Building Computational Science Portlets using Rapid—A Reference Manual

Jos Koetsier and Jano van Hemert

version 2.0

Abstract

This manual provides the main concepts required to use Rapid as well as a reference for the XML-based configuration used to develop portlets. Rapid is designed to quickly prototype, develop and deploy portlets that become part of a web portal. Its main purpose is to allow rapid development of user interfaces for dedicated tasks and applications that need access to remote compute resources. It has many features to make it easy to create these portlets as it can connect to different compute resources, handle remote file systems and deal with all aspects of submitting compute jobs.

Contents

Chapter 1

Introduction

In many scientific disciplines, the demand for computational power has increased dramatically and is likely to grow even further in the future. New techniques, such as micro array analysis in biology, generate ever larger data sets that need to be analysed. Sciences such as astronomy and physics are developing more accurate instruments, that, coupled with new and more complicated algorithms require ever larger computing resources. These new innovations and techniques mean that for many applications simple computation on a desktop computer can no longer be considered adequate.

Thankfully, the demand for more computing power is mirrored by the availability of newer and faster High Performance Computing resources. For example, the list of top 500 supercomputers in the world shows that available computing power has increased exponentially for the last fifteen years. New developments in Grid Computing enable users to access even more computing power by allowing users to submit computational tasks transparently to not one, but several, often heterogeneous, resources.

Unfortunately, high performance computing comes at the price of increased complexity. Submitting a computational task to a cluster or grid often requires issuing a set of complex commands through a command line interface. This is further complicated by the necessity of some form of security infrastructure to regulate access to often expensive resources. Users need to request certificates, download proxy certificates and understand the security implications of their actions. Accessing computing power is no longer a matter of double-clicking an application on a desktop, but more and more requires an in-depth knowledge of computer science, which not all scientists possess.

In order to alleviate this burden on the researcher, scientific applications are often wrapped in a user interface. In the grid community, the use of portals is often considered the preferred solution mainly due to its distributed nature. No complicated installation of software is necessary; users can log on to the portal from any computer, simply by using the browser supplied by their operating system. An added advantage is that security is dealt with by the portal. Most portals offer some sort of login mechanism and issue privileges using a role based access mechanism, allowing for fine-grained security to be applied.

When considering the state of the art of current portlets that allow users to submit tasks to clusters or grids, roughly two approaches can be identified. First there is the *generic* portlet. This type of portlet, in effect, wraps a com-

mand line job submission. For every parameter a text box or drop-down list is supplied which the user needs to fill out. These portlets allow a wide variety of computational tasks to be submitted, but at the cost of increased complexity. This type of portlet is far from ideal, especially because we set out to shield the user from all this complexity in the first place.

The other approach is to write a *custom* portal application for each computational task. We consider this type of portlet preferable to the generic portlet of the first approach, because of its simplicity. Any user can access the power of large compute clusters without the indepth knowledge required to perform a command-line job submission. However, this approach does come at the cost of increased development time spent on programming the portlets themselves. It is often the case that there is simply not enough time and money available to write a portlet for each new project. There is also the additional problem of maintainability. When the portlet is finished, it requires an expert to maintain it, fix bugs and perform upgrades.

The Rapid project is an attempt to address these issues. We propose an *automatic job submission portlet generation system*. This system allows an expert to specify information about a job submission portlet in an XML file. This can be information about file transfers, options to display to the user and the available compute clusters as well as the security information required to access them. The Rapid system takes this XML file and translates it into a fully working JSR 168 compliant job submission portlet.

This approach gives us the benefits of both portlet types we discussed earlier. On the one hand, users get a job submission portlet that shields them from any complexity and on the other hand the time and effort to write and maintain a generic portlet is greatly reduced. Generating a portlet in this way does reduce flexibility somewhat however; not each portlet that can be written by hand can be generated using Rapid. However, we think that using Rapid will be able to meet most users' needs and will allow scientists to make better use of the available computing power.

Chapter 2

Overview

This section describes the high level concepts required to understand the mechanisms that are used in the Rapid System. The job submission model used by Rapid is loosely based on the Basic Execution Service (BES) and the Job Submission Description Language (JSDL) standards. In these standards a method of job submission is described, which (in its most basic form) consists of the following stages:

1. Copy one or more files from one or more file servers into the host where the computation takes place. (Stage in, or *source* file transfers)
2. Execute and monitor the job as it is executing. This job may consist of several sub-jobs, for instance in the case of so-called parameter-sweeps.
3. Copy one or more files from the execution host (presumably the results of the computation) to one or more file servers. (Stage out, or *target* file transfers)
4. Visualise the results of the computation.

The Rapid system allows a *template* for such a job to be defined. The user can pre-define a set of 'source' and 'target' file transfers, using methods such as ftp, sftp, http or gsiftp. Defining the job itself follows the POSIX model, consisting of a set of environment variables, an executable and a number of parameters.

Obviously, submitting a pre-defined job is not very useful and therefore the Rapid system allows a user to change the pre-defined job template, by introducing a set of variables. Each variable is given a unique name and is initialised with a default value as part of the job template. In the user interface, the user is allowed to change these variables. The portlet developer can specify which user interface element to use to change a variable. Rapid supports a wide variety of input methods such as file browsers, radio buttons, lists and text entry fields.

When a job is submitted the variables are substituted in place holders within the job template. As an example, suppose we have an executable called 'factorial', which takes one parameter, which is a positive whole number for which this program computes the factorial. We would specify this in the job template as:

```
...
  <executable>usr/bin/factorial</executable>
  <parameter index="0">5</parameter>
...
```

This example always sets the first parameter to the value '5'. If we want the user to be able to change this value, we add a variable which we call 'parameterValue'.

```
...
  <variable name="parameterValue">
      <single>
         <value>5</value>
      </single>
  </variable>
...
```

This initialises the variable with a default value of '5'. To bind this variable to the parameter, we have to change the job template to:

```
...
  <executable>/usr/bin/factorial</executable>
  <parameter index="0">$(parameterValue)</parameter>
...
```

The `$(<name>)` notation allows a value to be substituted by the variable, when the job is submitted. This notation can be used in most parts of the Rapid XML file.

In the user interface section of the Rapid XML file we can now introduce a user interface element, that enables the user to examine and change the value of a variable. For example a list of possible values can be added, from which the user makes a selection:

```
<page name="configure job">
  ...
  <x:h1>Choose factorial</x:h1>
  <variable name="parameterValue">
      <list>
         <item value="1">1</item>
         <item value="5">5</item>
         <item value="20">20</item>
      </list>
  </variable>
 ...
</page>
```

In the following sections we will discuss all elements in the Rapid XML file.

Chapter 3

The Rapid XML Document

The `<rapid>` element denotes the root of the Rapid XML document and contains two namespaces: the *rapid* namespace: *http://www.ed.ac.uk/Rapid* and the *XHTML* namespace: *http://www.w3.org/1999/xhtml*. Elements from the Rapid namespace are used to set the job template, initialise file servers and configure user interface elements. The XHTML interface is used for markup, such as tables, fonts and incorporating images.

Element <rapid>

Root element of the Rapid XML document.

Child Elements

- `<condor>` Defines a new Condor job submission instance. (optional)
- `<sungridengine>` Defines a new Sub Grid Engine job submission instance. (optional)
- `<pbs>` Defines a new PBS job submission instance. (optional)
- `<fork>` Defines a new Fork submission instance. (optional)
- `<local>` Defines a local file system. (optional)
- `<ftp>` Defines an (s)ftp file system. (optional)
- `<http>` Defines an http file system. (optional)
- `<gsiftp>` Defines the gsiftp file system. (optional)
- `<initialise>` Initialises a new job template.
- `<page>` Defines a page int the user interface.
- `<persistence>` Defines state preservation. (optional)

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<rapid xmlns="http://www.ed.ac.uk/rapid"
            xmlns:x="http://www.w3.org/1999/xhtml">
   <!-- Contents go here -->
</rapid>
```

3.1 Job submission servers

A job submission server is the compute resource to which a compute job is submitted. In the *rapid* namespace, `<condor>`, `<sge>`, `<pbs>` and `<fork>` elements support Condor, Sun Grid Engine, PBS and simple forking a new process, respectively. They are the immediate child elements of the `<rapid>` root element.

3.1.1 Condor cluster

The `<condor>` element allows Rapid to submit to a Condor cluster.

Element <condor>

Attributes

- `name`: a unique identifier

Child Elements

- `<bin>`: This Element specifies the location of Condor execution files 'condor_submit' and 'condor_q'. (optional)
- `<condorconfig>`: Path to the condor_config file. (optional)
- `<universe>`: Defines the runtime environment under which Condor cluster should execute a job. (optional)
- `<filesystemname>`: Indicates through which file system (specified in 3.2) the condor submit host is accessed. Supports SSH and Local.
- `<submitline>`: Defines an additional line in a Condor submit file. (zero or more)
- `<pollingtime>`: The amount of time between two status updates in milliseconds. (optional. default is 10000)

Example

```
<condor name="condor">
   <bin>/home/condor/condor/bin</bin>
   <universe>vanilla</universe>
   <submitline>priority = 10</submitline>
   <submitline>image_size = 20</submitline>
   <filesystemname>CondorFS</filesystemname>
   <condorconfig>/etc/condor/condor_config</condorconfig>
```

```
    <pollingtime>5000</pollingtime>
</condor>
```

3.1.2 PBS

The <pbs> element allows Rapid to submit to a PBS scheduler.

Element <pbs>

Attributes

- name: a unique identifier

Child Elements

- <bin> Specifies the location of the PBS binary files 'qstat' and 'qsub'
- <filesystemname> Indicates through which file system (specified in 3.2) the condor submit host is accessed. Supports SSH and Local.
- <option> Defines an additional lines in the PBS submit file. (zero or more)
- <pollingtime> The amount of time between two status updates in milliseconds. (optional. default is 10000)

Example

```
<pbs name="pbs">
   <bin>/opt/pbs/bin</bin>
   <option>#PBS -l nodes=1:ppn=$(numberOfCPUS),cput=12:00:00</option>
   <filesystemname>PBSFS</filesystemname>
</pbs>
```

3.1.3 Sun Grid Engine

The <sungridengine> element allows Rapid to submit to a Sun Grid Engine cluster.

Element <sge>

Attributes

- name: A unique identifier

Child Elements

- <root>: Path to SGE_ROOT. (optional).
- <filesystemname>: Indicates through which file system (specified in 3.2) the Sun Grid Engine submit host is accessed. Supports SSH and Local.
- <option>: Lines to add to a SGE submit script. (zero or more)
- <pollingtime>: The amount of time between two status updates in milliseconds. (optional. default is 10000)

Example

```
<sungridengine name="sge">
   <root>/home/sge/sge</root>
   <filesystemname>LocalFS</filesystemname>
   <option>#$ -M admin@nesc.ac.uk</option>
</sungridengine>
```

3.1.4 Fork

The `<fork>` element configures a simple 'fork' submission resource where the job is simply run as a new process.

The <fork> Element

Attributes

- `name:` A unique identifier

Child Elements

- `<filesystemname>:` Indicates through which file system (specified in 3.2) the process is to be run. Supports SSH and Local.

Example

```
<fork name="fork">
    <filesystemname>LocalFS</filesystemname>
</fork>
```

3.2 File systems

There are five elements that are used to define file systems: `<ssh>`, `<http>`, `<ftp>`, `<local>` and `<gsiftp>`. These elements are child nodes of `<rapid>` and each requires a unique 'name' attribute used as a reference throughout the document. All filesystem tags contain a `<url>` child element of which the body specifies the complete URL of the file system. Some elements may encompass additional child nodes, depending on authorisation and authentication. Those file system elements are detailed in the following sections.

3.2.1 HTTP

The `<http>` element is used for both *http* and *https* servers. Neither of these requires authentication.

Element <http>

Attributes

- `name:` A unique identifier

Child Elements

- `<url>`: URL of the file system

Example

```
<http name="http">
   <url>http://www.inf.ed.ac.uk/</url>
</http>
<http name="http2">
   <url>https://www.omii.ac.uk/</url>
</http>
```

3.2.2 Local file system

The `<local>` element allows access to a local file system. Similar to `<http>`, there is no authentication. This file system refers to the file system the portal resides on and not the users' file system.

Element <local>

Attributes

- `name`: A unique identifier

Child Elements

- `<url>`: URL of the file system

Example

```
<local name="Local File System">
     <url>file:///home/portaluser/</url>
</local>
```

3.2.3 FTP

The `<ftp>` element is used for *ftp* servers and requires authentication using a username and password.

Element <ftp>

Attributes

- `name`: A unique identifier

Child Elements

- `<url>`: URL of the file system. Paths are relative to the users' home directory, unless preceded with a %2f sequence.
- `<username>`: Username used to log into the ftp server
- `<password>`: Password

Example

```
<ftp name="myftpserver">
   <url>ftp://host.university.ac.uk/full/path/</url>
   <username>myname</username>
   <password>mypassword</password>
</ftp>
```

3.2.4 SSH

The `<ssh>` element is used for *sftp* servers and requires authentication either using a username and password or by supplying an identity file.

Element <sftp>

Attributes

- `name`: a unique identifier

Child Elements

- `<url>` URL of the file system. Paths are relative to the users' home directory, unless preceded with a %2f sequence.
- `<username>` Username
- `<password>` Password
- `<identityfile>` File name of the identity file.

Example

```
<ssh name="mysftpserver">
   <url>sftp://host.university.ac.uk/full/path/</url>
   <username>myname</username>
   <password>mypassword</password>
</ssh>
```

3.2.5 GSIFTP

The `<gsiftp>` element describes a Grid FTP server. For authentication, a valid X509 proxy certificate must be obtained through a MyProxy server. Paths are relative to the users' home directory, unless preceded with a %2f sequence.

Element <gsiftp>

Attributes

- `name:` A unique identifier

Child Elements

- `<url>:` URL of the file system
- `<myproxyhost>:` Hostname of the myproxy server

- `<myproxyport>`: Port through which to access the myproxy server
- `<myproxyusername>`: Username
- `<myproxypassword>`: Password

Example

```
<gsiftp name="mygisftpserver">
   <url>gsifpt://host.university.ac.uk/full/path/</url>
   <myproxyhost>myproxy.university.ac.uk</myproxhost>
   <myproxyport>7512</myproxyport>
   <myproxyusername>myusername</myproxyusername>
   <myproxypassword>secret</myproxypassword>
</gsiftp>
```

3.3 Initialising a job

The `<initialise>` element is used to define a template for a new job. It can contain nine child elements. The `<datastage>` element involves the file staging details and the `<posix>` element is used to set parameters of a job according to the POSIX model. The compute resource to use is set in the `<submitto>` element and the `<variable>` and `<static>` elements can be used to declare variables and set initial values. The tags `<preprocess>` and `<postprocess>` can invoke a series of actions, when a new job is created and finished, respectively. The `<userunknown>` tag determines what to do when the user has not authenticated to the portal, and, finally, the `<plugin>` tag declares all available jython plugins.

Element Initialise

- `<datastage>` (optional)
- `<posix>` (optional)
- `<submitto>` (optional)
- `<preprocess>` (optional)
- `<postprocess>` (optional)
- `<userunknown>` (optional)
- `<variable>` (optional)
- `<static>` (optional)
- `<plugin>` (optional)

3.3.1 Variables

As explained in section 2, a job submission can be configured by inserting variables into the job template, filesystem definitions and compute resource definitions. These variables are initialised as specified and can be changed in various ways. For example, a user can change the value of a variable by choosing an item from a drop down box or a Jython script can be called that changes the variable.

Rapid distinguishes two types of variables: 'static' variables and 'regular' variables. 'Regular' variables are tied to a particular job submission. When a user finishes configuring a job and submits it, the current set of 'regular' variables are stored and bound to that job submission and a new set of 'regular' variables is initialised to be configured for the next submission. The job is queued, instantiated and executed using the stored set of 'regular' variables. Once a job has been submitted, the set of 'regular' variables which is bound to that submission cannot be changed anymore, but can be queried through the job monitoring tags.

In contrast to 'regular' variables, 'static' variables are defined across all jobs. When the portlet starts, all 'static' variables are initialised once and will not be bound, stored or reinitialised after job submissions. 'Static' variables can be modified in the same way as 'regular' variables, but cannot be used in the job template as this would mean that running jobs can be altered, resulting in unpredictable behaviour. Static variables can be displayed, modified and copied to regular variables using actions or Jython scripts.

Variables are declared using the `<variable>` or `<static>` element. The value of a variable can referred to by using the format: `$(VARIABLE)`.

Element <variable>

Defines a 'regular' variable

Attributes

- `name`: Name of the variable.
- `retainvalue`: Set to 'true' to retain the value of the previous job submission. (optional)

Element <static>

Defines a 'static' variable

Attributes

- `name`: Name of the variable.

Example

```
<initialise>
   <variable name="regularvariable" retainvalue="true">
      <single>
         <value>a value</value>
      </single>
```

```
   </variable>
   <static name="staticvariable">
      <uuid/>
   </variable>
</initialise>

<page>
   The value of the regular variable is $(regularvariable)
   The value of the static variable is $(staticvariable)
</page>
```

3.3.2 Data types

Each variable has a data type associated with it. There are currently four types, specified by the elements `<single>`, `<array>`, `<range>` and `<uuid>`. The `<single>` element is used to indicate simple, single values, the `<array>` element is used to specify multiple values and the `<range>` element is used to specify a range of numbers. The `<uuid>` element generates a single new Universally Unique IDentifyfier (UUID) for each new job that is generated. The values themselves are specified in the body of `<value>` elements which can appear as direct children of `<single>`, `<array>` or `<range>`. submit, where the first job will use the first value from each array, the second job will use the second value and so on.

Elements <single>, <array>

Child Elements

- `<value>` specifies a value (one or more if parent is `<array>`, one for `<single>`).
- `<min>` sets the minimum possible value. Implies `<value>` is a number.
- `<max>` sets the maximum possible value. Implies `<value>` is a number.
- `<regexp>` sets a regular expression the `<value>` elements must conform to.
- `<errormessage>` error message to display if `<value>` elements contain invalid values.

Example

```
<array>
   <value>12</value>
   <value>23</value>
   <value>44</value>
   <regexp>[0-9]+</regexp>
   <errormessage>Wrong Input! Please enter a whole, positive number</errormessage>
</array>
```

Element <range>

Child Elements

- `<min>` Sets the minimum of the range.
- `<max>` Sets the maximum of the range.
- `<step>` Sets the step size to use

Example

```
<range>
   <min>-1.0</min>
   <max>0.6</max>
   <step>0.3</step>
</range>
```

This example generates the range $-1.0, -0.7, -0.4, -0.1, 0.2, 0.5$

Element <uuid>

Automatically creates a new Universally Unique IDentifyer (UUID) when a new job is created.

Example

```
<uuid/>
```

3.3.3 Data staging

Data staging in the Rapid system follows the staging model as defined in JSDL. One *data stage instance* can contain a *source* and *target* transfer. The *source* file transfer copies a file or directory into the job execution engine and is performed before running a job. Once the job execution has finished, the *target* file transfer moves a file or directory out of the job execution engine.

Rapid defines three additional parameters for a data stage instance. First, a filename uniquely identifies the file called in the job execution engine. Second, a creation flag determines how the file will be created in the job execution engine and finally, a deleteontermination flag indicates whether the file will be removed from the execution host once the job is completed.

Element <datastage>

The `<datastage>` element defines a new data stage operation and has no attributes.

Child Elements

- `<source>` File or directory to copy into the compute resource. Performed before executing the job.
- `<target>` File or directory to copy results to from the compute resource . Performed after the job has finished executing.

- `<filename>` Name of the file or directory in the compute resource.
- `<deleteontermination>` Determines whether the file will be deleted from the compute resource, once the job has finished. (optional)
- `<creationflag>` Refers to the creationflag of the file. Its value can be either APPEND, OVERWRITE, or DONTOVERWRITE. (optional)
- `<dotarget>` Performs the target data stage, depending on the success state of the job. Values are: ALWAYS (always perform the target data stage), ONFAILURE (only when the job failed) and ONSUCCESS (only perform the target data stage when the job completed successfully).
- `<multiple>` This element is used if multiple files or directories should be staged.

Element <source> and <target>

Defines the file system and path used as a source or target file transfer.

Child Elements

- `<filesystem>` The name of the file system used in either the source or target file staging.
- `<path>` The full path of either 'source' or 'target' file stage.

Element <multiple>

This element is used to stage multiple files or directories at the same time. The number of files or directories is equal to the number of values contained in the variable it refers to. Tis element can be used if there are one or more multi-valued variables (<array> or <range>) embedded in the data stage definition and causes the execution engine to iterate over each value of the variables for each file transfer.

Attributes

- `variable` The name of the variable.

Example

```
<datastage>
   <source>
      <filesystem>datarepository</filesystem>
      <path>/var/data/input.data</path>
    </source>
    <target>
      <filesystem >httpserver</filesystem>
      <path>/var/www/htdocs/results.txt</path>
    </target>
    <filename>result.txt </filename>
    <deleteontermination>true</deleteontermination>
    <creationflag>OVERWRITE</creationflag>
```

```
    <dotarget>ONSUCCESS</dotarget>
    <multiple variable="numberoffiles"/>
</datastage>
```

3.3.4 Posix

Job execution in Rapid follows the Unix POSIX model and all its elements are grouped under the `<posix>` element.

Element <posix>

Child Elements

- `<executable>`: Denotes the name of the executable to run.
- `<stdin>`, `<stdout>`, and `<stderr>`: Refer to the standard input, output and error, respectively.
- `<workingdir>`: Element specifies the directory running the executable.
- `<parameter>`: Parameters of the executable.
- `<environmentvariable>`: Environment variables in a shell.
- `<multiple>`: Used for array jobs.

Example

```
<initialise>
   <posix>
      <executable>./java</executable>
      <environmentvariable>
         <name>CLASSPATH</name>
         <value>junit.jar:log4j.jar</value>
      </environmentvariable>
      <workingdir>/home/user/data001</workingdir>
      <parameter index="0">-jar</parameter>
      <parameter index="1">/opt/programs/science/calculate.jar</parameter>
      <parameter index="2">$(valuelist)</parameter>
      <multiple variable="valuelist"/>
  </posix>
       :
 </initialise>
```

Element <parameter>

Parameter of the executable.

Attributes

- `index`: Index of the parameter, starting at 0.

Example

```
<initialise>
   <parameter index="0">-alR</parameter>
</initialise>
```

Element <environmentvariable>

Shell environment variable defined as NAME=VALUE.

Child Elements

- `name:` Name part of the environment variable.
- `value:` Value part of the environment variable.

Example

```
<initialise>
   <environmentvariable>
      <name>CLASSPATH</name>
      <value>junit.jar:log4j.jar</value>
   </environmentvariable>
   <environmentvariable>
      <name>PATH</name>
      <value>/usr/bin:/bin:/home/user/mybin</value>
   </environmentvariable>
</initialise>
```

3.3.5 Pre and postprocessing

The user can execute a set of actions before a new job is edited by the user and after the job has completed executed. This can, for example, be used to load the result of an execution into a variable, so it can be displayed in a job monitor.

Elements <preprocess> and <postprocess>

Postprocessing and preprocessing.

Child Elements

The child elements of `<preprocess>` and `<postprocess>` are actions, such as defined in section 3.4.2, which are executed in order.

- `<loadfile>`
- `<savefile>`
- `<deletefile>`
- `<copyvariable>`
- `<callprogram>`
- `<runplugin>`

Example

```
<initialise>
   ...
  <preprocess>
      <loadfile variable="myfilevar" path="/path/to/my/file" filesystem="myfilesystem"/
      <callprogram variable="myprogramvar" path="/path/to/my/program.sh" filesystem="my
  </preprocess>
   ...
</initialise>
```

Element <userunknown>

This element determines which page to load if a user has not authenticated. If this element is not present, the first page is simply loaded and the portlet proceeds as normal

Child Elements

The child elements of `<userunknown>` can contain one 'navigate' action.

- `<navigate>`

Example

```
<initialise>
   ...
  <userunknown>
      <navigate nextpage="user_not_logged_in_page"/>
  </userunknown>
   ...
</initialise>
```

3.3.6 Execution node

By default, a job is submitted to the execution node specified under the `<submitto>` element. The value of this element refers to a submission engine, previously defined in section 3.1.

Element <submitto>

Example

```
<initialise>
   <submitto>mycluster</submitto>         :
</initialise>
```

3.3.7 Plugins

Plugins are declared using the `<plugin>` element. The name of the file as well as the class to use must be given. The plugins are assumed to reside in the `configuration/plugins` directory.

Element <plugin>

Declares a new plugin.

Attributes

- `name:` Unique name of the plugin.
- `file:` Name of the file to use WITHOUT the '.py' extension.
- `class:` Name of the class to use.

Example

```
<plugin name="myplugin" file="myjythonfile" class="rapidclass"/>
```

3.4 User Interface

The user interface (UI) is defined in a set of `<page>` elements, that are immediate children of the `<rapid>` root element. When a Rapid portlet starts up, the view associated with the first `<page>` element that is defined within the document is loaded up and displayed. The user can navigate between views by pressing a button, containing a 'navigate' action, as described in section 3.4.2.

Element <page>

A page containing markup and rapid user interface elements.

Attributes

- `name:` unique name of the page.

Child Elements

- `<variable>`
- `<button>`
- `<fileupload>`
- `<joblist>`
- `<subjoblist>`
- `<setjob>`
- `<onload>`
- elements from the XHTML namespace

Example

```
<page name="page1">
  <!-- page layout is defined here -->
</page>
```

Child elements of the `<page>` can be taken from both the *xhtml* and the *rapid* namespace. The markup of the user interface is written using standard XHTML. The logic of the portlet as well as user input is handled by elements from the *rapid* namespace.

3.4.1 User interface elements

The rapid system enables a user to change or display a variable by specifying a `<variable>` element as child element of the `<page>` element. How this is done, is defined in the child elements of the `<variable>` element, which are currently `<list>`, `<checkboxlist>`, `<checkbox>`, `<radio>`, `<browser>`, `<text>`, `<editor>` and `<output>`.

Element <variable>

Enables a user to change a variable or display the value of this variable.

Attributes

- `name:` Name of the variable to change or display.

Child Elements

- `<list>`
- `<checkbox>`
- `<checkboxlist>`
- `<radio>`
- `<browser>`
- `<text>`
- `<editor>`
- `<output>`

Example

```
<variable name="variableone">
   <text/>
</variable>
```

Element <list> and <radio>

Shows a list of items, a dropdown list or a set of radio buttons. If using a list with an 'array' value, the user can will be able to select multiple values.

Attributes

- `refresh:` if 'true', the page will be reloaded if a selection is made.
- `size:` `<list>` only. Length of the list. If the length is 1, this user interface element will be a dropdown list.

- `index: <radio>` only. An index from 0. If the value edited is an 'array' value, it refers to the index-th element in the array. If it is a 'range' value, then 0 refers to the minimum value, 1 refers to the maximum value and 2 refers to the step size.
- `class:` specifies a classname for the HTML element that will be generated. To be used with Cascading Style Sheets.
- `id:` specifies an id for the HTML element that will be generated. To be used with Cascading Style Sheets.

Child Elements

- `<item>` (one or more)

Example 1

```
<list refresh="true" size="1">
   <item value="itemone">select item one</item>
   <item value="itemtwo">select item two</item>
   <item value="itemthree">select item three</item>
</list>
```

Example 2

```
<radio refresh="true" index="2">
   <item value="itemone"/>1.0
   <item value="itemtwo"/>2.0
   <item value="itemthree"/>3.0
</radio>
```

Example 2 allows a user to set the step size of a range value.

Element <checkbox>

Adds a checkbox.

Attributes

- `checked:`the variable will get this value if the box is 'checked'
- `unchecked:`the variable will get this value if the box is 'unchecked'
- `class:` specifies a classname for the HTML element that will be generated. To be used with Cascading Style Sheets.
- `id:` specifies an id for the HTML element that will be generated. To be used with Cascading Style Sheets.

Example

```
<checkbox checked="you checked this option" unchecked="this option was not checked"/>
```

Element <checkboxlist>

Adds a list of checkboxes. This element can only be used with 'array' values. The value of each checkbox the user selects will be added to the 'array' value. Each checkbox that is not selected will be ignored. The resulting 'array' value will have as many elements as values the user selected.

Attributes

- `class:` specifies a classname for the HTML element that will be generated. To be used with Cascading Style Sheets.
- `id:` specifies an id for the HTML element that will be generated. To be used with Cascading Style Sheets.

Child Elements

- `<item>` (one or more)

Example

```
<checkboxlist>
   <item value="itemone"/>select item <x:h1>one</x:h1>
   <item value="itemtwo"/>select item <x:h1>two</x:h1>
   <item value="itemthree"/>select item <x:h1>three</x:h1>
</checkboxlist>
```

Element <item>

Adds a choice to a `<list>`, `<radio>`, or `<checkboxlist>`

Attributes

- `value:` The value to set into the job.

Child Elements

- `<list>` can contain text

Example

```
<checkboxlist refresh="true" size="1">
   <item value="itemone"/>select item <x:h1>one</x:h1>
   <item value="itemtwo"/>select item <x:h1>two</x:h1>
   <item value="itemthree"/>select item <x:h1>three</x:h1>
</checkboxlist>
```

Element <text>

Adds a text box.

Attributes

- `cols:` Size of the text box in columns.

- `rows:` Size of the text box in rows. If omitted, a value of one is assumed.
- `index.`An index from 0. If the value edited is an 'array' value, it refers to the index-th element in the array. If it is a 'range' value, then 0 refers to the minimum value, 1 refers to the maximum value and 2 refers to the step size.
- `password:`If the variable refers to a password in a filesystem, or is otherwise a value that should not be echoed to the screen, set this attribute to 'true'
- `class:` specifies a classname for the HTML element that will be generated. To be used with Cascading Style Sheets.
- `id:` specifies an id for the HTML element that will be generated. To be used with Cascading Style Sheets.

Example 1

```
<text cols="10" password="true"/>
```

Example 2

```
Specify Minimum Value: <text size="10" index="0"/> <br/>
Specify Maximum Value: <text size="10" index="1"/> <br/>
Specify Step Size: <text size="10" index="2"/>
```

Element <editor>

Adds a javascript editor to the page. This editor is taken from the 'editarea' project (http://www.cdolivet.com) and resides in the 'javascript' directory of the rapidportlet distribution.

Attributes

- `cols:` Number of columns of the editor.
- `rows:` Number of rows of the editor.
- `highlight:`True / False: use syntax highlighting.
- `syntax:`coldfusion, css, java, pas, php, sql, vb, basic, c, cpp, html, js, perl, python, ruby, tsql, xml
- `toolbar:`True / False. : display a toolbar.
- `fontsize:`number, indicating the size of the font.
- `fontfamily:`family of the font to use
- `wordwrap:`True / False. Use wordwrap.
- `tabtospaces:`True / False. Convert tabs to spaces.

Example 1

```
<variable name="editvariable>
   <editor rows="20" cols="120" highlight="true" syntax="java"/>
</variable>
```

Element <browser>

Displays a file browser, allowing the user to navigate and select a file or directory. The value of the variable will be the path of the file selected.

Attributes

- `filesystem` the name of the filesystem this browser uses. Can refer to a variable.
- `multiple` Set to 'true' to enable the user to select multiple files.
- `refresh` refresh after each selection.
- `size:` size of the file browser.
- `files:` set to 'true' to allow files to be set.
- `directories:` set to 'true' to allow directories to be set.
- `filestyle:` set a style to indicate the presence of a file.
- `directorystyle:` set a style to indicate the presence of a directory.

Example

```
<variable name="path">
   <browser filesystem="$(filesystemVar)" size="15"
       filestyle="color:blue" directorystyle="background-color:green"/>
</variable>
```

Element <output>

Displays the value of a variable. Can replace values.

Attributes

- `default:` String to display if the job value has not been set

Child Elements

- `<replace>` Replace the value by another string. Specify the value in a 'search' attribute and the replacement in the body.
- `<pre>` Used for `<array>` type. Displays a string before each value in the `<array>`. Can contain XHTML elements.
- `<post>` Used for `<array>` type. Displays a string after each value in the `<array>`. Can contain XHTML elements.

Example

```
<variable name="parameterValue">
   <output>
      <replace search="item1">this is item one</replace>
      <replace search="item2">this is item two</replace>
      <replace search="item3">this is item three</replace>
      <pre>before</pre>
      <post>after</post>
   </output>
</variable>
```

Element <fileupload>

Allows a user to upload a file to any of the filesystems. The file upload takes place immediately when updating the page.

Attributes

- `size:` size of the file upload input element (optional)
- `filesystem:` name of the filesystem to upload this file to.
- `path:`path to copy the file to. If this file is a directory, the file is copied into that directory.

Example

```
<fileupload filesystem="ftp" path="/path/to/destination/" size="10">
```

3.4.2 Buttons

Rapid allows the portlet designer to add buttons that perform one or more actions when pressed. Possible actions include navigating between pages, submitting jobs, loading and saving files, deleting jobs from a job monitor and changing the value of a variable.

Element <button>

This element defines a button. Its child elements are a list of actions that are performed in order.

Attributes

- `size:` size of the button.
- `display:` string to display on the button.
- `class:` specifies a classname for the HTML element that will be generated. To be used with Cascading Style Sheets.
- `id:` specifies an id for the HTML element that will be generated. To be used with Cascading Style Sheets.

Child Elements

- `<navigate>`
- `<submit>`
- `<loadfile>`
- `<savefile>`
- `<deletejob>`
- `<haltjob>`
- `<copyvariable>`
- `<callprogram>`
- `<runplugin>`
- `<refreshbrowsers>`

Example

```
<button display="navigate and submit">
   <navigate nextpage="secondpage"/>
   <submit/>
</button>
```

Element <navigate>

Navigates to another page

Attributes

- `nextpage:`next page to load.

Element <submit>

Submits the job. Contains no attributes or child elements.

Elements <loadfile> and <savefile>

The element `<loadfile>` loads the contents of a file into the value of a variable. Similarly, the element `<savefile>` enables a user to save the value of a variable to a file.

Attributes

- `filesystem:` The name of the filesystem.
- `path:` Full path of the file.
- `variable:`Name of the variable.

Example

```
...
<variable name="contents">
   <single><value/></single>
</variable>
<variable name="path">
   <single><value>/path/to/my/file.txt</value></single>
</variable>
...
<page name="pagename">
...
  <variable name="contents">
      <text cols="40" rows="20"/>
   </variable>
...
<button display="Load File">
   <loadfile filesystem="myfilesystem" variable="contents" path="$(path)"/>
</button>

<button display="Save File">
   <savefile filesystem="myfilesystem" variable="contents" path="$(path)"/>
</page>
```

Element <copyvariable>

The element `<copyvariable>` copies the value of one variable to another

Attributes

- `from`: Name of the variable to copy
- `to`: Name of the destination variable

Example

```
<button display="Copy">
   <copyvariable from="fromvar" to="tovar"/>
</button>
```

Element <deletejob> and <haltjob>

The element `<deletejob>` removes a job that has been submitted from the list of jobs the portal knows about.. `<haltjob>` stops a job that is running on an HPC. Both these elements are used in combination with a job monitor, which selects the job which is to be deleted or halted.

Attributes

- `selection`: Selection as defined in a job monitor.

Element <callprogram>

The element `<callprogram>` executes a program or script and loads the standard output into the value of a variable.

Attributes

- `filesystem:` The name of the filesystem.
- `path:` Full path of the file.
- `variable:`Name of the variable.

Example

```
...
<variable name="contents">
   <single><value/></single>
</variable>
<variable name="path">
   <single><value>/path/to/my/executable.sh</value></single>
</variable>
...
<page name="pagename">
...
  <variable name="contents">
      <text cols="40" rows="20"/>
   </variable>
...
  <button display="Execute Script">
    <callprogram  filesystem="myfilesystem" variable="contents" path="$(path)"/>
  </button>
```

Element <runplugin>

The element `<runplugin>` executes the plugin. Once the plugin has executed the output can be retrieved using the `<plugin>` element.

Attributes

- `name:` Name of the plugin to execute

Example

```
<button display="run plugin">
   <runplugin name="myplugin"/>
</button>
```

Element <refreshbrowsers>

Reloads the contents of all filebrowsers.

3.4.3 Onload event

Before loading a page a set of actions can be executed.

Element <onload>

This element defines a list of actions that are performed in order when a page loads.

Child Elements

- `<loadfile>`
- `<savefile>`
- `<copyvariable>`
- `<callprogram>`
- `<runplugin>`

Example

```
<page name="mypage">
   <onload>
      <runplugin name="myplugin"/>
   </onload>
</page>
```

3.4.4 Job monitor

Job monitoring can be implemented using the `<joblist>` , `<subjoblist>` and the `<setjob>` elements. The `<joblist>` element iterates through all submitted jobs and allows the user to select a job by clicking on a radiobutton. If a job contains a set of subjobs, the element `<subjoblist>` can be used to iterate through any subjobs selected in a `<joblist>` element. The `<setjob>` element refers to the selection made in the `<joblist>` for `<subjoblist>` element and allows a more detailed view of a particular job to be presented. Adding a `<deletejob>` element allows a user to delete a job from the job monitor and the `<haltjob>` element stops a job that is running on a cluster.

Elements <joblist>, <subjoblist> and <setjob>

`<joblist>` iterates through all previously submitted jobs. If a job consists of subjobs, `<subjoblist>` can be used to iterate through subjobs. `<setjob>` sets the job selected a `<selection>` element, which is a child element of `<joblist>` or of `<subjoblist>`.

The elements `<datastage>`, `<posix>`, `<submitto>` and `<variable>` elements have all been defined previously. However, if they appear as child elements of `<joblist>` and `<setjob>`, they can only be used to display values of jobs that have been submitted and therefore do not contain input or output child elements.

Attributes

- `selection:<setjob>` and `<subjoblist>` only. When used with `<setjob>`sets the (sub)job that was selected. When used with `<subjoblist>`, this attribute refers to a selection of a job in `<joblist>`.

Child Elements

- `<submitto>`
- `<variable>`
- `<jobid>`
- `<selection>` Used only in conjunction with `<joblist>` and `<subjoblist>`.
- `<date>`
- elements from the XHTML namespace

Element <jobid>

When a job has been submitted it is given a Job ID in the form of a UUID. This Job ID can be displayed using this tag.

Element <status>

The status of a submitted job.

Element <selection>

This tag displays a radiobutton that can be used to select a job.

Attributes

- `selection:` unique name identifying the selection

Element <date>

The date the job has been submitted.

Example

```
<joblist>
   <x:table border="1" width="800"> <x:tr> <x:td>
      <selection name="mySelection"/>
   </x:td> <x:td>
       <date/>
   </x:td> <x:td>
       <jobid/>
   </x:td> <x:td>
      <status/>
   </x:td> </x:tr>
   </x:table>
</joblist>
```

```
   <button display="delete job">
      <deletejob selection="mySelection" />
      <haltjob selection="mySelection" />
   </button>

   <subjoblist selection="mySelection">
      <x:table border="1" width="800"> <x:tr> <x:td>
         <selection name="mySubSelection"/>
      </x:td> <x:td>
          <date/>
      </x:td> <x:td>
          <jobid/>
      </x:td> <x:td>
         <status/>
      </x:td> </x:tr>
      </x:table>
   </subjoblist>

   <setjob selection="mySubSelection">
      <x:h1>executable:
$(executable)
      </x:h1>
   </setjob>
```

Element <plugin>

`<plugin>` element retrieves the output from a plugin that has previously been executed. More information about plugins can be found in section 4

Attributes

- `name` name of the plugin as declared in the 'initialise' section.

Example

```
   <page name="view">
      <plugin name="myplugin"/>
   </page>
```

3.5 Persistence

In the default persistence model the Rapid system preserves the state of the portlet as part of the current login session. As a consequence, all state information, such as which jobs were submitted, which user they belong to and what the status of each job is, is lost each time a user logs out or if the session expires.

If the state of the portlet is required to be maintained, a database or file must be specified under the `<persistence>` tag, in which the portlet can store this information.

When using persistence it is important that the database exists and is empty.

When upgrading Rapid to a new version, ensure that the database is cleared before running the portlet.

Element <persistence>

Child Elements

- `<username>` Username of the database to connect to. The username is specified in the `name` attribute.
- `<password>` Password to use to connect to the database. The password is specified in the `name` attribute.
- `<host>` Host name of the database. The host name is specified in the `name` attribute.
- `<dbms>` Type of Database Management System. Currently supported are `mysql`, `postgres`, `hsqldb`, `h2` and `file`. The type of database is specified in the `name` attribute.
- `<database>` Name of the database to use. If the under the `<dbms>` tag 'file' was chosen, this will refer to the full path of the file to use. The database is specified in the `name` attribute.

Example 1

```
<persistence>
   <username name="rapiduser"/>
   <password name="password"/>
   <host name="database.nesc.ac.uk"/>
   <dbms name="mysql"/>
   <database name="rapidDB"/>
</persistence>
```

Example 2

```
<persistence>
   <username name="admin"/>
   <password name="sa"/>
   <host name=""/>
   <dbms name="file"/>
   <database name="/home/portal/rapidDB"/>
</persistence>
```

Chapter 4

Plugins

Using plugins, a portlet developer can add more complex logic and dynamic user interface elements to the portlet. Plugins are written using the Jython programming language, which is a Python implementation written in Java. All plugin files should be placed in the 'plugins' directory, where the portlet will be able to find them.

Before running a plugin, it should first be declared in the 'initialise' section, where the name of the file and class is given. When it is declared, the plugin can be *executed* using the `<runplugin>` element, which can be part of a button press, the preprocess or postprocess steps or when a page loads using the `<onload>` element.

When the plugin has been executed, the *output* of a plugin can be inserted into a page using the `<plugin>` element.

4.1 Structure

Plugins are classes written in Jython that inherit the 'RapidPlugin' class. RapidPlugin itself is an abstract class and requires the 'doAction' method to be overloaded. This method is called every time a new page containing the `<plugin>` element is loaded.

4.1.1 Example

```
from uk.ac.nesc.rapid.plugin import RapidPlugin

class RapidTest(RapidPlugin):
      self.addHTML('<h1>Hello World</h1>')
```

The example above is a simple 'hello world' plugin, which simply outputs `<h1>Hello World</h1>`. The plugin should be declared in the 'initialise' section, executed as part of a button or other event and finally, the output should be retrieved using the `<plugin>` element. Interaction between the plugin and the portlet is done by calling methods of the RapidPlugin base class. In the following sections all available methods of RapidPlugin are described.

- void addHTML(String html)

Adds a string to the output of the plugin. Multiple calls result in strings being appended to the output. Variables using the `$(variable)` notation are substituted.

- void addTextInput(String variable, int rows, int cols [, int index])

  This method adds a new text input box to the portlet. The name of the variable is assumed to have been defined in the `<initialise>` section of the rapid.xml file. The parameters 'rows' and 'cols' describe the size of the text box and the optional parameter 'index' can be used to set the value of a particular sub job.

- void addRadioInput(String variable, String groupName, boolean refresh, String option [, int index])

  addRadioInput add a radio button to the portlet. The variable is assumed to exist. 'groupName' is a fixed string that groups radio buttons, where only one button can be selected at one time. The 'option' is the value the variable will take, when selection. The optional parameter 'index' is used to refer to a particular value of a sub job.

- void addCheckBoxInput(String variable, String checked, String unchecked)

  This method adds a checkbox to the portlet. The variable indicated is assumed to be defined, and, if selected will take the value of 'checked' and if unselected will take the value 'unselected'

- void addListInput(String variable, int size, boolean refresh, java.util.List<java.lang.String> optionList, java.util.List<java.lang.String> displayList)

  addListInput adds a list box input to the portlet. The variable is assumed to exist. The parameter 'size' determines the length of the list box; if the size is set to '1', the list box is rendered as a drop down box. The two lists 'optionList' and 'displayList' are a list of values the variable will be set to and a corresponding list which is displayed to the user.

- String getVariable(String variable [, int index [, uk.ac.ed.rapid.jobdata.JobID jobID]])

  This method gets the value of a variable. The first parameter refers to the name of the variable, the second, optional, parameter refers to particular subjob and the last parameter refers to a job ID if the value of a parameter of a previously submitted job is needed.

- int getVariableSize(String variable [, uk.ac.ed.rapid.jobdata.JobID jobID])

  The method 'getVariableSize' gets the number of values contained in a variable. The optional parameter 'jobID' can be used to query variables of a job that has been submitted previously.

- void setVariable(String variable, (java.util.List<java.lang.String> valueList — String value)

  This method sets a variable to either a list of values or to one single value.

- java.util.List<uk.ac.ed.rapid.jobdata.JobID> getJobIDList()

  Gets a list of jobs, ordered on the date of submission

- void addJobSelection(uk.ac.ed.rapid.jobdata.JobID jobID, String selectionID [, int subJobIndex])

  This method adds a radio button, that can be used to select a jobID and an optional sub job. The selection can be retrieved using the `getJobSelection(String selection)`

- uk.ac.ed.rapid.jobdata.JobID getJobID(String selectionID)

  Gets the Job ID for a selection.

- int getSubJobIndex(String selectionID)

  Gets the sub job index for a selection

- String getJobStatus(String jobID)

  Gets the status of a job.

- String getJobDate(uk.ac.ed.rapid.jobdata.JobID jobID)

  Gets the date the job was submitted at.

- String getUserName()

  Gets the username of the user currently logged in.

4.1.2 Jython Class path and Python path

Jython can import both Python modules and Java classes. Any jars should be placed in the 'plugins' directory and will be copied into the portlet WAR file where they can be found. Python modules be accessed by changing the python path within a plugin as follows:

```
import sys
sys.path.append("/home/me/mypy")
```

Rapid uses a very minimalist version of Jython 2.5 and does not include any of the modules that come as standard with the Jython distribution. To access those, download and install Jython 2.5 and append the python path as shown above.

Chapter 5

Generating the portlet

In this section we describe the steps necessary to generate and install the portlet.

5.1 Dependencies

This section reviews the dependencies for the Rapid system.

5.1.1 Portal

Because Rapid generates portlets, we require a JSR 168 compliant portlet container to deploy our portlet into. The build scripts are currently written the GridSphere portal (version 3.0 and higher), Pluto version 1 and 2 and the Liferay container. We also support the building of a 'general' JSR 168 compliant WAR file.

5.1.2 Apache Ant

Before installing a Rapid portlet, Apache Ant is required, which can be downloaded from the Apache website[1].

5.2 Rapid portlet generation and installation

In order to install a new rapid portlet, download and unpack the RapidPortlet tarball. Within the RapidPortlet directory, all the configuration options are set in the 'configuration' directory.

The file 'rapid.properties', sets properties such as the name and title of the portlet that is to be generated. Finally, the configuration directory should contain a file called 'rapid.xml', which specifies the content of the new portlet,

The portlet is generated by issuing the `ant` command.from the RapidPortlet directory. The default portal framework that Rapid targets is Liferay. If a portlet for Gridsphere is required, an additional parameter `gridsphere\` should be added to the `ant` command. For the Pluto container add `pluto1` or `pluto2` for version 1 or 2 respectively. Use `jsr168` if a minimal WAR file is required.

[1] http://www.apache.org/

When this process is completed, a 'portlet' directory containing a 'war' file is generated.

The final portlet will be packaged as a 'war' file into the 'portlet' directory together with a small ant install script. When deploying a GridSphere portlet, a build.xml script is generated that can be used to deploy the portlet. It assumes the `CATALINA_HOME` environment variable has been set and the portlet is installed by entering the `portlet` directory and issuing the `ant deploy` command. For the LifeRay portal, it is sufficient to copy the 'war' file into the LifeRay deployment directory.

When upgrading to a new version of Rapid, be careful to completely remove portlets generated by previous versions of Rapid. This is because older versions of libraries may not be removed automatically by the portal when redeploying a new portlet.

An in-depth example of Rapid portlet installation can be found in Tutorial: Rapid-based Portlet Installation.