Research360

Sakai-SWORD2 Integration Development Report

June 2012

1. Executive Summary

The Jisc-funded Research360[1] project at the University of Bath aimed to integrate data deposit seamlessly into the research workflow, by enabling users to transfer data from a virtual research environment (VRE) directly into a data repository. As the work was to use the SWORD2 profile to interface with the repository, the first stage of this involved developing an integration between the CLE version of the Sakai virtual research environment and the SWORD2 deposit specification. SWORD2 is supported for deposit by both the EPrints and DSpace repository platforms and as such, the planned work would be of interest to many UK and international universities.

Development work on the integration between SWORD2 and Sakai proved to be technically far more challenging than anticipated. The main unexpected difficulties faced were:

- Lack of a full plug-in architecture for the branch of the CLE version of Sakai used in development necessitated substantial modification of the existing source code;
- Installation of Sakai in a fully working development environment was complex;
- Documentation was out-dated, inaccurate, missing or difficult to find.

As such, the work was suspended in late 2012-early 2013. By this point an initial, although incomplete, compilation of new code had been achieved. No problems were encountered working with the swordapp Java library, which is smaller in scale than Sakai.

This report documents relevant experiences encountered during the development work and makes recommendations for future developers seeking to build on the initial work achieved. Suggestions for how Sakai's documentation should be improved are listed in Appendix 1 and will be passed to the Sakai community as a separate report. Support for such improvements would be one way in which development engagement with Sakai could be made easier in the future.

All additional code and changes made to the Sakai code, together with this report, will be made available to the wider community as a deliverable of the Research360 project. This will be a single patch file for Sakai, affecting the content tool (also known as the resources tool) module and the internationalisation information, as these were the only parts of Sakai modified during this work.

2. Recommendations

Two general recommendations are made to the wider Sakai development community working in this area:

1. Widely used open source projects such as Sakai should enhance their documentation, particularly guidance aimed at potential developers. It may be necessary for funding bodies to provide support for such activities. A full list of recommended improvements to Sakai documentation is provided in Appendix 1.
2. Sakai should ensure that all relevant files required to create a Sakai instance, and to develop new code, are included in every relevant branch of the subversion code

[1] http://blogs.bath.ac.uk/research360

repository. In particular, the *content-api* library must be included in future releases.

The following recommendations are aimed at developers who plan to work on Sakai:

1. If there is freedom to choose which version of Sakai is used, consider taking advantage of the widget architecture for Sakai OAE where possible rather than working with the more complex API in Sakai CLE.
2. Install a dedicated development copy of Sakai rather than relying on an existing production server.
3. Rather than carrying out the compilation to set up a demo or test site, instead consider a site installed from a pre-compiled binary. When testing modified code in the site, replace the modules that have been amended with compiled versions from the updated code. Configure Java, Tomcat and Maven to use as much memory as is available when compiling and when the Tomcat server is running: This must be at least 2GB of RAM, although 4GB is preferable.
4. If a fully compiled version of Sakai is required, then it should be installed using components downloaded directly from the relevant sources or software development kits (SDK) e.g. Tomcat and the Oracle/Sun Java SDK, rather than using packaged versions of these or other SDKs/Java VMs. If errors occur it can be problematic to re-install over an existing installation. Instead, a clean Tomcat needs to be set up for Sakai from scratch each time as a non-standard configuration of Tomcat is needed for Sakai to run. This can be achieved with a single installation of Tomcat using multiple instances[2].
5. When developing for the Sakai CLE, it is advisable not to install Sakai from source, nor to attempt to add the whole of Sakai to the Eclipse integrated development environment (IDE). When compiling for import into Eclipse, use the *sakai:deploy* profile and omit those Sakai modules that are not relevant to the development task from both the compilation and import.
6. The requirement for high memory usage of integrated development environments (IDEs) is particularly exposed by Sakai. If Eclipse is used, the source code must be compiled to import into the IDE. For this purpose, the development machine will require more than 3GB RAM. It is therefore worth considering not using an IDE at all. Development in a standard text editor, combined with standard Unix command line tools, was found to be easier than using Eclipse.
7. A relatively easy way to identify the Java code used to produce a specific page or function is to analyse the HTML source code displayed when a task is carried out on a Sakai server using the web browser. Use of the Unix command line tools *grep -r* and *find* was found to be faster than the search tool in Eclipse for finding relevant code in the Sakai source, the latter tool having a tendency to hang and not complete searches.
8. When working with the CLE, remember that it is a complex system including a lot of legacy code and allow additional time to learn how it works, looking in detail at the classes in the module(s) to be modified. As class files can be very large and complex,

[2] See the “Advanced Configuration - Multiple Tomcat Instances” in RUNNING.txt in the root of a standard Tomcat installation.

this is likely to be a time consuming process.

9. Join the Sakai development mailing list and make use of it earlier rather than later when problems arise. Contribute to improvements in Sakai documentation by updating key information with the helpful advice provided by the Sakai community.

3. Technical Aims

The purpose of the development work was to produce an extension to Sakai[3] that would allow the selection and submission of items from the resources tool (folders and individual files) to a repository using SWORD2[4].

The intended components of the extension were:

1. A mechanism to add “Archive” to the list of options available when one or more items from the resources tool were selected, to complement the existing list that includes such functions as “Copy” and “Delete”;
2. Integration with Sakai's permissions for resource actions so that individual users or groups could be permitted or denied the ability to archive items on a per resource, per directory, or per collection level. This is similar to the way that Sakai manages rights to read, amend, or delete items and directories;
3. A mechanism to allow a repository to be selected from a list so that more than one repository would be available to submit files to e.g. both a research publications repository and a data archive. A mechanism to allow, or possibly require, a user to add metadata to the collection of chosen file items to be submitted. This was intended to be a web form that would be displayed as an intermediate step in the deposit process;
4. A SWORD2 client that would operate in the background and perform the actual transfer of data between Sakai and the repository;
5. A mechanism that would notify the user when the archiving was complete, indicating success or failure;
6. Documentation and testing against DSpace and EPrints.

4. Difficulties Encountered During Development

4.1 Versions

The estimated time required for development was initially based on the assumption that an easily-discoverable plug-in architecture would be available[5]. Use of this would have meant that the development could utilise an API to be integrated into Sakai, without the need for substantial modification of the existing Sakai source code. It would also have simplified integration with existing components of Sakai.

Sakai currently comes in two major versions: the original CLE (Community Learning Environment) and the more recent OAE (Open Academic Environment). Although the OAE

[3] http://www.sakaiproject.org/
[4] http://www.swordapp.org/
[5] Note that an API is available at http://source.sakaiproject.org/release/2.8.2/apidocs/

version has a plug-in architecture and interface widgets[6], release 2.7.2 of the CLE version was already in use by the University of Bath Centre for Sustainable Chemical Technologies. The CLE version was therefore selected for modification, with a snapshot of the 2.8.x release branch from the Sakai subversion repository being patched against, this being considered more stable than the main development trunk branch.

Work on the CLE required developing a detailed understanding of many aspects of the Sakai CLE architecture, including Velocity templates and how they are managed in Sakai; the Sakai authorisation model; how internationalisation is handled; and how the state-dependent flow of the content tool works. Due to difficulties encountered in identifying suitable, accurate and current documentation for these topics, analyses to obtain this understanding proved to be more time consuming that originally anticipated. Additionally, the initial estimates of the time required for development assumed that a widget environment would be available for the CLE. Upon discovery that this was not the case, the timetable for this aspect of the work should, in hindsight, have been revised from 10 to at least 40 days.

4.2 Installation

Like many software products written in Java for use through web browsers, Sakai is implemented as a large collection of servlets that are housed in a web container, for which Apache Tomcat is recommended by the Sakai community. Compared with other open source products with a similar architecture such as DSpace and the Shibboleth Identity Provider[7], the installation process for Sakai was found to be complex and difficult.

It was intended to use Apache proxying to Tomcat to serve Sakai rather than Tomcat standalone, the architecture described in the Sakai documentation, as this would have provided additional flexibility and convenience by enabling the installation of other web-based software on the same machine. This was intended to make it easier to install and manage test targets such as EPrints and DSpace without using many virtual machines, although eventually no such further installation was required. As well as flexibility, proxying provides greater stability and for most applications is only trivially different[8] to setting up a Tomcat standalone environment.

Sakai configuration allows the use of a prefix such as */sakai* to the path of a URL, so that the URL for a Sakai component would be http://domain-name/prefix/component/, and thus should easily support proxying. However, hard-coded URLs in the software were not over-ridden by this configuration setting and the prefix to the path was ignored by some components, meaning that this installation method could not work without source code modification to ensure that the links in the user interface were actually valid.

Compiling a working installation of Sakai from the source code, and installing the source code into the Eclipse IDE, the development method described in the Sakai wiki, proved to be time consuming. Although ultimately neither of these activities proved necessary for a relatively small project such as this, both were helpful in gaining understanding of the code. Further details about problems encountered are detailed elsewhere[9].

---

[6] http://oae-widgets.sakaiproject.org/
[7] https://shibboleth.net/products/identity-provider.html
[8] E.g. about 10 lines of configuration in the Shibboleth Identity Provider.
[9] See links provided in Appendix 2 (blog posts two to five) for additional detail.

4.3 Insufficient memory for compilation

Sakai's installation documentation for version 2.8[10] recommends the following Java memory settings in the user environment of the Linux user carrying out the compilation:

```
export JAVA_OPTS='-server -Xms512m -Xmx1024m -
XX:PermSize=128m -XX:MaxPermSize=512m -XX:NewSize=192m -
XX:MaxNewSize=384m
```

These, however, proved to be insufficient. This also meant that the default RAM provided for a University of Bath virtual server was not sufficient and had to be increased to make the higher setting effective. All the figures given were doubled in an attempt to identify settings that worked, although further experimentation was not carried out to identify the exact values required.

The memory requirements for the Maven compilation tool must be set in the user environment separately from those for Java itself:

```
export MAVEN_OPTS='-Xms512m -Xmx1024m -XX:PermSize=64m -
XX:MaxPermSize=128m'
```

Doubling the memory settings for Java itself again proved to be insufficient, so it is recommended that they are increased further. It is worthwhile setting higher values for Java and Maven memory use from the start as the compilation process will run until the memory setting is used and then crash: something that took around 15 minutes in practice. The increased memory values for Java should also be used in the configuration of Tomcat, which is described in the Sakai documentation.

4.4 Missing components from the interface after compilation

The creation of some content in the installed Sakai instance, to be used for testing, was expected to be completed quickly. However, this important task proved difficult as, after compilation, the interface was crucially missing the form elements that allow the upload of material.

Help solicited from the Sakai development mailing list[11] suggested completely re-starting the installation. Successive re-installations emphasised the need to start a new installation in a completely unmodified Tomcat installation, but ultimately resulted in the same problems. The solution eventually identified was to set up the test installation from a downloaded binary rather than compiling from source. This enabled replacement of a binary installation of a specific module with a newly compiled version. This method of installation is therefore recommended for other developers.

4.5 Circular issues with library dependencies

Even though the test installation was not produced from compiled source code, it was still necessary to compile the source code in order to import it into the Eclipse IDE[12], the preferred development environment for Sakai. On the computer used for this work, this

---

[10] https://confluence.sakaiproject.org/pages/viewpage.action?pageId=75667802; the quoted figures have since been amended on the wiki page.
[11] http://collab.sakaiproject.org/pipermail/sakai-dev/2012-August/018448.html
[12] http://www.eclipse.org/

created different problems from those encountered with the attempts to compile the test installation, because the computer had a slightly different Java environment and a different compilation profile was used for this purpose.

The most time-consuming errors to resolve were circular problems with dependencies. As Maven compiles code, it checks for the existence of every dependency listed in the *pom.xml* configuration files that appear in each module, as well as those at the top of the Sakai distribution. It will then download missing libraries from online repositories, but will cease the compilation if it cannot find one that is required in the configuration. It is, however, usually possible to find an appropriate file and install it by hand so that Maven will then recognise its existence in its local cache of libraries.

Thus, as compilation progressed, a library would be indicated as missing and therefore installed. On re-compiling, this library would be recognised, but a second library would fail to be found. However, when that second library was installed, Maven would again proceed to indicate that the first library was not installed. A particularly challenging aspect of this problem is that at least one of the libraries involved, *org.sakaiproject.kernel:sakai-component-manager*, is itself a component of Sakai.

The solution identified was to change to a less comprehensive compilation profile, which ignores many modules, using the *mvn* command:

```
mvn -Pcafe sakai:deploy
```

4.6 Memory problems arising from the combination of Eclipse and the Sakai CLE

Memory usage problems of a severe nature also occurred when trying to compile the code for import into Eclipse, even with the expanded memory settings given above. These problems were encountered on a computer with 3GB RAM on which only the windowing environment, a terminal window and Maven were running, meaning that 50% more RAM was available than Maven was configured to use at maximum, so the compilation should have been safe. A particular problem that occurred on one compilation run involved memory filling leading to a crash, in the course of which */boot* was corrupted so that the computer would then not reboot from the hard drive. However, investigation of the precise cause of the problem was minimal due to a need to proceed with other aspects of the work.

The memory problems were eventually solved by the change to the minimal compilation profile. With a clean installation of Linux; new installations of Java, Tomcat, Maven and Eclipse; and with the reduced profile, the compilation was finally successful and the code could be imported into Eclipse.

4.7 Installation and use of an IDE

The various modules that make up the Sakai CLE are more independent than the existence of an overarching software project would suggest. It is therefore possible to replace individual modules of a Sakai installation, whether compiled or installed from pre-compiled components, one at a time, each module of which could also be compiled or pre-compiled. As such, a test development installation does not need to be entirely installed from source and although there are dependencies on the Sakai kernel modules which need to be

satisfied[13], not every module needs to be added to the IDE in order to work on a single module. Modules can therefore effectively be added as libraries, unless changes need to be made to them, as is the case with internationalisation. It is also possible to modify the *pom.xml* file in the root of a Sakai source installation to compile only those modules you are working with.

As previously noted, the Sakai wiki recommended the use of the Eclipse IDE for development work. However, development was eventually found to be easier using a standard text editor, in this case *gedit*, combined with standard Unix command line file and text search tools such as *locate*, *find*, and *grep –r.* These tools enabled files in the source containing specific strings to be identified faster than when using Eclipse's code search tool.

4.8 Documentation

Documentation is a difficult issue for open source software, and Sakai is no exception. Problems were experienced with the accuracy, availability, accessibility, and currency of information provided both in the source code, in automatically generated documentation, and in written documentation. Although Sakai does have a helpful and friendly developer mailing list[14], this proved difficult to search, an important problem for mailing lists where the etiquette is not to ask questions already answered. Together, these issues mean that newcomers to Sakai development may struggle to independently find any additional information they require.

A number of small-scale actions that might improve the quality and functionality of Sakai documentation were identified and are provided in Appendix 1. Recommendations to implement these actions are being fed back separately to the Sakai community.

5. Status of Development Work

5.1 Main Development Strands

The development work carried out to implement the required changes consisted of modifications and additions to the Sakai source in four main areas:

1. The addition of new Velocity interface templates for the content management module so that a user would be able to choose to send an item to a repository.
2. Configuration of new English language terms for use by this interface in Sakai's internationalisation interface layer.
3. Additions to the Java code that manages the content management module and interacts with the interface templates, to allow files selected using the new Velocity templates to be submitted to a repository using the swordapp library.
4. The addition of the Java swordapp library to those required for compilation of the content management module.

It was found not to be necessary to modify the swordapp SWORD2 library. Submission of multiple files in a single transaction had not been implemented in the current version of this library, but this could be worked around either by creating a ZIP file on the fly containing the

[13] See http://collab.sakaiproject.org/pipermail/sakai-dev/2012-August/018540.html for an example
[14] http://collab.sakaiproject.org/mailman/listinfo/sakai-dev

files selected for deposit or via use of the 'multipart' functionality. The impact of the former on the work of Sakai extension discussed here is that deposit of large items would require sufficient virtual memory on the Sakai server to create the ZIP file. As the deposit of large individual files should be fairly uncommon, it is unlikely that even multiple concurrent submissions would cause problems. However, too heavy a load would slow down the operation of Sakai and may even lead Tomcat, Apache, or the server itself to crash.

It is recommended that any subsequent work on the Sakai plugin should seek to ameliorate this issue if the relevant SWORD2 server end implementations, e.g. those used by DSpace and EPrints, support multiple file submissions. EPrints does not yet do this, as discussed on the eprints-tech mailing list in February 2013[15].

5.2 Progress against technical aims

It was originally estimated that this work would take 10 days to complete. Development was eventually stopped after 46 days of work. Progress against the original aims was as follows:

- Approximately 1,000 lines of new code had been written for aims 1 to 4.
- Aim 2 was developed by making amended copies of the existing read permission code.
- Some additional code for aim 3 was still required, in order to handle some form fields omitted from the processing up to this point. This would consist of a copy of two lines of existing code with altered web form parameters.
- When the work was suspended, code for aim 5 remained unwritten. Some preliminary investigation of the exact way in which Sakai handles the required type of functionality had been started.
- It was originally intended that documentation would be based on detailed records of the development work, also published as the series of blog posts listed in Appendix 2.
- The test requirements had been identified, along with the demo sites for DSpace (http://demo.dspace.org/) and EPrints (http://demoprints.eprints.org) to be used for testing.

5.3 Compilation

Compilation was the initial test for the newly written code. The developer's preference was not to attempt compilation until the bulk of the coding had been completed. It was felt that this would enable faster identification of missing requirements for such things as error catching, than would checking through the Sakai documentation for every library used, even assuming that the documentation was complete. Additionally, the structure of the Sakai Java classes modified in the course of development was unusual: One class was made up of over 7000 lines of code, describing a large number of methods, whereas it is usual with Java development to split code up into several smaller classes. It would therefore have been difficult to isolate changes to ensure that a successful compilation would have been possible at earlier stages: Errors would have indicated only that the new code depended on other parts of the code that had not yet been written.

A small amount of additional work was carried out in order to make the whole of the code

[15] http://www.eprints.org/tech.php/17352.html

ready for an initial attempt at compilation. This consisted of adding code to handle the response document produced by a SWORD2 server and to display the relevant fields to a user, some of the returned information being only of interest to a SWORD2 client.

5.4 Compilation Errors

The initial compilation found a large number of errors. Some of these were expected, as several relatively mechanical aspects of code production had been left for the compiler to uncover. Many of these errors were fixed by relatively simple amendments to the code, including typographical errors and references to external class libraries that had not been correctly loaded. However, unexpected errors included some from Java classes where the source code had not been altered[16]. Additionally, a new method had to be added to the *ResourcesAction* class to enable complete processing of a user request to archive data.

Ultimately, seven final compilation errors remained, all of which were associated with the *content-api* library, and mostly with the *ResourceToolAction* class. The *content-api* handles various common requirements for actions, such as determining whether the user is authorised to carry out the action on a specific resource or content item. For this to happen, the action code needs to call the *ResourceToolAction* class to register the action to be carried out.

The remaining seven compilation errors were related to the archive action not being permitted to be registered with the API. Theoretically, these could have been fixed by amending *ResourceToolAction*. However, this was problematic because of the way that Sakai code is distributed. Unlike other subversion repositories, extensive parts of the Sakai source code have not been included in the branches. The *content-api* library code is not included in the 2.8.x branch, nor in any branch since version 2.5[17], instead being included in one of the build files as an “overlay” along with many other APIs. These overlays are downloaded and added automatically by the Maven build tool during compilation, and so would never be seen by a developer who didn’t know specifically to look for them. It is therefore difficult to determine what APIs are available. While the interfaces these APIs provide are documented, this documentation is not easy to discover, and were only found via a web search after the developer had already undertaken significant work on the project.

The dependence on modification of the *content-api*, and the difficulties associated with doing this, indicated that in its current form the newly written code would not work with the library and further testing of the code was likely to be impossible.

6. Future Work

Late in the development process, an alternative class in the *content-api* library was identified that would allow a new action to be added without modifying the existing *content-api* code. The *CustomToolAction* class allows for an action of type “CUSTOM_TOOL_ACTION” in addition to the hard-coded actions in *ResourceToolAction*. This extension API would

---

[16] A quick check for the obvious possible cause, erroneous alteration of the contents tool *pom.xml* file to omit a required library previously there, found no errors. It would therefore be difficult to discover the actual cause of this error.

[17] It is possible that the lack of the *content-api* library might have caused some of the original compilation difficulties previously discussed.

potentially fulfil the original requirements of the SWORD2 extension.

However, as noted above, this class proved difficult to discover as the file is not included in source code downloads, initially suggesting that it was out-of-date for the current code. Further, the Sakai wiki does not provide information about either the CLE *content-api*[18], nor the *CustomToolAction*.

Future developers should be able to make use of the rudimentary plug-in architecture[19] and the *CustomToolAction* class to re-write the newly developed code as a CUSTOM_TOOL_ACTION, in a new class. It should be possible to re-use much of the existing code whilst doing this.

7. Appendix 1: Recommendations for Improvements to Sakai Documentation

This appendix contains recommendations for improvements to Sakai documentation, which were passed separately to the Sakai community.

7.1 Background

The Jisc-funded Research360 project at the University of Bath aimed to integrate data deposit seamlessly into the research workflow, by enabling data to be transferred from a virtual research environment directly into a data repository. The first stage of this involved developing an integration between the Sakai virtual research environment and the SWORD2 deposit specification. Development work on the integration between SWORD2 and Sakai proved to be technically far more challenging than anticipated. As such, the work was suspended in late 2012. A frequently encountered difficulty during the work was the poor quality of Sakai's documentation. In order to contribute to improvements in this area, this document contains a list of recommendations that, if implemented, are likely to improve the experience of future developers who are new to Sakai.

7.2 Recommendations

The following list suggests some ways in which improvements could be made, ordered by how easy it is anticipated they would be to implement.

1. On the Sakai wiki, replace the search box provided by the Confluence software with a domain-restricted search box for a major search engine e.g. Google – for example, a Confluence search for "authorization" returns many results for "authoring" instead. Ensure that the Sakai wiki pages are indexed by Google and other major search engines.
2. Update documentation for the *sakai.properties* file used for localisation of a Sakai installation. There are currently two sources of information: example files (one containing commonly needed customisation, the other every possible item that can be customised), and an out of date Word document listing the customisable values with minimal information on what they mean. Not all fields have clear functions or acceptable values that can be inferred from their names. This documentation on the Sakai wiki is quite out

[18] Searching the Sakai wiki for "*content-api*" returns a single page, https://confluence.sakaiproject.org/display/3AK/Content+API, which describes an unrelated content API for the OAE rather than the CLE.
[19] http://source.sakaiproject.org/release/2.8.2/apidocs/

of date.

3. Volunteers from the community should regularly scan the Sakai wiki to identify and then either fix or remove broken links.
4. Documentation matching frequently asked or anticipated questions, such as how the Sakai authorisation model works, should exist in the Sakai wiki.
5. Easy to find and regularly managed pathways into the Sakai wiki should be created and these should be linked directly from the Sakai website page for developers. These could pick up common issues for developers. Although the Programmer's Café goes some way toward this, it is not yet sufficient alone. In particular, links to the Programmers Café and other developer areas of the Sakai wiki are conspicuously absent from the Getting Started page for Technical Contributors[20].
6. Discoverability of APIs and their Javadocs should be improved, particularly for classes which are shipped as binary overlays and are therefore not in the standard source distribution. It is not at all clear to new Sakai developers that the Sakai CLE Kernel provides a wide array of useful APIs and extension points.

   This might involve a central index of all versions of each module, along with some way of determining which module a particular class belongs to. Without detailed knowledge of the project it is not immediately obvious whether a particular class will be part of the API, the kernel, or needs to be found in the individual release Javadocs.
7. Developers should be encouraged to include more comments in their work and to add descriptions to functions, constants, etc. If correctly formatted, the latter would then automatically appear in generated Javadocs.
8. The Sakai wiki would benefit from some usability testing of access to information.
9. The Sakai wiki would benefit from both removal of out-of-date information and the addition of more up-to-date information. In addition, links from discussion documents to information about the resolution of the discussion would be particularly useful.
10. Some overarching management of the Sakai wiki content and structure might help to reduce proliferation of out-of-date pages.
11. The wiki pages on development environment set up should be altered to make clear that using an integrated development environment (IDE) such as Eclipse is optional, and not required for successful Sakai development. The high memory requirements of IDEs are particularly exposed by Sakai, which itself requires a lot of memory.

8. Appendix 2

During development, comprehensive details of progress were recorded in a series of blog posts on the '*Matters of Opinion and Science'* blog. These blog posts were linked to from the Research360 blog, available at blogs.bath.ac.uk/research360. Full development blog posts can be found at:

- Prelude (4 July 2012): http://matters-of-opinion-and-

[20] http://www.sakaiproject.org/technical-contributors

science.blogspot.co.uk/2012/07/sakai-development-diary-prelude.html

- Post One (5 July 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/07/sakai-development-diary-day-one.html
- Post Two (18 July 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/07/sakai-development-post-two.html
- Post Three (31 July 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/07/sakai-development-post-three.html
- Post Four (7 August 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/08/sakai-development-post-four.html
- Post Five (4 September 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/09/sakai-development-post-five.html
- Post Six (5 September 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/09/sakai-development-post-six.html
- Post Seven (24 September 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/09/sakai-development-post-seven.html
- Post Eight (3 October 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/10/sakai-development-post-eight.html
- Post Nine (10 October 2012): http://matters-of-opinion-and-science.blogspot.co.uk/2012/10/sakai-development-post-nine.html