Using JSR-170 (Java Content Repository)

Added by Antranig Basman, last edited by Antranig Basman on Nov 07, 2007

This page will be a growing dump of notes from working with Sun's JSR-170.
The primary route for this integration for Sakai 2.5 will be the jackrabbit-service written by Ian Boston of CARET, which is currently hosted at https://source.sakaiproject.org/contrib/tfd/trunk/jackrabbitservice . This embeds an Apache Jackrabbit instance as a Sakai component, which it configures appropriately for the current Sakai database and clustering, and exposes via a standard JSR-170 API.

Experimentation is also underway with other JSR-170 implementations, such as Xythos, and Alfresco

Setting up repositories standalone for testing

Setting up Jackrabbit

Setting up Jackrabbit standalone is really quite hard due to the incredibly poor documentation. A simple "in-memory" demonstration as in their "First Hops" guide is straightforward, but setting up a realistic production environment using a database (e.g. MySQL) is extremely obscure. The main brunt of the work is done in this "JCRTests" project
https://saffron.caret.cam.ac.uk/svn/projects/amb26/trunk/jcr-tests
Many of the configuration options are not yet properly broken out but must be edited by hand in the project files - consult the README.txt for details.

Setting up Alfresco

The Alfresco content repository is another frontrunner in JSR-170 implementation. Their repository is actually part of a much wider CRM initiative, with many standalone components and servers. This information describes an attempt to set up Alfresco 2.1.0R1, at the time of writing the latest "stable" release of Alfresco. The main impediment in doing this is that Alfresco, despite being fairly enlightened Spring-wise, is still using an Ant build, with many peculiar, unlabelled and non-standard dependencies, some of which are not in the standard repositories.

MySQL instructions

The default user/password are alfresco/alfresco, database alfresco.

MySQL construction lines:

```
create database alfresco default character set utf8;
grant all privileges on alfresco.* to 'alfresco'@'localhost'
identified by 'alfresco';
flush privileges;
```

Unusual Afresco dependencies

- OpenOffice - the packaging information was taken from this "official" looking release, although the actual OO version for this Alfresco release predates any Maven release, at 2.0.3. http://wiki.services.openoffice.org/wiki/Uno/Java/MavenBundles

- SpringModules-jbpm - the actual version of this is unlabelled. The version in the official repositories demands a jboss dependency which does not appear in the Alfresco libs. NB - this was eventually discovered to be the JAR labelled jbpm-jpdl-3.2-patched.jar in disguise - entered into the repository manually under the jboss path.

- jug - unlabelled in the distro. I chose 1.1.2 as it is in the central repo. 2.0.0 has a POM but no JAR.

- jibx - unlabelled in the distro. I chose jibx-run-1.0.1.jar, which did not work. A JIBX exception is thrown "Binding information for class org.alfresco.repo.dictionary.M2Model must be recompiled with current binding compiler (compiled with jibx-rc0, runtime is jibx_1_0)". Note that this message is swallowed by M2Model.java at line 99

```
catch(JiBXException e)
{
throw new DictionaryException("Failed to parse model", e);
}
```

- JIRA this. The bundled JARs were thus relabelled into the rpo as jibx-rc0.

- ant - actually required at runtime because of the use of org.apache.tools.zip.ZipFile. I chose Ant 1.6.5.

- odf_util - quite a show-stopper. This JAR appears to exist nowhere, the official ODF site is at http://books.evc-cit.info/odf_utils/ and this situation is complained about at http://www.javalobby.org/java/forums/t77967.html. I placed it in catcode.com under version 05-11-29 to correspond to the only apparently visible source. This metadata never seems to have changed...

Alfresco - current status

Alfresco has now been demonstrated starting up in a standalone configuration, with a proper Maven 2 build held at https://saffron.caret.cam.ac.uk/svn/projects/amb26/trunk/jcr-tests . The dependencies which were not available in the standard repositories have been uploaded into the Caret Maven 2 repository, which is referenced in the POM file in that project. A crucial step is bundling a copy of xercesImpl-2.8.0 into the build, which seems to include a SAX parser that is

more lenient with the apparently invalid XML held in the Alfresco workflow definitions than the out of the box Java 5 parser. You will receive the following messages at startup

```
2007-11-07 16:48:10,406 WARN (JpdlXmlReader.java:127) - <process xml warning:
swimlane 'initiator' does not have an assignment>

2007-11-07 16:48:10,578 WARN (JpdlXmlReader.java:127) - <process xml warning:
swimlane 'initiator' does not have an assignment>

2007-11-07 16:48:10,671 WARN (JpdlXmlReader.java:127) - <process xml warning:
swimlane 'initiator' does not have an assignment>

2007-11-07 16:48:10,718 WARN (JpdlXmlReader.java:127) - <process xml warning:
swimlane 'initiator' does not have an assignment>
```

which seem essentially harmless. Without this JAR you will instead receive a fatal exception (see screenshots attached below).

A further deploy issue is that Alfresco *requires* a native code library (DLL under Windows) allowing it to operate its NetBIOS code. It appears that when not running under Windows it will automatically swap to another configuration where this JNI is not required http://wiki.alfresco.com/wiki/File_Server_Configuration but I have not tested this. For reference, I get this to run with the following JVM startup option on my box: **-Djava.library.path=E:\Source\alfresco-sdk-2.1.0\bin**

Alfresco as a Jackrabbit replacement

Now studying the extent to which Alfresco will be a "drop-in" for our current use of Jackrabbit. Some differences so far I have noted:

Dealing with Alfresco metadata and content model

- Dynamic registration of namespaces/node types is not supported. This doesn't seem to be baked into the underlying code design, but **is** hardwired in code at the top level of configuration for the JCR "facade". From Alfresco RepositoryImpl:

  ```
  namespaceRegistry = new NamespaceRegistryImpl(false,
  serviceRegistry.getNamespaceService());
  ```

- The first argument is "allowRegistration" which is set to false in this constructor and not further exposed through any setters.

- Some JCR types are not supported. An April 2006 post http://forums.alfresco.com/viewtopic.php?t=1659 suggests that only nt:base is supported, but in fact the "jcrModel.xml" file held at (config)/alfresco/model/jcrModel.xml from the recent release (2.1.0) shows that most of the core JCR types are supported (nt:base, nt:hierarchyNode, nt:file, nt:folder, nt:resource, mix:referenceable, mix:lockable). Notable by their absence are nt:unstructured (see next point) and mix:versionable. Note

that the April post indicates that the Alfresco web client will not recognise JCR nodes within the repository and the data dictionary page recommends extending the core Alfresco types.

- Unstructured nodes are not supported via the JCR API. The guide to the Alfresco "Data Dictionary Scheme" is at http://wiki.alfresco.com/wiki/Data_Dictionary_Guide. This documents a 3-step procedure for adding new namespaces and node types into Alfresco using their custom XML format. I believe the Jackrabbit format is proprietary too so no real loss here. The following forum posting includes more detailed assistance plus example for a user unable to get his metadata model working: http://forums.alfresco.com/viewtopic.php?p=17897

Independent attempts at a Maven 2 build

This Google cache link http://66.102.9.104/search?q=cache:zHh5ohywA40J:forums.alfresco.com/viewtopic.php%3Ft%3D7407%26view%3Dprevious%26sid%3Dea0233cf75bd8889d278bdd2e5686c6c (cache seems to disagree with current page, corresponding link is http://forums.alfresco.com/viewtopic.php?t=1017) to some traffic on the Alfresco forums from desperate people wanting a Maven 2 build. I tried to get in touch with a few of them but no answer. The "m2alfresco" Sourceforge project referred to at the end is a red herring, it is a dead project with no files and no traffic.

JSR Integration

Added by Antranig Basman, last edited by Steven Githens on Feb 16, 2008

Editor's note, this is version 3. The original dated 7 November 2007 is available at https://confluence.sakaiproject.org/pages/viewpage.action?pageId=42795093# See also below

Information

- Summary
- Overview of JCR in Sakai
- Content Hosting Stats

Sakai JCR Installation and Conversion Instructions
Alfresco in Sakai
Cambridge JCR migration experience

Summary

- Current issues that this addresses
    - Non-standard storage of content and limited interoperability support
    - Untested code in one of the most critical areas of Sakai
- Goals
    - Standard system for storing content which is easy to swap/configure
    - Industry standard and well tested code which is reliable
    - Simplification and clarification of the Content Hosting Service
- Benefits
    - JCR is an industry standard and Jackrabbit is well tested
    - JCR can interoperate and is highly configurable
    - Reduction of Sakai developed code of around 10000+ lines and overall reduction in code paths
    - Reduction of Content Hosting Service from 103 methods to around 20 methods (it becomes a simple abstraction layer and utility class for JCR)
    - Predictable performance (and we will be able to take advantage of any industry performance improvements)

Overview of JCR in Sakai

The Java Content Repository API is defined in 2 JCP specifications, JSR-170 and JSR-283. JSR-170 is the original specification, defining an API for a Node/Property based Content Repository, with features such as simple versioning, locking, and XPATH/SQL querying. The API is defined into 2 compatibily layers, with the Level 2 layer containing mostly optional features. JSR-283 supercedes JSR-170 and adds support for full versioning, sharable nodes, and other features.

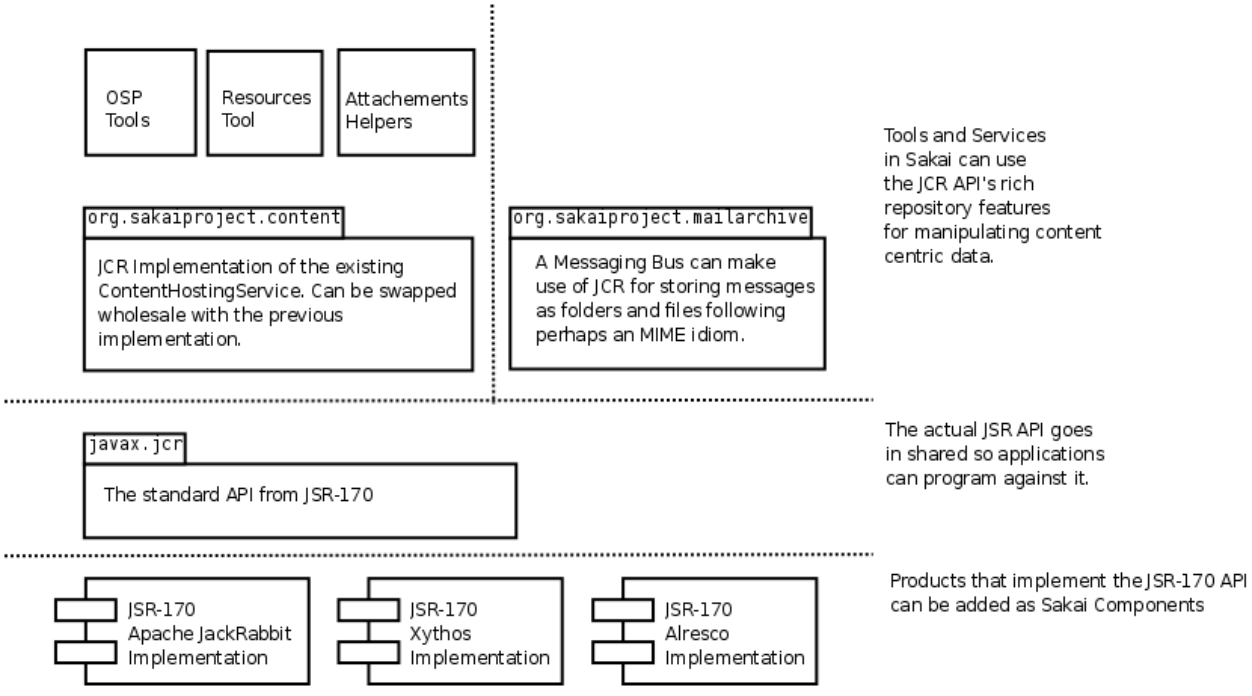
The JavaDocs, API jar, and PDF formatted specifications can be found here:
http://jcp.org/en/jsr/detail?id=170 [Final 17 June 2005]
http://jcp.org/en/jsr/detail?id=283 [Final 25 September 2009]

The API from JSR-170, jcr-1.0.jar, is currently going into shared/lib in Sakai.

We define an addition API in the maven module sakai-jcr-api located in the top level sakai subversion module 'jcr'.

https://source.sakaiproject.org/svn/jcr/

This API allows you to obtain a javax.jcr.Session object. This is one of the most top level interfaces in the JCR API that allows you to perform work on the repository. At the moment, obtaining this Session is much like obtaining a DB connection in that there is no default Sakai security performed (although the underlying implementation may choose to enforce some security parameters).



The above diagram outlines the basic idea. There can be a number of implementations of the JCR API that can be installed as Sakai Components. Currently there are working

implementations of Apache JackRabbit and Xythos. It's reasonable to say that these are currently of beta quality. Alfresco research is currently underway.

While either of the these implementations can be used, it's not likely that they can be swapped in an existing production instance using one or the other, as the different implementations have wildly different storage schemas. However, a goal of the JCR specification is to allow the copying of resources between implementations via the standard API with some mounting semantics. As an example then, it is conceivable that you could have both a JackRabbit, Xythos, and Implementation XYZ in Sakai, and expose them with some amount of interoperability. This is described in Section 3.2 (Page 14) of the JSR-170 PDF Specification.

On top of the JCR API tools and services can work on top of the Content Repository. A JCR implementation of the existing org.sakaiproject.content.api.ContentHostingService contract in currently underway (roughly Alpha quality at the moment). Other components can then build on top of this as usual. There are plans to eventually supercede the ContentHostingService API that relies heavily on the existing JCR API's. This could be some time away, so a current priority is to completely implement the original API as to support existing tools and allow a smooth migration/deprecation process.

Other system components can be built as well. Under research and development is the ability to store Messages in JCR Files and Folders. In this fashion you can model MIME messages and other social postings, as these can naturally be modeled using a file/folder structure. A good read is David's Model, a set of tips from David Nuescheler (a lead JackRabbit developer) concerning the structuring of content when using JCR.

Content Hosting Stats

These are stats collected from various schools related to access of Content Hosting.
(feel free to contribute your numbers to this)

- Max/Avg Usage - indicates the peak and average hits per second in writes and reads

Usage	UM	UCT
Max content.read / sec	8	2
Max content.write / sec	0.8	0.001?
Avg content.read / sec	2.5	0.5
Avg content.write / sec	0.1	0.001?

- Related activity levels - indicates the ratio of the various Content Hosting CRUD operations on an item (compared to read as that is the most prevalent activity), in other words, this shows that reads happen 200x more often than updates

Activity	Ratio
read	1
create	1/20
update	1/200
delete	1/60

JSR Integration

Added by Antranig Basman, edited by Steven Githens on Nov 12, 2007

In advance of some more authoritative pages from Ian, I (Antranig) will be using this page as a dumping ground for results of my integration experiments with various JCR repositories. An initial bulk of this information is held at Using JSR-170 (Java Content Repository) which will be migrated over here gradually.

Alfresco JCR compatibility for Content Hosting

Held here are the results of my exploration of the potential of Alfresco as an alternative implementation to Apache Jackrabbit for the new ContentHostingSystem implementation in Sakai 2.5, backed by a standard JSR-170 Java Content Repository. There are two main categories of results, i) performance, ii) JAR/environmental/API compatibility.
The code for this investigation is currently packaged as a standalone Maven 2/Eclipse JUnit test case held at https://saffron.caret.cam.ac.uk/svn/projects/amb26/trunk/jcr-tests.

API/Repository compatibility

Alfresco complies to level 2 of the JCR spec, however its specific interpretation of the spec differs in a number of substantial ways from Jackrabbit's that would obstruct its suitability for a drop-in replacement for Jackrabbit in CHS.

Unstructured schema support

Firstly these is the issue that Alfresco has **no** support for dynamic or unstructured schemas exposed via JCR (and may indeed have none at all internally, I have not investigated this). The nt:unstructured JCR mixin is not implemented in the Alfresco schema. Properties which are not registered on a node type will cause a NullPointerException if they are accessed or written. Since the CHS implementation has a requirement to support arbitrary resource properties attached to a resource this would seem to obstruct creating an CHS implementation that allowed different implementations to show the same JCR node structure to clients. However this may be possible to paper over at a higher level with some utilities or other API abstractions.

Schema leniency

Alfresco is also less lenient than Jackrabbit on property multiplicity. Accessing a multi-valued property (defined using <multiple>true</multiple> as a scalar property via the API will cause an exception, and vice versa. Alfresco also **insists** that the jcr:primaryType property be set on every node since this property is <mandatory> in its schema. Unfortunate for JackRabbit, this property

is <optional> in its schema, and so causes an exception if set. This difference is "papered over" in the test code with a construction like this:

```
testNode = root.addNode(testPath, "nt:folder");

    if (repositoryFactory.requiresPrimaryType()) {
      testNode.setProperty("jcr:primaryType", "nt:folder");
    }
```

Versioning

Despite Alfresco possessing built-in version management support, the mix:versionable mixin is not present in its builtin JCR schema. It may be possible to implement this easily, or it may be that Alfresco version management in this release differs too substantially from the JCR semantics.

JAR/Environmental compatibility

Alfresco is delivered "out of the box" in two main configurations. Firstly as a complete Tomcat distribution "ready to run", and secondly as a WAR. There is also an "embedded/SDK" configuration which was the basis of this experimentation. The complete Tomcat is obviously unsuitable for direct integration with Sakai, although could be accessible over some form of RMI/Web Services bridge. The current CHS/JCR implementation expects the backing impl to be delivered as a Sakai component. This impedes the ability to use the WAR configuration, although it would be possible to construct some form of webapp/component bridge. The problem with this approach is ensuring that the lifecycles matched up and that the webapp were started before any use were made of the system.

This exploration intended to establish the obstacles to packaging Alfresco **directly** as a Sakai component. The "jcr-tests" project has succeeded in establishing a Maven 2 build profile containing the minimal set of JARs necessary to start up a functioning Alfresco repository accessible over JCR. These JARs could in theory be delivered as an alternate implementation of JCR CHS. The obstacles are the following:

Spring/ClassLoader compatibility

(see discussion on Sakai's Component Manager at Component Manager Upgrade). Sakai's ClassLoader structure within components is nonstandard. This would prevent most independently Spring-configured applications from starting up correctly, since references to resource within Spring configuration files will be referred to the wrong ClassLoader. Whilst Alfresco is bundled with Spring 2.0.2, I **have** verified that it starts up correctly with Spring 2.0.6, the version currently used within Sakai, so Spring version compatibility specifically is not an issue.

Hibernate compatibility

Alfresco is implemented using Hibernate. Unfortunately Hibernate is present at the shared ClassLoader level within Sakai, and so is visible in code throughout the entire system. The version of Hibernate bound to Alfresco 2.1.0 is 3.2.1ga, and the current Sakai version is 3.2.5ga. Unfortunately Alfresco will not run correctly with Sakai's version of Hibernate, due to a change in the syntax tree structure for HQL queries. On startup with 3.2.5ga Alfresco delivers the following exception

```
2007-11-09 19:44:22,373 ERROR (SessionFactoryImpl.java:363) - <Error in named
query: node.patch.GetNodesWithPersistedSerializableProperties>
org.hibernate.QueryException: illegal attempt to dereference collection
[nodeimpl0_.id.properties] with element property reference
[serializableValue] [
      select distinct
         node
      from
         org.alfresco.repo.domain.hibernate.NodeImpl as node
      where
         node.properties.serializableValue is not null and
         node.properties.multiValued = false
   ]
```

The complete exception text is present as an attachment to this page.

This indicates that without specific ClassLoader work, it would be impossible to deliver Alfresco within Sakai. See the Component Manager Upgrade page for discussion of some related work, especially the section relating to Hibernate visibility. The most direct route would seem to be to achieve "Stage 1" of the upgrade plans, restoring the correct use of Context ClassLoaders, and then to provide a custom Hibernate-isolating ClassLoader for shielding the Alfresco module. Unless Hibernate were somehow removed from shared, it would even be impossible to deploy the standard Alfresco WAR within Sakai.

Comparative performance

In the limited testing done so far, Alfresco performance seems considerably worse than that of Jackrabbit, both in terms of memory and CPU usage. The testing was somewhat unrealistic in terms of real-life loads, since it consisted of a large number of updates made to the repository made within a single Session. However, a significant degradation in JCR query performance was observed. I have very little understanding of how to configure Alfresco appropriately for good performance and it is certain that many of the problems in this section could be significantly relieved through better configuration, especially that of the Alfresco cache.

Also this testing was done on a Windows box with a very crowded disk partition, and so the absolute timings are an extremely poor guide to what could be achieved in a production environment. However the relative timings between Alfresco and Jackrabbit are probably reasonably accurate.

Repository loading

Alfresco warns at startup if it is configured in a JVM with less than 512Mb heap. When run in the default configuration (64Mb heap), it fails to execute a test which tries to create 1000 nodes due to heap exhaustion. When given 512Mb heap, it still fails to execute a test creating 10000 nodes.

On a 1000 node test for Jackrabbit:

```
Root node has 1000 descendents
Saved 10240000 bytes in 168298ms: 59.42 K/s, 5.94 nodes/s
```

However for Alfresco:

```
Root node has 665 descendents
Saved 6809600 bytes in 903344ms: 7.36 K/s, 0.74 nodes/s
```

Many of the nodes "appeared" lost due to the excessive time to complete the test causing expiration from the cache. However the nodes were correctly saved to the backing storage.

Querying

JCR XPath queries of increasing length were presented to both Alfresco and Jackrabbit. Alfresco queries were up to two orders of magnitude slower than the same Jackrabbit queries on the same node sets, with Alfresco queries typically taking more than a second to return result sets of a few hundred nodes. Some representative transcripts, after a few runs to let any caching bed down:

Jackrabbit:

```
Begin run 4
Query /jcr:root/JCRTestPath/element(*, nt:file) concluded: 47ms
Matched 1000 nodes
Counted 10239000 chars in 6812ms: 1467.85 K/s, 146.80 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0' ]
concluded: 16ms
Matched 200 nodes
Counted 2047800 chars in 828ms: 2415.22 K/s, 241.55 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1' ] concluded: 47ms
Matched 400 nodes
Counted 4095600 chars in 1875ms: 2133.12 K/s, 213.33 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2' ] concluded: 63ms
Matched 600 nodes
Counted 6143400 chars in 3079ms: 1948.49 K/s, 194.87 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2'  or @sakaijcr:aclkey = '3'
] concluded: 46ms
```

```
Matched 800 nodes
Counted 8191200 chars in 4203ms: 1903.22 K/s, 190.34 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2'  or @sakaijcr:aclkey = '3'
or @sakaijcr:aclkey = '4' ] concluded: 93ms
Matched 1000 nodes
Counted 10239000 chars in 6187ms: 1616.13 K/s, 161.63 nodes/s
Logged in as username to a Jackrabbit repository.
```

Alfresco:

```
Begin run 4
Query /jcr:root/JCRTestPath/element(*, nt:file) concluded: 1484ms
Matched 1000 nodes
Counted 10239000 chars in 58078ms: 172.17 K/s, 17.22 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0' ]
concluded: 1859ms
Matched 200 nodes
Counted 2047800 chars in 7359ms: 271.75 K/s, 27.18 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1' ] concluded: 1969ms
Matched 400 nodes
Counted 4095600 chars in 14406ms: 277.63 K/s, 27.77 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2' ] concluded: 2063ms
Matched 600 nodes
Counted 6143400 chars in 23344ms: 257.00 K/s, 25.70 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2'  or @sakaijcr:aclkey = '3'
] concluded: 2391ms
Matched 800 nodes
Counted 8191200 chars in 35078ms: 228.04 K/s, 22.81 nodes/s
Query /jcr:root/JCRTestPath/element(*, nt:file)[@sakaijcr:aclkey = '0'  or
@sakaijcr:aclkey = '1'  or @sakaijcr:aclkey = '2'  or @sakaijcr:aclkey = '3'
or @sakaijcr:aclkey = '4' ] concluded: 2563ms
Matched 1000 nodes
Counted 10239000 chars in 48547ms: 205.97 K/s, 20.60 nodes/s
Logged in as admin to a Alfresco Content Repository (Community Network)
repository.
```