárte, Rosa…"
TransactionContext.USER_CLASS=Directores
TransactionContext.OBJECT_DOM="…<mod

As can be seen therefore, the processes run operations which store values in the TransactionContext, based on a particular key and the rules can test some of these values to implement the negotiation rule.

GetProfile

Description: This Process obtains a Profile from the active Policy and stores it in the TransactionContext under the well-known name TransactionContext.PROFILE ("profile").

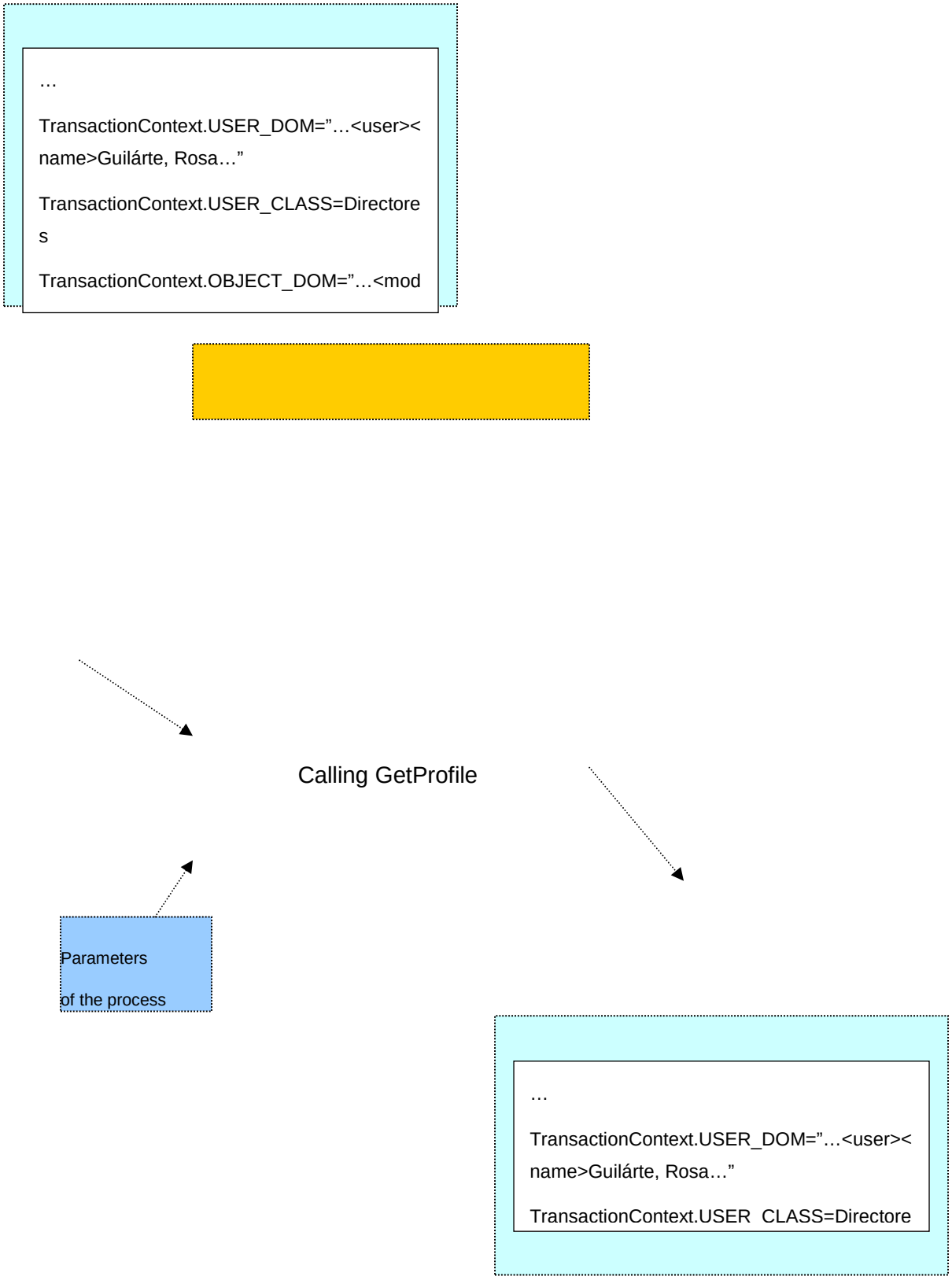
To select a Profile, it uses the "userClass" and "objectCategory" stored in the TransactionContext.

A "priority" parameter with valid values of "userClass" or "objectCategory" may be passed to the Process.

The behaviour is as follows:

- ✓ It attempts to get a Profile that matches (userClass, objectCategory)
- ✓ If such a Profile does not exist in the currently active Policy then:
    1. If the "priority" parameter is set to "userClass", attempt to get a Profile in this order:
        1. (TC.userClass, *)
        2. (*, TC.objectCategory)
        3. (*, *)
    2. If the "priority" parameter is set to "objectCategory", attempt to get a Profile in this order:
        1. (*, TC.objectCategory)
        2. (TC.userClass, *)
        3. (*, *)
    3. If the "priority" parameter does not exist, return an error
- ✓ If a Profile still couldn't be found, return an error

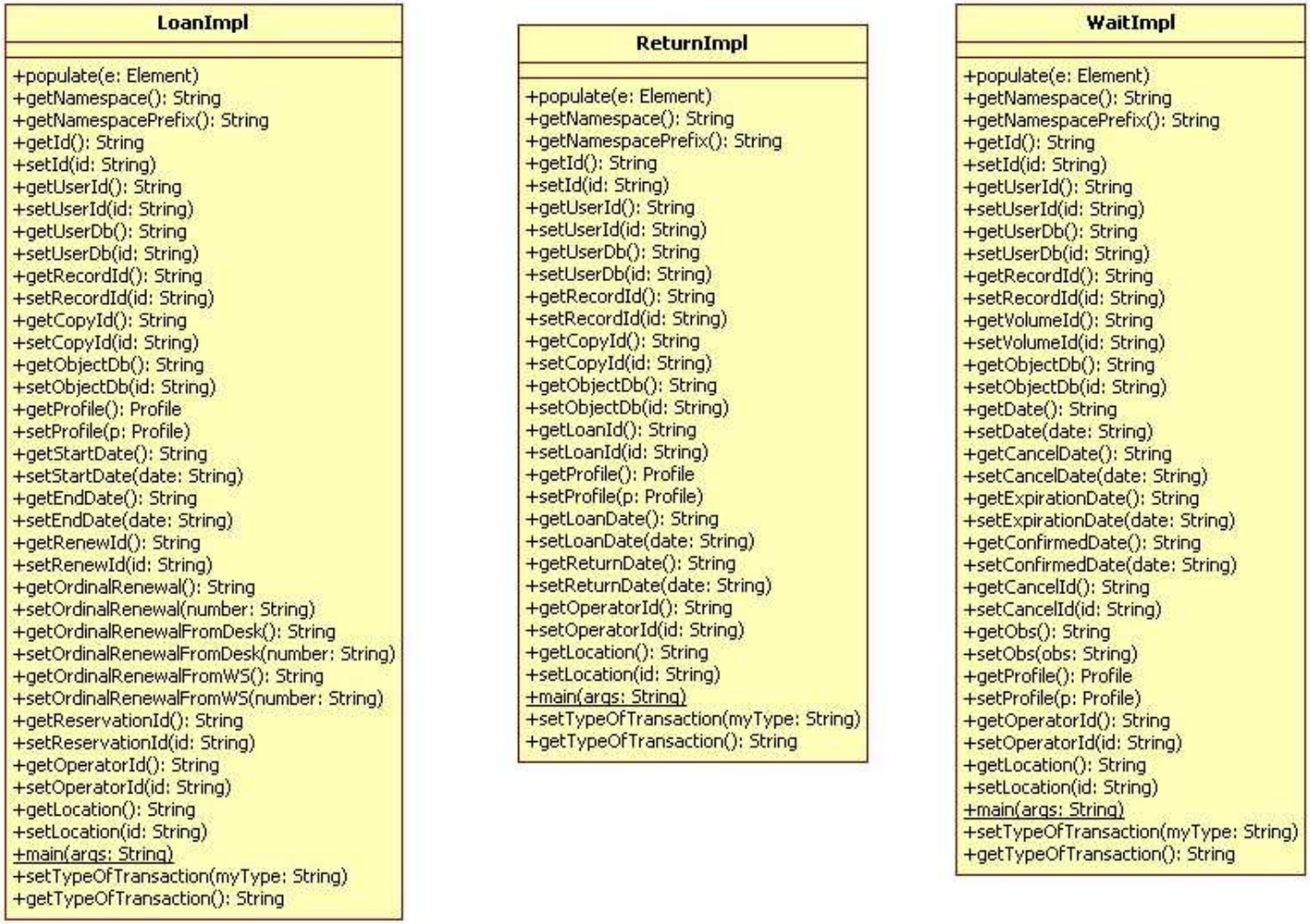
TC.userClass represents the userClass in the TransactionContext (and similarly for TC.objectCategory)


…
TransactionContext.USER_DOM="…<user><name>Guilárte, Rosa…"
TransactionContext.USER_CLASS=Directores
TransactionContext.OBJECT_DOM="…<mod
Calling GetProfile
Parameters
of the process
…
TransactionContext.USER_DOM="…<user><name>Guilárte, Rosa…"
TransactionContext.USER_CLASS=Directore

In the case of GetProfile we can see more clearly that in the TransactionContext objects are stored, which can later on be accessed by their properties or methods.

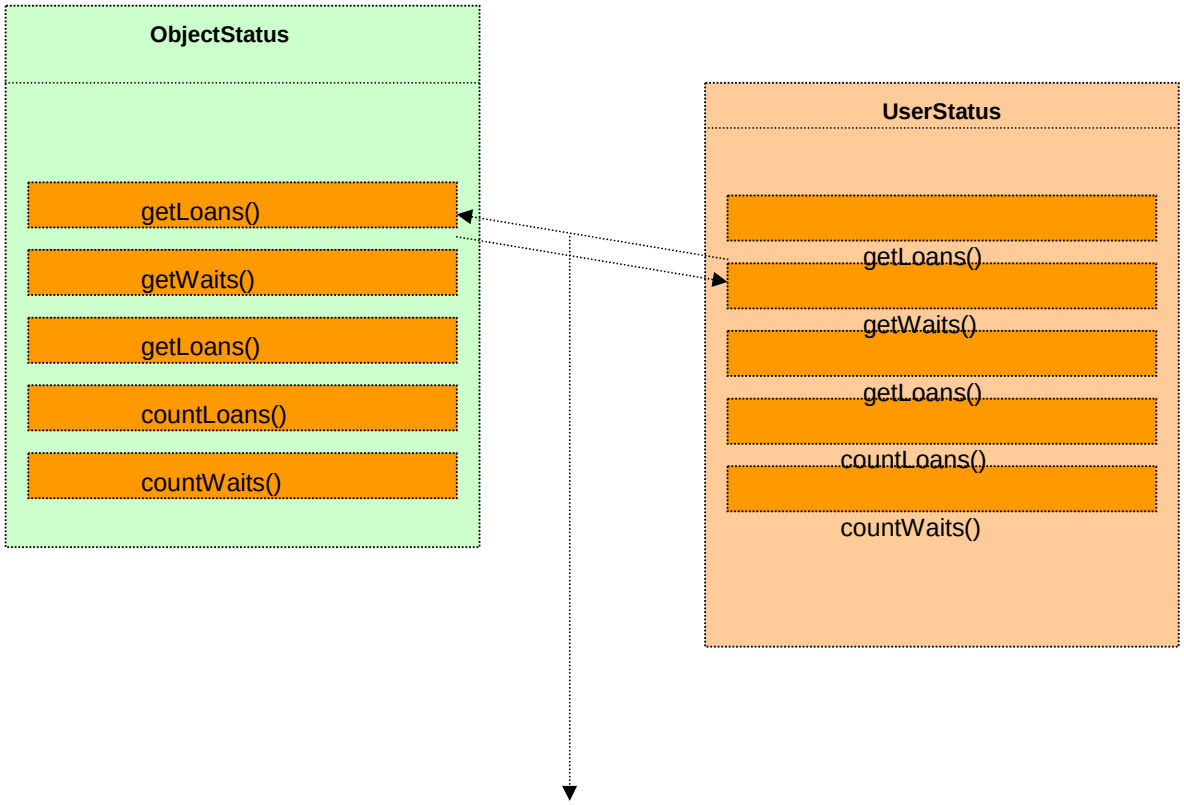
5. Object defined in EmpWeb.

The following is a basic diagram with the objects which are accessible by Groovy scripts, together with their public access methods.


LoanImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(id: String)
+getUserDb(): String
+setUserDb(id: String)
+getRecordId(): String
+setRecordId(id: String)
+getCopyId(): String
+setCopyId(id: String)
+getObjectDb(): String
+setObjectDb(id: String)
+getProfile(): Profile
+setProfile(p: Profile)
+getStartDate(): String
+setStartDate(date: String)
+getEndDate(): String
+setEndDate(date: String)
+getRenewId(): String
+setRenewId(id: String)
+getOrdinalRenewal(): String
+setOrdinalRenewal(number: String)
+getOrdinalRenewalFromDesk(): String
+setOrdinalRenewalFromDesk(number: String)
+getOrdinalRenewalFromWS(): String
+setOrdinalRenewalFromWS(number: String)
+getReservationId(): String
+setReservationId(id: String)
+getOperatorId(): String
+setOperatorId(id: String)
+getLocation(): String
+setLocation(id: String)
+main(args: String)
+setTypeOfTransaction(myType: String)
+getTypeOfTransaction(): String
ReturnImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(id: String)
+getUserDb(): String
+setUserDb(id: String)
+getRecordId(): String
+setRecordId(id: String)
+getCopyId(): String
+setCopyId(id: String)
+getObjectDb(): String
+setObjectDb(id: String)
+getLoanId(): String
+setLoanId(id: String)
+getProfile(): Profile
+setProfile(p: Profile)
+getLoanDate(): String
+setLoanDate(date: String)
+getReturnDate(): String
+setReturnDate(date: String)
+getOperatorId(): String
+setOperatorId(id: String)
+getLocation(): String
+setLocation(id: String)
+main(args: String)
+setTypeOfTransaction(myType: String)
+getTypeOfTransaction(): String
WaitImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(id: String)
+getUserDb(): String
+setUserDb(id: String)
+getRecordId(): String
+setRecordId(id: String)
+getVolumeId(): String
+setVolumeId(id: String)
+getObjectDb(): String
+setObjectDb(id: String)
+getDate(): String
+setDate(date: String)
+getCancelDate(): String
+setCancelDate(date: String)
+getExpirationDate(): String
+setExpirationDate(date: String)
+getConfirmedDate(): String
+setConfirmedDate(date: String)
+getCancelId(): String
+setCancelId(id: String)
+getObs(): String
+setObs(obs: String)
+getProfile(): Profile
+setProfile(p: Profile)
+getOperatorId(): String
+setOperatorId(id: String)
+getLocation(): String
+setLocation(id: String)
+main(args: String)
+setTypeOfTransaction(myType: String)
+getTypeOfTransaction(): String


FineImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+getTableName(): String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(id: String)
+getUserDb(): String
+setUserDb(id: String)
+getDate(): String
+setDate(date: String)
+getType(): String
+setType(t: String)
+getObs(): String
+setObs(obs: String)
+getAmount(): String
+setAmount(amount: String)
+getLocation(): String
+setLocation(id: String)
+getOperatorId(): String
+setOperatorId(id: String)
+getObject_copyId(): String
+setObject_copyId(id: String)
+getObject_objectDb(): String
+setObject_objectDb(id: String)
+getObject_recordId(): String
+setObject_recordId(id: String)
+getObject_profile(): Profile
+setObject_profile(p: Profile)
+getObject_loanStartDate(): String
+setObject_loanStartDate(date: String)
+getObject_loanEndDate(): String
+setObject_loanEndDate(date: String)
+getObject_daysOverdue(): int
+setObject_daysOverdue(days: int)
+getPaid_date(): String
+setPaid_date(date: String)
+getPaid_amount(): String
+setPaid_amount(amount: String)
+getRefId(): String
+setRefId(id: String)
+main(args: String)
+setTypeOfTransaction(myType: String)
+getTypeOfTransaction(): String
ProfileImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+toXMLString(): String
+toDOMElement(): Element
+getId(): String
+setId(id: String)
+getUserClass(): String
+setUserClass(uclass: String)
+getObjectCategory(): String
+setObjectCategory(ocategory: String)
+getPolicyId(): String
+setPolicyId(id: String)
+getTimestamp(): String
+setTimestamp(ts: String)
+getLimitNames(): String
+getLimitMap(): Map
+getLimit(limitName: String): String
+getLimit(limitName: String, defaultValue: String): String
+setLimit(limitName: String, limitValue: String)
+main(args: String)
SuspensionImpl
+populate(e: Element)
+getNamespace(): String
+getNamespacePrefix(): String
+getId(): String
+setId(id: String)
+getUserId(): String
+setUserId(id: String)
+getUserDb(): String
+setUserDb(id: String)
+getDate(): String
+setDate(date: String)
+getStartDate(): String
+setStartDate(date: String)
+getDaysSuspended(): int
+setDaysSuspended(days: int)
+getEndDate(): String
+setEndDate(date: String)
+getType(): String
+setType(type: String)
+getObs(): String
+setObs(obs: String)
+getCancelId(): String
+setCancelId(id: String)
+getLocation(): String
+setLocation(id: String)
+getOperatorId(): String
+setOperatorId(id: String)
+getObject_copyId(): String
+setObject_copyId(id: String)
+getObject_objectDb(): String
+setObject_objectDb(id: String)
+getObject_recordId(): String
+setObject_recordId(id: String)
+getObject_profile(): Profile
+setObject_profile(p: Profile)
+getObject_loanStartDate(): String
+setObject_loanStartDate(date: String)
+getObject_loanEndDate(): String
+setObject_loanEndDate(date: String)
+getObject_daysOverdue(): int
+setObject_daysOverdue(days: int)
+main(args: String)
+setTypeOfTransaction(myType: String)
+getTypeOfTransaction(): String

6. Special objects used in EmpWeb



Parallel methods. In the rules these will be frequently used, like e.g. to know or browse the list of loan and/or reservations which the user or objects has pending.

7. **Structure of a process or rule in a pipeline.**

As can be observed, the processes or rules have a first entry-box dedicated for the documentation, a second one for the specifications fo the limits and a third one in which the paremeters of the process are specified. Let's take the relevant ones one by one.

8. Limits published by a process or rule.

When a process or rule is dependent of a particular profile, let's assume e.g. that we need a process which sends an alert to the operator when the actual number of loans is superior to X.

X is a limit which can have a value set as x1 for the profile (Students, Books) and a value x2 for the profile (Directors, Books).

In this case, it is indicated that X is a limit which will be used within the process or rule with the correct value for the profile.

The syntax for putting values in the limits is as follows :

```
<limits>
  <limit name="name">default value</limit>
</limits>
```

Let's suppose now that a new limit is created by a particular rule or process. This will imply that the value will need to be filled in in all profiles which EmpWeb currently has registered for the active policy. In the case that the profile to be used does not have a value filled in for the correct profile, the default value will be used.

It is important to emphasize that before the use of any process or rule which uses limits, The basic class GetProfile should have been called, given that if not the default value will always be used.

9. Parameters for a rule or process.

The parameters of a particular rule or process can have specific individual behaviours specified during the use of the same process or rule in different pipelines.

The syntax of the parameters is :

```
<params>
    <param name="attribute">value</param>
</params>
```

An example can be obtained from the basic class UpdateDb, which is used in several pipelines, and in which different values could be used for the parameters for each use.

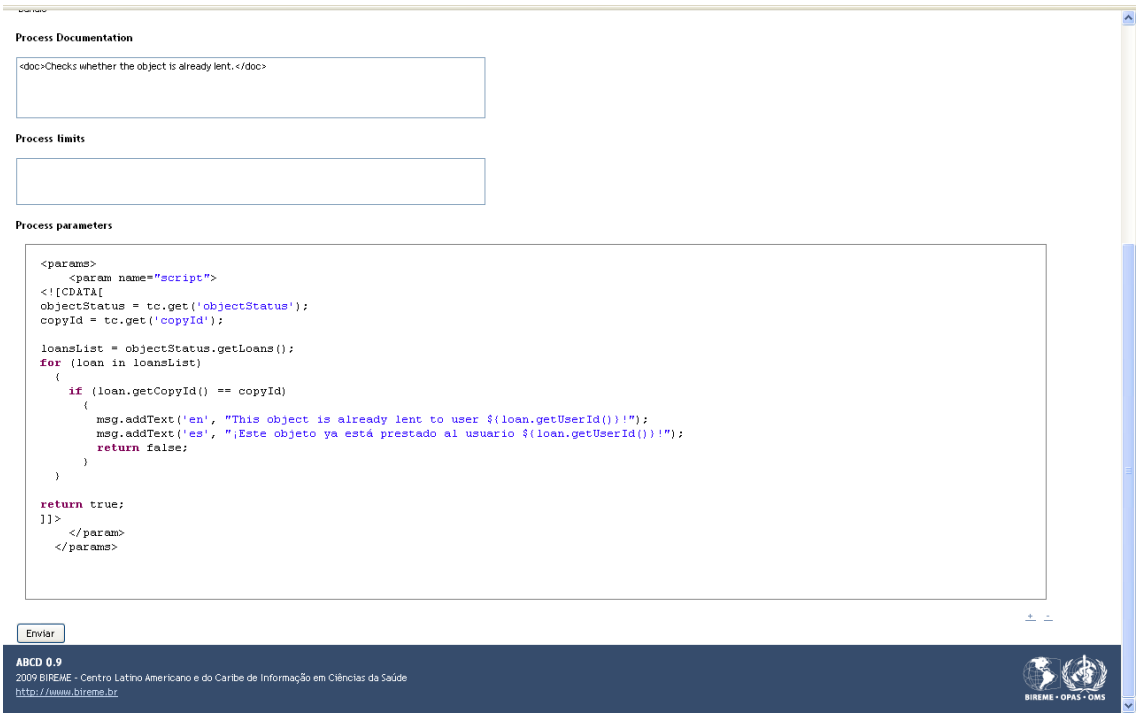
Example :

```
<params>
    <param name="transactionKeys">paymentFine,pendingFine</param>
    <param name="ignoreTransactionNotFound">true</param>
    <param name="storeUserStatus">true</param>
    <param name="storeObjectStatus">false</param>
</params>
```

10. Rule or process types of Groovy scripts.

Scripted processes or rules are special rules or processes which do not need the file of the application to be compiled fully and which allow interaction with the EmpWeb objects like for implementing particular negotiation rules.

The structure of a scripted rule or process that can be used consists of :



The same files are taken into account as with a basic rule or process of EmpWeb, only the Groovy script deals with the parameters.

The structure of this is similar to the one of any parameter of a rule or process.

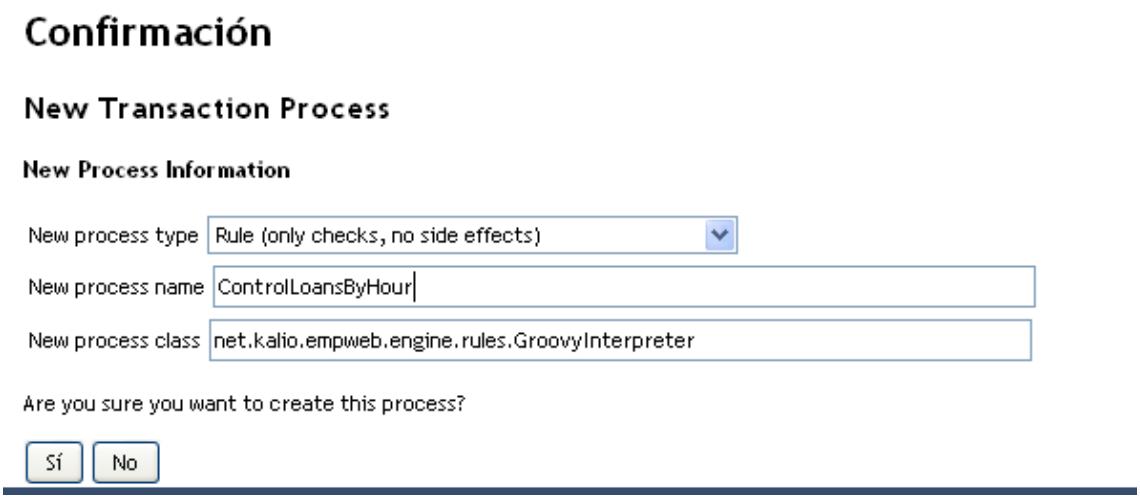
```
<params>
<param name="script">

        <![CDATA[



        …. instructios of the script.


        ]]>
</param>
</params>
```

A Groovy script deals with a set of variables which we can call “magic variables” with which one can interact freely :

11. First example of a Groovy script.

The first step for aggregating a Groovy script consists of creating a rule or process for the correct pipeline. Continuing the previous example, we will create a new scripted process in the Loans pipeline, which will send a message to the operator.

Clicking on the option Create Process the following window will appear :

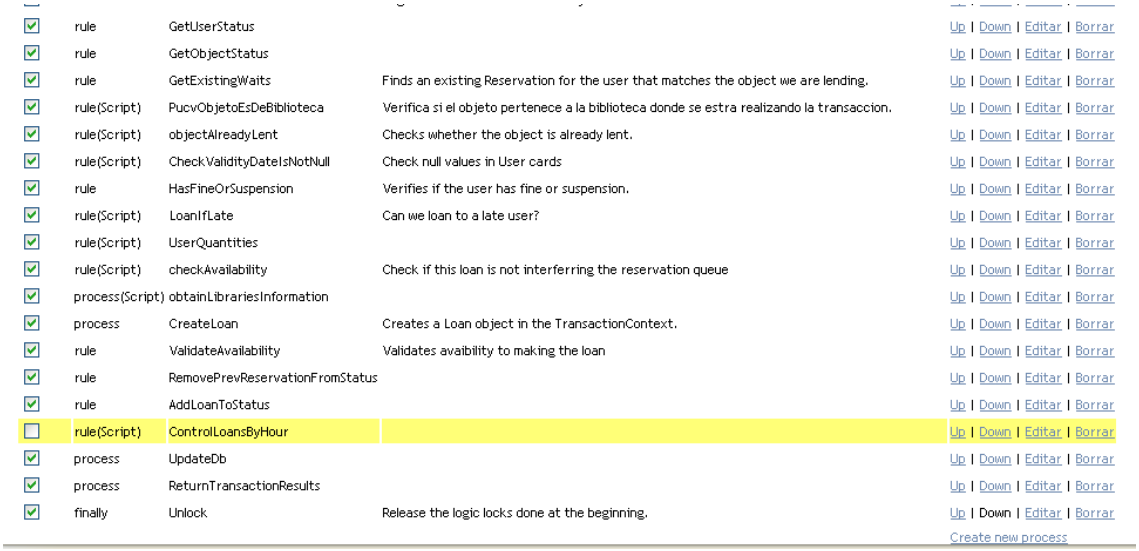


We select the name of a process or rule and indicate that the class which will interprete our script will be net.kalio.empWeb.engine.rules.GroovyInterpreter.

Once filled in this window, the process which we have just created will apper in the pipeline, inactive and at the end of the list.

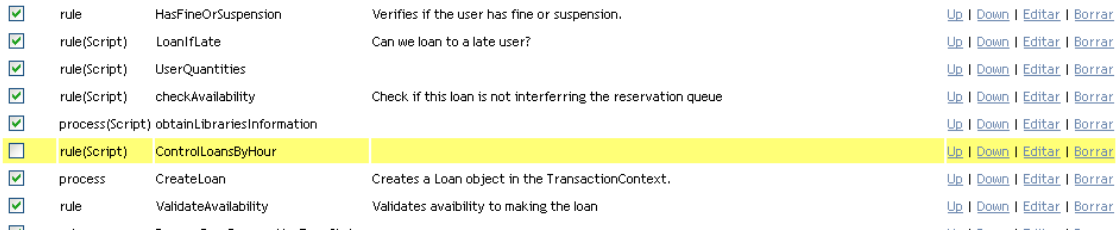
Important therefore, since the pipeline is an ordered sequence, to put the process within the sequence of the pipeline at the right position.

This will imply, given that we pretend to create a validation rule, that the said rule executes before the registration of the transaction. Therefore, using the options 'Up/Down' allowing repositioning the process, one has to locate the process at the right position for its execution :



☑	rule	GetUserStatus		Up | Down | Editar | Borrar
☑	rule	GetObjectStatus		Up | Down | Editar | Borrar
☑	rule	GetExistingWaits	Finds an existing Reservation for the user that matches the object we are lending.	Up | Down | Editar | Borrar
☑	rule(Script)	PucvObjetoEsDeBiblioteca	Verifica si el objeto pertenece a la biblioteca donde se estra realizando la transaccion.	Up | Down | Editar | Borrar
☑	rule(Script)	objectAlreadyLent	Checks whether the object is already lent.	Up | Down | Editar | Borrar
☑	rule(Script)	CheckValidityDateIsNotNull	Check null values in User cards	Up | Down | Editar | Borrar
☑	rule	HasFineOrSuspension	Verifies if the user has fine or suspension.	Up | Down | Editar | Borrar
☑	rule(Script)	LoanIfLate	Can we loan to a late user?	Up | Down | Editar | Borrar
☑	rule(Script)	UserQuantities		Up | Down | Editar | Borrar
☑	rule(Script)	checkAvailability	Check if this loan is not interferring the reservation queue	Up | Down | Editar | Borrar
☑	process(Script)	obtainLibrariesInformation		Up | Down | Editar | Borrar
☑	process	CreateLoan	Creates a Loan object in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	ValidateAvailability	Validates avaibility to making the loan	Up | Down | Editar | Borrar
☑	rule	RemovePrevReservationFromStatus		Up | Down | Editar | Borrar
☑	rule	AddLoanToStatus		Up | Down | Editar | Borrar
☐	rule(Script)	ControlLoansByHour		Up | Down | Editar | Borrar
☑	process	UpdateDb		Up | Down | Editar | Borrar
☑	process	ReturnTransactionResults		Up | Down | Editar | Borrar
☑	finally	Unlock	Release the logic locks done at the beginning.	Up | Down | Editar | Borrar
				Create new process

Pressing repeatedly on the link Up we drop the new rule to be created before the execution of the creation of the Loan entity. So it stays in the position as can be seen under here :



☑	rule	HasFineOrSuspension	Verifies if the user has fine or suspension.	Up | Down | Editar | Borrar
☑	rule(Script)	LoanIfLate	Can we loan to a late user?	Up | Down | Editar | Borrar
☑	rule(Script)	UserQuantities		Up | Down | Editar | Borrar
☑	rule(Script)	checkAvailability	Check if this loan is not interferring the reservation queue	Up | Down | Editar | Borrar
☑	process(Script)	obtainLibrariesInformation		Up | Down | Editar | Borrar
☐	rule(Script)	ControlLoansByHour		Up | Down | Editar | Borrar
☑	process	CreateLoan	Creates a Loan object in the TransactionContext.	Up | Down | Editar | Borrar
☑	rule	ValidateAvailability	Validates avaibility to making the loan	Up | Down | Editar | Borrar

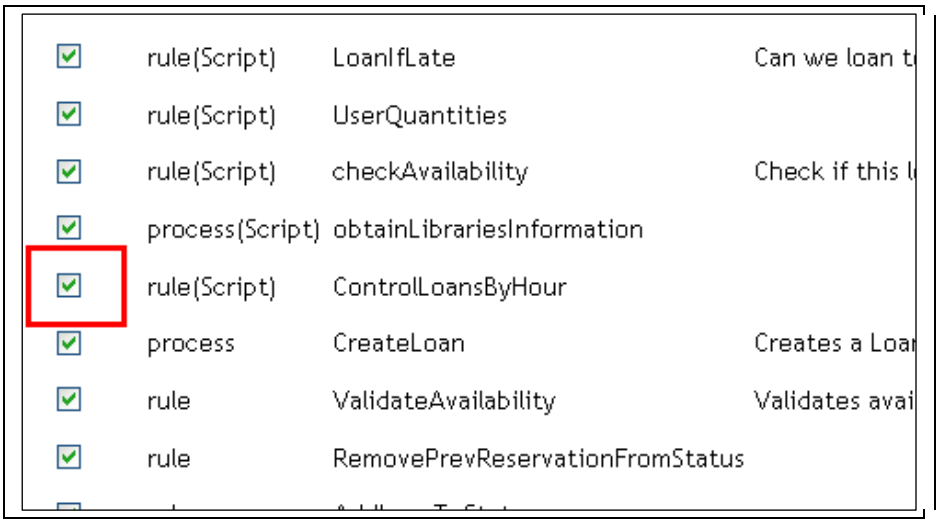
Then we press the link Edit and we are ready to create our first Groovy script which validates a particular situation.

Each Groovy script will have a syntax mentioned earlier in its parameters :

Process parameters

```
<params>
    <param name="script">
<![CDATA[

]]>
    </param>
  </params>
```

Warning : If the rule is to be included in the process of basic execution, remember to activate it after edition, i.e. :



The rule or process has to be ticked for it to executed.

So now we can work on our first script.

The case to be evaluated, as we envisage, wil lbe :

"…The users of type Coordinators will not be allowed to have more than 2 books at the same time per reading room or per hour …"

If we put this in a pseudo-code acting on the Groovy objects, we will have :

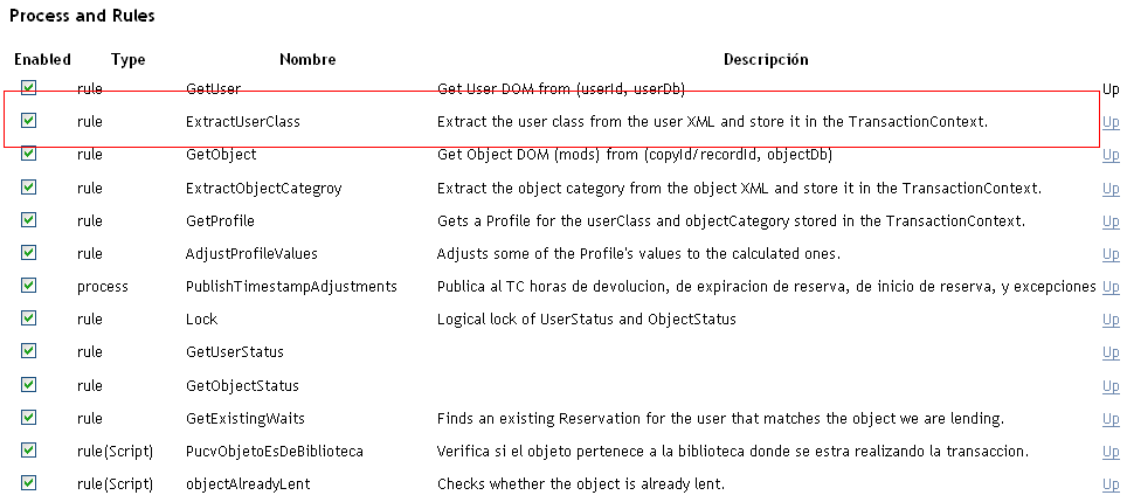
```
Get the user type;
If the user type is Coordinators then
    For (loans by this user)
        If the loan is an object loanable per hour entonces
            Count
    End For
    If number counted > 1 then

        Send message
        Return that the rule failed

    End If
End If
Return that the rule accomplished.
```

In this step it will be important to document that in every Groovy script a set of "magic variables" can be used :

- Variable tc (TransactionContext)
- Variable params (the parameters of the current rule or process)
- Variable msg (messages which can be produced by the current script)

Therefore, reviewing the objects which we have available in the EmpWeb Objects universum, we can analyze how we can get the actua user's type.

If we look into the list of processes actually existent in the pipeline, we would see that there is already a process present which is named ExtractUserClass. This one fills in the value for TransactionContext. USER_CLASS



Process and Rules

Enabled	Type	Nombre	Descripción	
☑	rule	GetUser	Get User DOM from (userId, userDb)	Up
☑	rule	ExtractUserClass	Extract the user class from the user XML and store it in the TransactionContext.	Up
☑	rule	GetObject	Get Object DOM (mods) from (copyId/recordId, objectDb)	Up
☑	rule	ExtractObjectCategroy	Extract the object category from the object XML and store it in the TransactionContext.	Up
☑	rule	GetProfile	Gets a Profile for the userClass and objectCategory stored in the TransactionContext.	Up
☑	rule	AdjustProfileValues	Adjusts some of the Profile's values to the calculated ones.	Up
☑	process	PublishTimestampAdjustments	Publica al TC horas de devolucion, de expiracion de reserva, de inicio de reserva, y excepciones	Up
☑	rule	Lock	Logical lock of UserStatus and ObjectStatus	Up
☑	rule	GetUserStatus		Up
☑	rule	GetObjectStatus		Up
☑	rule	GetExistingWaits	Finds an existing Reservation for the user that matches the object we are lending.	Up
☑	rule(Script)	PucvObjetoEsDeBiblioteca	Verifica si el objeto pertenece a la biblioteca donde se estra realizando la transaccion.	Up
☑	rule(Script)	objectAlreadyLent	Checks whether the object is already lent.	Up

Therefore we could start the Groovy script by checking this value :

```
tipousuario = tc.get(TransactionContext.USER_CLASS);

If (tipousuario==’Students’)

{


}

return true;
```

With this we have implemented the first step, related to the check of the user type. Now we will check the current loans which the users has on his name.

For this, we will use the special object type indicated previously, in this case the UserStatus. Using this, we can get a list of the objects which correspond to the loans of the actual user.

But how can we get the UserStatus in the current script ?

As in the previous case, when we check the actual pipeline, we will see that there is another process, already in the TransactionContext, giving the userStatus needed.

Process and Rules

Enabled	Type	Nombre	Descripción	
☑	rule	GetUser	Get User DOM from (userId, userDb)	Up
☑	rule	ExtractUserClass	Extract the user class from the user XML and store it in the TransactionContext.	Up
☑	rule	GetObject	Get Object DOM (mods) from (copyId/recordId, objectDb)	Up
☑	rule	ExtractObjectCategroy	Extract the object category from the object XML and store it in the TransactionContext.	Up
☑	rule	GetProfile	Gets a Profile for the userClass and objectCategory stored in the TransactionContext.	Up
☑	rule	AdjustProfileValues	Adjusts some of the Profile's values to the calculated ones.	Up
☑	process	PublishTimestampAdjustments	Publica al TC horas de devolucion, de expiracion de reserva, de inicio de reserva, y excepciones	Up
☑	rule	Lock	Logical lock of UserStatus and ObjectStatus	Up
☑	rule	GetUserStatus		Up
☑	rule	GetObjectStatus		Up
☑	rule	GetExistingWaits	Finds an existing Reservation for the user that matches the object we are lending.	Up
☑	rule(Script)	PucvObjetoEsDeBiblioteca	Verifica si el objeto pertenece a la biblioteca donde se estra realizando la transaccion.	Up
☑	rule(Script)	objectAlreadyLent	Checks whether the object is already lent.	Up
☑	rule(Script)	CheckValidityDateIsNotNull	Check null values in User cards	Up

So, we can access a key from the TransactionContext, which will return the userStatus of the current user.

Using this object userStatus, we can browse the loans of this user.

```
uStatus = tc.get(TransactionContext.USER_STATUS);
```

Which methods support userStatus? One of them is getLoans(), which will give us the loans of the current user :

```
loans = uStatus.getLoans();
```

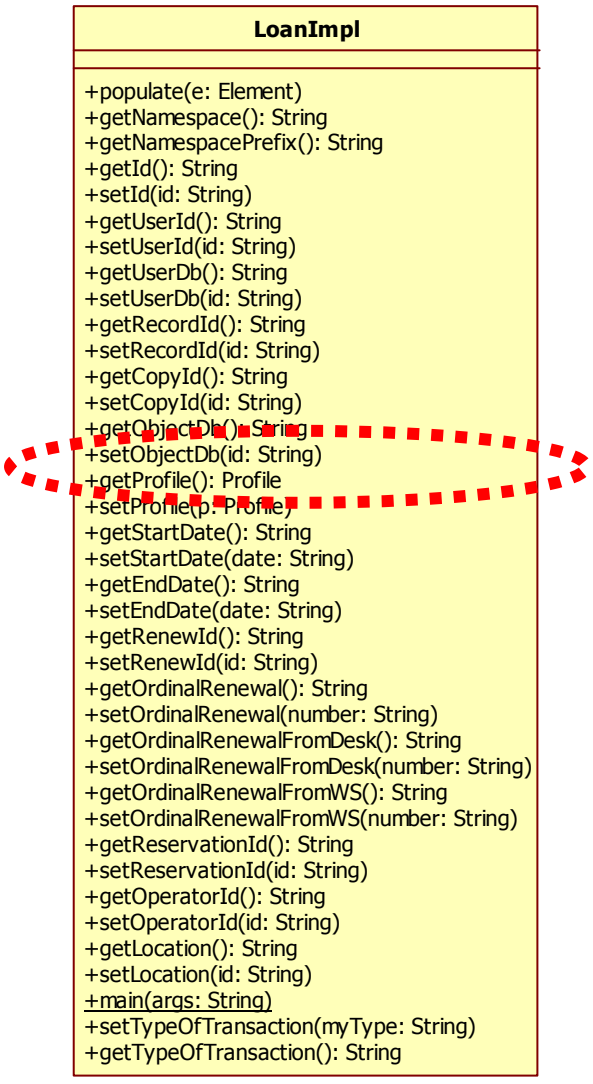
We will now try to browse this list returned by the object userStatus and note the loans which are relevant to us.

We use a Groovy loop for this :

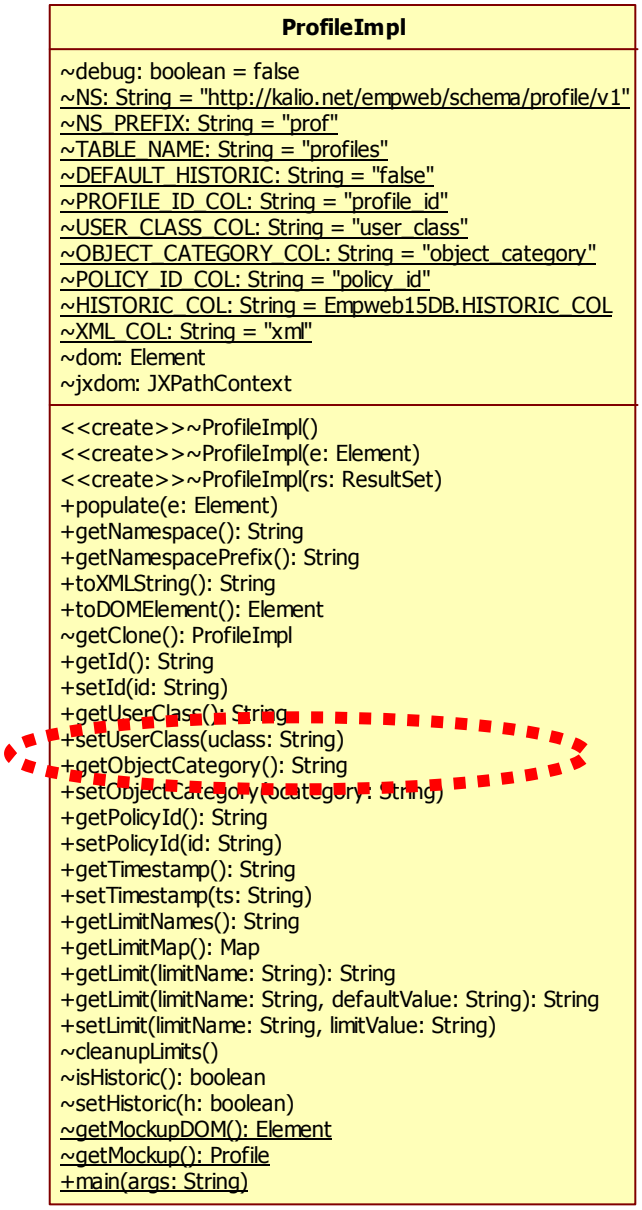
for (loan in loansList) { // Here we will analyze each individual loan}

If we analyze the object type Loan, we will find out that there is no method which gives us the loaned object type.

This is due to the fact that the Loan object has a reference to the profile (object type, user type) in which is was created.



What we can do now is obtain the profile, from which the indeed can obtain the string which represents the loan object type:



Thus, this part of the code in Groovy would be as follows :

```
Profile prof = loan.getProfile();

If (prof.getObjectCategory='LBH')

{
```

```
    //count
}
```

Now the final version of our code would read :

```
<![CDATA[

usertype = tc.get(TransactionContext.USER_CLASS);
if (usertype =='Coordinators')
{
    uStat = tc.get(TransactionContext.USER_STATUS);
    loans = uStat.getLoans();
    counter = 0;
    for (loan in loans)
    {
        Profile prof = loan.getProfile();
        objCateg = prof.getObjectCategory();
        if (objCateg=='LBH')
        {
              counter++;
        }

    }

    if (counter>1)
    {
      msg.addText("en_CL_UCV","It's impossible to loan to Coordinators more than 2 concurrent by hour books");
      msg.addText("es_CL_UCV","No es posible prestar mas de dos libros concurrentes por hora a Coordinadores");
```

```
        return false;
    }
}
return true;
]]>
```

If we dwell on the msg area, it displays a result of the transaction, and its syntax is as follows:

```
msg.add ("language identifier", "message");
```

Interestingly, in evaluating the entered script execution stopped; this is due to the fact that there is a loan per hour. This is because we have a loan for which the rule when applied does not meet the originally specified rule.

We can now conclude that the logics of the pseudo-cod was not correct, and we propose the correct version in what follows :

```
Get the user type;
Get the object type which the user wants to take;

If the user type is Coordinators  AND  the object type is for reading room only then
    For ( loans by this user)
        If the loan is of object type per hour hen
            Count
    End For
    If number counted > 1 then

        Produce message
        Return rule failed
```

```
        End If
End If
```

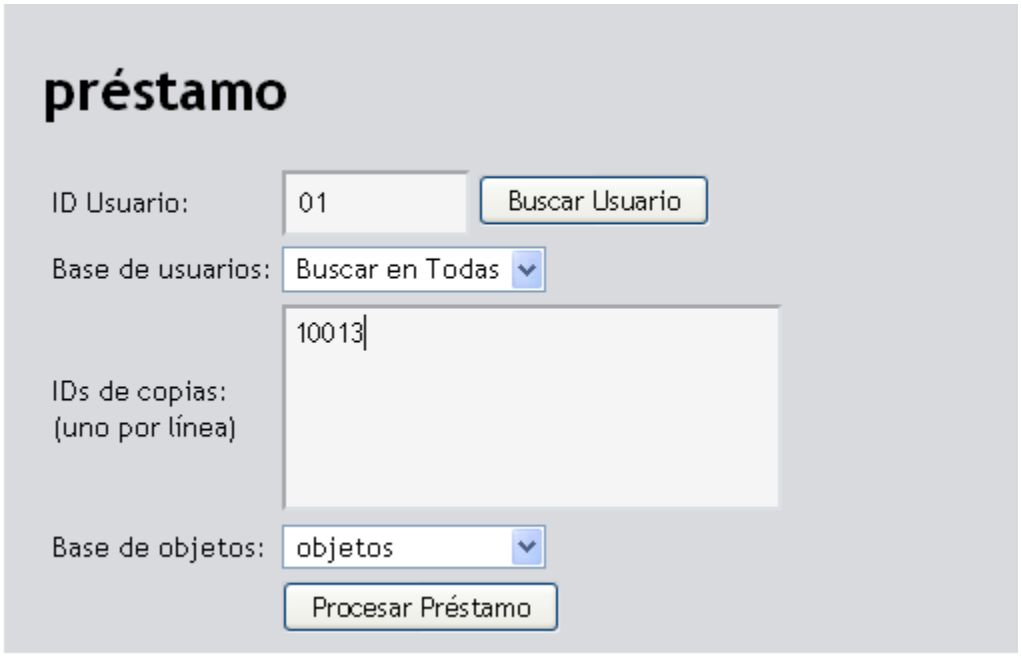
Return rule accomplished.

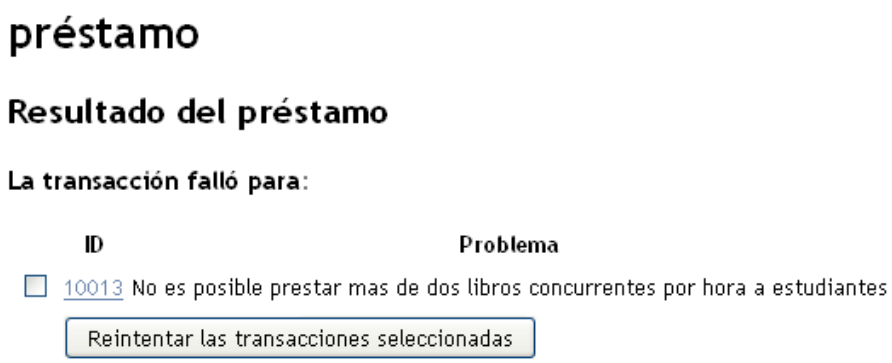
In the implementation we see that it comes out rather simple to assemble this operation :

```
<![CDATA[

UserType = tc.get(TransactionContext.USER_CLASS);
ActualObjectType = tc.get(TransactionContext.OBJECT_CATEGORY);

if (UserType=='Coordinators' && ActualObjectType =='LBH' )
{
   uStat = tc.get(TransactionContext.USER_STATUS);
   loans = uStat.getLoans();

        counter = 0;
   for (loan in loans)
   {
       Profile prof = loan.getProfile();
       objCateg = prof.getObjectCategory();
       if (objCateg=='LBH')
       {
              counter++;
       }
```

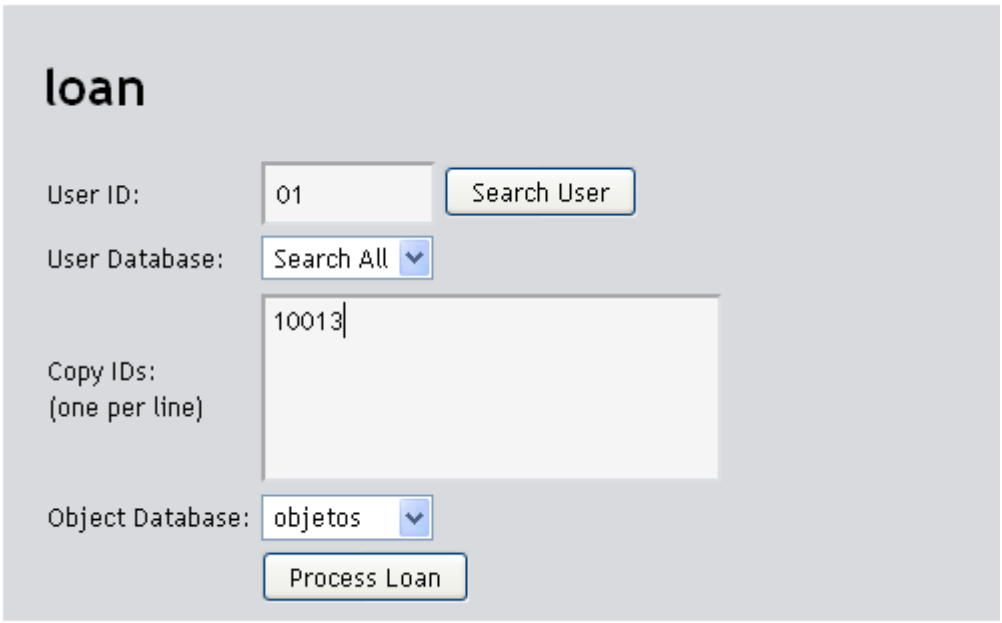
```
    }


    if (counter > 1)
    {
        msg.addText("en_CL_UCV","It is impossible to loan to Students more than 2 concurrent by hour books");
        msg.addText("es_CL_UCV","No es posible prestar mas de dos libros concurrentes por hora a estudiantes");
        return false;
    }
}


return true;
```

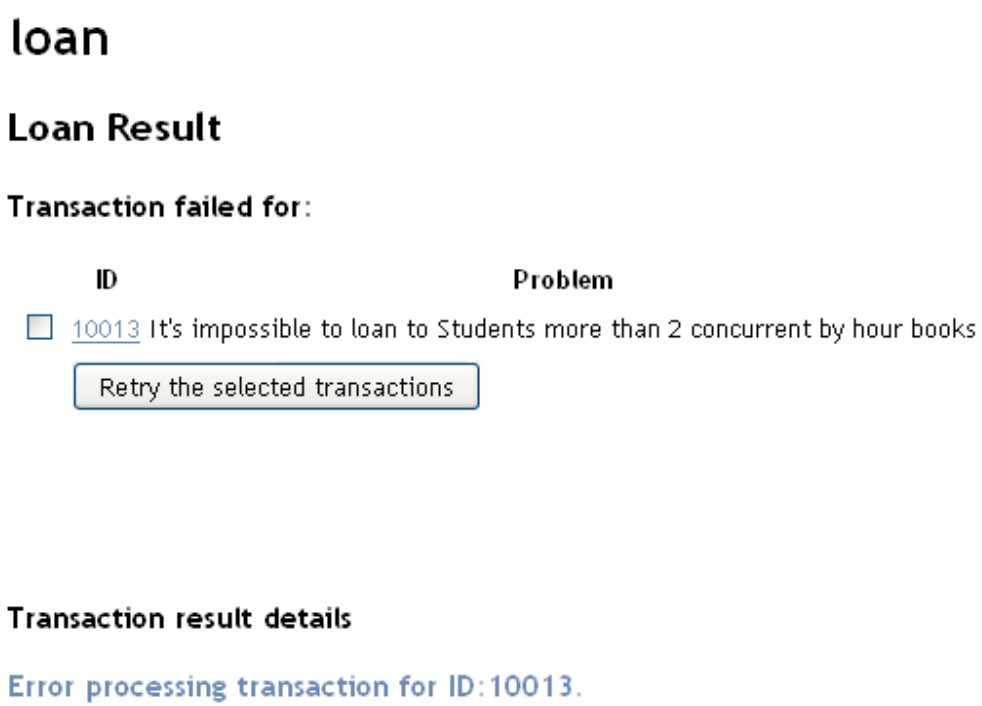
When we execute this we can observe, for example with the user 01 who is a Coordinator and how has the item 10012 on loan, the followin screens :





The same situation in English would show as :





With this we can see clearly the usefulness of associating each message with the identifier fo the corresponding language.

Attention : when sending messages to the MySite these ones would have to be repeated with the identifier en and es (or pt, fr, etc) but without the identifier part xx_CL_UCV.
