Phase 1 Consumer Guide

Version Information
Some of the following information applies only to Sakaibrary work done for Sakai 2.3 (Sakaibrary's Phase 1). This work has progressed and will be included in Sakai 2.4 as the Citations Helper in the Resources Tool. Updated documentation to configure the Citations Helper is available at the Citations Helper page in Resources.

Description
The Sakaibrary Repository OSID Implementation (SROI) provides access to metasearch engines for searching an institution's licensed electronic databases for scholarly resources.

Contents
1. Installation Notes (applies only to Sakai 2.3)
2. Out-of-Band Agreements (valid for Sakai 2.4)
3. Implementation Notes (valid for Sakai 2.4)

Installation Notes
At this time, due to changes that have been made to the Resources tool, Sakaibrary software can only be run from within a modified version of Sakai. This means that Sakaibrary software cannot neatly be dropped into an existing version of Sakai. We are currently working on this and hope to have this capability soon. If you have a test or pilot server in use and would like to import other experimental tools into our version of Sakai (with a modified Resources tool), you should be able to do so. Please contact Gaurav if you’d like to work together to do this.

Installation Walkthrough

Development Environment Setup
Below you will be given directions on what to do to get Sakaibrary up and running and not how to get your development environment (the tools you will need) set up. You will, however, be informed of the tools you will need for each step and linked to the Programmer's Cafe Working Group's Development Environment Setup Walkthrough documentation on setting up individual tools. If you do not find enough context information on the individual tool pages, try the Development Environment Setup Walkthrough main page.

1 Checkout the code
The Sakaibrary Project has its own branch within the ‘contrib’ section of Sakai’s subversion repository. You will need to check out the most current tag to get our latest version.

<table>
<thead>
<tr>
<th>What to do</th>
<th>Tools you will need</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkout the latest Sakaibrary code (sakaibrary_phase-1-006)</td>
<td>Subversion</td>
<td>Do not use Eclipse's Subversion plugin to checkout. Checkout will take a couple of minutes.</td>
</tr>
</tbody>
</table>

svn checkout https://source.sakaiproject.org/contrib/sakaibrary/branches/sakaibrary_phase-1-006/

This will checkout a full version of Sakai 2.3 along with Sakaibrary code and place it into a folder with the same name as the tag.

2 Configure
There are a few configuration parameters that you must attend to.

<table>
<thead>
<tr>
<th>What to do</th>
<th>Tools you will need</th>
<th>Notes</th>
</tr>
</thead>
</table>
### 3 Build and deploy

You can build and deploy Sakai as you normally would using Java, Maven, Tomcat and perhaps a database such as MySQL or Oracle.

<table>
<thead>
<tr>
<th>What to do</th>
<th>Tools you will need</th>
<th>Notes</th>
</tr>
</thead>
</table>
| **Configure Maven's build.properties** <br>Open (or create) build.properties in your user home directory (i.e. gbhatnag) <br>/Users/gbhatnag/build.properties | **Maven 1.0.2**, Tomcat 5.5.x <br>Maven 1.0.2, Tomcat 5.5.x | - The trailing slashes are required  
- You must use forward slashes  
- You must have spaces around the "="  
- DO add c: to the path if using windows |

Make sure the following properties are defined:

- maven.tomcat.home = /opt/apache-tomcat-5.5.17/ (your Tomcat install)  
- maven.compile.source = 1.4  
- maven.compile.target = 1.4  
- maven.test.skip = true

| **Build Sakai** <br>From a shell prompt, navigate to your Sakai source root directory (i.e. sakaibrary_phase-1-005) and run: maven bld | **Java 1.4.2+**, Maven 1.0.2 <br>Java 1.4.2+, Maven 1.0.2 | This will take a couple of minutes... |

This will compile Sakai and make it ready to deploy.

| **Deploy Sakai** <br>From a shell prompt, navigate to your Sakai source root directory (i.e. sakaibrary_phase-1-005) and run: maven dpl | **Maven 1.0.2**, Tomcat 5.5.x <br>Maven 1.0.2, Tomcat 5.5.x | This will take less time than building.  
For future build/deploy cycles, you can use the maven sakai command, which cleans, builds and deploys the source code (maven sakai = maven cln bld dpl) |

This will deploy Sakai to your Tomcat installation (defined in build.properties).
### 4 Start Tomcat

You should now be able to start Tomcat and create Citation Lists using Sakaibrary software.

<table>
<thead>
<tr>
<th>What to do</th>
<th>Tools you will need</th>
<th>Notes</th>
</tr>
</thead>
</table>
| **Start Tomcat**  
From a shell prompt, run the command: | **Tomcat 5.5.x**  
$CATALINA_HOME/bin/startup.sh | **Start up should take a minute or two - you can tail the log at:** $CATALINA_HOME/logs/catalina.out |

**Verify Tomcat is running**  
Open http://localhost:8080/  
- you should see the generic Tomcat web page.  
**Verify Sakai is running**  
Open http://localhost:8080/portal  
- you should see the generic Sakai homepage.  

**Access the Citation List Editor and Search Library Resources Search Dialog**

- Login to Sakai using username: admin, password: admin
- Go to Resources
- Choose to Add a resource.
- From the 'Add item Type' drop-down selection box, choose "Citation List" - the page below should change to the Citation List Editor.
- Clicking on the 'Search Library Resources' button should pop-up a search dialog window with the search categories you have defined in RepositoryManager.properties configuration (for the X-Server) or searchsource.xml configuration (for the Web2Bridge).

### Out-of-Band Agreements

The Sakaibrary Repository OSID Implementation (SROI) is capable of conducting an asynchronous search of multiple databases. A number of out-of-band agreements (OBAs) need to be met by the Consumer to properly specify and manage such a search, as well as to utilize the results of such a search. For definitions of all SROI Types, please see the Repository Types page. Following is a summary of the major Types, Properties formats and criteria format needed to specify and manage a search using the SROI.

**getAssetsBySearch()**

The Repository method, getAssetsBySearch(), requires the following input parameters:

- java.io.Serializable searchCriteria
- org.osid.shared.Type searchType
- org.osid.shared.Properties searchProperties

**searchType**

The SROI defines and supports the sakaibrary / search / asynchMetasearch org.osid.shared.Type for searching. This search Type is used to initiate an asynchronous search of a metasearch engine and defines the other two fields (searchCriteria and searchProperties) associated with the getAssetsBySearch() method.

**searchCriteria**

The searchCriteria must be passed as a java.lang.String in CQL Format limited to the fields: **keyword, title, author, subject, year** and the relations: **{ = , and }**.

Examples are:

- (keyword = clara+clairvoyant) and (keyword = lyrics)
- (((((keyword = clara+clairvoyant) and (keyword = lyrics)) and (title = open+road)) and (author = donovan)) and (subject = rock+and+roll+music)) and (year = 1970)
Classes from the org.sakaibrary.common.search.impl package can be used to effectively parse user input and create proper CQL according to the above rules.

searchProperties

The SROI defines and supports the sakalibrary / properties / asynchMetasearch org.osid.shared.Type for properties. It is the responsibility of the Consumer to pass an object of type org.osid.shared.Properties to getAssetsBySearch() with the following properties:

- required properties are in red and starred
- * guid - A globally unique identifier. This is an alpha-numeric java.lang.String that uniquely identifies the specific user's search session.
- * username - A java.lang.String username used to login to the SROI's metasearch engine.
- * password - A java.lang.String password used to login to the SROI's metasearch engine.
- sortBy - A java.lang.String with one of the following values:
  - 'rank' - results are returned sorted by the metasearch engine's internal ranking algorithm.
  - 'title' - results are returned sorted by title in ascending order (A-Z).
  - 'year' - results are returned sorted by year in descending order (newer to older).
  - 'author' - results are returned sorted by author's last name in ascending order (A-Z).
  - 'database' - results are returned sorted by database in metasearch engine order.
  - if undefined, sortBy defaults to 'rank'.
- pageSize - A java.lang.Integer indicating how many records to display per page. If undefined, pageSize defaults to 10.
- startRecord - A java.lang.Integer indicating which record to begin displaying results at. Records start at 1 and not 0. If undefined, startRecord defaults to 1 (the first record).
- databaseIds - A java.util.List of identifiers used to identify the databases to be searched by the metasearch engine. In Sakai 2.3, this field is not used.

A convenient way to pass an object of type org.osid.shared.Properties to getAssetsBySearch() with the above properties is to implement the org.osid.shared.Properties interface creating a class that wraps a java.util.HashMap as illustrated below.

getPropertiesByType( org.osid.shared.Type propertiesType )

The SROI defines the sakalibrary / properties / metasearchStatus Type for retrieving status information regarding an initiated asynchronous search. The Repository's getPropertiesByType() method can be used to retrieve search status information from the Repository. The org.osid.shared.Properties object of Type sakalibrary / properties / metasearchStatus returned has a unique key-value structure. Below is a table describing the structure of the Properties object:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>delayHint</td>
<td>A java.lang.Integer representing the number of milliseconds the SROI recommends the Consumer wait before calling nextAsset() again. This number is only a suggestion and does not guarantee an Asset will be retrieved if the Consumer waits the indicated number of milliseconds.</td>
</tr>
<tr>
<td>databaseIds</td>
<td>a java.util.ArrayList containing the &lt;databaseId&gt; (java.lang.String) of each database being searched.</td>
</tr>
<tr>
<td>status</td>
<td>A java.lang.String with status notification (searching, fetching, ready, error, timeout) for the entire search set.</td>
</tr>
<tr>
<td>statusMessage</td>
<td>A java.lang.String with status details for the entire search set.</td>
</tr>
<tr>
<td>numRecordsFound</td>
<td>A java.lang.String representing how many total records have been found for the entire search set.</td>
</tr>
<tr>
<td>numRecordsFetched</td>
<td>A java.lang.String representing how many records have been fetched for the entire search set.</td>
</tr>
<tr>
<td>numRecordsMerged</td>
<td>A java.lang.String representing how many records have been merged for the entire search set.</td>
</tr>
<tr>
<td>&lt;databaseId&gt; - the id of a database being searched (returned by databaseIds)</td>
<td>A java.util.Map containing all of the above fields except for databaseIds, delayHint and numRecordsMerged. These fields provide data about the specific database (with &lt;databaseId&gt;) being searched as opposed to data pertaining to the entire search set.</td>
</tr>
</tbody>
</table>

AssetIterator methods hasNextAsset() and nextAsset()

Access to individual Assets is encapsulated by the AssetIterator returned by Repository.getAssetsBySearch(). Calling hasNextAsset() returns a boolean indicating whether or not the end of the AssetIterator has been reached: true indicates there are more Assets in the AssetIterator, false indicates that there are no more Assets left. Calling nextAsset() returns an Asset, if it is available.

Due to the nature of an asynchronous search, there are a number of situations in which the hasNextAsset() and nextAsset() methods may not behave as described above. Consequently, the SROI provides special exception behavior for these two methods.

Method Behavior
The only Exception that may be thrown is an org.osid.repository.RepositoryException. This Exception is always thrown with a specific message indicating a specific problem that has occurred. These RepositoryException messages and the problems they indicate are described below.

**hasNextAsset()**
- returns true if the AssetIterator cursor has not yet reached the total number of records found.
- returns false if the AssetIterator cursor has reached the total number of records found.
- throws a RepositoryException with the SESSION_TIMED_OUT message if the session associated with this user's guid has been timed out by the metasearch engine.
- throws a RepositoryException with the METASEARCH_ERROR message if the search has failed due to a metasearch engine error.
- throws a RepositoryException with the OPERATION_FAILED message if resources the method needs cannot be gathered.

**nextAsset()**
- returns an Asset if the AssetIterator cursor has not yet reached the total number of records found and an Asset is readily available.
- throws a RepositoryException with the SESSION_TIMED_OUT message if the session associated with this user's guid has been timed out by the metasearch engine.
- throws a RepositoryException with the METASEARCH_ERROR message if the search has failed due to a metasearch engine error.
- throws a RepositoryException with the NO_MORE_ITERATOR_ELEMENTS message if the AssetIterator cursor has reached the total number of records found (equivalent to hasNextAsset() returning false).
- throws a RepositoryException with the ASSET_NOT_FETCHED message if the AssetIterator cursor has not yet reached the total number of records found (equivalent to hasNextAsset() returning true) but the current Asset has not yet been fetched.
- throws a RepositoryException with the OPERATION_FAILED message if resources the method needs cannot be gathered.

### Implementation Notes

1. Get a RepositoryManager with org.sakaibrary.osid.repository.xserver or org.sakaibrary.osid.repository.web2 as the implementation package name:

```java
org.osid.repository.RepositoryManager repositoryManager = OsidLoader.getManager(  
    "org.osid.repository.RepositoryManager", "org.sakaibrary.osid.repository.xserver", new org.osid.OsidContext(),  
    null );
```

2. Get the correct Repository Type from the RepositoryManager:

```java
// find Repository Types
org.osid.shared.TypeIterator typeIterator = repositoryManager.getRepositoryTypes();

// we are interested in "sakaibrary / repository / metasearch" type
org.osid.shared.Type tempType;
org.osid.shared.repositoryType = null;

while( typeIterator.hasNextType() ) {
    tempType = typeIterator.nextToken();
    if( tempType != null && tempType.getAuthority() != null &&  
        tempType.getDomain() != null &&  
        tempType.getKeyword() != null ) {
        if( tempType.getAuthority().equals( "sakaibrary" ) &&  
            tempType.getDomain().equals( "repository" ) &&  
            tempType.getKeyword().equals( "metasearch" ) ) {
            repositoryType = tempType;
            break;
        }
    }
}
```

3. Get Repositories of Type `sakaibrary / repository / metasearch` and find the selected Repository to search on (repository with 'repositoryName' has been selected):
org.osid.repository.RepositoryIterator repositoryIterator = repositoryManager.getRepositoriesByType(
    repositoryType);

while( repositoryIterator.hasNextRepository() ) {
    org.osid.repository.Repository repository = repositoryIterator.nextRepository();
    
    if( repository.getDisplayName().equals( repositoryName ) ) {
        /* found a matching repository - do search */
        // format search criteria to CQL
        // setup search properties
        // call getAssetsBySearch()
        // poll for results
    }
}

4. Setup search properties and call getAssetsBySearch()

// setup search properties
java.util.HashMap searchPropertiesMap = new java.util.HashMap();
searchPropertiesMap.put( "guid", "i9309we90rn203942309487kh0" );
searchPropertiesMap.put( "sortBy", "rank" );
searchPropertiesMap.put( "pageSize", new Integer( 10 ) );
searchPropertiesMap.put( "startRecord", new Integer( 1 ) );
searchPropertiesMap.put( "maxRecords", new Integer( 500 ) );

// ConsumerSearchProperties implements org.osid.shared.Properties and wraps a HashMap
org.osid.shared.Properties searchProperties = new ConsumerSearchProperties( searchPropertiesMap );

// Conduct search
org.osid.repository.AssetIterator assets = repository.getAssetsBySearch( criteria, searchType,
    searchProperties );

5. Poll for results.
try {
    while( assets.hasNextAsset() ) {
        try {
            asset = assets.nextAsset();
            // Consumer gets the Asset - does something with it...
        } catch( RepositoryException re ) {
            if( re.getMessage().equals( org.sakaibrary.osid.repository.MetasearchException.SESSION_TIMED_OUT ) ||
                re.getMessage().equals( org.sakaibrary.osid.repository.MetasearchException.METASEARCH_ERROR ) ||
                re.getMessage().equals( org.osid.shared.SharedException.NO_MORE_ITERATOR_ELEMENTS ) ||
                re.getMessage().equals( org.osid.OsidException.OPERATION_FAILED ) ) {
                // search is over, all assets that have been retrieved have been
                // optionally check searchStatus Properties for further details or information to present in UI
                break;
            } else if( re.getMessage().equals( org.sakaibrary.osid.repository.MetasearchException.ASSET_NOT_FETCHED ) ) {
                // need to wait some time and then try again
                Thread.sleep( ... );
            }
        } // catch
    } // while
} catch( RepositoryException re ) {
    if( re.getMessage().equals( org.sakaibrary.osid.repository.MetasearchException.SESSION_TIMED_OUT ) ||
        re.getMessage().equals( org.sakaibrary.osid.repository.MetasearchException.METASEARCH_ERROR ) ) {
        // search is over, all assets that have been retrieved have been
        // optionally check searchStatus Properties for further details or information to present in UI
    }
}