

Rapid Tutorial

Using Rapid to create portals

Introduction

In this tutorial we begin with building a very simple portlet, which we will refine into something more complex.

What is Rapid?

Rapid is a unique way of quickly designing and delivering web portal interfaces to applications that require computational resources, such as utility computing infrastructures or high-performance computing facilities. It focuses on the requirements of the end-user by designing customised user interfaces for domain-specific applications that allow users to achieve particular tasks.

How is this Tutorial structured?

To make a distinction between text, code and screen output, in this tutorial filenames and directory names are written in `courier`.

Screen output is written in `courier` and highlighted as shown below:

```
1
2
3
5
```

Support

For help with this tutorial, visit the Rapid website (<http://research.nesc.ac.uk/rapid>) or contact the Rapid team (j.vanhemert@ed.ac.uk).

For more information about OMII-UK, visit the OMII-UK website (www.omii.ac.uk) or contact support@omii.ac.uk.

Before starting the tutorial

The first task is to set up the development environment. We require the following software packages:

1. Java (version 1.5+)
 2. Maven 2 (maven.apache.org)
 3. Liferay Portal (www.liferay.com)
 4. The rapidportlet distribution (research.nesc.ac.uk/rapid)
- Install Java, Maven2 and Liferay (standard edition).
 - Unpack the rapidportlet distribution.

Welcome Bruno Admin!

- Home
- Control Panel
- My Account
- Sign Out
- Add Application**
- Layout Template
- Manage Pages

Add Application

Search applications (searches as you type):

- Collaboration
- Community
- Content Management
- Entertainment
- Finance
- Google
- News
- Religion
- Sample
- Shopping
- Soclel
- Tools
- WSRP
- Wiki
- World of Liferay
- rapid
- NewRapidPortlet

NewRapidPortlet Add

Drag a portlet to place it on the page.

Install More Applications

Web Content Display - Look and Feel - Content

Welcome to Liferay

Liferay is a complete all-in-one open source portal framework for building enterprise portals worldwide. Liferay customers include governments, educational institutions, and many other organizations and many others like:

including:

INSTANT WEBSITES

Use Liferay Web Content Manager with minimal effort and cost.

- Create
- Custom
- Work
- Live
- Office
- Life
- See
- Life
- Site
- Tools

```
...
</rapid>
```

- Add the missing second page, containing some XHTML and a button that refers back to the first page.

Deployment

The portlet is now ready to be translated and the resulting WAR file deployed.

- Call this file `rapid.xml` and put it into the `configuration` directory.
- Issue the command `mvn package -Pliferay` from the `rapidportlet` directory.
- Copy the `war` file from the `portlet` directory into the `deploy` directory of the portal.
- Wait a few seconds, log into the portal and add the portlet (see figure 1).

FileBrowser

In this section we will add a file browser to the portlet which will enable the user to select a file from a remote sftp server. The full path of the file the user selects will be displayed.

Initialise variables

Rapid uses variables to maintain state. Before a variable can be used it has to be initialised and given a default value. For this exercise we require three variables: two variables are needed to authenticate to the sftp server, one for the username and one for the password. A third variable is used to store the full path of the file the user selected.

```
<initialise>
  <variable name="usernameVar">
    <single><value>username</value></single>
  </variable>
  <variable name="passwordVar">
    <single><value></value></single>
  </variable>
  ...
  ...
</initialise>
```

- Declare and initialise variables to contain the Username, Password and the Filename.

Adding the sftp server

Next we define the sftp server that the user will be able to browse. The sftp server requires a unique name which will be used to refer to it throughout the `rapid.xml` file. Furthermore, we have to specify the URL, a username and a password. In this instance, we use the values of the variables we specified in the previous section for the username and password. The notation Rapid uses for variable substitution is a dollar sign followed by the name of the variable in brackets. To retrieve the value of the username we use `$(usernameVar)` and for the password we use `$(passwordVar)`. This notation can be used throughout the Rapid XML file.

```
<ftp name="mysftpserver">
  <url>sftp://my.host/path</url>
  <username>$(usernameVar)</username>
  <password>$(passwordVar)</password>
</ftp>
```

- Add a sftp file server definition.

Entering a username and password

Of course simply defining and displaying the values of variables is not very interesting. We need to allow the user to manipulate these variables. Rapid can use different methods to enter a value, such as radio buttons, drop-down lists and text boxes. In this exercise we will add two text boxes, one for the username and one for the password.

```
<page name="...">
...
  <variable name="usernameVar">
    <text cols="20"/>
  </variable>
...
</page>
```

- Add a textbox to allow a user to enter the username
- Add a textbox to allow a user to enter the password. Use the password="true" attribute to stop characters echoing.

Add a file browser

Finally, we add a file browser so the user can select a file. We also want to display the path of the file that was chosen. Adding a filebrowser is very similar to adding a text box as we did in the previous section. The filebrowser is simply another way for the user to manipulate the value of a variable. The value of the file chosen by the user can displayed simply by using the `${filenameVar}` notation.

```
<page name="...">
...
<variable name="filenameVar">
<browser filesystem="mysftpserver"/>/>
</variable>
...
</page>
```

- Add a file browser.
- Display the full path of the file chosen by the user.
- Deploy the new portlet, login to the sftp server and select a file.

Word Count

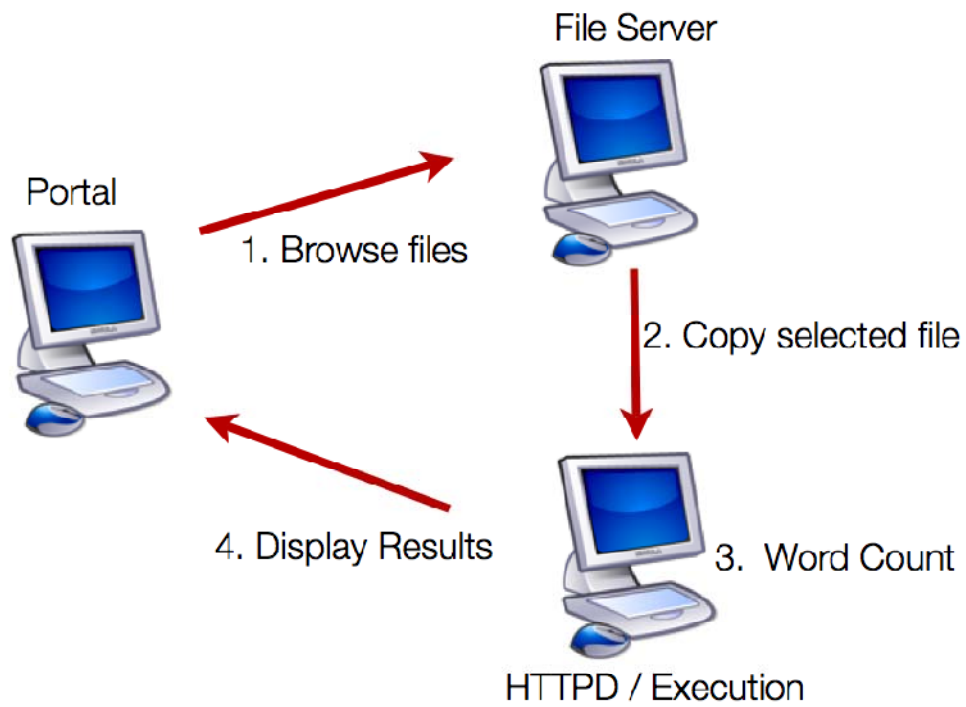
In this section, we will create a portlet to submit a very simple computational job. In the previous section, we allowed a user to select a file and, we will now extend this portlet to use the Unix *word count* program (`/usr/bin/wc`) to determine how many characters, words and lines the selected file has.

Figure 2: Selecting a file and executing the 'word count' program.

The user will be able to select a file from a file server, which gets copied to a second computer where the *computation* takes place. The result will be copied into a directory which is exported by HTTP, which means that the portal can easily access it.

Initialise the job

In the *initialise* section of the portlet we have to add a simple outline of the job. We have to specify the location of the executable and the parameters it uses. The word count program (in its simplest form) takes



```

    <single><value>mysftpserver</value></single>
  </filesystem>
  <path type="source" datastage="file_in">
    <single><value>$(filenameVar)</value></single>
  </path>
  <filename datastage="file_in">
    <single><value>/tmp/inputfile</value></single>
  </filename>
</datastage>
</initialise>

```

The section above copies the file selected by the user, which will be contained in the variable `$(filenameVar)`, into the `/tmp` directory of the execution host, where it will be called `inputfile`.

Synchronisation

Unfortunately, there is a synchronisation issue in the code above. If two users were to access the portlet at the same time, the input and output files could be overwritten and the wrong word count might be returned. We need to make the input and output filename unique for every computational job. This is done using a `uuid` variable. This variable will be set to a random new `uuid` everytime a job is initialised.

```

<initialise>
...
  <variable name="uuidVar">
    <uuid/>
  </variable>
...
</initialise>

```

- Add a `uuid` variable to the initialisation section.

Using this variable, we can change the `<filename>` element of our data transfer.

```

<initialise>
...
  <datastage>
...
    <filename datastage="mydatastage">
      <single><value>/tmp/inputfile-$(uuidVar)</value></single>
    </filename>
...
  </datastage>
...
</initialise>

```

The name of our input file will now look like:

```
/tmp/inputfile-fe7f0cb0-f606-4801-85ac-b20867443290
```

- Append a `uuid` to the name of the input file.

Similarly, we have to ensure the output file, containing the word count is unique. We therefore also need to append the `uuid` to the name of the output file.

- Append a `uuid` variable to the name of the output file.

Setting up the computational resource

Now we have to specify the computational resource the job runs on. This consists of two steps. First we have to indicate how to reach the computational resource, which we do by specifying a filesystem resource. Then we define a computational resource which will be accessed through the filesystem indicated in the first step.

Currently, we can access computational resources through a *local* filesystem, if the computational resource is the same as the portal, or, if the computational resource is installed on a remote system we can access it through ssh, using the sftp filesystem.

- Add a new sftp filesystem that refers to the portal to the *initialise* section. Call it `computeFS`.

Next we have to define a computational resource and tie it to the filesystem.

```
<fork name="compute">
  <filesystemname>computeFS</filesystemname>
</fork>
```

- Add the 'fork' execution engine

Submitting the job

Since we can define multiple execution engines in Rapid, we need to specify which execution engine to use. This is done with a *submitto* element.

```
<initialise>
...
  <submitto>
    <single><value>compute</value></single>
  </submitto>
...
</initialise>
```

- Set the 'fork' execution engine to the execution engine to submit jobs to.

Submitting the job is performed by pressing a button. To indicate that Rapid should submit a job, add a submit element to button.

```
...
<button display="submit">
  <navigate nextpage="pageone"/>
  <submit/>
</button>
...
```

- Add a new button or edit an existing one to allow the user to submit a job.

Job Monitoring

When the job has been submitted, we would like to be able to monitor it and check the results. However, as the job has been submitted all variables have been reset to the initial values. We need a way to retrieve those variables that were set when the job was submitted. There are two important elements that can do this: `<joblist>` and `<setjob>`.

The element `<joblist>` iterates through each job. Within this element, the values of previously submitted jobs are retrieved and can be displayed. Additionally, a number of other useful tags are defined.

1. `<status/>` Displays the current status of the job (suspended, running, completed, error)
2. `<date/>` Displays the date on which the job was submitted
3. `<jobid/>` A uuid Rapid associates with each job to uniquely identify it
4. `<selection name="selectionname"/>` Displays a radio button that allows the user to select a job. This is used in conjunction with `<setjob>`.

```
<page name="...">
  <x:h1>Select a job</x:h1>
  <x:br/>
  <joblist>
    <selection name="myselection"/>
    <jobid/>
    <status/>
    <x:br/>
  </joblist>
</page>
```

The element `<selection>` is used to allow a user to select a job which he or she wants to have a closer look at. It displays a radio button that the user can click. The selected job can later be retrieved using the `<setjob>` element.

In this section we display the contents of the output file we generated using the word count command. The simplest way to accomplish this using XHTML is by using an *iframe*.

```
<page name="...">
  <x:h1>Results</x:h1>
  <x:br/>
  <setjob selection="myselection">
    File: $(filenameVar)
    <x:br/>
    <x:iframe src ="http://my.httpd.ac.uk/outputfile-$(uuid) "
      width="40%" height="50"/>
    <x:br/>
  </setjob>
</page>
```

- Add the job monitor and
- Deploy the portlet!