<u>User’s Guide for PServer</u>

Collaboration between National Center for Scientific Research “Demokritos” ***(Institute of Informatics & Telecommunications)*** *and SciFY*

Table of Contents

1) Introduction

1.1) What is PServer?

PServer is a general purpose personalization server. So PServer is a program of functionality that can be used by different kinds of applications that need to provide personalized services.

PServer is designed to be platform and application independent. That means it can run on different operating systems, and can provide generic personalization services to any applications regardless of their content. To accomplish these tasks, PServer's implementation has been done with the Java programming language and without any usage of operating system API, and is usable under any known operating system that has a port of Java virtual machine version 1.5+.

We have tested PServer under Linux, Windows, and Solaris, and works perfectly. PServer needs a RDBMS to store its data and we have chosen MySQL (version 5+), which is a very mature, open source, and has a version for all the known operating systems software. Furthermore, to be platform independent, PServer is designed to communicate over HTTP protocol. Applications can make simple HTTP requests that contain the parameters of the request, and gather results through XML documents. This is pretty much the same way that RESTful web services work, but it is even “lighter”.

1.2) PServer Architecture

1.2.1) Logical Level

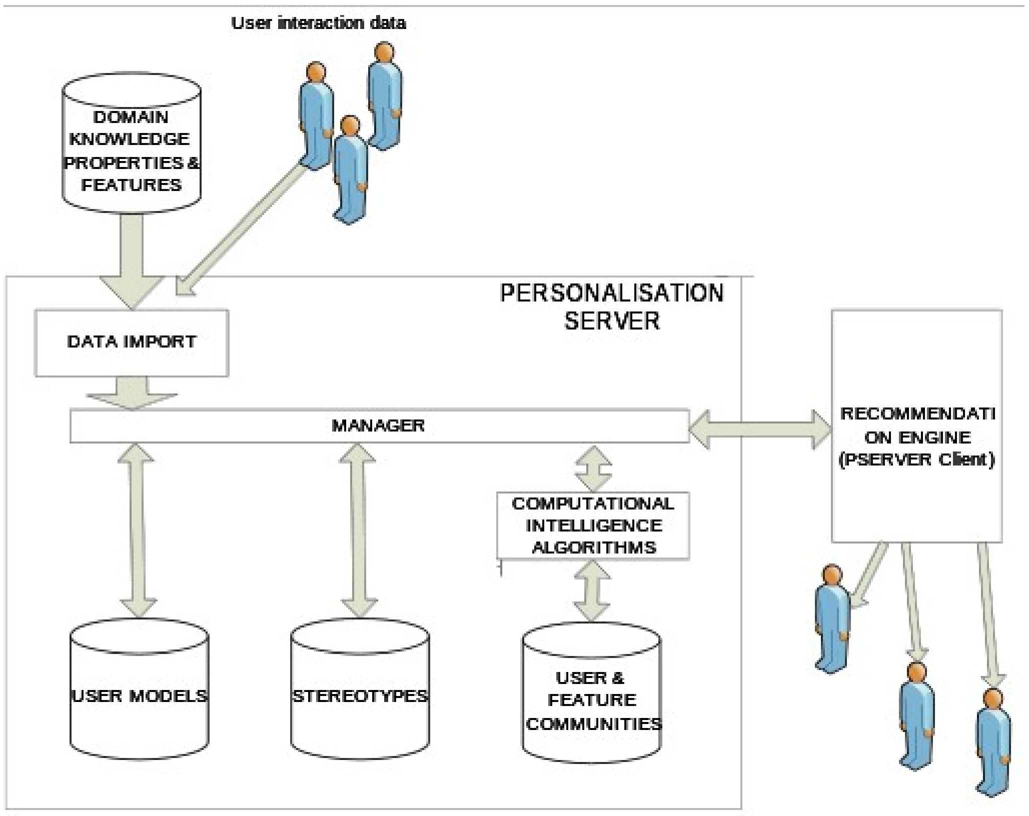
PServer’s central point is the user’s modeling, where three different models are provided. Specifically, Pserver can create and update user models (one for each user of a specific application), stereotype models, and also community models.

<u>Personal user models:</u> They are regarded as information that is stored for each user. For example, they could be exhibits that the user has visited, or how many times each exhibit has been seen. Users are determined by their specific attributes and characteristics. *Attributes* are related to the personal information of the user, and they remain unchanged during his interaction with the system (e.g. the language in which he receives the information, the age, and so on). *Features* refer to the ontology of the specific application, and they have arithmetic values, which manifest the importance for the specific user; for example, one «micro-plan» can be suitable for adults and unsuitable for young children. So the relative features will have a high value for adults and a low for children.

<u>Stereotypes:</u> They are groups of users with common attributes; for example, adults or children. Stereotypes have attributes and features like user models, but they have a basic difference. It’s not necessary for each stereotype to have the same number of features and attributes as the rest of the stereotypes.

<u>User Communities:</u> By use of machine learning algorithms it is possible for PServer to create groups of users and groups of features based on users' interaction with the system. Users’ groups are created through the finding of similarities between users, by use of clustering algorithms. For the groups of features, we would like to learn which have approximately the same values for users. For example, if a feature is as important as another for many users, we have to know it, because if a new user

wants something, then something else can be proposed to him, which the user will possibly be interested in.


User interaction data
DOMAIN KNOWLEDGE PROPERTIES & FEATURES
PERSONALISATION SERVER
DATA IMPORT
MANAGER
COMPUTATIONAL INTELLIGENCE ALGORITHMS
USER MODELS
STEREOTYPES
USER & FEATURE COMMUNITIES
RECOMMENDATI ON ENGINE (PSERVER Client)

1.2.2) Physical Level

On a physical level, PServer may reside as an application in a different machine and is implemented as a Web server that listens to a dedicated port, where all requests have the form of HTTP messages. Web browsers can be used as PServer clients, and responses from Pserver are encoded in XML syntax, and specially made XSL style sheets allow them to be displayed on browsers. Also, to facilitate applications, a client-side library of classes is available, that can be incorporated into the application to handle all low-level communication details.

1.3) Benefits from using PServer

- Personalization functionality as a **web service**:
    - **Centrally** installed and maintained.
    - Completely **separated** from applications.
    - Possible to be used by many applications **concurrently**.
    - Easily **accessible** through HTTP.
- Towards an adaptable model.
    - central concepts: application **features**, **users**.
    - Feature semantics not part of PServer.
    - Features organization in a **graph-like** manner facilitates querying.

2) PServer’s Components & Explanation

The main component of PServer is its data model, by which we have an abstract way to express users' likes and interests, and to apply (data mining) algorithms to extract information that can be useful to different kinds of applications. This model makes PServer application independent because we do not store any application semantics and we express different content types by using generic types. The generic types that we use are the following.

User: is the entity whose interests and likes we want to store.

Attribute: is any physical characteristic that a user has; for example, age is a user attribute.

Feature: is any object that we want to express how much the user likes or is interested in. A user can express his preference for and interest about a feature. We make this distinction because a user might be interested in features that others do not like at all.

User stereotype: is a group of users that have some common attributes. We use it for collaborating filtering. A user stereotype has users that belong to it, and its profile is calculated by a mathematical formula (sum etc) that takes as input the user profiles that are part of the stereotype.

User communities: is a group of users that has common likes about certain features.

Feature group: is a group of features that many users like concurrently.

3) Configuration & Running

3.1) Necessary Tools before Installation

As we have said, PServer is implemented in Java and uses MySQL as its RDBMS. That means that if you want to install Pserver, first you have to install MySQL version 5+; it is also advisable to install the MySQL Admin Tools, that provide a GUI, and JVM version 1.5+. You can download these applications from the sites of MySQL and Sun for free. After that, you should download the latest version of PServer files.

- PServer
- MySQL (it's also a good idea to install administration tools, or another visual environment)
- Java runtime edition (JRE) and preferably a java compiler (JDK) as well as a development environment (Netbeans)
- CMake (it is necessary for the community mode)
- C++ compiler like gcc (it is necessary for the community mode)

3.2) Configuration

3.2.1) MySQL Configuration

First, you have to configure the Pserver.sql file.
We open it with a text editor and replace **PServerDBname** with our DB name.
After that at the end of the file we find

grant all privileges on PServerDBname. to 'PServerDBUsername'@'localhost' identified by 'PServerDBPassword' with grant option;*

Replace *PserverDBname*, *PserverDBUsername*, *PserverDBPassword*
with your values.

After file's configuration, you have to start MySQL, and import the database schema. To do this, start a command prompt, and type the following:

i) *for windows:*
cd *<path to mysql/bin folder>*
e.g.: C:\xampp\mysql\bin
mysql –u root –p

for linux:
sudo mysql –u root –p

ii) To import now the database schema from the pserver.sql file that comes with PServer files, type the following:

\. *<replace this with the correct path of db file>***pserver.sql**
e.g.: for windows: \. C:\pserver\pserver.sql

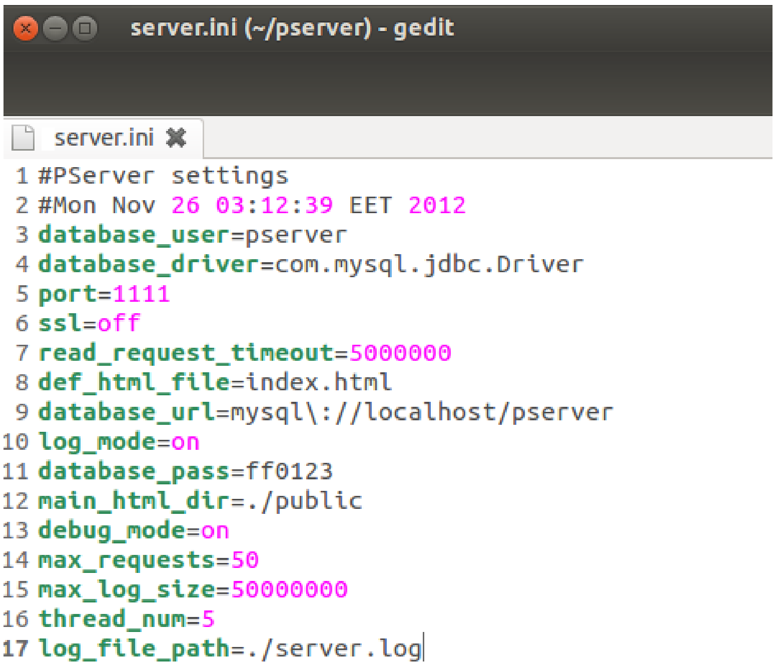
e.g.: for linux: \. /home/name/pserver/pserver.sql

3.2.2) Basic PServer Configuration

You have to set up variables in the server.ini and pbeans.ini files that exist in the pserver folder, as follows.

3.2.2.1) Configuration server.ini

You provide the user/name and password
database_user=pserver (no quotes, the user that you created)
database_url= mysql\://127.0.0.1/pserver (or the host of your mysql)
database_pass=ff0123 (no quotes, with the password that you entered previously)



```
#PServer settings
#Mon Nov 26 03:12:39 EET 2012
database_user=pserver
database_driver=com.mysql.jdbc.Driver
port=1111
ssl=off
read_request_timeout=5000000
def_html_file=index.html
database_url=mysql\://localhost/pserver
log_mode=on
database_pass=ff0123
main_html_dir=./public
debug_mode=on
max_requests=50
max_log_size=50000000
thread_num=5
log_file_path=./server.log
```

e.g.: screenshot with server.ini configuration

<u>**Server.ini Variables Explanation:**</u>

database_user=pserver
This parameter defines the user that is used by the PServer to connect to the database

database_driver=com.mysql.jdbc.Driver
This parameter defines the jdbc driver that will be used to connect to the database.

port=1111
This is the TCP port that PServer will bind to, in order to listen for requests.
Hint: *If you can’t see the PServer Administration Panel, or while running PServer the server's; port is -1, then it seems that another application is listening to port 1111, and you must change the PServer's port.*

ssl=off
This parameter defines if the communication is done by secure shell mode or not.

read_request_timeout=5000
This parameter defines the maximum time in milliseconds, for which PServer will keep a connection open waiting for a request.

main_html_dir=./public
Pserver is a simple HTTP server, and can serve files. This parameter defines the directory that contains the serve files.

administrator_name=root
This parameter defines the username of the administrator.

administrator_pass=root
This parameter defines the password of the administrator.

def_html_file=index.html
This parameter contains the default name of the HTML file that PServer serves from its web content directory.

database_url=mysql\://127.0.0.1/pserver
This parameter defines the jdbc URL that is used to connect to the database

log_mode=on
This parameter defines if messages are stored into a log into a file

database_pass=0000
This parameter defines the password, for the RDBMS user, that is used to connect to the database

debug_mode=on
This parameter defines if we want to log extra debug messages from the PServer

max_requests=50
This parameter defines the maximum requests that can be accepted concurrently.

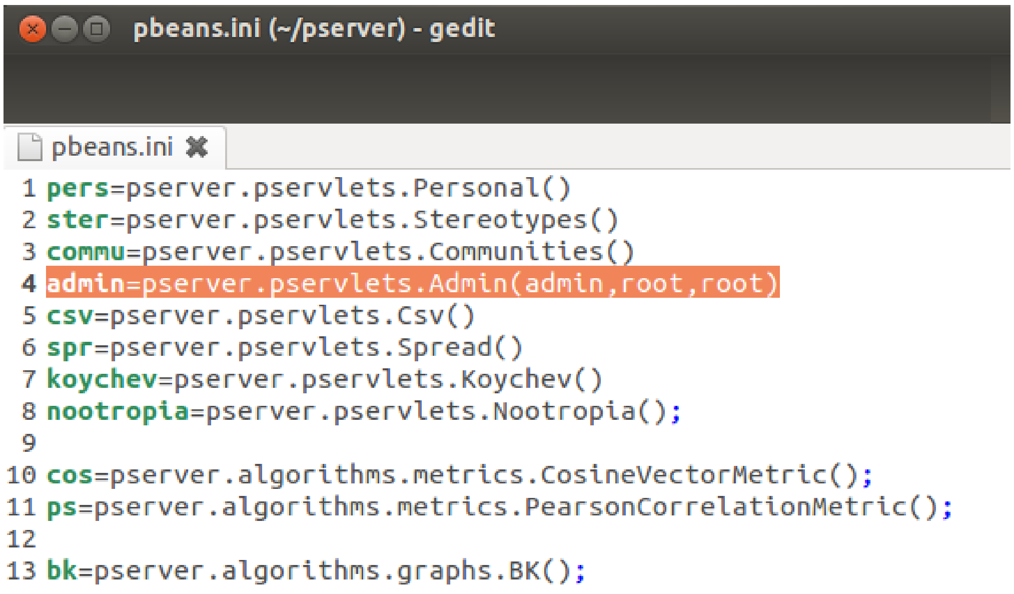
max_log_size=50000000
This parameter defines the maximum log file size in bytes

log_file_path=./server.log
This parameter defines the file where the log messages will be stored in.

thread_num=5
This parameter defines the maximum threads that PServer can use to execute processing. PServer might have to process a great amount of data, and this takes time. We use threads for parallel processing to speed up the server responses. These days every CPU has more than one cores and you have to parameterize PServer to use these cores. For an Intel Core 2 duo, our experiments have shown that the performance is increased up to 4 threads and for a Intel Core 2 quad, up to 5 threads. An I7 CPU has greater scaling but we have not determined its limit yet.

3.2.2.2) Configuration pbeans.ini

admin=pserver.pservlets.Admin (admin, admin_name, admin_pass)
Replace admin name and admin pass with the values that you want to use when you log in to the administration panel of Pserver. The word admin is a constant, and should stay as it is.



```
pers=pserver.pservlets.Personal()
ster=pserver.pservlets.Stereotypes()
commu=pserver.pservlets.Communities()
admin=pserver.pservlets.Admin(admin,root,root)
csv=pserver.pservlets.Csv()
spr=pserver.pservlets.Spread()
koychev=pserver.pservlets.Koychev()
nootropia=pserver.pservlets.Nootropia();

cos=pserver.algorithms.metrics.CosineVectorMetric();
ps=pserver.algorithms.metrics.PearsonCorrelationMetric();

bk=pserver.algorithms.graphs.BK();
```

e.g.: screenshot with pbeans.ini configuration

<u>pbeans.ini variables explanation:</u>

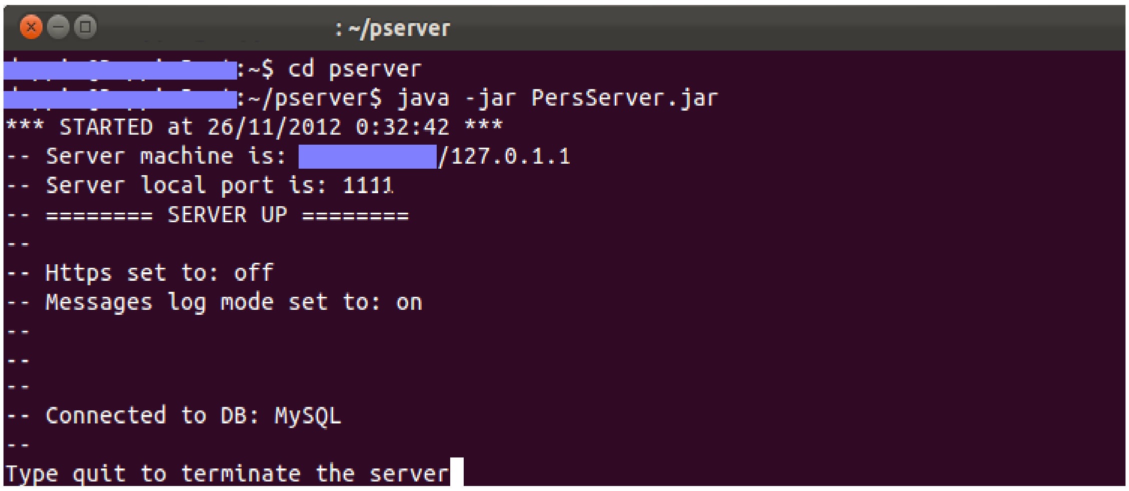
This informs the PServer to load objects of different classes (pbeans), which pserver will put into hash maps, and it can share their functionality through Java reflection. We will explain how to write pbeans and use them later. For now just remember that any plug-in that you want to insert and use must be defined in this file by a unique name. The definition is done by typing a name, the operator of equality, and a class name. The parenthesis after the class name defines the pbean initialization parameters. The preinstalled pbean admin contains the functionality for the administration interface of PServer that we have shown before. It takes three parameters: first, the name that is defined into pbeans.ini, the login name (the first root), and the login password that the user must type to make a successful log in. When you want to use the administration tool, you have to provide these two words to gain access. If you want to change these words, just make the changes here, save the file, and make a restart of the PServer.

3.3) Run PServer

You have already downloaded the latest PServer file. To run Pserver, open the terminal, and go inside the pserver folder, and run the PersServer.jar.
For example, in linux you must type these two commands shown below:

1) **cd** <*the path of pserver folder*>**pserver**
2) **java -jar PersServer.jar**

Now, PServer runs, and is ready for your requests.



```
:~/pserver
[redacted]:~$ cd pserver
[redacted]:~/pserver$ java -jar PersServer.jar
*** STARTED at 26/11/2012 0:32:42 ***
-- Server machine is: [redacted]/127.0.1.1
-- Server local port is: 1111
-- ======== SERVER UP ========
--
-- Https set to: off
-- Messages log mode set to: on
--
--
--
-- Connected to DB: MySQL
--
Type quit to terminate the server
```

Hint: if you are a developer and you want to modify the code, follow the steps below

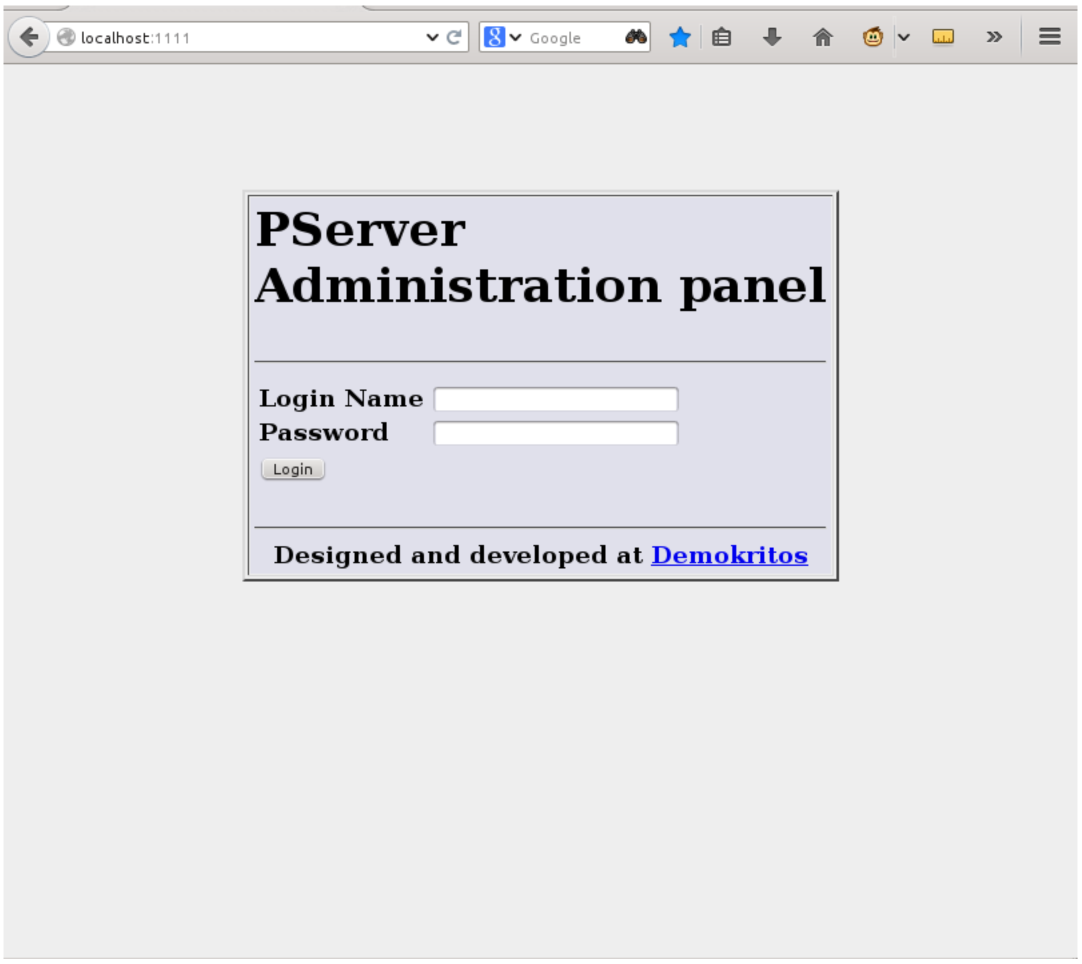
1. Download the project from https://github.com/iit-Demokritos/PServer
2. Compile the project
3. Take the new PersServer.jar, and replace it inside the pserver folder that you have downloaded before.
4. Open the terminal, and go inside the pserver folder, and run the PersServer.jar.

3.4) Using the Administration Panel

From now on, with administration panel help you can create or delete clients, change many other settings of Pserver, and use the online help for the requests that we see below.

3.4.1) Logging in

- Open a browser, and type http://localhost:1111
- You should see the PServer Administration Panel

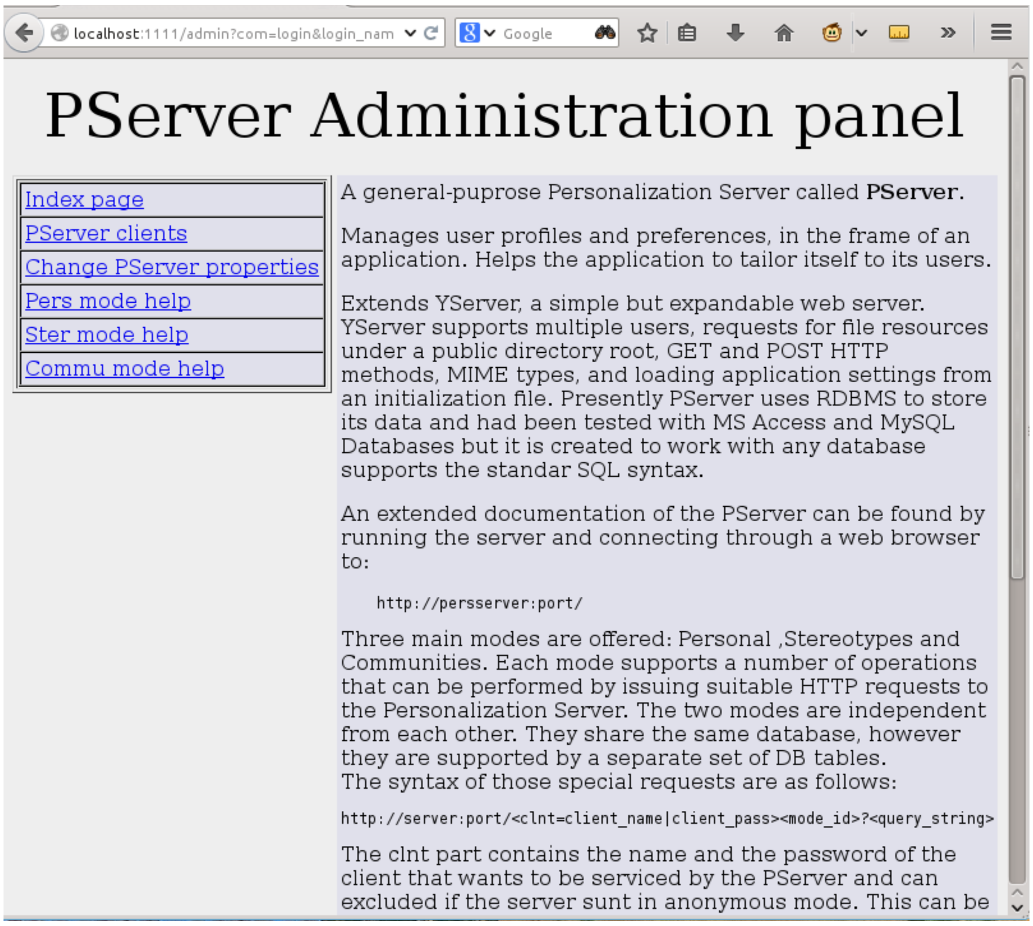

localhost:1111

PServer
Administration panel

Login Name
Password
Login

Designed and developed at Demokritos

Now, log in with your username and password you set in the pbeans.ini file.



PServer Administration panel

- Index page
- PServer clients
- Change PServer properties
- Pers mode help
- Ster mode help
- Commu mode help

A general-puprose Personalization Server called **PServer**.

Manages user profiles and preferences, in the frame of an application. Helps the application to tailor itself to its users.

Extends YServer, a simple but expandable web server. YServer supports multiple users, requests for file resources under a public directory root, GET and POST HTTP methods, MIME types, and loading application settings from an initialization file. Presently PServer uses RDBMS to store its data and had been tested with MS Access and MySQL Databases but it is created to work with any database supports the standar SQL syntax.

An extended documentation of the PServer can be found by running the server and connecting through a web browser to:

```
    http://persserver:port/
```

Three main modes are offered: Personal ,Stereotypes and Communities. Each mode supports a number of operations that can be performed by issuing suitable HTTP requests to the Personalization Server. The two modes are independent from each other. They share the same database, however they are supported by a separate set of DB tables.
The syntax of those special requests are as follows:

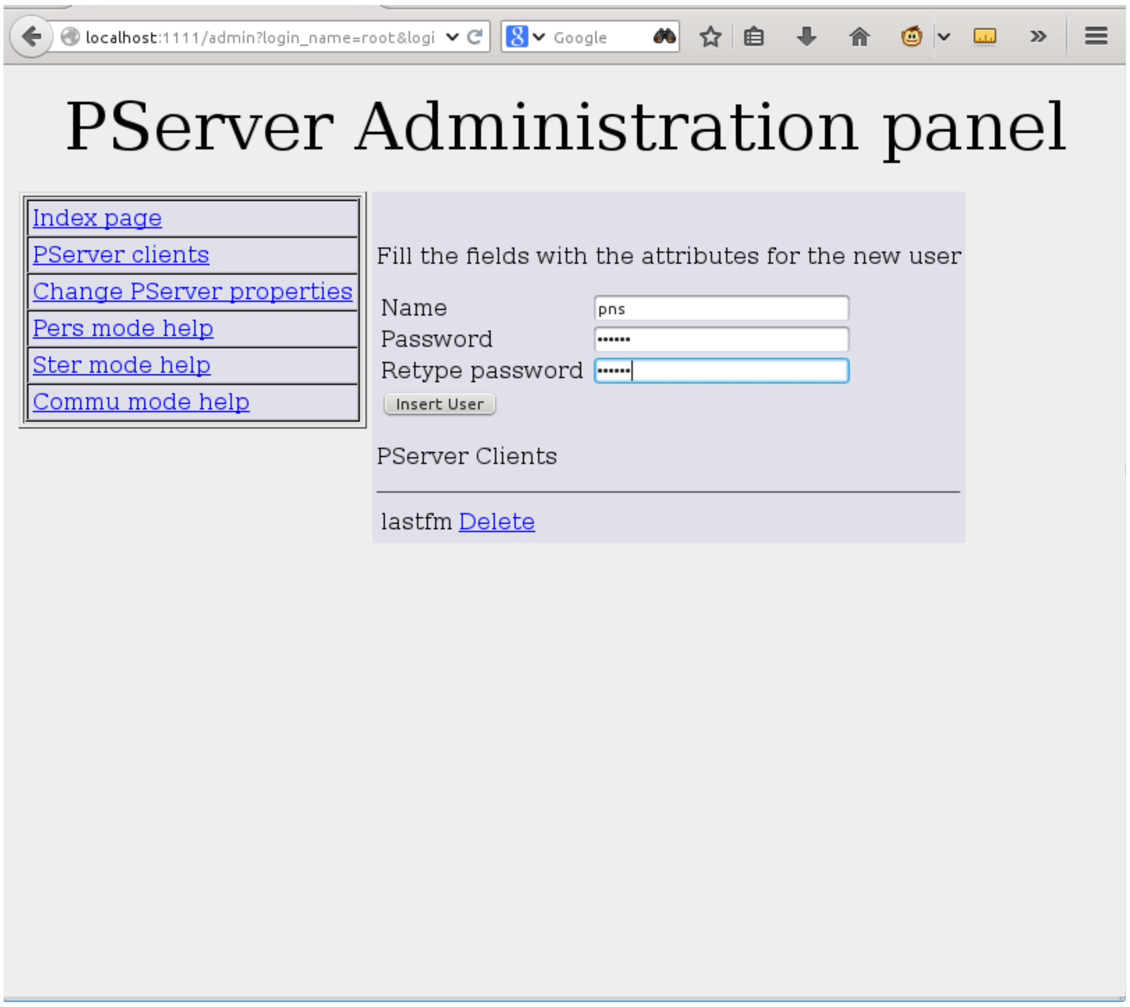
```
http://server:port/<clnt=client_name|client_pass><mode_id>?<query_string>
```

The clnt part contains the name and the password of the client that wants to be serviced by the PServer and can excluded if the server sunt in anonymous mode. This can be

e.g.: The homepage of the PServer administration panel

3.4.2) Creating Pserver Clients

As we mentioned before, among PServer's benefits is that we can connect more than one applications or sites on PServer. In order to distinguish them, we create one unique client with username and pass, so as to direct properly the information to PServer. To create a new client, you should go to «*PServer clients*» page, type in your client username and password, and click on the *insert user* button.



localhost:1111/admin?login_name=root&logi

PServer Administration panel

- Index page
- PServer clients
- Change PServer properties
- Pers mode help
- Ster mode help
- Commu mode help

Fill the fields with the attributes for the new user

Name	pns
Password	••••••
Retype password	••••••

Insert User

PServer Clients

lastfm Delete

4)System specifications according to user needs

This chapter describes the required steps that an administrator needs to follow in order to integrate PServer into his site - application.

4.1) Define the User Model

In order to define the User Model, an admin must decide which PServer personalization features he wants to integrate into his site or application, according to his special needs and requirements. The existing PServer features are already described in chapter 1.2. For example, the admin must decide if he wants data about Personal User Models, Stereotypes, User Communities, or all of the above.

4.2) Define the data to be stored

At this step, the Admin must specifically determine the data that needs to be stored from the PServer, in order to fulfill the User Model requirements which are described in the above step. For example, if the admin wants to use only the Personal user model, he is not obliged to store their specific User Attributes and characteristics.

4.3) Data Feeding to PServer

There are two proposed ways of adding data to PServer. Real Time Data feeding, or periodical data feeding. Here, an admin should decide whether he would like to feed real time data to PServer, or whether he would like to feed data periodically any time he wants.
For example, if a given site has too much traffic, the recommended way is to save the data to a log file and periodically feed PServer with this file, in low traffic periods.

4.4) Recommendation Engine

At this last step, an engine should be developed in order to accept the PServer responses, and after proper processing - according to the admins requirements- to present the Personalised results to the end-user. In the next chapter we show how to communicate with Pserver, and handle the responses.

4.5) Setting up PServer on a pre-existing system

In order to acquire PServer’s full functionality in a predefined system, the admin should follow all of the four above steps, and additionally, he should develop a wrapper, which would acquire data from the existing DB, according to the User Model requirements (Steps 1 and 2). The wrapper takes the data, and feeds PServer accordingly.

5) Make PServer requests

PServer is an HTTP application server that runs plug-ins (pservlet pbeans), serves the requests, and returns responses. Based on this logic, the requests that you make to PServer have specific structure.

http://url/<pbean_name>?clnt=name|pass&com=some_com&param1=val1&param2=val2...

First, you define the URL that links to PServer, then the pservlet that you want to use, and then the client name/pass that makes the request and the pservlet parameters. Every pservlet defines its own parameters. The pservlets that we have written have a common structure; they need to get a com parameter, which defines the pserver command that needs to be called, and then defines the parameters of the specific command that has been specified by the com parameter. The pservlets can return any type of document (HTML,XML,JSON etc), but the ones that we have implemented and we use return XML. The admin pservlet is an exception; it returns HTML documents, and this is needed to access it through a web browser. The XML that is returned is always structured like a 2d array. You can see the structure by reading the xsl files that we provide with Pserver, and exist in the resp_xsl folder inside the public folder. For example, we are printing one of these:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
    <head>
      <title>View from Table: up_features</title>
    </head>
    <body>
      <br></br>
      <h2>features and default values</h2>
      <p></p>
      <b>Tables:</b> up_features
      <br></br>
      <b>Description:</b> A_selection_of_(feature,_default value)\-pairs.
      <p></p>
      <table border="1" cellpadding="4">
        <xsl:for-each select="result/row">
          <tr>
            <th>
              <xsl:value-of select="ftr"/>
            </th>
            <td>
              <xsl:value-of select="defval"/>
            </td>
          </tr>
        </xsl:for-each>
      </table>
      <br></br>
      <a href="/">Back to home</a>
      <p></p>
      </body>
   </html>
</xsl:template>
```

These xsl files are used to provide formatted output to web browsers. Type commands into a web browser (like Firefox), and you will get results that show the 2d arrays philosophy, like above, for example.
Here is the XML document from the execution of a command:
*http://127.0.0.1:1111/pers?clnt=test1|test1&com=getusrs&whr=**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="/resp_xsl/user.xsl"?>
<result>
<row><usr>1</usr></row>
<row><usr>10</usr></row>
<row><usr>100</usr></row>
….
<row><usr>194</usr></row>
<row><usr>195</usr></row>
<row><usr>196</usr></row>
<row><usr>197</usr></row>
<row><usr>198</usr></row>
<row><usr>199</usr></row>
<row><usr>2</usr></row>
</result>
```

Inside the public folder of PServer there are subdirectories that contain help about every implemented pservlet and its API.

There is full documentation about the methods that PSClientRequest provides in the Java docs that are bundled with the Pserver.

6) Enter data into PServer

You have two ways to enter information into the P-server:
a) the web browser
b) through a program with http requests.
In any case, it is important to remember that you do not enter data directly into the database, but rather you must do it through a client application to the PServer.

http://localhost:1111/pers?clnt=testclient|testpass&com=addattr&job=1

'pers'=personal mode (i.e. not stereotypes, or communities. In other words individual users).
'testclient'=the client
'testpass'=password to the client
'\&' separator
'com=addattr' = command add attribute
'job=1', the hypotherical attribute job assumes the value 1

Provided you have created the database schema with 'pserver.sql', you should be able to see the results in the 'attributes' and user 'attributes' tables.

6.1) Attributes and Features

6.1.1) Inserting Attributes

Add attributes:
pers?clnt=testclient|testpass&com=addattr&age=null&sex=null&occupation=null
This adds age, sex, occupation and their default values (null, null, null) as attributes in 'attributes' table.

Add attribute values along with user:
pers?clnt=testclient|
testpass&com=setusr&usr=user1&attr_age=12&attr_sex=male&attr_occupation=journalist
This adds the user1 to table 'user' and the attributes in 'user attributes' table.

Remove attributes:
pers?clnt=testclient|testpass&com=remattr&attr=sex&attr=age
pers?clnt=testclient|testpass&com=remattr&attr=* (deletes all attributes and user attributes)

6.1.2) Inserting Features

Add features:
pers?clnt=testclient|testpass&com=addftr&movieID=0
This adds movieId and default value 0 as feature in ‘up_features table’.

Add feature values to a user:
pers?clnt=testclient|testpass&com=setusr&usr=user1&ftr_movieID=12
This adds to user1, the feature movieID with value 0 in table 'user_attributes'.

Hint: Be careful with the setusr *command: It is the same command as the (add attributes command* setusr, see 6.1.1)*, with the only difference that the prefix to each attribute inserted is changed.*
*e.g.:***attr_**age → mention to attribute age
ftr_age → mention to feature age

Remove features:
pers?clnt=testclient|testpass&com=remftr&ftr=&ftr=movieID
pers?clnt=testclient|testpass&com=remftr&ftr=* (deletes all up_features and user features)

Get default values for features and attributes:
pers?clnt=testclient|testpass&com=getattrdef&attr=*

Increase the value of a feature for a user:
pers?clnt=testclient|testpass&com=incval&usr=user1&movieID=1

6.2) Stereotypes

Creating Stereotypes:
ster?clnt=testclient|testpass&com=addstr&str=str18_25GR10male&rule=age>18|and|country:"Greece"|and|sex:"male"
(create a stereotype with attributes: age>18, country: Greece. sex: male)

Adding users to a stereotype:
ster?clnt=testclient|testpass&com=addusr&usr=user1&str18_25GR10male=1
ster?clnt=testclient|testpass&com=addusr&usr=kostas&visitor=0.78&expert=0.9

(add the kostas to stereotype visitor with degree 0.78 and expert with degree 0.9)

Increase the value of a feature for a stereotype:
ster?clnt=testclient|testpass&com=incftr&str=testStr.1&ftr.1=15&ftr.test.5=0.5

Remove a stereotype:
ster?clnt=testclient|testpass&com=remstr&str=str18_25GR10male
ster?clnt=testclient|testpass&com=remstr&str=test.*

List all stereotypes:
ster?clnt=testclient|testpass&com=liststr&str=*

Hint: Pay attention to the form of each attribute, so that it can be read correctly.

6.3) Communities

In order to create user communities, first you have to create the user distances using one of the two metrics provided below.

Using metric cos *(Cosine Vector Metric)*
commu?clnt=testclient|testpass&com=calcuassoc&algorithm=cos

Using metric ps *(Pearson Correlation Metric)*
commu?clnt=testclient|testpass&com=calcuassoc&algorithm=ps

To make communities you must choose an algorithm and type association
commu?clnt=testcl|pass&com=makecommunities&algorithm=metis&association=cos

6.4) Feature Groups

In order to create feature groups, previously you must have created the feature distances using one of the two metrics.

Using metric cos *(Cosine Vector Metric)*
commu?clnt=testclient|testpass&com=calcuftrassoc&algorithm=cos

Using metric ps *(Pearson Correlation Metric)*
commu?clnt=testclient|testpass&com=calcuftrassoc&algorithm=cos

To make feature groups you must choose an algorithm and type association
commu?clnt=testcl|pass&com=makeftrgroups&algorithm=metis&association=cos

6.5) Deleting a PServer client

To delete a client such as the testclient along with the relevant data, you must perform the following:

1. Run Pserver
2. Login in http://localhost:1111
3. You should see the PServer Administration Panel
4. Go to PServer clients.
5. Click on Delete button next to the client you are interested in.

7) About Models - Detailed

7.1) PersonalMode

In Personal Mode, the server functions as a repository oriented to store user profiles. The profile of a user is a set of tuples (feature, value). Features are entities relevant to specific applications, while values give an estimation about a user relevance to corresponding features. All users have values for all application features. Features may have default values that are assigned to new users. Also, features can be organized in a tree or graph based manner, in order to easily manage conceptual hierarchies. This organization is encoded in the name of every feature as a path expression, and is set up by applications. The DB structure: up_features (uf_feature, uf_defvalue, uf_numdefvalue) with key 'uf_feature', user_profiles (up_user, up_feature, up_value, up_numvalue) with key 'up_user' and 'up_feature'. If a field in 'up_features' is deleted, the deletion is cascaded to 'user_profiles' because of a referential integrity constraint. The two fields 'uf_numdefvalue' and 'up_numvalue' are "invisible": they are not part of the results of 'select' queries, they contain the numeric equivalent of the string value (in

the other value fields), or NULL if the string cannot be converted to numeric. Those duplicate fields are used mainly to allow two types of value comparisons: string and numeric. Note that the primary data type for values is always string, as it is more general, and that the numeric version always corresponds to the string version. Also, note that values intended to be numeric must use '.' for the decimal part when given as strings. If ',' is used, the string will not be successfully converted to numeric, and its numeric equivalent will be NULL.
Personal Mode also offers a separate 'DECAY' functionality. In decay, the server keeps a record of every user interaction with certain features, marking the date/time the interaction occurred. Then, it is possible to calculate a value that shows how much a user is interested in a feature, and that takes into account not only how many times the user has visited the feature, but also if the user has visited other features in the meanwhile (therefore, if the feature has been 'forgotten'). Most recently visited features receive higher scores, therefore, to forget means to lose interest. The formula depends on a variable called here 'decay rate' between [0,1] inclusive, which determines the rate of forgetting. If the rate is set to 0, the user does not forget (or lose interest), and the decay mechanism is reduced to sorting features based only on how frequently a user has visited them (not when). The application can define a number of feature groups and a rate for each group. Each group represents a set of features that compete for the user's interest under the decay formula. The application must inform the server about any user interaction with such features, and can subsequently ask for features a user is most interested in. In case the rate of a feature group is 0, the decay value calculated by the server for any feature of the group for a specified user is simply the total number of visits the user has paid to the feature.

The DB structure: decay_groups (dg_group, dg_rate) with key dg_group, decay_data (dd_user, dd_feature, dd_timestamp) with key dd_user, dd_feature and dd_timestamp. If a field in 'up_features' is deleted (or a feature name updated), the deletion (or update) is cascaded to 'decay_data' because of a referential integrity constraint. Note that 'decay_groups' is not connected with any other table through referential integrity constraints, so data from this table must be deleted explicitly when initializing the Personal Mode database. The role of 'decay groups' is secondary - actually the decay can function without this table, and the application is not obliged to declare groups by using decay. It only needs to notify PServer about user-feature interactions. Another function of the Personal Mode has to do with "numeric features". For some features it is not meaningful to count how often a user has visited them in order to determine of how much interest they are to the user. Such features have numeric values, and meaningful operations are aggregates of those values. For example, a user interested in laptop computers can visit laptops of different weight. In this case, a number relevant to the profile of the user may be the average weight of all laptop descriptions the user has visited (or showed interest in). A single table in the DB supports this functionality: num_data (nd_user, nd_feature, nd_timestamp, nd_numvalue) with key nd_user, nd_feature, nd_timestamp. The table 'num_data' is not connected with any other table through referential integrity constraints (not even with table 'up_features'), so data from this table must be deleted explicitly when initializing the Personal Mode database.

7.2) StereotypeMode

In Stereotype Mode, the server offers support for stereotypes. Stereotypes are categories of users with specific characteristics. Each stereotype has a profile that is defined by means of (feature, value) tuples. Features can be entities relevant to specific applications, while values give an estimation about the stereotype relevance to corresponding features. Each stereotype may have its own different features. Features can be organized in a tree or graph based manner, in order to easily manage conceptual hierarchies. This organization is encoded in the name of every feature as a path expression, and is set up by applications. Users can be assigned stereotypes, together with a degree of relevance, showing how relevant a stereotype is to a user. A user may be assigned several stereotypes (not the same twice).

The DB structure: stereotypes (st_stereotype) with key 'st_stereotype', stereotype_profiles (sp_stereotype, sp_feature, sp_value, sp_numvalue) with key 'sp_stereotype' and 'sp_feature', stereotype_users (su_user, su_stereotype, su_degree) with key 'su_user' and 'su_stereotype'. If a field in 'stereotypes' is deleted, the deletion is cascaded to 'stereotype_profiles' and 'stereotype_users' because of referential integrity constraints. The field 'sp_numvalue' is "invisible": it is not part of the results of 'select' queries, and contains the numeric equivalent of the string value in field 'sp_value', or NULL if the string cannot be converted to numeric. This duplicate field is used mainly to allow for two types of value comparisons: string and numeric. Note that the primary data type for values is always string, as it is more general, and that the numeric version always corresponds to the string version.

Also note that values intended to be numeric must use '.' for the decimal part when given as strings. If ',' is used, the string will not be successfully converted to numeric, and its numeric equivalent will be NULL. The field 'su_degree' is numeric (double), and when its values are exchanged as strings they follow the rules described above. This field also contains NULLs for values that could not be converted to numeric.

7.3) Community Mode

In Community Mode, the server provides functions to create, store and retrieve groups/communities of Users or Items(features).

With this mode, PServer makes possible the extraction of knowledge about the interests of each User based on the interests of the other Users of the system (Collaborative recommendations). So we aim at the production of communities of users who are 'close' based on some metric. Actually User communities are an abstraction of groups of Users that are strongly connected somehow (have common interest, are friends, etc)

To achive this we follow a two step processes.

1. Calculating or Feeding the user associations.
   - In this step the client can explicitly declare that two users are associated (e.g two users are friends in a social network)
   - Or the client can use the methods provided by PServer (through the API) which compare all the user profiles and store the association level(weight, similarity of profiles) between every pair of Users.

2. Now that we have the information of who is connected (has something in common) with whom, we can start producing Communities of strongly connected users.

This is made by applying one of the available clustering or partitioning algorithms, that are included in PServer, on the User graph.

The same process is also followed for producing Feature Groups.

8) Implement a PServlet

To Implement your own pservlet, you must create a Java class implement, the pserver.pservlets.PService interface. This interface has the following methods.

```
   public abstract String getMimeType();
```

This method returs the MIME type of the document that your pservlet returns. For example, text/xml if you want to return XML documents.

```
   public abstract void init( String[] params ) throws Exception;
```

This method will be called when the PServlet will be loaded.

```
        public abstract int service( VectorMap parameters, StringBuffer response,
DBAccess dbAccess );
```

The method contains the functionality that the pservlet provides. It gets the request parameters into a VevtorMap, the response object will contain the returned document after the execution of the pservlet, and dbAccess is an object that provides access to PServer database.

A typical pservlet implementation will look like this.

```
public class SomeServlet implements pserver.pservlets.PService {
public void init( String[] params ) throws Exception {
   }

   public String getMimeType() {
      return pserver.pservlets.PService.xml;
   }

      public int service( VectorMap parameters, StringBuffer response, DBAccess
dbAccess ) {
      int respCode;
      VectorMap queryParam;
      StringBuffer respBody;

      respBody = new StringBuffer();
      queryParam = parameters;

      //removes the password from the clnt parameter, the validation has been done by
PServer before this statement
      int clntIdx = queryParam.qpIndexOfKeyNoCase( "clnt" );
      String clientName = (String) queryParam.getVal( clntIdx );
      clientName = clientName.substring( 0, clientName.indexOf( '|' ) );
      queryParam.updateVal( clientName, clntIdx );

      int comIdx = parameters.qpIndexOfKeyNoCase( "com" );
      if ( comIdx == -1 ) {
         respCode = PSReqWorker.REQUEST_ERR;
         WebServer.win.log.error( "-Request command does not exist" );
         return respCode;  //no point in proceeding
      }

      //recognize command encoded in request
      String com = (String) queryParam.getVal( comIdx );
      if ( com.equalsIgnoreCase( "someCommand" ) ) {//calculetes user distances
         respCode = comSomeCommand( queryParam, respBody, dbAccess );
      } else if ( com.equalsIgnoreCase( "calcftrdist" ) ) {//calculetes feature distances
          else {
         respCode = PSReqWorker.REQUEST_ERR;
         WebServer.win.log.error( "-Request command not recognized" );
      }

      response.append( respBody.toString() );
      return respCode;
   }

….
```

9) FAQ

9.1) Questions and Answers

Q: What happens if I have many users and many features, but only a few features are relevant to each user? That is, *relevant* << *o*(*users* × *features*)?
A: Nothing, actually, the table 'user profiles' will contain users×features records.

For more questions and answers visit PServers FAQ site or PServer's Forum

9.2) Troubleshooting

Problem 1 *I run correctly the PServer but when i type http://localhost:1111, I cannot see the* Administration Panel.
Solution 1 If you can't see the PServer Administration Panel, check the terminal on which PServer runs, and see what port it has taken. If the port is -1, it seems that another application is listening to 1111, so you must change the PServer's port.

Problem 2 *I type http://localhost:1111, then I provide the correct user name and password, and i cannot log in.*
Solution 2 *in the file pbeans.ini the admin=pserver.pservlets.Admin(admin,<your username>,<your password>)' should either be empty or have different content.*

Problem 3 *I got the following error message in mozilla firefox Error loading stylesheet: Parsing an XSLT stylesheet failed.*
Solution 3 *Try Explorer. Due to an unknown reason, it is assumed that you have accessed the 'pserver' database.*