DEVELOPER MANUAL

APPLICATION PORTAL

WP3.1

Document Filename:	**CG3.1-v1.0-ALGO-ApplicationPortalDeveloperManual**
Work package:	**WP3.1 Portal**
Partner(s):	**ALGO**
Lead Partner:	**ALGO**
Config ID:	**CG3.1-v1.0-ALGO-ApplicationPortalDeveloperManual**
Document classification:	**PUBLIC**

Document Log

Version	Date	Summary of changes	Author
1.0	5/01/2005	First draft version	Miltos Kokkosoulis

CONTENTS

1 COPYRIGHT NOTICE 4

2 INTRODUCTION 5

2.1 OBJECTIVES OF THIS DOCUMENT 5
2.2 APPLICATION AREA 5

3 APPLICATION PORTAL - A JAVA WEB-BASED INTERFACE 6

4 CONTACT INFORMATION AND CREDITS 12

5 EDG LICENSE AGREEMENT 13

1 COPYRIGHT NOTICE

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

2 INTRODUCTION

2.1 OBJECTIVES OF THIS DOCUMENT

This document describes the Application Portal from the developer's point of view. It aims at providing the developer with all necessary information in order to deploy and set up the portal correctly as well as understand its architecture and its inner structure and mechanism.

2.2 APPLICATION AREA

The CrossGrid Application Portal is based on the portlets framework, which basically provides the developers with the ability to plug extensions (portlets) into the portal. The main idea behind this framework is that "windows" to grid services are created and they are presented in a user-friendly manner to the potential users. The portal makes use of the Roaming Access Server (RAS) machine and the Job Submission Services (JSS), which are installed on that machine.

3 APPLICATION PORTAL - A JAVA WEB-BASED INTERFACE

The Application Portal is a front-end web interface built on Java, XML, HTML and JavaScript. It aims at providing an entry-point to Grid services for the average user. The Portal interface is designed in a user-friendly way and it has been taken into consideration that the users will not necessarily have any knowledge of IT or computer programming. All technical details have been hidden and there are comments inside the various portlets' areas that offer complete guidance to the users.

The structure of the directory tree containing the portal code in the FZK CVS repository is:

(under http://gridportal.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/)

wp3_1-portals/portal_jetspeed/src/org/apache/jetspeed/modules/actions/portlets is where the code for all the portlets is put (.java files)

wp3_1-portals/portal_jetspeed/src/org/apache/jetspeed/modules/actions/portlets/common is where the PortalConfig.java is put, which is needed by the other java files and it is imported in them at the beginning of their code lines.

wp3_1-portals/portal_jetspeed/conf is where the .xreg files for all the portlets are. These are used by a Jetspeed portal mechanism in order to identify the portlets that are registered with the portal system and make them available for the user to select them and see them.

wp3_1-portals/portal_jetspeed/templates is where the .vm files (velocity modules) for all the portlets are. These files are the ones that contain the HTML code for each one of the portlets and determine how the whole interface will look.

wp3_1-portals/portal_jetspeed/lib is where the necessary jar files can be found.

wp3_1-portals/portal_jetspeed/scripts is where the script files for some applications are located (for example, flood application script files, WAM-sea wave model application script files, etc.)

wp3_1-portals/portal_jetspeed/sql is where the “tables.sql” is located, which is used for the creation of the submitted jobs database.

There follows a list of all the java files and the methods that are included in them.

The first one is the PortalConfig.java, which is located in the portlets/common directory. This file contains all those methods that are common and necessary for the other java files, each of which corresponds to a particular portlet.

PortalConfig.java

static public String getStatusString(String job_id) throws Exception → the status as a string

static public String getDoneString(String job_id) throws Exception → the Done flag

static public boolean getjobStatus(String job_id, GSSCredential proxy) throws Exception → the status of a submitted job

static public void check4UserDir(RunData rundata, GSSCredential proxy) throws Exception → it checks for the user’s directory (based on whether the user’s system is Linux-based or Windows-based

static public String gethost() throws Exception → it retrieves the host

static public int getport() throws Exception → it retrieves the port

static public String getWShost() throws Exception → the Web Services host

static public int getWSport() throws Exception → the port for the Web Services host

static public String getRBhost() throws Exception → the Resource Broker host

static public int getRBport() throws Exception → the port for the Resource Broker host

static public String getLBhost() throws Exception → the host for Logging and Bookkeeping

static public int getLBport() throws Exception → the port for LB host

static public String getTargetURL(GSSCredential proxy) throws Exception → it forms the path to which the file(s) will be transferred.

static public String getUserHome() throws Exception → the user's home directory is determined based on the user's operating system

static public String getUserName(GSSCredential proxy) throws Exception → it determines the user's name with the use of grid-mapfile

static public byte[] getGlobusProxyAsByteArray (GSSCredential gssCred) throws GlobusCredentialException, GSSException

static public String execCommand(String commandString) throws Exception

airpollutionappAction.java

protected void buildNormalContext(VelocityPortlet portlet,

Context context,

RunData rundata)

public void doSubmit(RunData rundata, Context context) throws Exception → used for the submission of an air pollution application job to the Grid and the action that takes place as soon as the submit button in the portlet area is pressed

public void doGetresults(RunData rundata, Context context) throws Exception → it transfers the results from the physical location in which they were produced and exist there, to the portal system

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet (after the job submission has returned with a specific Job ID) as soon as the respective button is pressed

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table.

floodappAction.java

protected void buildNormalContext(VelocityPortlet portlet,

Context context,

RunData rundata)

public void doSubmit(RunData rundata, Context context) throws Exception → this is used for the submission of a flood application job to the Grid

public void doSubmit_vis(RunData rundata, Context context) throws Exception → it submits a visualization job for any of the jobs that have previously returned with numerical data and results

public void doGetresults(RunData rundata, Context context) throws Exception → it is used for the retrieval of the numerical results

public void doGetresults_vis(RunData rundata, Context context) throws Exception → it is used for the retrieval of the visualization results

protected void load_davef_param(Context context, RunData rundata, GSSCredential proxy) throws Exception → this method loads specific parameters for the davef model

protected void load_davef_vis_param(Context context, RunData rundata, GSSCredential proxy) throws Exception → it loads parameters for the davef visualization process

protected void save_davef_vis_param(Context context, RunData rundata, GSSCredential proxy) throws Exception → it is used for saving the visualization parameters to a file

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet (after the job submission has returned with a specific Job ID) as soon as the respective button is pressed

protected List getRows(RunData rundata, boolean vis) throws Exception → this is used in order to formulate the results in a listing table.

protected boolean getjobStatus(RunData rundata, String jobname, boolean vis) throws Exception

public class Filter implements FilenameFilter

public boolean accept(File dir, String name)

jobgetoutputAction.java

protected void buildMaximizedContext(VelocityPortlet portlet,
Context context,
RunData rundata)

buildNormalContext(portlet, context, rundata);

protected void buildNormalContext(VelocityPortlet portlet,
Context context,
RunData rundata)

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

public void doSubmit(RunData rundata, Context context) throws Exception → it is used in order to connect the job submission portlet and the specific job ID with the output

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet (after the user has observed the results for the specific Job ID) as soon as the respective button is pressed

joblistmatchAction.java

protected void buildNormalContext(VelocityPortlet portlet,
Context context,
RunData rundata)

public void SubmitJDL(RunData rundata, Context context, String jdl) throws Exception → it is used for the submission of a JDL script (Job Description Language parameters).

public void doFilesubmit(RunData rundata, Context context) throws Exception → it is used for the submission of a file

public void doSubmit(RunData rundata, Context context) throws Exception → it performs a simple submission to the Grid

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

protected void buildMaximizedContext(VelocityPortlet portlet,
Context context,
RunData rundata)
buildNormalContext(portlet, context, rundata);
protected void buildNormalContext(VelocityPortlet portlet,
Context context,
RunData rundata)

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

public void doSubmit(RunData rundata, Context context) throws Exception → it performs a simple submission to the Grid

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

jobloginfoAction.java

protected void buildMaximizedContext(VelocityPortlet portlet,
Context context,
RunData rundata)
buildNormalContext(portlet, context, rundata);
protected void buildNormalContext(VelocityPortlet portlet,
Context context,
RunData rundata)

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

public void doSubmit(RunData rundata, Context context) throws Exception → it is used in order to connect the job submission portlet with the log info of the specific job ID

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

jobstatusAction.java

protected void buildMaximizedContext(VelocityPortlet portlet,
Context context,
RunData rundata)
buildNormalContext(portlet, context, rundata);
protected void buildNormalContext(VelocityPortlet portlet,
Context context,
RunData rundata)

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

public void doSubmit(RunData rundata, Context context) throws Exception → it is used in order to connect the job submission portlet with the status of the submitted job

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

jobsubmitAction.java

protected void buildNormalContext(VelocityPortlet portlet,

Context context,

RunData rundata)

public void SubmitJDL(RunData rundata, Context context, String jdl) throws Exception → this method is used for submitting a JDL script

public void doFilesubmit(RunData rundata, Context context) throws Exception → it submits a file

public void doSubmit(RunData rundata, Context context) throws Exception → this method is activated as soon as the "submit" button is pressed

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

neuralappAction.java

protected void buildNormalContext(VelocityPortlet portlet,

Context context,

RunData rundata)

public void doSubmit(RunData rundata, Context context) throws Exception → it performs a submission of a neural network application job to the Grid

public void doGetresults(RunData rundata, Context context) throws Exception → it retrieves the results from their location to the portal server

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

WAMappAction.java

protected void buildNormalContext(VelocityPortlet portlet,

Context context,

RunData rundata)

public void doSubmit(RunData rundata, Context context) throws Exception → used for the submission of a WAM (sea-wave modeling application) job to the Grid and the action that takes place as soon as the submit button in the portlet area is pressed

public void doGetresults(RunData rundata, Context context) throws Exception → it retrieves the results from their location to the portal server

public void doGoback(RunData rundata, Context context) throws Exception → it is used to return to the introductory screen of the portlet as soon as the respective button is pressed

protected List getRows(RunData rundata) throws Exception → this is used in order to formulate the results in a listing table

protected void save_WAM_params(Context context, RunData rundata, GSSCredential proxy) throws Exception → it saves the WAM application job's parameters to a file

There are .vm files that are connected with the respective .java files internally: airpollutionapp.vm, floodapp.vm, jobgetoutput.vm, joblistmatch.vm, jobloginfo.vm, jobstatus.vm, jobsubmit.vm, neuralapp.vm, WAMapp.vm

These files include the HTML code as well as some JavaScript code where necessary. They are responsible for the way the portlets are displayed inside the portal page.

There are also .xreg files which are used for registering the portlets into the portal system.

4 CONTACT INFORMATION AND CREDITS

For further information, the responsible team persons for the development and support of the Application Portal are:

Miltos Kokkosoulis (mkokkos@algosystems.gr)

Yannis Perros (yperros@algosystems.gr)

5 EDG LICENSE AGREEMENT

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 cgoffice@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.