



DEVELOPER MANUAL
MIGRATING DESKTOP AND ROAMING ACCESS SERVER

WP3.1

Document Filename:	CG3.1-v1.0-PSNC-MD_RASDeveloperManual.doc
Work package:	WP3.1 MD and RAS
Partner(s):	PSNC
Lead Partner:	PSNC
Config ID:	CG3.1-v1.0-PSNC-MD_RASDeveloperManual
Document classification:	PUBLIC

Document Log

Version	Date	Summary of changes	Author
0.1	20/12/2004	First draft version	Marcin Płóciennik
0.2	22/12/2004	Description of plug-ins added	Bartek Palak
1.0	15/01/2004	Adapting document to crossgrid template, with related sections	Marcin Płóciennik
	26/01/2005	Verified by the QE	Robert Pajak

CONTENTS

1	COPYRIGHT NOTICE.....	5
2	INTRODUCTION.....	6
2.1	OBJECTIVES OF THIS DOCUMENT	6
2.2	APPLICATION AREA.....	6
2.3	APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS	6
2.4	ABBREVIATIONS AND ACCRONYMS	6
2.5	INSTALLATION AND SOURCE CODE	7
3	IMPLEMENTATION STRUCTURE	8
3.1	PRODUCT USE CASES	8
3.1.1	<i>Interactivity based on Java plug-in use case.....</i>	8
3.1.2	<i>Running interactive Legacy application use case.....</i>	9
3.2	PRODUCT COMPONENT MODEL	9
3.3	DETAILED IMPLEMENTATION MODEL	10
4	PRODUCT INTERFACES	13
4.1	MIGRATING DESKTOP – JAVA APPLICATION PROGRAM INTERFACE.....	13
4.1.1	<i>Tool Plug-in</i>	<i>13</i>
4.1.1.1	Description	13
4.1.1.2	Tool Plug-in API	15
4.1.1.3	Interaction between Plug-in and Container.....	18
4.1.1.4	Passing parameters to plug-in.....	19
4.1.1.5	Dynamic loading of Tool Plug-IN classes	21
4.1.1.6	XML description of plug-in.....	21
4.1.1.7	security	21
4.1.1.8	Jar signing	21
4.1.1.9	Additional remarks	22
4.1.2	<i>JS Plug-in.....</i>	<i>22</i>
4.1.2.1	Description of application plug-in.....	23
4.1.2.2	Procedure of preparation and execution of the JSS Plug-in.....	24
4.2	ROAMING ACCESS SERVER - WEB SERVICES INTERFACE	30
4.2.1	<i>Server Login Web Service</i>	<i>30</i>
4.2.1.1	String getServerAddressForVO(String hostname, byte[] input).....	30
4.2.1.2	Vector getServersAddressForVO(java.lang.String hostname, byte[] input).....	30
4.2.1.3	boolean exist()	31
4.2.2	<i>File Tunnel Web Service</i>	<i>31</i>
4.2.2.1	open	31
4.2.2.2	close.....	32
4.2.2.3	exists.....	32
4.2.2.4	createNewFile.....	33
4.2.2.5	createNewDir.....	34
4.2.2.6	rename	34
4.2.2.7	deleteDir.....	35
4.2.2.8	deleteFile	35
4.2.2.9	listDir.....	36
4.2.2.10	goUpDir.....	37
4.2.2.11	getRootCurrDirPath.....	37
4.2.2.12	thirdPartyTransfer.....	38
4.2.2.13	getFileInformation	39
4.2.3	<i>Auxiliary Web Service</i>	<i>39</i>
4.2.3.1	getTrustedCertificates.....	39
4.2.3.2	getSecurityInfo	40
4.2.3.3	addSecurityInfo	40
4.2.3.4	removeSecurityInfo	41
4.2.3.5	getAllFileViewerXML	42

4.2.3.6	getFileViewerXML	42
4.2.3.7	addFileViewerXML	43
4.2.3.8	updateFileViewerXML	43
4.2.3.9	removeFileViewerXML	44
4.2.4	<i>Application Information Web Service</i>	44
4.2.4.1	addNewApplicationInfo	44
4.2.4.2	deleteApplicationInfo	45
4.2.4.3	getAllApplicationInfo	45
4.2.4.4	getApplicationInfo	45
4.2.4.5	getApplicationInfoById	46
4.2.5	<i>Virtual Directory Web Service</i>	46
4.2.5.1	getRoots	47
4.2.5.2	getMetaElem	47
4.2.5.3	getNodeElemPath	47
4.2.5.4	getPathAsNodeElems	48
4.2.5.5	addFileLogical	48
4.2.5.6	changeUserFileName	49
4.2.5.7	removeUserFile	50
4.2.5.8	removeUserFileLocation	50
4.2.5.9	makeUserFileLocation	50
4.2.5.10	getUserFileLocation	51
4.2.5.11	getUserFileLocations	51
4.2.5.12	confirmUserFileUpload	52
4.2.5.13	updateFileInfo	52
4.2.5.14	replicateFile	53
4.2.5.15	addAlias	53
4.2.5.16	registerFiles	54
4.2.5.17	registerDirectory	54
4.2.5.18	addUserDirectory	54
4.2.5.19	changeUserDirectoryName	55
4.2.5.20	removeUserDirectory	55
4.2.5.21	getDirectoryList	56
4.2.5.22	listAvailableSE	56
5	PRODUCT TESTING	59
6	CONTACT INFORMATION AND CREDITS	60
7	EDG LICENSE AGREEMENT	61

1 COPYRIGHT NOTICE

Copyright (c) 2005 by *CrossGrid Consortium*. All rights reserved.

Use of this product is subject to the terms and licenses stated in the Crossgrid license agreement. Please refer to Chapter 7 for details.

JAVA is a registered trademark of *SUN Microsystems Inc.* All rights reserved.

This research is partly funded by the European Commission IST-2001-32243 Project “CrossGrid”.

2 INTRODUCTION

2.1 OBJECTIVES OF THIS DOCUMENT

This document is the Java and Web Services developer guide for the CG Migrating Desktop and Roaming Access Server. The usage of the Migrating Desktop and Roaming Access Server is described in the user guide.

2.2 APPLICATION AREA

WP3.1 Migrating Desktop and Roaming Access Server

The Roaming Access Server (RAS) offers a well-defined set of web-services that can be used as an interface for accessing Grid systems and services in a common, standardized way. It consists of several independent parts responsible for job submission, job monitoring, user profile management, data management, authorisation, and application information management. RAS may support wide variety of clients including personal computers, laptops, and in the future PDA, and mobile phones. It is specifically targeted for supporting mobile grid users.

Migrating Desktop is an advanced user-friendly graphical shell that serves as uniform grid working environment independent on specific grid infrastructure. It provides a Java based GUI designed especially for mobile users and is independent of OS platform (MS Windows, Linux, Solaris) and hardware (personal computers, laptops, workstations). MD is a complex environment that integrates many tools and allows working with many grids transparently and simultaneously. It exploits capabilities provided as Web Services from RAS and JSS (Job Submission Service).

2.3 APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

Applicable documents

[A1] Architecture and Design Document.

[A4] CrossGrid Architecture

[A5] User Guide

[A6] Install Guide

[A7] Test Procedures Description

Reference documents

2.4 ABBREVIATIONS AND ACCRONYMS

GUI:	Graphical User Interface
MPI:	Message Passing Interface
MD:	Migrating Desktop
RAS:	Roaming Access Server
JDL:	Job Description Language
EDG:	EU-DataGrid Project
WN:	Worker Node

CG:	CrossGrid
WM:	Workload Manager
VNC:	Virtual Network Computing
LDAP:	Lightweight Directory Access Protocol.
WP:	Work-package
URL:	Uniform Resource Locators
FTP:	File Transfer Protocol
GridFTP:	Grid File Transfer Protocol
XML:	Extended Markup Language
TLS:	Transport Layer Security
SSL:	Secure Socket Layer
LCG:	LHC Computing Grid
VO:	Virtual Organization
RM:	Replica Manager
L&B:	Logging and Bookkeeping

2.5 INSTALLATION AND SOURCE CODE

The source code of this application and how to install it is fully described in the Installation Guide of the application.

The source code can also be browsed at the following address:

http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_1-portals/src/

3 IMPLEMENTATION STRUCTURE

3.1 PRODUCT USE CASES

Migrating Desktop and Roaming Access Server are part of complex system, therefore it have lot of possible use cases. As CrossGrid project focused on interactive application we present the most advanced cases that is interactivity schemas. Other less advanced ones are described in next sections.

3.1.1 Interactivity based on Java plug-in use case.

A sample use case of such scenario is presented in Fig. 1 and includes the following actions:

- 1) The user will prepare the interactive job submission using the Application Wizard in MD. It concerns choosing the appropriate application-specific parameter input and output files and needed resources.
- 2) The job is submitted to the RAS server (using the above data). The appropriate JDL file is being created basing on input parameters and restrictions.
- 3) RAS receives the request and starts the Job Shadow locally, adding the info on Job Shadow port and name of the pipes to JDL
- 4) The user submission request is sent to the EDG 2 WM Server component.
- 5) The EDG 2 WM Server finds an available CE that matches the requirements (including the capability to run interactive jobs) and submits the job there to be run on one of CE's WN.
- 6) The Cushion Process on WN and the Job Shadow start to exchange i/o messages until the job stops running.
- 7) MD starts the application-related Java Visualization client, which is in touch with the Interactive Job Channels Forwarding Service on the RAS machine, which will be in charge of reading/writing the content of the relevant Job Shadow pipes.
- 8) In case of any problem MD takes care of killing the launched process.

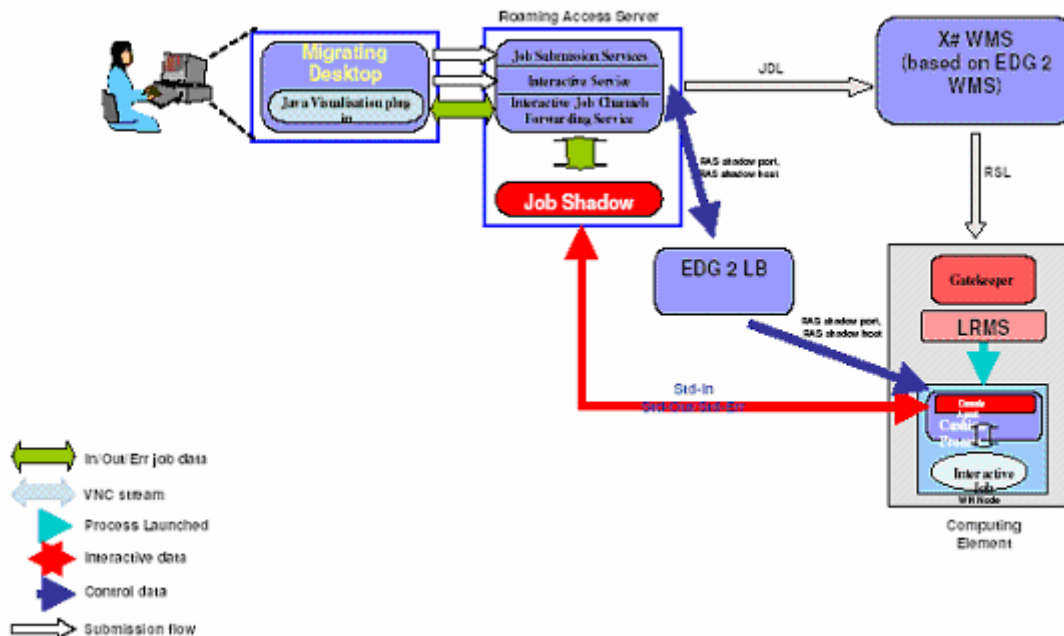


Fig. 1 Interactivity "Java plug-in based" schema.

3.1.2 Running interactive Legacy application use case.

A sample use case of such scenario is presented in Fig. 2 and includes following the actions:

- 1) The user will prepare the interactive job submission using the Application Wizard in MD. It concerns choosing appropriate application-specific parameter input and output files and needed resources.
- 2) After submitting the job by the user, a piece of Java code inside MD will launch a script on the Application Client Machine, which will start the Job Shadow, a legacy client application and the VNC server, as a result sending back the id's of the launched processes, the port where of the relevant Job Shadow pipes.
- 3) The job is submitted to the Roaming Access Server. The appropriate JDL file is being created basing on input parameters and restrictions.
- 4) The user submission request is sent to the EDG 2 WM Server component.
- 5) The EDG 2 WM Server finds an available CE that matches the requirements (including the capability to run interactive jobs) and submits the job there to be run on one of CE's WN.
- 6) The Cushion Process on WN and the Job Shadow start to exchange i/o messages until the job stops running.
- 7) The user using VNC from MD can see the legacy client application GUI and interact with the running job.
- 8) In case of any problem MD takes care of killing the launched process

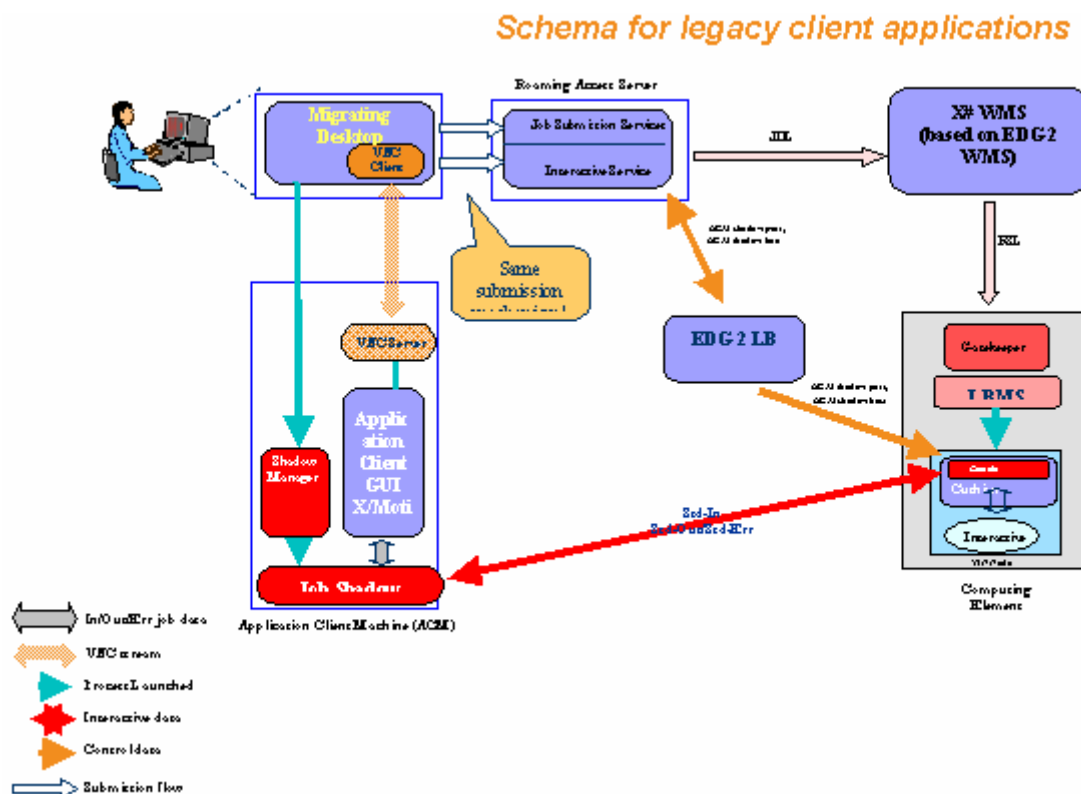


Fig. 2 Interactivity “legacy application based” schema

3.2 PRODUCT COMPONENT MODEL

The Migrating Desktop framework allows the user to access the Grid resources, run interactive applications, monitoring and visualization, and manage data files. MD provides a front-end framework for embedding some of the application mechanisms and interfaces, and allows the user to have virtual access to Grid resources from other computational nodes.

Roaming Access Server is a set of web services that mainly provide grid functionality. It is the integration level for interfaces of different middleware. It provides services like job submission service, job monitoring service, interactive session manager and interactive job channels forwarding service.

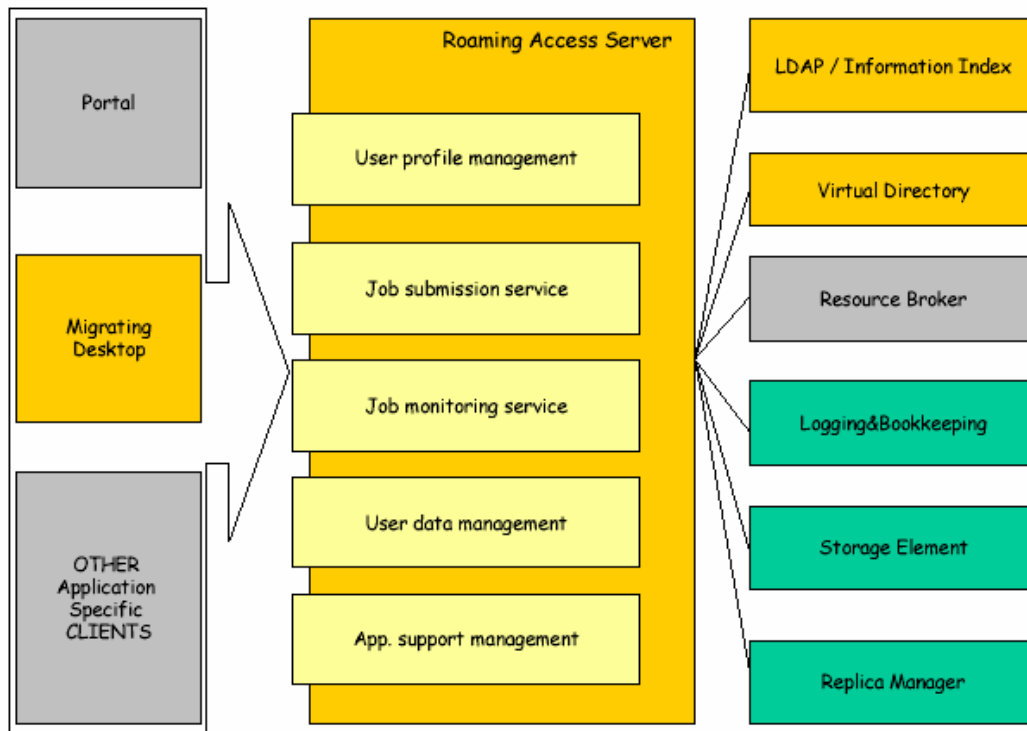


Fig. 3 RAS/MD components

3.3 DETAILED IMPLEMENTATION MODEL

The Roaming Access Server [4] offers a well-defined set of web-services that can be used as an interface for accessing HPC systems and services (based on various technologies) in a common, standardised way. All communication bases on web services technology. This way it may support wide variety of client including personal computers, laptops, and in the future PDA, and mobile phones.

The Roaming Access Server is a set of modules and plug-ins that provides interfaces to work with grids. It consists of several independent parts responsible for job submission, job monitoring, user profile management, data management, authorisation, and application information management. Provided functionality contains a wide range of different grid services that are common for many various grid projects.

Job submission

One of the most important and common functionality in a grid environment is a job submission. RAS job submission interface gives a uniform access to different resource brokers. Clients can submit all jobs with this interface and then appropriate plug-ins are called to convert job description to the specific language. Currently we are using plug-in for CrossGrid/DataGrid resource broker to convert job description to JDL. In the future Job submission interface will support XML job description, which is under development by Job Submission Language group from Global Grid Forum.

Job monitoring

Job monitoring is an interface that allows retrieving job status information. Job monitoring module uses grid plug-ins to retrieve job status information in grid dependent way and presents uniform status information for all jobs.

User profile management

User profile management provides functionality that allows operations on user profiles. User profiles contain all information that define current user working environment including information about graphical configuration (e.g. desktop background, icon locations, and colours) and information needed to access specific grid infrastructures (e.g. user name and password). LDAP protocol is currently used for saving and retrieving stored information.

Data management

In every grid system data management is a very important and complicated part. After many analyses a common set of interfaces that allow operation on data and/or metadata was created. The designed framework allows extending RAS infrastructure easily so that many management systems could be attached.

One of the main additional functionalities is the User Virtual Directory that is an abstract filesystem that contains information about all user files independently of their physical location. Each branch in the Virtual Directory tree can be physically placed on different location or even can be placed on storage with different way of accessing (e.g. ftp, gridftp or any other project native protocol). Virtual Directory was designed to standardise data access, and to create a user-friendly, uniform view of Grid and local files.

Application information management

To submit application to grid and to visualise results some information about application is needed. An interface for application information management is provided. LDAP protocol is currently used for retrieving application information.

Other services

We are going to extend the functionality of the Roaming Access Server not only in a field of grid services. We will design functionality that will simplify the communication between users like scientists for data exchanging. In scope of our research there are also ways of noticing the user via e-mails, SMS etc. about important events connected with the status of his/her submitted jobs.

To add support for new grid infrastructure requires RAS plug-in is necessary. RAS plug-ins provides grid specific information and are used by job submission, job monitoring and data management modules. RAS plug-in is a Java class that inherits from RASPluginBase and implements all its abstract functions.

The following diagram shows the functional dependencies between MD&RAS modules and other CrossGrid components. Reliability of Migrating Desktop & Roaming Access Server can be derived from reliability of each of their sub-modules that they depend on.

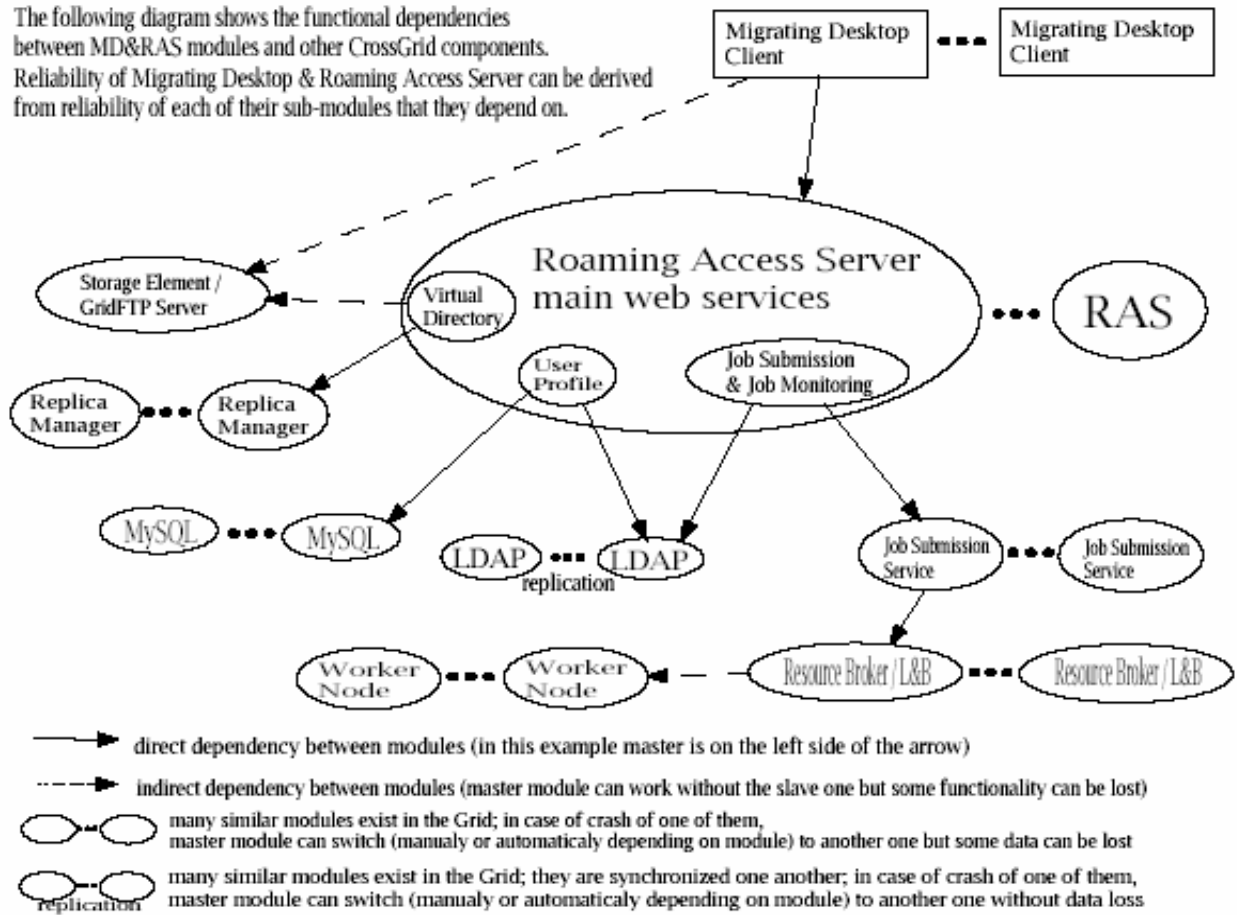


Fig. 4 Functional dependencies between RAS/MD modules

4 PRODUCT INTERFACES

4.1 MIGRATING DESKTOP – JAVA APPLICATION PROGRAM INTERFACE

Migrating Desktop defines interface of some plug-ins - entities designed to act similar to those used in popular browsers. Such modules that can be easy implemented and integrated with Migrating Desktop as these “parent” application. There are two main groups of plug-ins: Job Submission plug-ins and Tool Plug-ins.

4.1.1 Tool Plug-in

Tool Plug-in is designed to act similar to used in popular browsers “plug-ins” – it is a module that can be easy implemented and integrated with Migrating Desktop as its “parent” application.

Fields of applications of the Tool Plug-in are:

- visualization of grid application output;
- integration of Java application (like e.g. grid tools implemented in confines of WP2, VNC viewer, etc) with Migrating Desktop;
- file viewers - visualisation of files;

Tool Plug-in was designed because of the need of integration several application and tools with Migrating Desktop, in order to standardize the integration and to give them easy access to some resources (like user proxy, some remote files, etc.).

4.1.1.1 Description

All Tool Plug-in developers should do, is to write class that extends ToolPluginBase class (that inherits from JPanel class) and defines all its abstract methods. That class can be put inside Tool Plug-in Container – usually JFrame or JDialog (see picture below) that implements ToolPluginContainerInterface and defines set of methods that can be called from TPlug.

Tool Plug-in shall be delivered as a set of java archive files (available in some network location) from which Migrating Desktop will load classes dynamically using Java reflection mechanism.

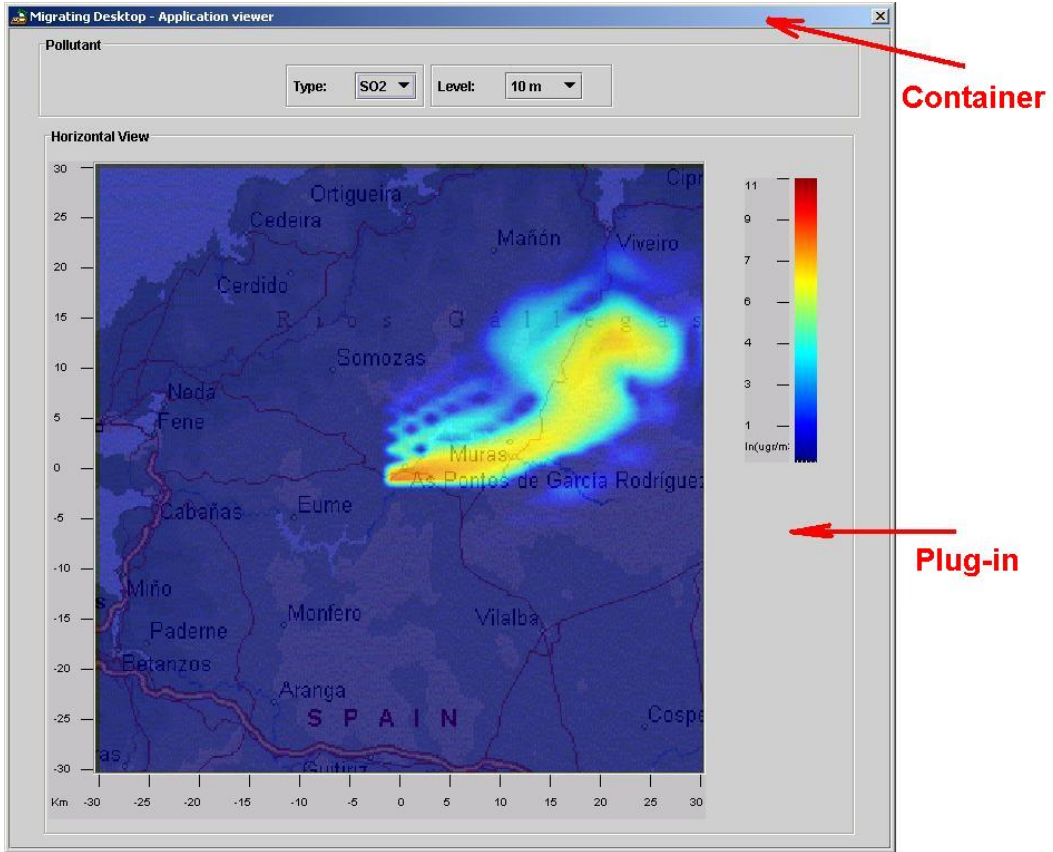


Fig. 5 Tool Plug-in used for visualisation of air pollution application output

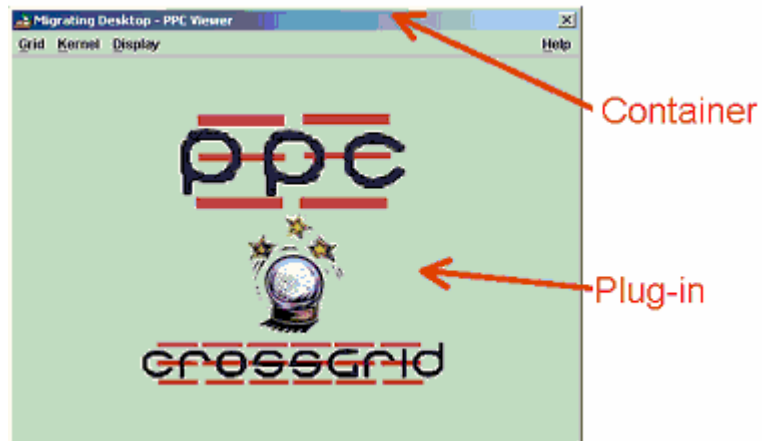


Fig. 6 Tool Plug-in used for integration Performance Prediction tool with MD

4.1.1.2 Tool Plug-in API

Abstract ToolPluginBase class defines set of methods – part of them is already defined and just call method from container the other are abstract and should be defined by Tool Plug-in developers.

4.1.1.2.1 “Ready to use” methods;

Tool Plug-in classes may use some methods of ToolPluginBase that call methods from container.

4.1.1.2.1.1 *setToolContainer*

- public void setToolContainer(ToolContainerInterface toolContainer)
Sets parent tool container for plug-in. This method is called (only!) by container just after creating plug-in object.
- Parameters:
 - ✓ *toolContainer* – parent tool container;

4.1.1.2.1.2 *getGSSCredential*

- public GSSCredential getGSSCredential() throws ToolPluginException
Gets user credential;
- Returns:
Object that implements GSSCredential interface - user credential. Now, credential create at MD side is GlobusGSSCredentialImpl class, so it can be cast to that class for getting GlobusCredential etc.

4.1.1.2.1.3 *browseFile*

- public String browseFile() throws ToolPluginException
Opens Grid Explorer (in “file selection” mode) and return URL (as a String) of file that user chose;
- Returns:
URL¹ of chosen file as a String;

4.1.1.2.1.4 *browseFile*

- public String browseFile(String startDirId) throws ToolPluginException
Opens Grid Explorer (in “file selection” mode) and return URL of file that user chose; If related file system is already open in Grid Commander or grid Explorer, Grid Explorer starts in directory specified by **startDirId** parameter.
- Parameters:
 - ✓ startDirId – identifier of initial directory. For local file system, FTP or GridFTP file system startDirId shall be the same as directory URL¹ (e.g. gsiftp://any.host.name:2811//path/startdir)

¹ There are some inconsistencies between SUN implementation of Java and RFC 1738. RFC 1738 document states that “the ‘/’ between the host (or port) and the url-path is NOT part of the url-path” (see <http://www.faqs.org/rfcs/rfc1738.html> for details). So, there should be 2 slashes after port number for

- Returns:
URL¹ of chosen file as a String;

4.1.1.2.1.5 *browseDirectory*

- public String[] browseDirectory() throws ToolPluginException
Opens Grid Explorer (in “directory selection” mode) and return array of file URLs belonging to directory chosen by user;
- Returns:
Array of file URLs¹ belonging to directory chosen by user;

4.1.1.2.1.6 *browseDirectory*

- public String[] browseDirectory(String startDirId) throws ToolPluginException
Opens Grid Explorer (in “directory selection” mode) and return array of URL of files belonging to directory chosen by user; If related file system is already open in Grid Commander or grid Explorer, Grid Explorer starts in directory specified by **startDirId** parameter.
- Parameters:
 - ✓ startDirId – identifier of initial directory. For local file system, FTP or GridFTP file system startDirId shall be the same as directory URL¹ (e.g. gsift://any.host.name:2811//path/startdir)
- Returns:
Array of URLs¹ of files belonging to directory chosen by user;

4.1.1.2.1.7 *getRemoteOutputStream*

- public OutputStream getRemoteOutputStream(String fileURL, boolean append, boolean passiveMode, int transferMode) throws ToolPluginException
Opens remote output stream to file pointed in fileURL;
- Parameters:
 - ✓ fileURL – URL of file
 - ✓ append – shall file contents be overridden ?
 - ✓ passiveMode – server or client choose FTP ports;
 - ✓ transferMode – binary or ascii, defined in org.globus.ftp.Session

4.1.1.2.1.8 *getRemoteInputStream*

- public InputStream getRemoteInputStream(String fileURL, boolean passiveMode, int transferMode) throws ToolPluginException –
opens remote input stream to file pointed in fileURL
- Parameters:

GridFTP and FTP and URL of local Unix file starts with “file:///dir1/...” (four slashes!). – MD uses the same scheme as Commodity Grid (CoG) organisation.

- ✓ fileURL – URL of file
- ✓ passiveMode – server or client choose FTP ports;
- ✓ transferMode – binary or ascii (FTP_TRANSFER_MODE_IMAGE or FTP_TRANSFER_MODE_ASCII) defined in ToolPluginConstants

4.1.1.2.1.9 readRemoteFile

- public byte[] readRemoteFile(String fileURL, boolean passiveMode, int transferMode) throws Exception

Read remote file and returns its contents as byte array

- Parameters:
 - ✓ fileURL – URL of file
 - ✓ passiveMode – server or client choose FTP ports;
 - ✓ transferMode – binary or ascii (FTP_TRANSFER_MODE_IMAGE or FTP_TRANSFER_MODE_ASCII) defined in ToolPluginConstants

4.1.1.2.1.10 getFileLocation

- public String getFileLocation(String fileId) throws Exception
Gets URL of file based on “Virtual Directory id” passed as an argument.
- Parameters:
 - ✓ fileId – unique virtual directory identifier
- Returns:
URL of file

4.1.1.2.1.11 closeContainer

- public void closeContainer() throws Exception
Close parent container window.

4.1.1.2.1.12 getBaseVersion

- public String getBaseVersion()
Returns the version of ToolPluginBase;
- Returns:
Version of ToolPluginBase;

4.1.1.2.2 Abstract methods

This set of methods is called from container and shall be defined inside the plug-in.

4.1.1.2.2.1 setProperties

- abstract public void setProperties(Properties props);
Called by Tool Container to set some initialisation values.
- Parameters:
 - ✓ props – java.util.Properties – specific properties passed from container to plug-in;

4.1.1.2.2.2 *getProperties*

- abstract public Properties getProperties();
used for getting some values from Tool Plug-in - not defined yet, may be used in future

4.1.1.2.2.3 *init*

- abstract public int init();
Called by Tool Container to let Tool Plug-in know that it should perform some initialisation actions. It may be empty method
- Returns:
 - ✓ ToolPluginConstants.STATUS_OK – if function returns correctly
 - ✓ ToolPluginConstants.STATUS_ERROR – if an error occurs

4.1.1.2.2.4 *start*

- abstract public int start();
Tells TPlug that it has been added to container, it is now visible and should perform some (tool specific) actions.
- Returns:
 - ✓ ToolPluginConstants.STATUS_OK – if function returns correctly
 - ✓ ToolPluginConstants.STATUS_ERROR – if an error occurs

4.1.1.2.2.5 *stop*

- abstract public int stop();
Tells TPlug to stop its execution. Called by Tool Container to let Tool Plug-in know that it is no longer visible and will be removed from Tool Container. It may be empty method
- Returns:
 - ✓ ToolPluginConstants.STATUS_OK – if function returns correctly
 - ✓ ToolPluginConstants.STATUS_ERROR – if an error occurs

4.1.1.2.2.6 *destroy*

- abstract public int destroy();
Called by Tool Container to let Tool Plugin know that it will be no longer used and should perform some "cleanup" operation. It may be empty method
- Returns:
 - ✓ ToolPluginConstants.STATUS_OK – if function returns correctly
 - ✓ ToolPluginConstants.STATUS_ERROR – if an error occurs

4.1.1.3 **Interaction between Plug-in and Container**

Tool Plug-in main class extends JPanel, so it may be put inside any Java “graphic” container (like every Java component).

The sequence of calls TPlug methods from container is:

- constructor ← create class;

- `setToolContainer` ← passes to TPlug class that implement ToolPluginContainerInterface
- `init` ← perform some initialisation actions;
- `setProperties` ← passes to TPlug all necessary parameters (using java.util.Properties class – see next paragraph)
- `start` ← container shows the TPlug that should start its execution (e.g. starts animation for visualisation plug-ins);
- `stop` ← TPlug stops execution
- `destroy` ← container is destroying, so it call TPlug to perform some “cleaning” actions;

4.1.1.4 Passing parameters to plug-in

4.1.1.4.1 General parameters

Property name	Description
Name	Job name set by the user
Status	Job status (in format "number:string")
JobId	Unique job identifier (this does not need to be EDG JobId or Globus JobId)
EndDate	Job finishing date as returned by RB
User	Distinguish name of submitter (certificate subject)
SubmissionDate	Date of job submission
StartDate	Date when job was starter
Host	Hostname on which job was run
Type	Grid type. For crossgrid jobs it is always "crossgrid"
Application	Application identifier

4.1.1.4.2 Arguments

For every argument:

Property name	Description
<code>Argument.<arg_name>.Value</code>	Value of an argument
<code>Argument.<arg_name>.Prefix</code>	Prefix of the argument as it should appear in command line (eg. -f=)
<code>Argument.<arg_name>.CommandLi</code>	"true" or "false" indicates weather

ne	<p>this argument should be added to command line (in a form Prefix+Value eg. -f=5)</p> <p>Possible use of this field is when one command line argument is generated from several graphical components. For example application plugin can show three input boxes and ask about RGB color components. Then arguments with names Rcolor, Gcolor and Bcolor should have CommandLine flag set to "false" and argument RGBcolor with value computed from RGB components should have CommandLine flag set to "true" and will be passed to command line.</p>
----	---

4.1.1.4.3 Resources

For every resource:

Property name	Description
Resource.<res_name>.Value	Name and Value of job resource requirements.

4.1.1.4.4 Files

For every file:

Property name	Description
File.<file_log_name>.Id	Virtual Directory unique file identifier
File.<file_log_name>.Type	"in", "out", "inout", "refr", "temp", "StdInput", "StdOutput", "StdError";
File.<file_log_name>.RefreshTime	Time (in seconds) after which "visualizer" should reload file

4.1.1.4.5 Environment

For every environment variable:

Property name	Description
Variable.<var_name>.Value	Environment variable (like PATH, USER_HOME, etc)

4.1.1.5 Dynamic loading of Tool Plug-IN classes

There are several applications and tools that will be integrated with Migration Desktop, so delivering all Tool Plug-ins as a subset of applet archives will extremely enlarge applet. So, every Tool Plug-in will be kept in separate network location and load them (using Java reflection mechanism) only when given visualization or tool will be called by user. The advantages of that solutions are:

- makes Migrating Desktop not so „heavy”;
- changes in plug-in code can be introduced immediately;
- plug-in developer has better control over his code

4.1.1.6 XML description of plug-in

Container requires some parameters to correctly load a plug-in: full qualified name of main plug-in class, set of libraries used by plug-in classes, etc... These parameters that had previously to be registered directly in MD can be now defined in simple XML file placed “somewhere on the net”.

Using XML plug-in developer has control over plug-in load mechanism – he can change plug-in parameters on his own (and these changes will be visible in MD immediately).

See <http://ras.man.poznan.pl/crossgrid/plugins/tool/GridBench.xml> for example and <http://ras.man.poznan.pl/crossgrid/plugins/tool/ToolPlugin.xsd> for XML schema that should be used for XML validation. Most of the XML tags are “self-explaining” – fields can be filled based on example.

Note: Field “file_extensions” is optional. It should be filled only if Tool Plug-in can be used as file viewer for specific type of files.

4.1.1.7 security

To increase security level:

- sign all plug-in libraries (using certificate signed by one of the X# Certificate Authorities, if it is possible);
- make all plug-in files (XML and libraries) available using *https* protocol which use TLS (SSL) protection.

4.1.1.8 Jar signing

The security policy is still a matter of discussion of Integration Team, so there are no restrictive rules defining certificates that should be used for jar signing. To sign jar using existing certificate verified by one of X# CA, certificate (and private key) should be first exported into file of PKCS12 format that can be use as java “keystore”.

- Exporting certificate

To export certificate OpenSSL toolkit² tool can be used.

Command syntax:

```
openssl pkcs12 -export -chain \  
-inkey <private key file path> \  
-in <certificate file path> \  

```

² OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them. The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. [from openssl manual]

```
-out <keystore user defined name>
-CApath /etc/grid-security/certificates/ \
-name <certificate alias>
```

(see open ssl documentation for details <http://www.openssl.org/docs/>)

- Signing jar

Command syntax:

```
jarsigner \
  -keystore <keystore user defined name> \
  -storetype PKCS12 \
  -storepass <keystore passwd> \
  <jar file name> <certificate alias>
```

See <http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html#security> for detailed jarsigner tool description.

4.1.1.9 Additional remarks

- Do not use container methods in constructor

Please remember that sequence of calls Tplugin methods from container is: constructor → setToolContainer → init → setProperties → start, so don't use any container methods (as getRemote/OutputStream) in your constructor.

- Don't open new window as main Tool Plug-in window

Please make TPlug content pane main pane of your tool. TPlug content pane will be put in dialog (or other java container) created specially for that purpose. Please avoid situation, where dialog (container for TPlug) will occur empty and your tool starts its own window/frame as the main frame (however plug-in may open its "child" windows/frames)

- Use InputStreams for reading.

To avoid allocating large amount of memory please use rather InputStreams for reading large remote files than buffers (read file sequentially, not use readRemoteFile method);

- Use „remote streams” carefully!

Please be careful while using InputStream.read(byte[]) method: for input streams opened for "remote files" it may returns data partially (as data frames come from network(?)) so use rather InputStream.read()

- Store Tool Plug-in *.jar files in „safe” but available network location!

Java archive files network location should be available via HTTP protocol. URLs to jar's shouldn't change.

- Write „no graphic blocking” code – make use of threads;

Please remember that every action “fired” by any GUI event (like pressing toolbar button, etc) is handled by Java EventQueue. Performing time consuming operations inside EventQueue will hang application graphic (see <http://java.sun.com/docs/books/tutorial/uiswing/overview/event.html#thread> for details)

- TPlug start(), stop() methods can be called in loop!

- Tool Plug-in class are loaded dynamically from network location;

So all changes in TPlug code will be available for TPlug developer (and other users) immediately after creating archive and putting it on web.

- Java archive (cog-jglobus.jar) is needed to compile plug-in.

It can be downloaded from <http://www-unix.globus.org/cog/java/1.1/>

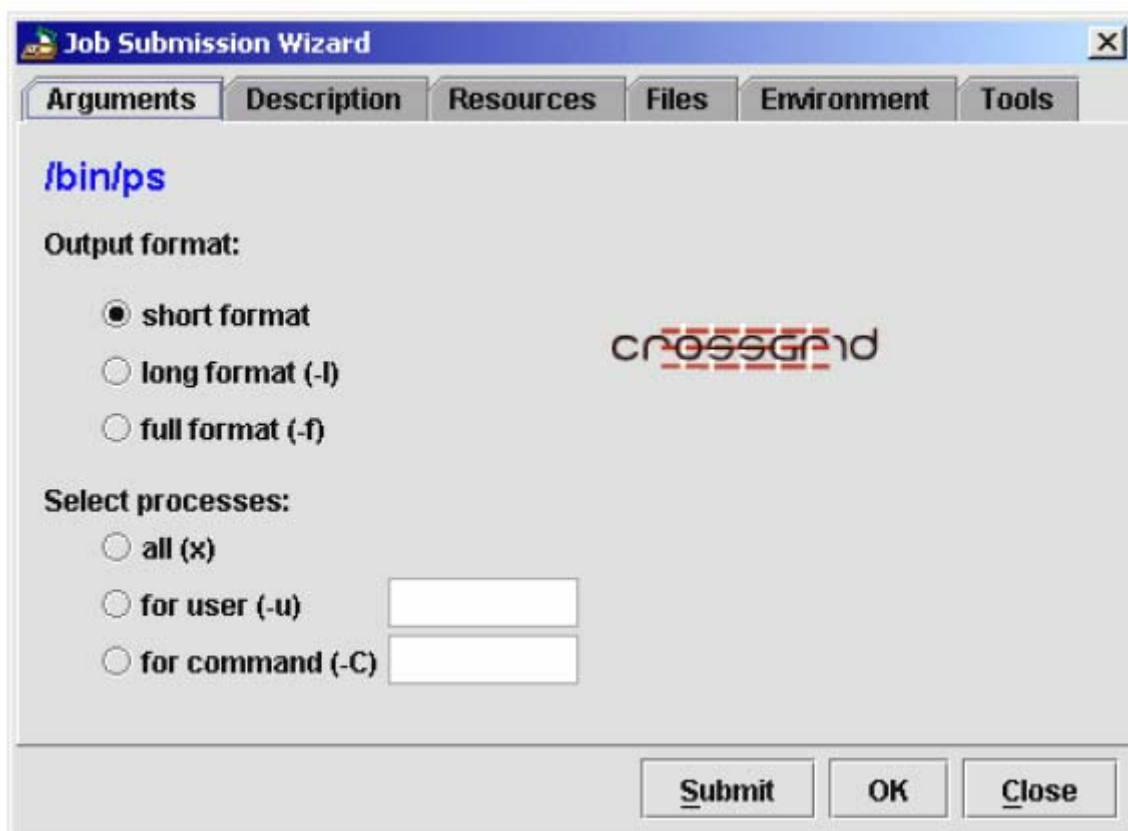
4.1.2 JS Plug-in

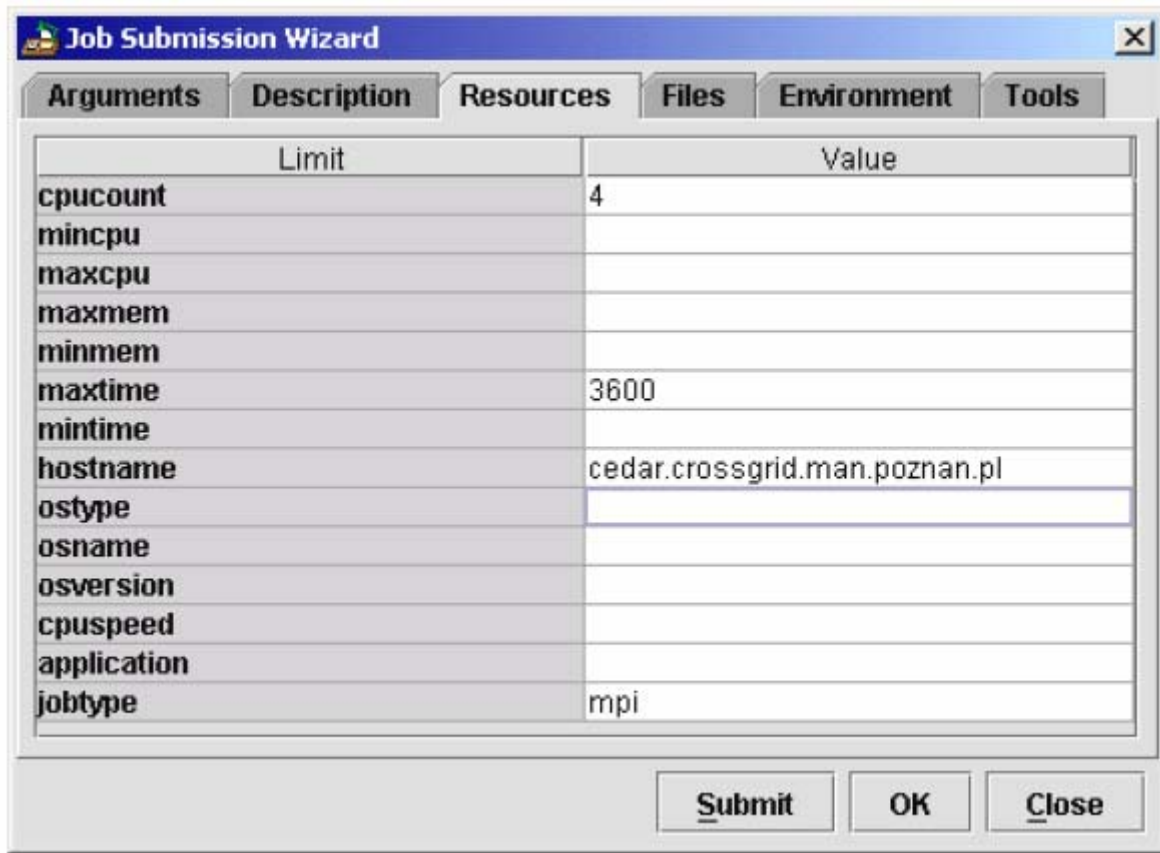
4.1.2.1 Description of application plug-in.

JobSubmissionWizard supports two kind of plug-ins – JSSApplicationPlugins and JSSToolPlugins.

Application plug-in in general is used to get application specific input parameters. This will allow prepare portal framework that is independent of application type so that we could easily extend it and add next applications.

Tool plug-in is used to modify job parameters according to the specific tool needs. Desktop provides a wizard that user can use to specify job details. This wizard will simplify process of specifying parameters and limits, suggesting user default or lastly used parameters. That wizard will consist of several panels. The first panel is reserved for application specific plug-in – Arguments panel. (Contents of that panels that mean “application plug-in” will be implemented by applications). List of tools plug-ins appears in the last panel of job submission wizard.





JSSApplication plug-in is a java class that inherits from JSSApplicationPlug-inBase class and implements a set of its methods.

JSSTool plug-in is a java class that inherits from JSSToolPluginBase class and implements a set of its methods.

Both plug-in bases inherit from JSSPluginBase which inherits from JPanel.

Full class names are:

org.crossgrid.wp3.portals.plugins.jss_plugin.JSSApplicationPluginBase

org.crossgrid.wp3.portals.plugins.jss_plugin.JSSToolPluginBase

Documentation and examples can be found at:

<http://ras.man.poznan.pl/crossgrid/plugins/JSSplugins.html>

4.1.2.2 Procedure of preparation and execution of the JSS Plug-in

Migrating Desktop will have information (URL, class name, JAR file name etc.) how to get to the tool plug-in through the Internet. The application plug-in developers should provide such information. For example URL like `http://ras.man.poznan.pl/gridApplication.jar` will allow Migrating Desktop for downloading tool plug-in class from this web page, create appropriate object and place it inside JobSubmission Dialog. The JAR file pointed by that URL should contain all the classes, resources and libraries required (imported) by the tool

plug-in.

After the tool plug-in class is downloaded, plug-in object is created and initialised inside the JobSubmissionDialog. The next task of the JobSubmissionDialog is to add the application plug-in to the proper GUI container (where the plug-in will be displayed). Plug-ins should inherit from appropriate base class and implements some methods:

```
public String getVersion()
```

Returns the version of a specific ApplicationPlug-in.

```
public String getName()
```

Returns the Name of the plug-in. This name will be used in the list of available plug-ins.

```
public String getDescription()
```

Should return short description of the plug-in.

```
protected boolean onSubmit() throws JSSInvalidJobException
```

Called by JobSubmission Wizzard, when user clicks Submit button. If some data or scripts should be prepared, they should be created here. This function can also check if parameters are set correctly. If return is false, job will be not submitted. Plug-in can show message dialog itself or it can pass a message to job submission dialog by throwing JSSInvalidJobException.

```
public void init()
```

Called by JobSubmission Wizzard, when the instance of plugin is created. This can be place for graphical setup of the plugin

```
public void start()
```

Called by JobSubmission Wizzard, when the plugin is put into its Container. Application plug-ins must also implement:

```
public Argument[] getArguments();
```

Called by JobSubmission Dialog when the dialog is closed. The Arguments are used to save the job and to create a command line to start a job. Should return the table of Arguments

Tool plug-ins should implement also:

```
- public void configure()
```

Called by job submission wizard when user clicks the button for toll configuration. Can show user a dialog for parameter configuration.

- public Properties getConfiguration()

Should return the configuration of the plug-in in the form of Properties. The configuration will be stored together with a job and when the Job Submission Wizard will be restored, this configuration will be passed to plug-in.

- public void setConfiguration(Properties configuration)

Sets Properties with plug-in configuration values. These are the same values which were taken by getConfiguration() method during last plugin configuration.

JSS plug-ins can implement some methods defined in plug-in base classes. Function common for JSSApplicationPlugin and JSSToolPlugin:

public File[] getAllFiles()

throws JSSPluginException

Allows getting information about all files currently defined in the "Files" tab of the Job Submission Dialog.

public void addFile(File f)

throws JSSPluginException

Allows adding a file to the list of files in the "Files" tab in the Job Submission Dialog.

public void addFile(File f,

java.lang.String fileId)

throws JSSPluginException

Allows adding a file with known fileId to the list of files in the "Files" tab in the Job Submission Dialog. FileId is returned by browseFile.

ApplicationPlugin

public void removeFile(File f)

throws JSSPluginException

Allows removing a file from the list of files in the "Files" tab in the Job Submission Dialog.

public Resource[] getAllResources()

throws JSSPluginException

Allows getting information about all resource requirements currently defined in the "Resources" tab of the Job Submission Dialog.

public java.lang.String getResource(java.lang.String name)

throws JSSPluginException

Allows getting the value of the resource with a given name

public java.lang.String getResource(int type)

throws JSSPluginException

Allows to get the value of the resource with a given type

public void addResource(Resource res)

throws JSSPluginException

Allow to add a resource requirements to the list of resource requirements in the "Resources" tab of the Job Submission Dialog. It calls addResource(Resource res, int force) with the RECOMMEND flag

public void addResource(Resource res,
int force)

throws JSSPluginException

Allows to add a resource requirements to the list of resource requirements in the "Resources" tab of the Job Submission Dialog.

public void removeResource(Resource res)

throws JSSPluginException

Allows to remove a resource requirements from the list of resource requirements in the "Resources" tab of the Job Submission Dialog.

public Variable[] getAllVariables()

throws JSSPluginException

Allows to get information about all environment variables currently defined in the "Environment" tab of the Job Submission Dialog.

public java.lang.String getVariable(java.lang.String name)

throws JSSPluginException

Allows to get the value of the environment variable with a given name

public void addVariable(Variable var)

throws JSSPluginException

Allows to add an environment variable to the list of variables in the "Environment" tab of the Job Submission Dialog. It calls addVariable(Variable var, int force) with the RECOMMEND flag

```
public void addVariable(Variable var,  
int force)
```

throws JSSPluginException

Allows to add an environment variable to the list of variables in the "Environment" tab of the Job Submission Dialog.

```
public void removeVariable(Variable var)
```

throws JSSPluginException

Allows to remove an environment variable from the list of variables in the "Environment" tab of the Job Submission Dialog.

```
public java.lang.String getJobDescription()
```

throws JSSPluginException

Returns Job Description from Description panel of JobSubmissionDialog

```
public java.lang.String getJobName()
```

throws JSSPluginException

Returns Job Name from Description panel of JobSubmissionDialog

```
public javax.swing.JDialog getParentDialog()
```

throws JSSPluginException

Returns the parent dialog. If Application Plugin will open new dialog the result of getParentDialog() should be passed to its constructor e.g. newDialog = new JDialog(getParentDialog());

In addition JSS Application plug-ins can use following methods:

```
public FileDialog browseFile()
```

throws JSSPluginException

Starts a Grid Explorer dialog and allows to choose a file from user Virtual Directory

```
public java.io.OutputStream getRemoteOutputStream(java.lang.String fileId,  
boolean append)
```

throws JSSPluginException

Opens a remote file from a user VirtualDirectory. This function can be used to

write a file using the OutputStream. If this file should be automatically copied to CE were the job will be executed it is necessary to add this file by `addFile(file,fileId)`

`public java.io.InputStream getRemoteInputStream(java.lang.String fileId)`
throws `JSSPluginException`

Opens a remote file from a user `VirtualDirectory`. This function can be used to read a file using the `inputStream`.

`public void writeFile(java.lang.String fileId,`
`boolean append,`
`byte[] buffer)`
throws `JSSPluginException`

Writes a buffer into a file in a user `VirtualDirectory`. This function can be used to prepare set of data for application. If this file should be automatically copied to CE were the job will be executed it is necessary to add this file by `addFile(name,fileId)`

`public void writeTempFile(java.lang.String name,`
`byte[] buffer)`
throws `JSSPluginException`

Writes a buffer into temporary file in a user `VirtualDirectory`. File is removed when the job is finished. This function can be used to prepare a script which is run on CE. This file will be automatically copied to CE were the job will be executed. There is no need to add this file by `addFile()`

`public java.lang.String getPhysicalLocation(java.lang.String fileId)`
throws `JSSPluginException`

Returns the physical location connected with given `fileId`.

`public byte[] readFile(java.lang.String fileId)`
throws `JSSPluginException`

Reads a file from user `VirtualDirectory`. Use it only with small files. For large files use `getRemoteInputStream()`

`public javax.swing.ImageIcon loadImageIcon(java.lang.String filename)`
throws `JSSPluginException`

Auxiliary function for reading image from jar archive.

`public javax.swing.ImageIcon loadImageIcon(java.net.URL url)`
throws `JSSPluginException`

Auxiliary function for reading image from URL.

4.2 ROAMING ACCESS SERVER - WEB SERVICES INTERFACE

4.2.1 Server Login Web Service

This web service provides functions used during logging to the system. Client can check if the webservice gives any response, what is the performance of that server (MD can connect to RAS server that has best performance). Server checks also to which VO user belongs and what are the appropriate servers for his VO. It can occur that server that user are login to serve only different VO but if only the credentials are correct and user exist in gridmap file a list of appropriate other servers would be returned to client.

4.2.1.1 *String* `getServerAddressForVO(String hostname, byte[] input)`

Return the server name that serves VO that user belongs to. Hostname that is given as parameter is returned in case if this server serves user VO. Otherwise first from the rest of RAS servers for specified VO is returned.

PARAMETERS

hostname – server name that user connects to (localhost name)

input – table of bytes with credential

RETURN VALUE

The server name that serves VO that user belongs to.

EXCEPTIONS

java.rmi.RemoteException - in case of

EXAMPLE

```
getServerAddressForVO("https://ras.man.poznan.pl:8443", input);
```

4.2.1.2 *Vector* `getServersAddressForVO(java.lang.String hostname, byte[] input)`

Return the list of servers name that serve VO that user belongs to.

PARAMETERS

hostname – server name that user connects to (localhost name)

input – table of bytes with credential

RETURN VALUE

Vector with the server's name that serves VO that user belongs to.

EXCEPTIONS

java.rmi.RemoteException - in case of following exception:

“User does not exist in grid-map file” – user should be added to VO, this information is propagate

“Unable to get list of available servers” – user exist in grid-map file but there is no any server that serves his VO

EXAMPLE

```
getServersAddressForVO(“https://ras.man.poznan.pl:8443”, input);
```

4.2.1.3 *boolean exist()*

Check existence of server. Can be used for checking server response time.

RETURN VALUE

Always return TRUE.

3.1.4 long checkServerPerformance()

Return the server performance.

RETURN VALUE

Long integer value the lower value the better performance.

4.2.2 File Tunnel Web Service

File Tunnel Web Service was designed to solve common problems with firewalls that blocks FTP/GridFTP connections. It provides basic functionality of the FTP/GridFTP over the http connection. Web-service methods are responsible for performing “control commands” - for data transmission File Tunnel Servlet can be used.

4.2.2.1 *open*

```
public java.lang.String open(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData) throws java.rmi.RemoteException;
```

Method opens connection to FTP or GridFTP server based on passed connection parameters. It returns user home directory for opened file system.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

RETURN VALUE

User home directory URL.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.2 close

public void close(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData) throws java.rmi.RemoteException;

Method closes connection to FTP or GridFTP server described by connection parameters passed to method.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

RETURN VALUE

None.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.3 exists

public boolean exists(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String nodeURL) throws java.rmi.RemoteException;

Method checks if file or directory of given URL exists on FTP or GridFTP server described by connection parameters.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

nodeURL – URL of the file or directory to be checked if exists on server;

RETURN VALUE

True – if file or directory exists on specified server

false – otherwise.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.4 createNewFile

```
public org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable createNewFile(java.lang.String  
userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String currentDirId,  
java.lang.String name) throws java.rmi.RemoteException;
```

Method creates new zero-length file in directory of given URL.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

currentDirId – URL of the directory in which new file shall be created;

name – name of the file to be created;

RETURN VALUE

Object of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing new created file. Class FileInfoTransferable is “a container” for data taken from org.globus.ftp.FileInfo class object to transfer it through web-services.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.5 createNewDir

```
public org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable createNewDir(java.lang.String  
userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String currentDirURL,  
java.lang.String name) throws java.rmi.RemoteException;
```

Method creates new subdirectory of directory of given URL..

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

currentDirURL – URL of the directory in which new directory will be created;

name – name of the new directory to be created;

RETURN VALUE

Object of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing new created file. Class FileInfoTransferable is “a container” for data taken from org.globus.ftp.FileInfo class object to transfer it through web-services.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.6 rename

```
public void rename(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData,  
java.lang.String nodeURL, java.lang.String newName) throws java.rmi.RemoteException;
```

Method renames file of given URL.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

nodeURL – URL of the file or directory to be renamed;

newName – new name of the file;

RETURN VALUE

None

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.7 deleteDir

public void deleteDir(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String dirURL) throws java.rmi.RemoteException;

Method deletes directory of given URL on remote server.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

dirURL – URL of the directory to be deleted;

RETURN VALUE

None

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.8 deleteFile

public void deleteFile(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String fileURL) throws java.rmi.RemoteException;

Method deletes file of given URL on remote server.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

fileURL – URL of the file to be deleted;

RETURN VALUE

None

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.9 listDir

```
public org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable[] listDir(java.lang.String userId,
byte[] input, java.lang.String fsType, byte[] connData, java.lang.String dirURL) throws
java.rmi.RemoteException;
```

Method lists contents of directory of given URL.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

dirURL – URL of the directory which contents is listed;

RETURN VALUE

Array of objects of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing files or folder belong to listed directory. Class FileInfoTransferable

is “a container” for data taken from `org.globus.ftp.FileInfo` class object to transfer it through web-services.

EXCEPTIONS

`java.rmi.RemoteException` – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.10 *goUpDir*

```
public java.lang.String goUpDir(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData, java.lang.String dirURL) throws java.rmi.RemoteException;
```

Method returns “parent” directory of the directory which URL is specified as input parameter;

PARAMETERS

`userId` – unique identifier of the user;

`input` – user proxy stored as byte array;

`fsType` – file system type, should be one of the following:

- `org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP` for FTP connections;
- `org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP` for GridFTP connections;
- `org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS` for Private Storage connection;

`connData` – connection specific parameters stored as byte array;

`dirURL` – URL of the directory which contents is listed;

RETURN VALUE

URL of the “parent” directory.

EXCEPTIONS

`java.rmi.RemoteException` – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.11 *getRootCurrDirPath*

```
public org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable[]  
getRootCurrDirPath(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData,  
java.lang.String currentDirURL) throws java.rmi.RemoteException;
```

Method returns set of directories placed between root and given directory;

PARAMETERS

`userId` – unique identifier of the user;

`input` – user proxy stored as byte array;

`fsType` – file system type, should be one of the following:

- `org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP` for FTP connections;

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
 - org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;
- connData – connection specific parameters stored as byte array;
currentDirURL – URL of the directory;

RETURN VALUE

Array of objects of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing folders placed between root and given directory. Class FileInfoTransferable is “a container” for data taken from org.globus.ftp.FileInfo class object to transfer it through web-services.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.12 *thirdPartyTransfer*

```
public org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable  
thirdPartyTransfer(java.lang.String userId, byte[] input, java.lang.String fsType, byte[] connData,  
java.lang.String sourceURL, java.lang.String destURL) throws java.rmi.RemoteException;
```

Method performs third file party transfer between two GridFTP servers.

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

sourceURL – URL of the source file;

destURL – URL of target file;

RETURN VALUE

Object of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing transferred file. Class FileInfoTransferable is “a container” for data taken from org.globus.ftp.FileInfo class object to transfer it through web-services.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.2.13 *getFileInformation*

```
public                               org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable  
getFileInformation(java.lang.String  userId, byte[]  input, java.lang.String  fsType, byte[]  connData,  
java.lang.String  nodeURL) throws java.rmi.RemoteException;
```

Method returns information about file of the given URL

PARAMETERS

userId – unique identifier of the user;

input – user proxy stored as byte array;

fsType – file system type, should be one of the following:

- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_FTP for FTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_GRIDFTP for GridFTP connections;
- org.crossgrid.wp3.portals.common.Constants.FS_TYPE_PS for Private Storage connection;

connData – connection specific parameters stored as byte array;

nodeURL – file URL ;

RETURN VALUE

Object of the class org.crossgrid.wp3.portals.common.io.ftp.FileInfoTransferable containing data describing transferred file. Class FileInfoTransferable is “a container” for data taken from org.globus.ftp.FileInfo class object to transfer it through web-services.

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3 Auxiliary Web Service

This web-service offers two groups of methods that perform some auxiliary functions:

- methods for managing security information;
- methods for managing MD plug-ins configuration;

4.2.3.1 *getTrustedCertificates*

```
public java.util.Vector  getTrustedCertificates(java.lang.String  userId, byte[]  input)  throws  
java.rmi.RemoteException;
```

Method returns set of trusted CA certificates.

PARAMETERS

userId – unique identifier of the user;
input – user proxy stored as byte array;

RETURN VALUE

Vector of trusted CA certificates. Each element of the vector is a object of class `java.security.cert.X509Certificate` stored as a byte array.

EXCEPTIONS

`java.rmi.RemoteException` – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.2 *getSecurityInfo*

```
public java.util.Vector getSecurityInfo(java.lang.String userId, byte[] input) throws java.rmi.RemoteException;
```

Method returns user-defined set of “security information” - data describing two kinds of entities:

- trusted (or rejected) SSL connections, or
- URLs to java archive files which can (or cannot) be loaded by application.

PARAMETERS

userId – unique identifier of the user;
input – user proxy stored as byte array;

RETURN VALUE

Vector of objects of class `org.crossgrid.wp3.portals.migratingdesktop.security.TrustInfo` stored as a byte array. Class `TrustInfo` stores information about SSL connections or URLs to jar which have been accepted (or rejected) by the user.

EXCEPTIONS

`java.rmi.RemoteException` – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.3 *addSecurityInfo*

```
public int addSecurityInfo(java.lang.String userId, java.lang.String secId, byte[] securityData, byte[] input) throws java.rmi.RemoteException;
```

Method adds “security information” - data describing one of two kinds of entities:

- trusted (or rejected) SSL connections;
- URLs to java archive files which can (or cannot) be loaded by application.

PARAMETERS

userId – unique identifier of the user;
secId - unique identifier of stored “security information”;
securityData – object of TrustInfo class stored as byte array;
input – user proxy stored as byte array;

RETURN VALUE

org.crossgrid.wp3.portals.common.Constants.OPERATION_SUCCESSFUL – if methods ends correctly;
org.crossgrid.wp3.portals.common.Constants.OPERATION_FAILED – otherwise;

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.4 *removeSecurityInfo*

public int removeSecurityInfo(java.lang.String userId, java.lang.String infoId, byte[] input) throws java.rmi.RemoteException;

Method removes “security information” of given identifier;

PARAMETERS

userId – unique identifier of the user;
secId - unique identifier of stored “security information”;
input – user proxy stored as byte array;

RETURN VALUE

org.crossgrid.wp3.portals.common.Constants.OPERATION_SUCCESSFUL – if methods ends correctly;
org.crossgrid.wp3.portals.common.Constants.OPERATION_FAILED – otherwise;

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.5 *getAllFileViewerXML*

```
public java.util.Vector getAllFileViewerXML(java.lang.String userId, byte[] input) throws java.rmi.RemoteException;
```

Method returns set of all XML descriptions of “Tool Plug-ins” registered as file viewers;

PARAMETERS

userId – unique identifier of the user;
input – user proxy stored as byte array;

RETURN VALUE

Vector of XML descriptions (each stored as String object) of “Tool Plug-ins” registered as file viewers (see chapter Migrating Desktop Plug-ins for details);

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.6 *getFileViewerXML*

```
public java.lang.String getFileViewerXML(java.lang.String userId, java.lang.String xmlId, byte[] input) throws java.rmi.RemoteException;
```

Method returns file viewer XML description for given identifier;

PARAMETERS

userId – unique identifier of the user;
xmlId – unique identifier of file viewer XML description;
input – user proxy stored as byte array;

RETURN VALUE

XML descriptions (stored as String object) of “Tool Plug-in” registered as file viewers (see chapter Migrating Desktop Plug-ins for details);

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.7 *addFileViewerXML*

```
public int addFileViewerXML(java.lang.String userId, java.lang.String xmlId, java.lang.String xmlDescription, byte[] input) throws java.rmi.RemoteException;
```

Method adds (registers) file viewer XML description;

PARAMETERS

userId – unique identifier of the user;

xmlId – unique identifier of file viewer XML description;

xmlDescription - XML description (String object) of file viewer (see chapter Migrating Desktop Plug-ins for details);

input – user proxy stored as byte array;

RETURN VALUE

org.crossgrid.wp3.portals.common.Constants.OPERATION_SUCCESSFUL – if methods ends correctly;

org.crossgrid.wp3.portals.common.Constants.OPERATION_FAILED – otherwise;

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.8 *updateFileViewerXML*

```
public int updateFileViewerXML(java.lang.String userId, java.lang.String xmlId, java.lang.String xmlDescription, byte[] input) throws java.rmi.RemoteException;
```

Method updates (removes old and adds new) file viewer XML description;

PARAMETERS

userId – unique identifier of the user;

xmlId – unique identifier of file viewer XML description;

xmlDescription - XML description (String object) of file viewer (see chapter Migrating Desktop Plug-ins for details);

input – user proxy stored as byte array;

RETURN VALUE

org.crossgrid.wp3.portals.common.Constants.OPERATION_SUCCESSFUL – if methods ends correctly;

org.crossgrid.wp3.portals.common.Constants.OPERATION_FAILED – otherwise;

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.3.9 *removeFileViewerXML*

public int removeFileViewerXML(java.lang.String userId, java.lang.String xmlId, byte[] input) throws java.rmi.RemoteException;

Method removes XML description of file viewer for given id;

PARAMETERS

userId – unique identifier of the user;

xmlId – unique identifier of file viewer XML description;

input – user proxy stored as byte array;

RETURN VALUE

org.crossgrid.wp3.portals.common.Constants.OPERATION_SUCCESSFUL – if methods ends correctly;

org.crossgrid.wp3.portals.common.Constants.OPERATION_FAILED – otherwise;

EXCEPTIONS

java.rmi.RemoteException – exception thrown in case of any errors. It among others contains information about original exception cause.

4.2.4 Application Information Web Service

All information concerning application are saved in LDAP. This webservice provides functionality that allows to operate on LDAP for adding removing and browsing application info. This webservice is used by MD for getting application information and by Application Management Applet for adding, changing and removing application info.

4.2.4.1 *addNewApplicationInfo*

public void addNewApplicationInfo(

org.crossgrid.wp3.portals.roamingaccessserver.applicationinfows.ApplicationInfo appInfo, String userId, byte[] input)

PARAMETERS

appInfo – application info details
userId – user id (created basing on credential)
input – table of bytes with credential

RETURN VALUE

EXCEPTIONS

java.rmi.RemoteException - in case of error with LDAP connection

4.2.4.2 *deleteApplicationInfo*

```
public void deleteApplicationInfo(java.lang.String id, java.lang.String userId, byte[] input)
```

PARAMETERS

id – application id
userId – user id (created basing on credential)
input – table of bytes with credential

EXCEPTIONS

java.rmi.RemoteException - in case of error with LDAP connection/operation

4.2.4.3 *getAllApplicationInfo*

```
public org.crossgrid.wp3.portals.roamingaccessserver.applicationinfows.ApplicationInfo[]  
getAllApplicationInfo()
```

RETURN VALUE

Array of all application info details

EXCEPTIONS

java.rmi.RemoteException - in case of error with LDAP connection

4.2.4.4 *getApplicationInfo*

```
public org.crossgrid.wp3.portals.roamingaccessserver.applicationinfows.ApplicationInfo  
getApplicationInfo(java.lang.String appId, java.lang.String projectId)
```

PARAMETERS

appId – application id
projectId – project id

RETURN VALUE

application info details

EXCEPTIONS

java.rmi.RemoteException - in case of error with LDAP connection

4.2.4.5 *getApplicationInfoById*

```
public org.crossgrid.wp3.portals.roamingaccessserver.applicationinfos.ApplicationInfo  
getApplicationInfoById( java.lang.String id)
```

PARAMETERS

Id – application id

RETURN VALUE

application info details

EXCEPTIONS

java.rmi.RemoteException - in case of error with LDAP connection

4.2.5 Virtual Directory Web Service

Virtual Directory concept extends DataGrid mechanism for grid file management. From technical point of view it is MySQL database with user-friendly hierarchical filesystem mapped to guid's. All operations performed on Virtual Direcorey can be divided into two cases: file operations that are based entirely on Replica Manager client, currently LCG_UTILS package and on directory operations that are MySQL queries.

The first function that should be called is getRoots that for given user returns root element(NodeElem structure). Having this root it is possible to perform function like getDirectoryList that return all elements in root.

There is also one additional method public listAvailableSE that return all currently available Storage Elements for virtual organization that user belongs to.

Virtual Directory operates on elements called NodeElem that represents all possible types of nodes.

All NodeElement are identified unique only by id. Names are not unique so it is possible that in one directory two different nodes would have the same name. To avoid such situation some restriction should be introduced on the level of presentation level.

4.2.5.1 *getRoots*

A function that is intended for the use at the beginning of work on Virtual Directory.

```
public java.util.Vector getRoots(java.lang.String userId,  
    String projectId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

projectId – deprecated parameter, can be null

input – table of bytes with credential

RETURN VALUE

org.crossgrid.wp3.portals.common.NodeElem in Vector as a first element. This is because there was assumption that for one Virtual Directory there can be more the one root (example: for shared resources)

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.2 *getMetaElem*

This function allows getting information about NodeElem by giving its id

```
public org.crossgrid.wp3.portals.common.NodeElem getMetaElem(String elementID,  
    String userId, byte[] input)
```

PARAMETERS

elementID - element id that we are asking for

userId – user id (created basing on credential)

projectId – deprecated parameter, can be null

input – table of bytes with credential

RETURN VALUE

org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.3 *getNodeElemPath*

Virtual Directory path indicating Elements

```
public String getNodeElemPath(java.lang.String userId,  
    String elemId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

elemId - element id that we are asking for

input – table of bytes with credential

RETURN VALUE

Path to element like /root/myfiles/file1. This path doesn't exist physically or even logically, it is created dynamically basing on MySQL table. This path is created starting from the leaf and visiting parents of current one as long as parent is null that mean it is root element for given user.

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.4 *getPathAsNodeElems*

Function that returns path to elemId as NodeElems

```
public Vector getPathAsNodeElems(java.lang.String userId,  
    String elemId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

elemId - element id that we are asking for

input – table of bytes with credential

RETURN VALUE

Path to element like /root/myfiles/file1 presented as an array of successive NodeElems . This path doesn't exist physically or even logically, it is created dynamically basing on MySQL table. This path is created starting from the leaf and visiting parents of current one as long as parent is null that mean it is root element for given user.

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.5 *addFileLogical*

This function adds file to data management system. It is added only as a logical object without mapping to any physical localisation

```
public org.crossgrid.wp3.portals.common.NodeElem addFileLogical(  
    java.lang.String fileName, String currentDirId, java.lang.String userId, byte[] input, String url)
```

PARAMETERS

fileName - name of file added to Virtual Directory
currentDirId - id of directory that file is added to
userId – user id (created basing on credential)
input – table of bytes with credential

RETURN VALUE

org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.6 *changeUserName*

This function allows change file name (only logical operation, affects Virtual Directory).

```
public org.crossgrid.wp3.portals.common.NodeElem changeUserName(  
    java.lang.String newFileName, String fileId, java.lang.String userId,  
    byte[] input)
```

PARAMETERS

newFileName - new name of file
fileId - file identifier
userId – user id (created basing on credential)
input – table of bytes with credential

RETURN VALUE

org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.7 removeUserFile

This function removes user file logical and all his physical location.

```
public int removeUserFile(String fileId, java.lang.String userId, byte[] input)
```

PARAMETERS

fileId - file identifier

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

OPERATION_SUCCESSFUL or OPERATION_FAILED

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.8 removeUserFileLocation

This function allows removing file from given location.

```
public int removeUserFileLocation(String fileId, java.lang.String fileLocation, java.lang.String userId, byte[] input)
```

PARAMETERS

fileId - file identifier

fileLocation – URL of file location that should be removed

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

OPERATION_SUCCESSFUL or OPERATION_FAILED

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.9 makeUserFileLocation

This function allows getting information where to save file physical and it returns exact URL.

```
public java.lang.String makeUserFileLocation(java.lang.String fileId, long fileSize, java.lang.String protocolType, long timePeriod, boolean isExactSize, java.lang.String userId, byte[] input)
```

PARAMETERS

fileId - file identifier

fileSize - exact fileSize

protocolType - protocol that will be used

timePeriod - reservation time. After that time file can be accessed

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

String with URL to localization where file should be copied.

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.10 *getUserFileLocation*

This function allows getting information about file physical localisation

```
public String getUserFileLocation(String fileId, String protocolType,  
    long timeLock, String userId, byte[] input)
```

PARAMETERS

fileId - file identifier

protocolType - protocol that will be used

timeLock - reservation time. File s locked for any other operation // currently not used

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

String with URL to physical localization of file.

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.11 *getUserFileLocations*

This function allows getting information about file physical localisations

```
public String[] getUserFileLocations(String fileID, String userId,
```

byte[] input) throws java.rmi.RemoteException

PARAMETERS

fileId - file identifier

protocolType - protocol that will be used

timeLock - reservation time. Files locked for any other operation // currently not used

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

Array of URLs (that are physical localization of files).

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.12 confirmUserFileUpload

That function confirms successful upload of file. State of file is updated.

```
public org.crossgrid.wp3.portals.common.NodeElem confirmUserFileUpload(  
    String fileId, String userId, String destinationURL, byte[] input)
```

PARAMETERS

fileId - id of file that is updated

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

Updated file as org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.13 updateFileInfo

This function updates in Virtual Directory information about file like size, modification date

```
public void updateFileInfo(  
    org.crossgrid.wp3.portals.common.NodeElem nodeElem, java.lang.String userId, byte[] input)
```

PARAMETERS

nodeElem – NodeElem with newer information to be updated
userId – user id (created basing on credential)
input – table of bytes with credential

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.14 replicateFile

```
public java.lang.String replicateFile(java.lang.String fileId, java.lang.String protocolType,  
java.lang.String hostname, java.lang.String userId, byte[] input)
```

PARAMETERS

fileId – id of file that should be replicated
protocolType – used protocol
hostname – which host should be used for replication
userId – user id (created basing on credential)
input – table of bytes with credential

RETURN VALUE

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.15 addAlias

This function is created for compatibility with DataGrid File Management System. This aliases can be used with EDG command line commands on UI.

```
public java.lang.String addAlias(java.lang.String newAlias, java.lang.String fileId, java.lang.String  
userId, byte[] input)
```

PARAMETERS

newAlias – alias
fileId – id of file
userId – user id (created basing on credential)
input – table of bytes with credential

RETURN VALUE

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.16 registerFiles

```
public java.util.Vector registerFiles(java.lang.String userId,  
    java.util.Vector filesArray, String currentDirId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

filesArray -

currentDirId -

input – table of bytes with credential

RETURN VALUE

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.17 registerDirectory

```
public java.util.Vector registerDirectory(java.lang.String userId,  
    boolean subDirectories, String currentDirId, byte[] input, String url)
```

PARAMETERS

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.18 addUserDirectory

This function adds directory to Virtual Directory. (Operation is equivalent to "mkdir").

```
addUserDirectory(java.lang.String    directoryName,    java.lang.String    currentDirectoryId,  
java.lang.String userId, byte[] input)
```

PARAMETERS

directoryName - directory name

currentDirectoryId - id of directory in which you want to create new directory

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

Added directory as org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.19 changeUserDirectoryName

This function changes name of directory. This change involves only Virtual Directory (logical)

```
public org.crossgrid.wp3.portals.common.NodeElem changeUserDirectoryName(  
    java.lang.String newDirectoryName, String directoryId, java.lang.String userId, byte[] input)
```

PARAMETERS

newDirectoryName - new name of directory

fileId - directory identifier

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

org.crossgrid.wp3.portals.common.NodeElem with all information like creation date, modification date, size, name

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.20 removeUserDirectory

This function deletes user directory and all files and subdirectories.

```
public int removeUserDirectory(String directoryId, java.lang.String userId, byte[] input)
```

PARAMETERS

directoryId – identifier of directory that should removed

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

return OPERATION_SUCCESSFUL or OPERATION_FAILED

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.21 *getDirectoryList*

List directory pointed by rootId parameter.

```
public java.util.Vector getDirectoryList(java.lang.String userId,  
    String rootId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

rootId – identifier of directory that should be list

input – table of bytes with credential

RETURN VALUE

Vector containing list of NodeElems that represents contents of directory (similar to ‘ls’)

EXCEPTIONS

There are number of MySQL exceptions (connection error, MySQL wrong state error)

4.2.5.22 *listAvailableSE*

This function is based on information index. It retrieves list of all Storage Elements for user (VO).

```
public java.util.Vector listAvailableSE(java.lang.String userId, byte[] input)
```

PARAMETERS

userId – user id (created basing on credential)

input – table of bytes with credential

RETURN VALUE

Vector containing list of Storage Elements available for current user/current VO

EXCEPTIONS

There are number of Information Index/LDAP exceptions



5 PRODUCT TESTING

Ras/MD system was tested in details during development phase. Two ways of testing can be distinguished:

- validation tests – product was a subject of several exhausted “validation” procedures;
- “Real life” tests. – almost from the beginning of Crossgrid operation version of MD/RAS was available to X# members. Everyday work of X# community of users with MD/RAS was an excellent occasion to test system in real heterogeneous “grid” environment, on different software and hardware platforms.

Most of MD features strongly dependents on RAS web services, so they cannot be checked in test environment (without connection to working RAS server). From the other side, testing MD means in fact testing also RAS functionality and API. It is important because due to complex nature of RAS API input parameters, RAS itself cannot be tested automatically using e.g. JUnit tests.

MD/RAS tester should follow test procedures described in one of the CrossGrid deliverables [A6].

6 CONTACT INFORMATION AND CREDITS

For more information ore problems, please contact the developers.

Mirosław Kupczyk miron@man.poznan.pl

Rafał Lichwała syriusz@man.poznan.pl

Bartosz Palak bartek@man.poznan.pl

Marcin Płóciennik marcinp@man.poznan.pl

Paweł Wolniewicz pawelw@man.poznan.pl

7 EDG LICENSE AGREEMENT

Copyright (c) 2005 CrossGrid. All rights reserved.

This software includes voluntary contributions made to the CrossGrid Project. For more information on CrossGrid, please see <http://www.eu-crossgrid.org>.

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.

2. The user documentation, if any, included with a redistribution, must include the following notice: “This product includes software developed by the CrossGrid Project (<http://www.eu-crossgrid.org>).”

Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.

3. The names “CrossGrid” and “CG” may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by cgooffice@cyfronet.krakow.pl.

4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features, functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in the CrossGrid Project a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

5. DISCLAIMER

THIS SOFTWARE IS PROVIDED BY THE CROSSGRID PROJECT AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. THE CROSSGRID PROJECT AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

6. LIMITATION OF LIABILITY

THE CROSSGRID PROJECT AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.