Heliophysics Integrated Observatory

Project No.: 238969
Call: FP7-INFRA-2008-2

HELIO Processing Service (HPS) User Manual

Version 0.3

Title:	**Community Interaction Service – User Manual**
Document No.:	HELIO_TCD_S2_003_UM_HPS_User_Guide
Date:	01 October 2012
Editor:	**Dr. Gabriele Pierantoni**, Trinity College Dublin
Contributors:	
Distribution:	Project

Revision History

Version	Date	Released by	Detail
0.1	25/06/2012	Gabriele Pierantoni	First Draft
0.2	03/07/2012	Anja Le Blanc	Added workflow section
0.2	09/08/2012	Gabriele Pierantoni	Added SOAP Methods
0.3	01/10/2012	Gabriele Pierantoni	Minor changes

Note: Any notes here.

Table of Figures

Introduction

HELIO Processing Service (or HPS) offers services for the execution of applications on different computational resources.

About the HPS

The HELIO Processing Service (HPS) is an interface to different computational resources to execute pre-defined and user-defined sets of applications.

For the moment being, HPS supports two different computational resources:

- a fast execution engine suitable for short lived jobs and,
- a more powerful grid-based resource for longer running and parallel jobs.

According to its dual nature, the HPS can be deployed in two modalities as sketched in Figure 1

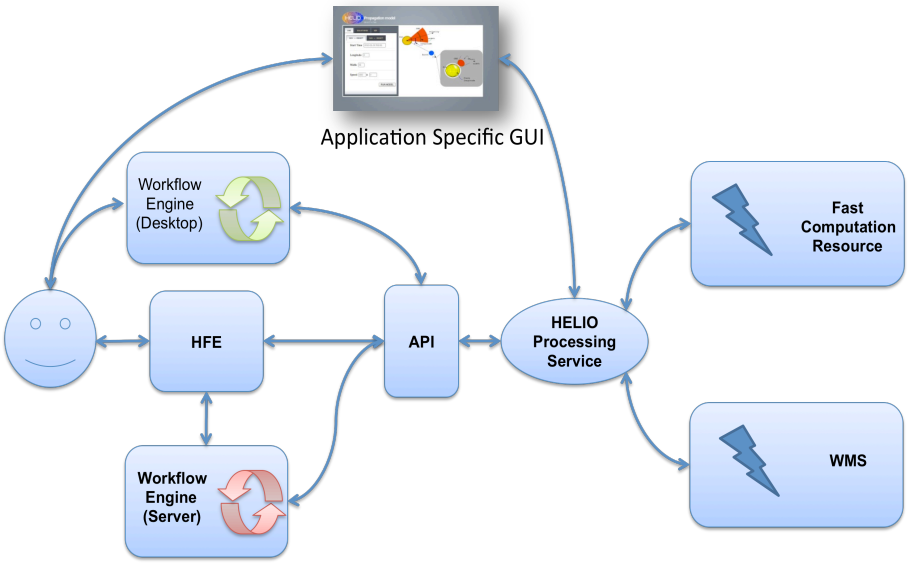

Figure 1, Overview of the HPS.

How to Access the HPS

The service is now accessible at http://cagnode58.cs.tcd.ie:8080/helio-hps-server/

The Graphical User Interface

The HPS exposes two different kinds of interfaces: a programmatic generic interface used to submit generic, user-defined jobs and graphical specific interfaces to different publicly available applications.

Propagation Model Interface

At the moment, the HPS exposes only one specific application interface, the Propagation Model GUI. To execute the Propagation Model, the user does not need any authentication information. To access the Propagation Model, the user has to access http://cagnode58.cs.tcd.ie:8080/PropagationModelGUI/.

The Propagation Model interface supports 6 different types of computation:

- **Forward CME**: Models the propagation of Coronal Mass Ejections from the sun to the other objects in the solar system.
- **Backward CME**: Models the propagation of Coronal Mass Ejections from a selected object of the solar system backward to the sun.
- **Forward Solar Wind**: Models the propagation of Solar Wind from the sun to the other objects in the solar system.
- **Backward Solar Wind**: Models the propagation of Solar Wind from a selected object of the solar system backward to the sun.
- **Forward SEP**: Models the propagation of Solar Energetic Particles from the sun to the other objects in the solar system.
- **Backward SEP**: Models the propagation of Solar Energetic Particle from a selected object of the solar system backward to the sun.

Selecting models and data

All the available operations are selected with the tabs on the two top layers as shown from Figure 2 to Figure 7.

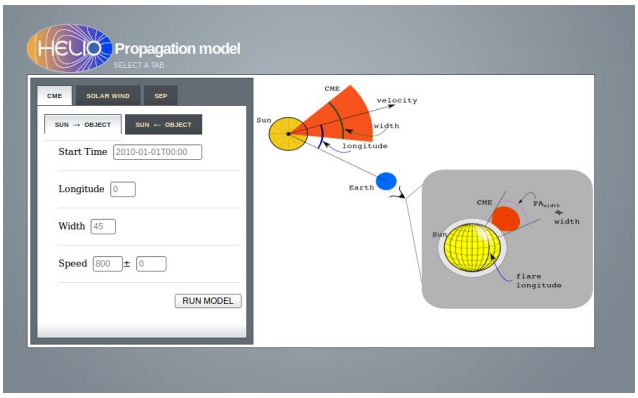

Figure 2, Forward CME Propagation Model

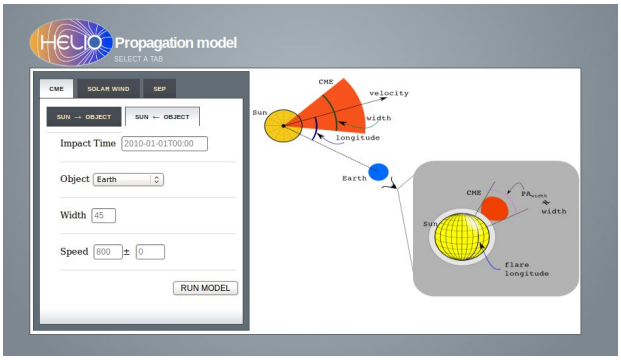

Figure 3, Backward CME Propagation Model

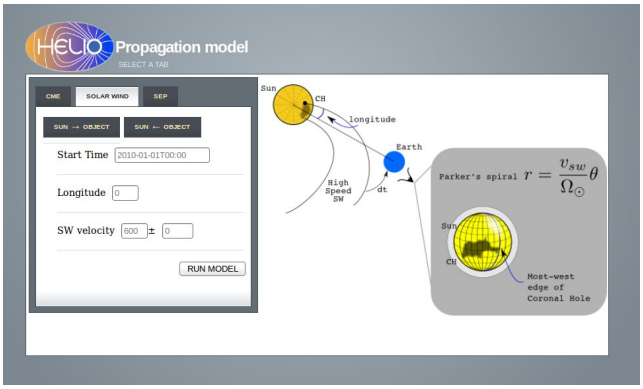

Figure 4, Forward Solar Wind Propagation Model

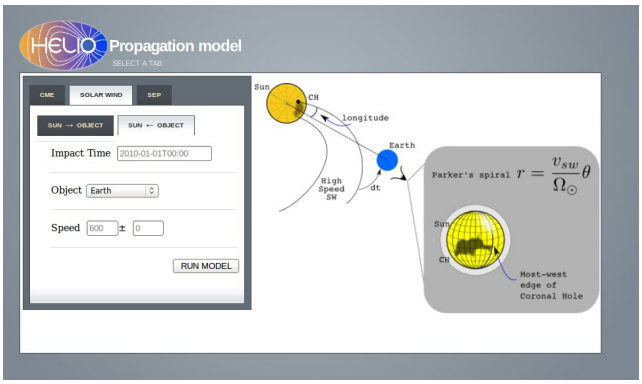

Figure 5, Backward Solar Wind Propagation Model

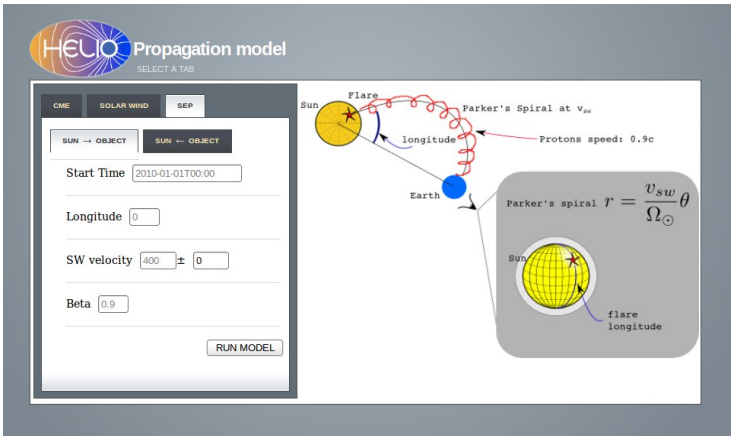

Figure 6, Forward Solar Energetic Particles Propagation Model

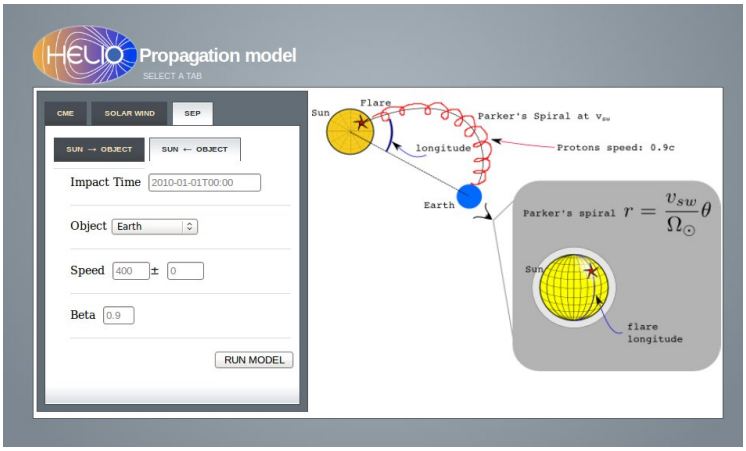

Figure 7, Backward Solar Energetic Particle Propagation Model

Accessing the results

After the model and data are selected, the screen of Figure 8 is shown when the results are available. By clicking on the link here at the bottom of the page, the user is re-directed to the output of the results (these results are managed by the HELIO Storage Service).

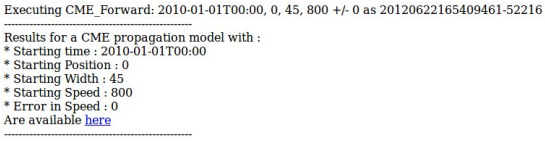

Figure 8, Screen showing the availability of the results of the Propagation Model

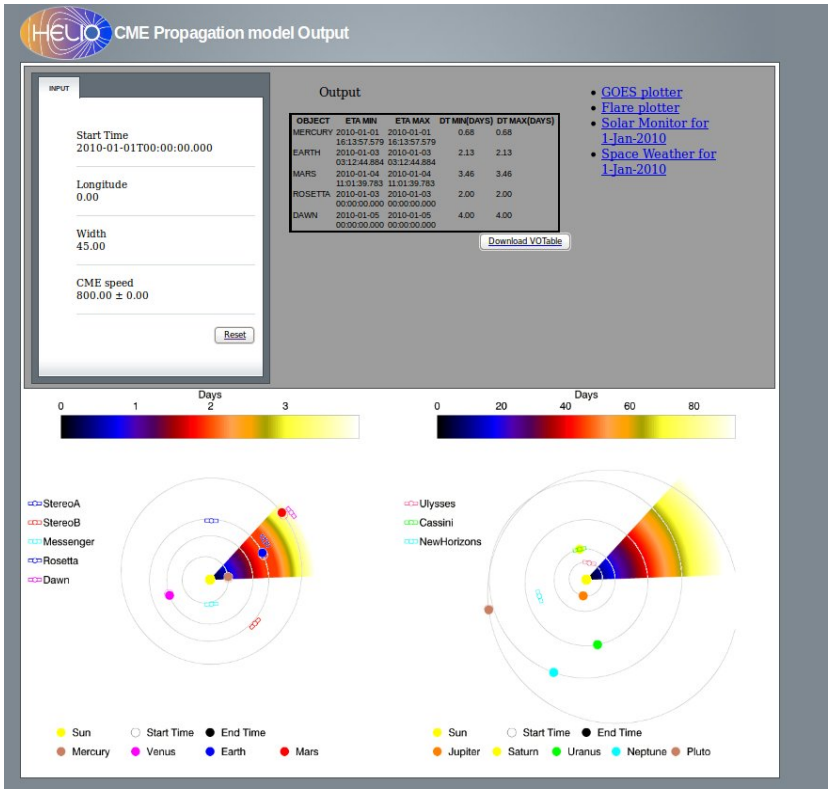

Figure 9, Results of a Forward CME Propagation Model

One such result is shown in Figure 9.

HPS SOAP Interface

The HPS functionalities are also accessible through SOAP protocol using the WSDL present at http://cagnode58.cs.tcd.ie:8080/helio-hps-server/ hpsService?wsdl.

SOAP function	Description
executeApplication	Selects application, sets required parameters, and starts computation
getOutputOfExecution	Returns the location where HPS has saved the output
getPresentApplications	Returns all the applications that are present in the application library of the HPS
getStatusOfExection	Returns the status of the execution. The HPS status can be of three different values: Running, Done and Failed. The status Running and Failed encompass different status on the Grid backend.
test	A simple test to control that the service is running.

To run user defined code on the secured Grid infrastructure the user has to use the following SOAP interface:
http://cagnode58.cs.tcd.ie:8080/helio-hps-server-ws/hpsService?wsdl

SOAP function	Description
test	A simple test to control that the service is running.
putFile	Uploads a file (that can be later executed) to the HPS, the Helio Identity Token has to be passed as parameter in this function.
executeUserDefinedApplication	Executes the defined file, the Helio Identity Token has to be passed as parameter in this function.
getStatusOfExecution	Returns the status of the execution. The HPS status can be of three different values: Running, Done and Failed. The status Running and Failed encompass different status on the Grid backend.
getOutputOfExecution	Returns the location where HPS has saved the output

Sample Workflows

HPS service implements an asynchronous service environment only, that means that there are several steps required to run a application on the HPS.

- Submit a job
- Check status of this job
- Retrieve results once job is completed

How to Use the HPS

Request available applications

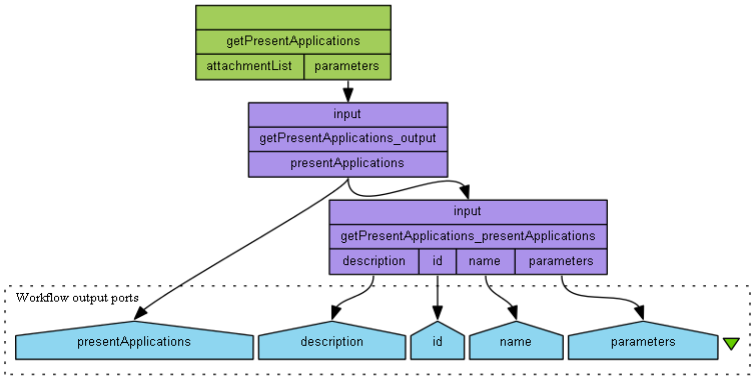

Figure 10: getPresentApplications call in the Taverna Workflow System

The workflow shown in Figure 10 requests all available applications which do not require a CIS authentication to execute. The result contains the name of the application, a description and the details of all parameters to execute it.

Execute an application

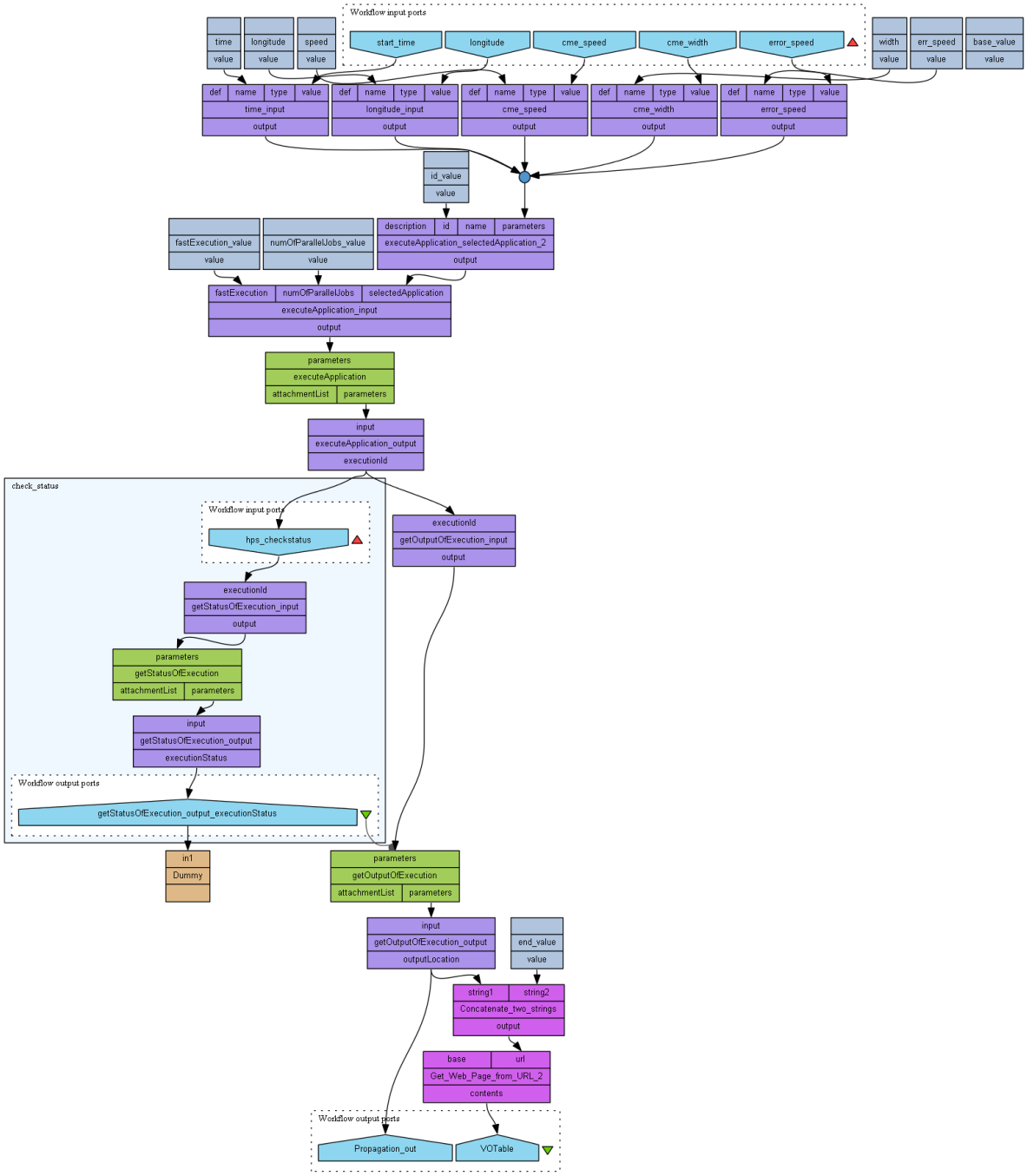

Figure 11: Workflow which runs a CME forward propagation on the HPS

Figure 11 shows a workflow which performs a forward propagation of a CME event from the sun. All applications are executed in the same way on the HPS system.

1. executeApplication: it requires that the *fastExecution* is set to true (run on the Fast Execution Resource - unauthenticated), *NumberOfParallelJobs* is set to 1 (for the fast execution), the application can be selected by the ID only (retrieved from output of getPresentApplications), and for each of the parameters the name and the value are required (name as retrieved from output of getPresentApplications). The return value of this function contains an execution ID required in all following steps
2. getStatusOfExecution: requires the execution ID as an input and checks the current status of the execution. Returned values can be either ‘Running’, ‘Completed’, or

'Failed'. getStatusOfExecution has to be repeatedly called until the returned status is 'Completed' or 'Error'

3. If the status of the execution is Completed getOutputOfExecution can be called again with the execution ID as an input parameter. The returned value is application dependent. In case of the propagation model code it is a URL to the result web page. If it is required as a VOTable the result URL can be statically altered.

The execution of all available applications are implemented as workflows and can be retrieved from the myExperiment repository (http://www.myexperiment.org) – shared with group 'helio'.

Execution of user defined code on the Grid

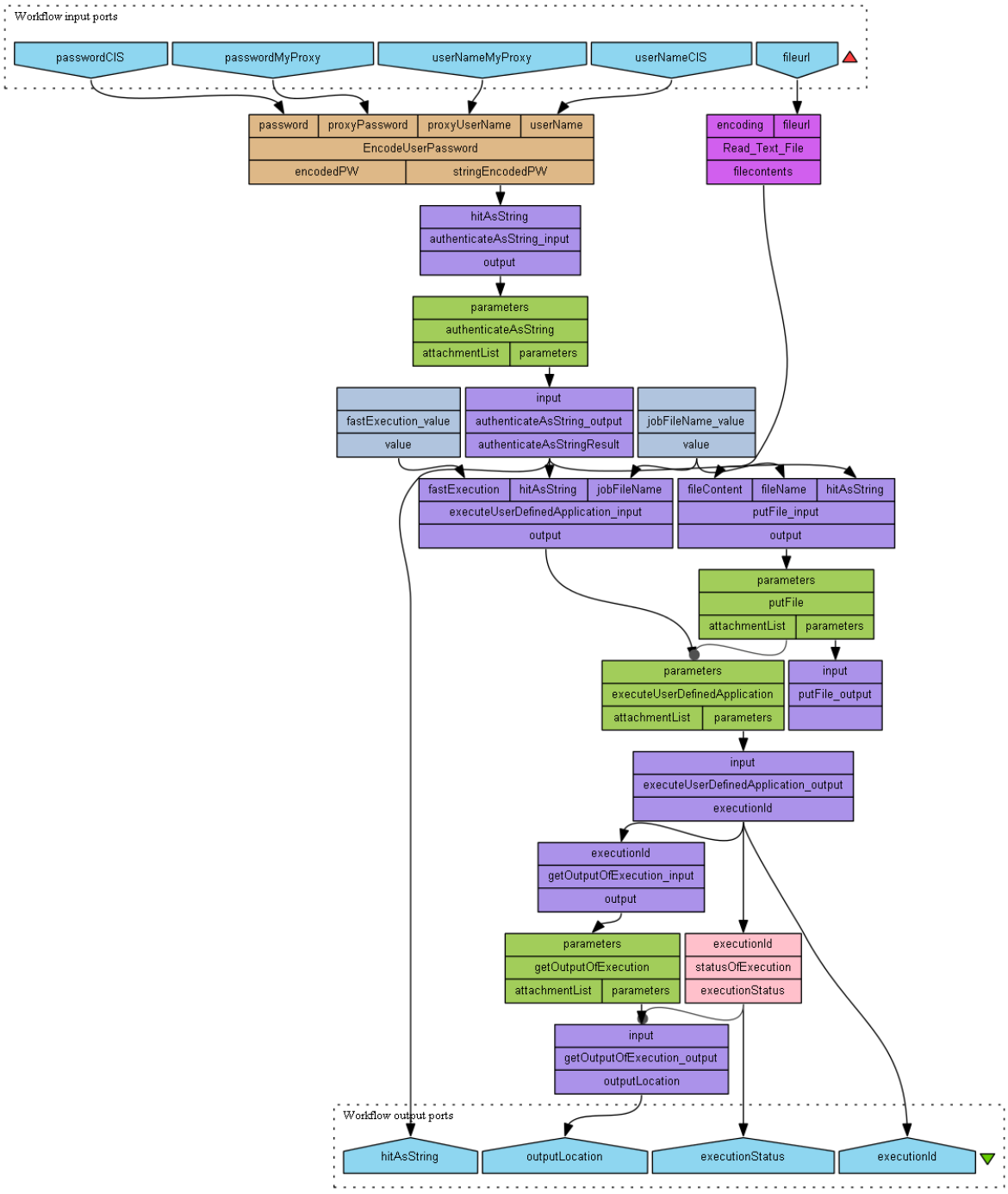

Figure 12: Secured execution of user defined code

Authentication is required to execute any user defined code on the processing service. Please read the documentation about CIS for more information.
After authenticating to the CIS the user receives a string token which is needed to upload files and execute code. The following steps provide information about how to execute user defined code on the processing service:

1. authenticate yourself to the CIS; function ‘authenticateAsString’ the parameter is a string which contains the encoded user name, password, myProxy user name and password

2. upload of executable code; function ‘putFile’ parameters file content – the string of the code (‘fileContent’), the file name of the code (‘fileName’), and the returned security token from the CIS (‘hitAsString’)
3. execute the code on the processing service; function ‘executeUserDefinedApplication’ with the parameters ‘fastExecution’ (supported values: true, false), ‘jobFileName’ that is the name you have given your code in Step 1, and the ‘hitAsString’, the returned security token from the CIS; the returned value is the ‘exectuionId’ which you will need to access your computation.
4. check the status of the execution; function ‘getStatusOfExecution’ the parameter is the execution Id returned by the previous step. This function need to be called until the status has changed from ‘Running’. Please note that the execution on Grid facilities usually requires the queuing of the job until it will be executed. It is not advisable to check very frequently (twice a minute is sufficient).
5. request results; function ‘getOutputOfExecution’ with the input parameter of the execution Id (output of step 2). As result of this operation you receive the URL to the location of your output directory. There are usually two directories in that, one with the results and one with the error message.