Getting Started with the Serpens extension for Kepler-2.3

Working with vine:toolkit module

{marcinp, michalo, tzok}@man.poznan.pl

vine:toolkit module description covers:

Data management using RAS/vine:toolkit
Job management using RAS/vine:toolkit
Example workflows
Template workflow for job submission

This paper covers Serpens 2.4 release

January 30, 2012

Getting Started with the Serpens – vine:toolkit module

The *Getting Started with the Serpens – vine:toolkit module* guide is for scientists who would like to use the Serpens suite in Kepler.

The Serpens was created to assist scientists who would like to use Kepler to manage GRID based job execution from the level of Kepler-2.3 workflow.

Table of Contents

1. Introduction

The *Getting Started with the Serpens – vine:toolkit module* guide introduces the main components and functionality of the *Serpens – vine:toolkit* module. The *Serpens – vine:toolkit module* is an add-on module suite for Kepler, a software application for the analysis and modeling of scientific data, and it provides functionality for managing data management and job management within GRID environment from the level of Kepler workflow.

1.1 What is vine:toolkit?

From official website http://vinetoolkit.org:

Vine is a modular, extensible Java library that offers developers an easy-to-use, high-level Application Programmer Interface (API) for Grid-enabling applications. Vine can be deployed for use in desktop, Java Web Start, Java Servlet 2.3 and Java Portlet 1.0 environments with ease. Plus Vine supports a wide array of middleware and third-party services, so you can focus on your applications and not lose focus on the Grid!

Supported Middleware and Standards:

- *QosCosGrid*
- *BES*
- *gLite3*
- *Globus Toolkit 4.0.x, 4.2.1*
- *GRIA 5.3*
- *Unicore 6*
- *JSDL*
- *Storage Resource Broker*
- *Storage Resource Manager*
- *RUS*
- *OGSA-DAI 2.2*
- *Active Directory*

1.2 What is the Serpens – vine:toolkit module*?*

The *Serpens – vine:toolkit module* provides a way for accessing GRID resources through the Kepler workflow.

The *Serpens – vine:toolkit module* allow users to:

- upload/download data to/from UNICORE storage
- submit jobs into UNICORE
- check statuses of the running jobs
- retrieve *Standard Output* and *Standard Error* of HPC job
- retrieve HPC job outcome

Serpens – vine:toolkit module is based on vine:toolkit, thus it is heavily based on so called Domain concept. You cane read more about Domains at vine:toolkit home page[1]. Simply saying - domain name describes resources

1 http://vinetoolkit.org/content/domainxml-multi-domain-configuration

available for user. Within this document DN or simply domain will refer to Domain Name. It is required by user to know names of domains available at given Vine:toolkit installation.

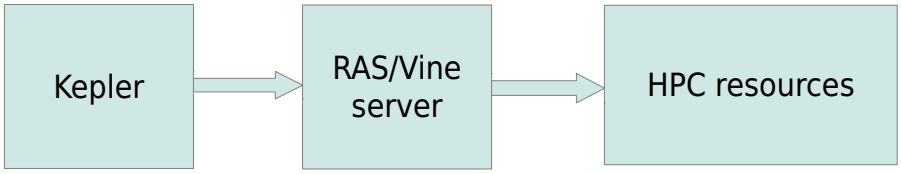
Important, Serpens vine:toolkit module was tested with UNICORE versions up to 6.3.1.

1.3 Accessing HPC resources via RAS/Vine

Serpens - vine:toolkit module is based on RAS - Roaming Access Server. RAS is a middleware that allows external clients to access HPC/GRID resources. Access to resources is granted basing on user's GRID certificates (X.509).

Each actor performing tasks related to HPC access uses RAS to achieve the task.

Architecture of the solution follows:



This architecture allows users to benefit following ways:

1. HPC resources are exposed via RAS/Vine - this way configuration takes place only at server side

2. RAS/Vine provides access to other infrastructures - this way client can benefit from universal GRID/HPC access layer

3. RAS/Vine is able to expose resources that are not available due to security related concerns (e.g. are hidden behind firewall)

4. RAS/Vine allows clients to access resources from behind the firewall (e.g. client is using network from highly secured environment - in that case there is a need for only one, http, port to be opened).

Most of the workflows presented in this documentation use so called ***hostAddressRAS***. This is the address of RAS/Vine server used for communication between client (Kepler) and resources (HPC).

2. Using the Serpens – vine:toolkit module

Once you have installed the Serpens – vine:toolkit module, you will notice new actors available at the Component list.

Actor name	Purpose
RASUploadFile	Uploads file into UNICORE storage via vine:toolkit
RASDownloadFile	Downloads file from the UNICORE storage via vine:toolkit
RASRegisterFile	Registers file at UNICORE storage via vine:toolkit (required for output files)
RASGetError	Retrieves job's error stream
RASGetOutput	Retrieves job's output stream
RASJSDLGenerator	Generates JSDL description of the job basing on input parameters
RASStatusLoop	Checks job's status
RASSubmit	Submits job into UNICORE via vine:toolkit
RASUName	Converts file name relative to job into UNICORE description of file
RASGenerateProxy	Creates user's proxy basing on X.509 certificate
RASVomsProxy	Creates user's VOMS proxy basing on user's proxy.

2.1 Authentication/authorization within Serpens vine:toolkit

Access to HPC resources is restricted and requires authentication/authorization to reach certain resources. Some actors require user to be properly authenticated before an action can take place. These actors are: *RASUploadFile*, *RASDownloadFile*, *RASRegisterFile*, *RASSubmit*, *RASStatusLoop*, *RASGetError*, *RASGetOutput*.

Serpens vine:toolkit authentication is based on VOMS[2] proxy concept. VOMS proxy can be generated using two actors:

- RASGenerateProxy – generates proxy basing on user's X.509 certyficate
- RASVomsProxy – generates VOMS proxy basing on the outcome of RASGenerateProxy actor

2 http://www.globus.org/grid_software/security/voms.php

Example usage of these actors is shown on following picture:

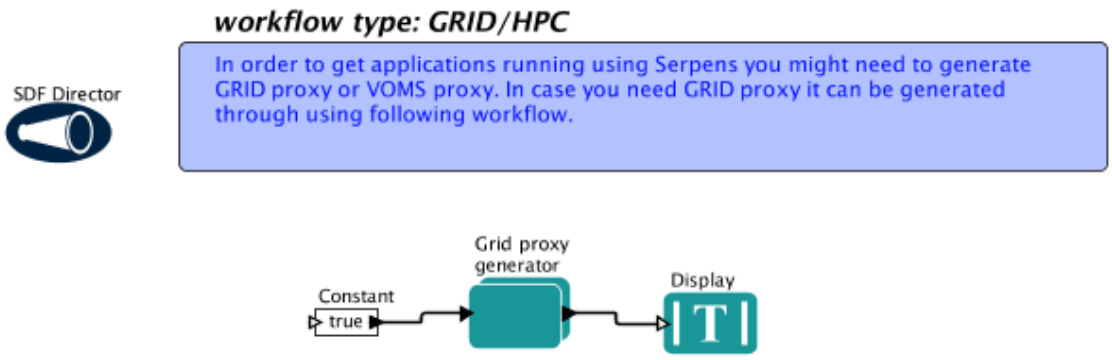


Illustration 1: Generating grid proxy basing on X.509 certificate

This workflow can be found at following location after Serpens demo workflows are installed: *$SERPENS_MODULE/vine-toolkit/demos/gridproxygenerator.xml*[3]. Please note that you will need your X.509 certificate (both certificate and private key) to be located at your $HOME directory. Typically certificate and key files will have following names usercert.pem and userkey.pem respectively.

2.2 Uploading files into UNICORE storage via vine:toolkit

To use external data for processing one must upload data into UNICORE storage to use it during HPC job execution. This can be done using RASUploadFile actor. RASUploadFile actor allows uploading local input files into UNICORE storage specified within vine:toolkit's domain.

NOTE: RASUploadFile/RASDownloadFile actors are much less efficient than UnicoreUploadFile/UnicoreDownloadFile actors. However, they can operate within restricted environment where Internet traffic goes through firewall.

To upload a file into UNICORE storage (using RASUploadFile actor) it is required to provide following information:

Input port name	Description
proxy	String representing user's proxy
domainName	Name of the Domain within vine:toolkit. This parameter allows to change target machine transparently by redirecting vine:toolkit calls to different target registry.
fileName	Local file name. This file will be uploaded into UNICORE storage. Base name will be used as file name at storage.

3 Throughout the document it is assumed that $SERPENS_MODULE points to directory where Serpens module was installed. This is the location where you have checked out Kepler 2.0 and Serpens related modules.

targetDirectory	Target directory at UNICORE storage. If not specified, root directory will be used.
hostAddressRAS	URL of RAS.

2.2.1 RASUploadFile – example workflow

Following example shows example usage of RASUploadFile actor. This workflow will perform following action:

- it will read file from $SERPENS_MODULE/vine-toolkit/demos/input.txt
- it will upload a file into UNICORE storage
- it will display file location at UNICORE storage

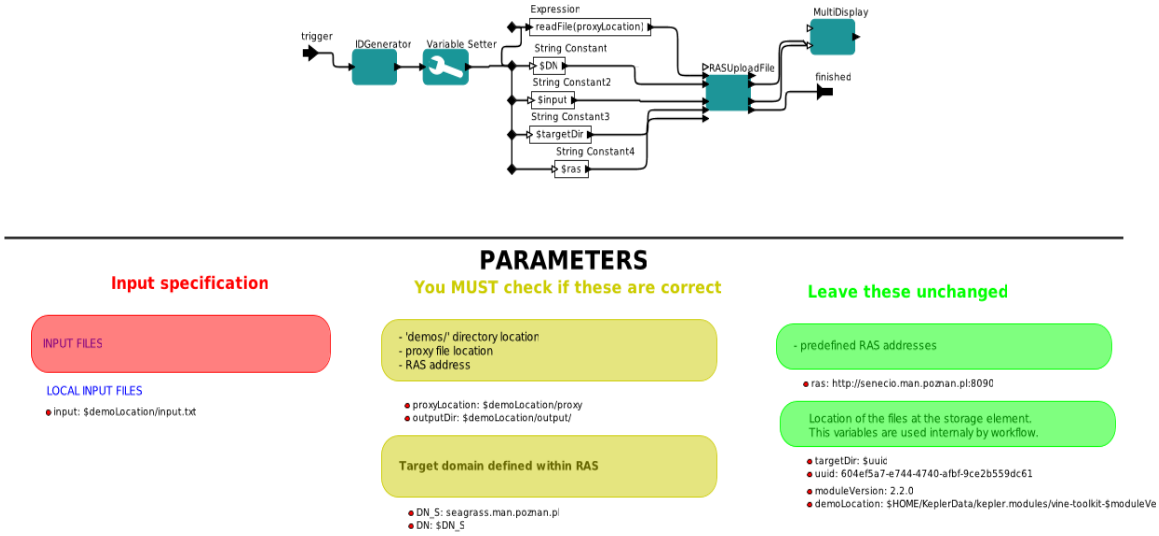


Illustration 2: Uploading local file into UNICORE storage - RASUploadFile

You can find this workflow at following location:

$SERPENS_MODULE/vine-toolkit/demos/vine_upload.xml

2.3 Downloading files from UNICORE storage

Vine:toolkit module allows downloading files from UNICORE storage. RASDownloadFile actor allows user to specify file location at UNICORE storage and download file into local directory. Note, that file location at UNICORE storage is internally stored by vine:toolkit for each domain and each user. Thus, users can operate on files with virtual names. Instead of specifying file's URI one can provide only directory name and file name at UNICORE storage (e.g. /my_data/input_file.txt).

To retrieve file from UNICORE storage, following parameters must be specified:

Input port name	Description
proxy	String representing user's proxy

domainName	Name of the Domain within vine:toolkit. This parameter allows to change target machine transparently by redirecting vine:toolkit calls to different target registry.
fileName	Location of file at UNICORE storage.
targetDirectory	Target directory at local file system (download area)
hostAddressRAS	URL of RAS.

2.3.1 RASDownloadFile – example workflow

Following example shows example usage of RASDownloadFile actor. This workflow will perform following action:

- upload a local file into UNICORE storage
- display file address at UNICORE storage
- download a file from UNICORE storage
- store it within local directory
- display local, downloaded file name

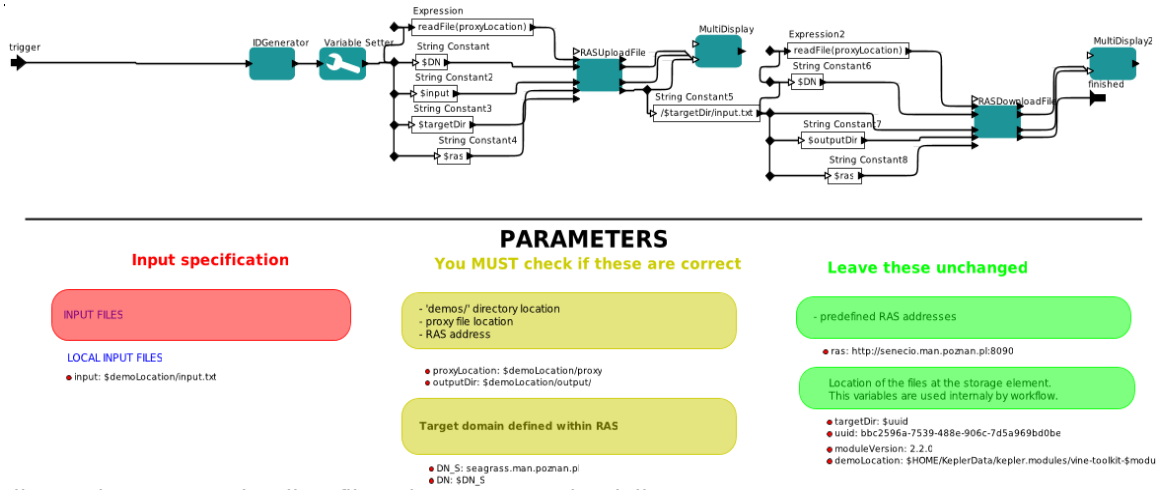


Illustration 3: Downloading file using RASDownloadFile actor

Workflow can be found at:

$SERPENS_MODULE/vine-toolkit/demo/vine_upload_download.xml

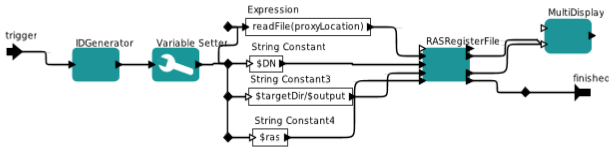
2.4 Registering output files at UNICORE storage

To retrieve files that are generated by HPC job user must specify output location for the files so they can be staged out from job's directory after it is finished. In order to create target location user can register file using RASRegisterFile actor. This actor creates file's URI that describes file location at UNICORE storage. Once created it can be passed to RASJSDLGenerator actor as output file location.

RASRegisterFile actor requires following input parameters:

Input port name	Description
proxy	String representing user's proxy
domainName	Name of the Domain within vine:toolkit. This parameter allows to change target machine transparently by redirecting vine:toolkit calls to different target registry.
fileName	Location of file at UNICORE storage (it can contain parent's names as well, e.g. /my_data/my_job/output_file.txt).
hostAddressRAS	URL of RAS.

As a result it output single String token with URI of the file.



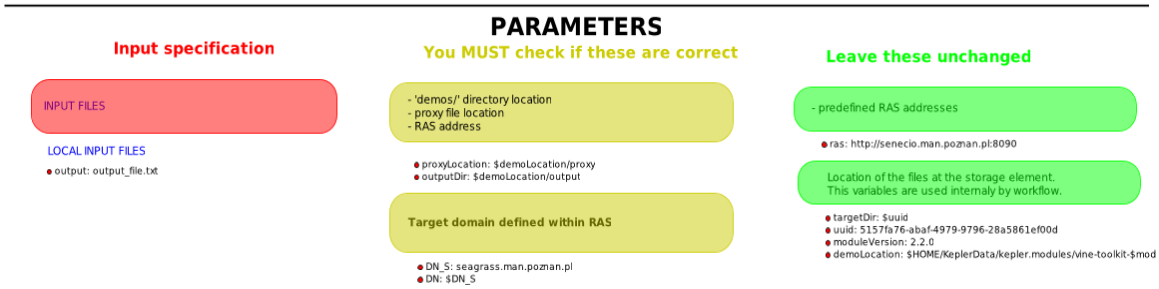


Illustration 4: Registering output file using RAS/Vine

Workflow can be found at:
$SERPENS_MODULE/vine-toolkit/demos/vine_register_file.xml

2.5 Creating JSDL based description of the job - RASJSDLGenerator

To submit a job into UNICORE one must define it via JSDL format. This format allows user to define all the predefined parameters of the job. However, defining JSDL by your own is not convenient – JSDL is an XML subset. In order to provide users with easier way of defining job description Serpens module provides RASJSDLGenerator actor. This actor creates JSDL descritpion of the job basing on input parameters.

Input port name	Description
inputStreamFileName	specifies that given file should be treated as input stream of the application. File must be present in the list of input files
outputStreamFileName	specifies that given file should be treated as output stream of the application. File must be present in the list of output files
inputFileNames	list of file names that will be staged into job's directory
inputFileNamesIURLs	list of input files location at UNICORE storage. NOTE! inputFileName.length must be equal to inputFileNamesIURLs.length
outputFileNames	list of file names that will be staged out from the job's directory
outputFileNamesURLs	list of output file locations at UNICORE storage. NOTE! outputFileName.length must be equal to outputFileNamesIURLs.length
executable	executable name at target HPC machine (NOTE. you can either use full path "/bin/ls" or application name when it is system wide available)
arguments	executable arguments
nodes	number of nodes that will be used (NOTE! meaning of this parameter may be HPC machine specific - it might reserve either cores or whole CPU nodes)
individualCPU	number of CPUs reserved for the application (NOTE! meaning of this parameter may be HPC machine specific - it might reserve either cores or whole CPU nodes)
cpuTime	application's wall time

As a result it outputs JSDL description of the job.

2.5.1 RASJSDLGenerator – example workflow

Below there is presented simple RASJSDLGenerator workflow which generates JSDL description of the job. This job is designed to start /bin/ls application with "-la" arguments (vine_jsdl.xml).

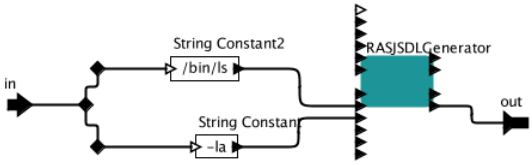


Illustration 5: RASJSDLGenerator example

Workflow can be found at:

$SERPENS_MODULE/vine-toolkit/demos/vine_jsdl.xml

2.6 Submitting jobs using RASSubmit actor

To submit a job into UNICORE one must use RASSubmit actor. By providing this actor with input parameters we can define what RAS/Vine host will be used for job submission, which Domain will be used and how job description will look like.

RASSubmit actor accepts following input data:

Input port name	Description
proxy	String representing user's proxy
resubmit	Number of times actor will try to submit the job in case of failure
Jsdl	JSDL based description of the job
hostAddressRAS	URL of RAS.
DomainName	Name of the Domain defined at RAS/Vine host

As an output RASSubmit actor generates job id that was submitted via Vine:toolkit library.

2.6.1 RASSubmit – example workflow

Below you can se an example of workflow that submits job after JSDL description is created using RASJSDLGenerator actor (vine_submit.xml).

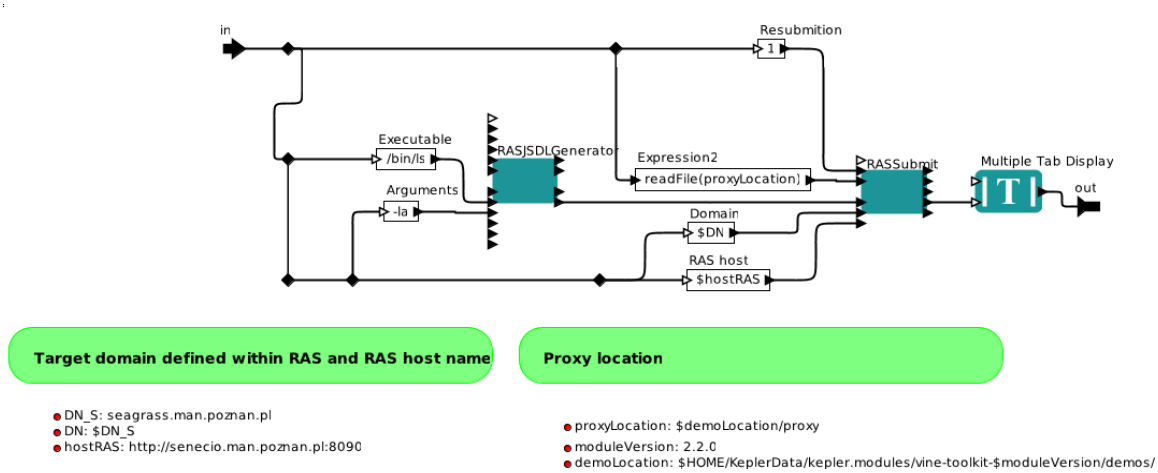


Illustration 6: Submitting jobs using RASSubmit actor

In this particular case JSDL is passed directly from RASJSDLGenerator actor, however it can be read from file as well. It can be also generated by other means.

Workflow can be found at:
$SERPENS_MODULE/vine-toolkit/demos/vine_submit.xml

2.7 Checking status of the job using RASStatusLoop actor

RASStatusLoop actor allows user to check the status of previously submitted job. Actor accepts following inputs:

Input port name	Description
proxy	String representing user's proxy
JobId	Id of the job
domainName	Name of the domain that will be used during status check.
hostAddressRAS	URL of RAS.

As a result, actor outputs current status name.

2.7.1 RASStatusLoop – example workflow

Usually, checking the status of the job is executed within the loop. This way, workflow can “wait” till particular job is finished and proceed with the results. You can see example workflow at following image (vine_status.xml):

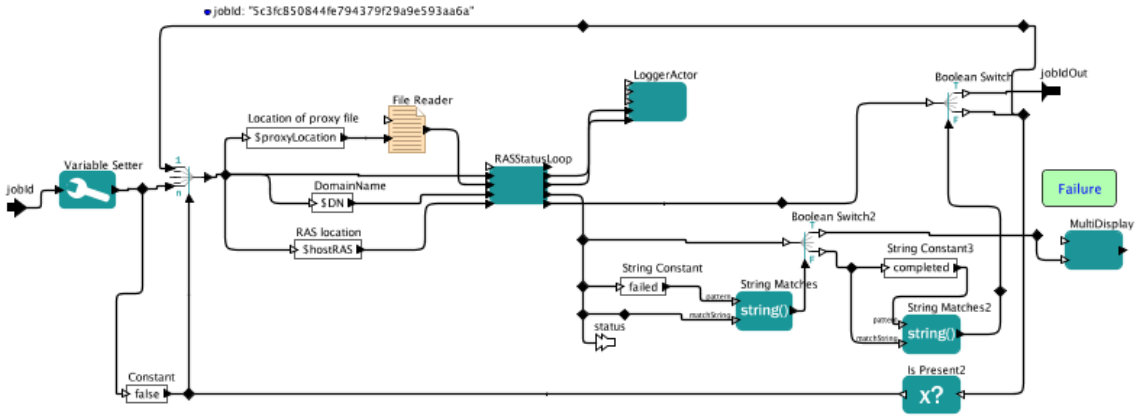


Illustration 7: Checking job's status within loop

Workflow can be found at:
$SERPENS_MODULE/vine-toolkit/demos/vine_status.xml

2.7 Retrieving job's output/error streams

To retrieve job's output/error stream user should use RASGetOutput/RASGetError actor. These actors accept following input data:

Input port name	Description
proxy	String representing user's proxy
JobId	Id of the job
domainName	Name of the domain that will be used during status check.
hostAddressRAS	URL of RAS.

They return standard output and standard error respectively.

Following picture presents example usage of these actors:

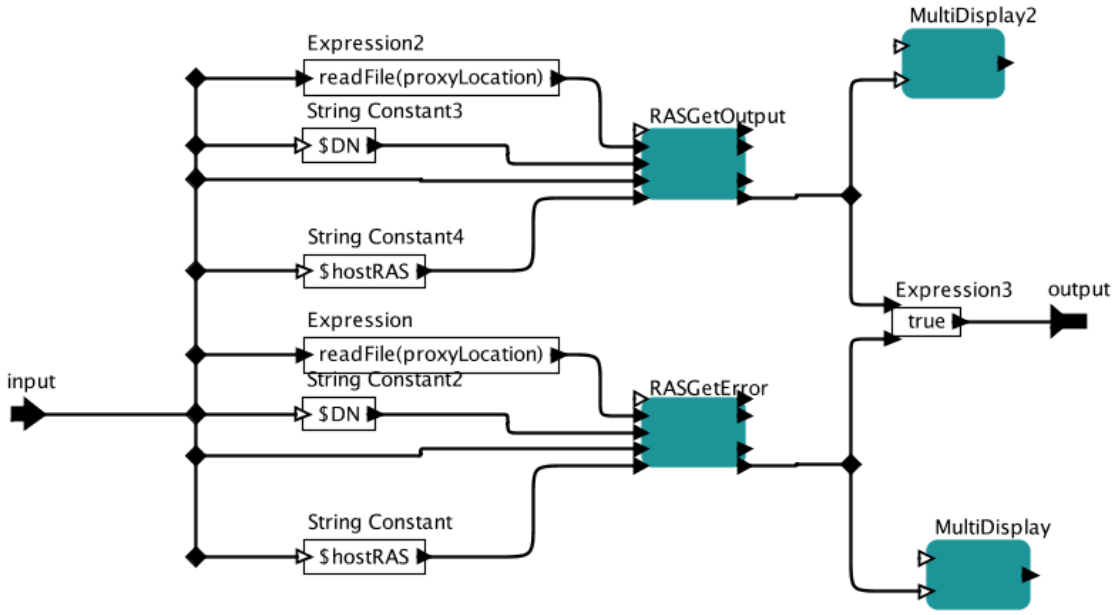


Illustration 8: Retrieving std out/std error of job

This workflow is a part of template workflow described in next section.

2.8 Putting things together

In general, users are not required to use all RAS/Vine related actors in order to build workflow by themselves. Serpens module, apart from actors, provides composite workflows that perform more complicated tasks. An example of such workflow is vine_template.xml workflow that allows user to submit job into UNICORE by specifying parameters defined within composite actor. This way, user can simply copy paste one actor in order to get the whole HPC related scenario running – that is: upload files, submit job, check it's status, retrieve outputs. You can see an example of such a workflow at following picture

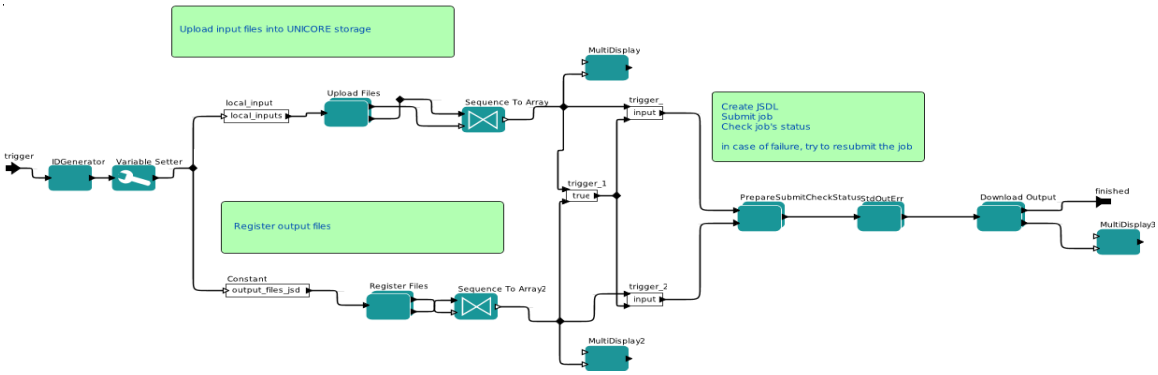


Illustration 9: HPC job submission template

Workflow can be found at:

$SERPENS_MODULE/vine-toolkit/demos/vine_template.xml

What you can see on the picture is the workflow that is divided into sections – each section is responsible for an action related to HPC job management. After workflow is finished, output of the job will be stored in specified location.

References

1. Serpens - http://serpens.psnc.pl/
2. UNICORE - http://www.unicore.eu/
3. Kepler-2.3 - https://kepler-project.org/
4. vine:toolkit - http://vinetoolkit.org/

Acknowledgments

This research has also received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement n°211804 (EUFORIA).