# Installation Guide

## Information

Installation instructions for SakaiAdminX.

**Required:**

- Sakai 2.4.x or higher
- Maven 2 configured and ready to go.
- A Tomcat instance, or you can deploy alongside Sakai. Once configured and built, the SakaiAdminX webapp will be a single WAR file that you can deploy anywhere you like. If you deploy alongside Sakai, SakaiAdminX will be much faster. In a future release, there will be an 'API mode' where SakaiAdminX uses Sakai's API and DB instead of webservice calls/embedded db.

### Get the source code

Checkout the code from the Sakai Contrib SVN repository:

```
svn checkout https://source.sakaiproject.org/contrib/sakaiadminx/trunk sakaiadminx
```

There are a number of different parts of SakaiAdminX that all work together to perform the various tasks required. There is a tag library, a set of quartz jobs, an embedded Derby database, some additional webservices and the webapp itself. Each needs to be configured and then built in a specific order, then deployed into the correct location. Please read the following instructions carefully, before building and deploying.

Each component of SakaiAdminX needs to be configured and built in a specific order because some have dependencies on each other. The configuration/build order is this:

1. Tag library
2. Database configuration
3. Configuring Sakai
4. Web Services install
5. Webapp configuration, building and deployment
6. Quartz job installation (+ appropriate additions to sakai.properties)

The next few sections will take you through, step-by-step how to configure and build each component

### Tag-library

#### Configuration:

Navigate to:

```
SAKAIADMINX-SRC/taglib/src/main/java/uk/ac/lancs/e_science/taglib/common/
```

and open **Page.java**.

Change the values of the configuration params to match your environment. There are examples in the file. This process will eventually be replaced with a taglib.properties file which will automatically configure the tag-library at compile-time.

#### Building:

Navigate to:

```
SAKAIADMINX-SRC/taglib/
```

Build the tag library using Maven by issuing the following command:

```
mvn clean install
```

The taglibrary will be built and installed to your Maven repository. For reference, the taglibrary documentation will also be created and packaged as a jar, located at:

```
SAKAIADMINX-SRC/taglib/target/sakaiadminx-taglib-VERSION-tlddoc.jar
```

which can be unzipped and viewed in a web browser.

## Database install and configuration

SakaiAdminX uses an embedded Derby database to store some configuration information, as well as acting as a cache for the heaviest queries. It runs in embedded mode which is very lightweight.

You could, however, use MySQL or Oracle if you prefer and you will just need to adjust your settings and use the appropriate drivers in place of the Derby ones. SakaiAdminX ships with DB scripts for both Derby and MySQL. If you want to run SakaiAdminX on Oracle, you might need to tweak the scripts - if you do, send them to me for inclusion in the release. All SQL in SakaiAdminX should be database independent.

⚠ **If you are using a database other than Derby**

You will need to create your database using either the MySQL console or your favourite database editor. You should create your database with the UTF-8 character set, ie 'create database sakaiadminx default character set utf8;'

The following is for those who wish to use Derby, if you are using MySQL/Oracle, you can skip this section - BUT, make sure you **populate your database with the included scripts**.

### Download, install and configure Apache Derby

Download the binary Apache Derby distribution from the Derby web site at http://db.apache.org/derby/derby_downloads.html. This installation guide uses version 10.4.1.3 but if a more recent release is available, download that, then substitute that version number for 10.4.1.3 in the following instructions.

After downloading db-derby-10.4.1.3-bin.tar.gz, extract it to

```
/usr/local/db-derby-10.4.1.3-bin
```

then issue the following commands:

```
cd /usr/local
sudo ln -s db-derby-10.4.1.3-bin derby
```

You now need to set an environment variable called DERBY_HOME:

```
DERBY_HOME=/usr/local/derby
export DERBY_HOME
```

Now, edit your CLASSPATH environment variable to include the following Derby jars:

```
CLASSPATH=$CLASSPATH:$CATALINA_HOME/common/lib/:$DERBY_HOME/lib/derbytools.jar:$DERBY_HOME/lib/derby.jar
export CLASSPATH
```

To verify you have Derby installed correctly, run the following command

```
java org.apache.derby.tools.sysinfo
```

Successful output will look something like this:

```
------------------ Java Information ------------------
Java Version:    1.5.0_13
Java Vendor:     Apple Inc.
Java home:       /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home
Java classpath:  :/Users/steve/dev/tomcat/current/common/lib/:/usr/local/derby/lib/derbytools.jar:/usr/local
/derby/lib/derby.jar
OS name:         Mac OS X
OS architecture: i386
OS version:      10.5.4
Java user name:  steve
Java user home:  /Users/steve
Java user dir:   /usr/local/db-derby-10.4.1.3-bin
java.specification.name: Java Platform API Specification
java.specification.version: 1.5
--------- Derby Information --------
JRE - JDBC: J2SE 5.0 - JDBC 3.0
[/usr/local/db-derby-10.4.1.3-bin/lib/derby.jar] 10.4.1.3 - (648739)
[/usr/local/db-derby-10.4.1.3-bin/lib/derbytools.jar] 10.4.1.3 - (648739)
------------------------------------------------------
----------------- Locale Information -----------------
Current Locale :  [English/United States [en_US]]
Found support for locale: [cs]
.
.
.
Found support for locale: [zh_TW]
         version: 10.4.1.3 - (648739)
------------------------------------------------------
```

The output on your system will probably be somewhat different from the output shown above, but it should reflect the correct location of jar files on your machine and there shouldn't be any errors. If you see an error like the one below, it means your class path is not correctly set:

```
$ java org.apache.derby.tools.sysinfo
Exception in thread "main" java.lang.NoClassDefFoundError:
     org/apache/derby/tools/sysinfo
```

Echo your CLASSPATH, as shown below, then double check each entry in your class path to verify that the jar file is where you expect it:

```
echo $CLASSPATH
```

gives:

```
:/Users/steve/dev/tomcat/current/common/lib/:/usr/local/derby/lib/derbytools.jar:/usr/local/derby/lib/derby.jar
```

If sysinfo outputs valid information, you're ready to move on.

## Install the SakaiAdminX database tables and configuration data

The next few steps create the SakaiAdminX database and populate it with the initial information. First you should decide where you want the SakaiAdminX DB files to reside. This is similar to a MySQL setup. Once you have your filesystem path, we then need to create the database and populate it with the provided SQL script.We do this by running an application included with Derby called 'ij'.
Start up ij with this command:

```
java org.apache.derby.tools.ij
```

You should see the output shown below:

```
ij version 10.4
ij>
```

The error below means the classpath isn't set correctly:

```
java org.apache.derby.tools.ij
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/derby/tools/ij
```

If all is well, we can create the SakaiAdminX database and populate it:

With ij running, issue the following command to create the database:

```
ij> connect 'jdbc:derby:/path/to/where/you/want/your/db/sakaiadminx;create=true';
```

Note down this jdbc:derby:xxx line as you will need it for configuring the webapp later.

Now you need to import the SakaiAdminX SQL script, so issue the following command, where the path needs to be the full path to the sql script in the SakaiAdminX source:

```
ij> run 'SAKAIADMINX-SRC/sql/derby-sakaiadminx-v1.0.sql';
ij> exit;
```

**Upgrading SakaiAdminX**

You will also need to run any database upgrade scripts via this same method above. Note that for a first time installation, you can either run the creation script then the upgrade script, or the combined script to do it in one go.

### Add the Derby DB driver to Tomcat:

Since Tomcat will be connecting to our embedded Derby DB, make sure Tomcat has the Derby JAR:

```
cp /usr/local/derby/lib/derby.jar $CATALINA_HOME/common/lib/
```

Congratulations, you have now setup Derby for user with SakaiAdminX!

**Note about Derby in embedded mode:** Derby in embedded mode means only one JVM can use Derby at a time. Don't be alarmed, this DOES NOT mean that only one user on the web can access the application, it simply means that you cannot have ij, Tomcat and a connection via Eclipse (if using the Eclipse Derby plugin) all connected to the same Derby instance at the same time. So just make sure you are not running ij or have Eclipse connected to your Derby db when you start Tomcat, or you will get connection errors.

## Sakai configuration

This section describes what configuration must be done to Sakai to ensure it will work with SakaiAdminX. Do not be alarmed, these are minor adjustments and will not affect an existing sites or users in Sakai (although your existing sites should be imported to SakaiAdminX so they can be managed there - more on that later)

### sakai.properties

You must ensure that the following key-value is set:

```
webservices.allowlogin=true
```

### !site.template or !site.template.project

This is the base realm for your initial template site, and for any sites where you choose a completely blank site (generally only if you are creating a new template from scratch). If you do not have a !site.template.project, then !site.template will be used.

Ensure this realm has the following roles:

access
maintain
guest
helpdesk
admin

where access and maintain are the defaults, guest and helpdesk are simply copies of the access role, and admin is a copy of the maintain role. You can add extra roles if you wish, and you can manage them in SakaiAdminX automatically - so long as you create the correct role mappings.

**Note:** If your roles in these realms are different, you will need to change the corresponding role mappings in SakaiAdminX (see the Data Management section).

### /site/!admin

This realm acts as a container for the admin and helpdesk users. This realm should already have the admin role (with a single permission: site.upd), and admin users will be allocated this role, however in order for the helpdesk users to be added to this realm as well, you need to add a new role called 'helpdesk' to this realm that has no permissions. This realm should then contain the following roles:

admin
helpdesk

remembering that the helpdesk role should not have any permissions, because this role is simply a container for the users in the realm.

### !site.template.clientconfig

This is a new realm that needs to be created in Sakai and is the base realm for client configuration sites. It should contain the following roles:

clientadmin
clienthelpdesk

however these roles MUST NOT HAVE ANY PERMISSIONS. The reason for this is that this realm is simply a container for the users in the realm.

**Reason:** When a client is created, it will use this realm as the base. Then, when users are added as either clientadmin or clienthelpdesk to a client, they will be added with these roles to the appropriate client site. Since they have no privileges, they cannot do anything in this site and they cannot even see the site in Sakai. This is the idea! The Quartz jobs that sync up the sites will see these users and load them into the correct sites, and since the correct sites have the correctly configured roles (see !site.template configuration above), they will inherit the right permissions.

You should also set the 'Maintain Role' in this realm to be 'clientadmin'. This won't actually do anything, but it will stop Sakai from complaining in the log files that the 'maintain' role doesn't exist, as when you save this realm Sakai will add 'maintain' into this field by default, but this role doesn't exist. So replace it with 'clientadmin'.

**Note:** It's actually quite important that these roles do not have any permissions otherwise the site list that is rendered for the users in these roles may include these Client Configuration sites.

### A template site.

You should create at least one template site, that contains a minimal set of pages/tools etc. You should do this within SakaiAdminX choosing 'No Template' in the Create a Site form. You will still need to setup the pages and tools in Sakai though, but the SakaiAdminX DB record is created for you, plus you get control over the siteid, which is important for SakaiAdminX.

I tend to call my templates template_common or template_everything etc.

Since this requires an operational SakaiAdminX, you will need to come back to this step after you have completed the rest of the install steps.

**Note:** You MUST give the template site the type of 'Template' so that this site is then available in the list for creating other sites from. Please also ensure that you have properly configured your roles as per point 1 above, as once you start creating sites based on other sites, it's a little tedious to go back and fix up the roles in the sites if they weren't created properly to start with.

### A 'helpdesk' user

Using either SakaiAdminX or the Sakai Admin tools, create a user account with the username 'helpdesk'. If this conflicts with your local username list or some other policy, you can use a different username, but be sure to amend the property in sakaiadminx.properties to be the right one.

This user will be added to every site with the role 'helpdesk' so that helpdesk users are able to view all sites.

You MUST ensure that the above sites/realms/settings are created or configured correctly.

## Web services install

SakaiAdminX requires additional web services to be installed into Sakai to complete the integration between SakaiAdminX and Sakai itself. SakaiAdminX ships with a replacement SakaiScript.jws for the Sakai web services module located at:

```
SAKAI_SRC/webservices/axis/src/webapp/SakaiScript.jws
```

which is exactly the same as the base SakaiScript.jws in Sakai 2.4 and 2.5, but has a few extra methods included. If you have made local modifications to your SakaiScript.jws, you will need to merge the files together.

The following section assumes you have already preserved any local modifications to your Sakai web services.

### Installation:

Navigate to:

```
SAKAIADMINX-SRC/ws/
```

and issue the following command:

```
cp SakaiScript.jws SAKAI_SRC/webservices/axis/src/webapp/SakaiScript.jws
```

Now you need to rebuild the Sakai web services by issusing the following command in SAKAI-SRC/webservices/

```
mvn clean install sakai:deploy -Dmaven.tomcat.home=/path/to/your/sakai/tomcat/
```

Or build the web services module of Sakai in your usual way.

## Tomcat configuration

SakaiAdminX doesn't require any special Tomcat configuration, except making sure the correct database driver jar is in TOMCAT /common/lib (which was done in the database section above).

SakaiAdminX does cache much of its data into memory though, so ensure you have enough memory allocated to your JVM, set via your JAVA_OPTS, ie:

```
JAVA_OPTS='-Xmx1500m -Xms1500m -Xmn1000m -XX:NewSize=400m -XX:MaxNewSize=400m -XX:PermSize=256m'
```

## Webapp config and install

Almost there! This is the easiest part:

### Configuration

Navigate to:

```
SAKAIADMINX-SRC/webapp/src/main/config
```

Copy sakaiadminx.properties.sample to sakaiadminx.properties if you don't have a sakaiadminx.properties already.

Edit the values of the configuration parameters in sakaiadminx.properties to match your environment. There are guidelines and examples in the file. These correspond with params in the various configuration files (web.xml, context.xml, log4j.properties etc) and will be replaced at buildtime.

### Building

Navigate to:

```
SAKAIADMINX-SRC/webapp/
```

SakaiAdminX will automatically deploy to $CATALINA_HOME/webapps/sakaiadminx if you do not make any changes to the pom.xml file. If you want to change the location, edit the pom.xml and edit

```
<webappDirectory>${env.CATALINA_HOME}/webapps/sakaiadminx</webappDirectory>
```

to the location you want.

If you don't want SakaiAdminX to be deployed automatically, simply comment this line out by surrounding it with <!-- and -->

When you are ready to build and deploy, issue the following command:

```
mvn clean package
```

The taglibrary that you built previously will be bundled, and all other dependencies will either be downloaded automatically by Maven, or fetched from your local repo. The webapp will be packaged as a WAR file and deployed to your Tomcat container in $CATALINA_HOME/webapps/ if you have the pom.xml setup to do this, or will be ready for deployment from:

```
SAKAIADMINX-SRC/webapp/target/sakaiadminx-webapp-VERSION.war
```

## SakaiAdminX Quartz jobs

To fully complete the integration between SakaiAdminX and Sakai, a set of Quartz jobs need to be installed and configured to run at appropriate times on the Sakai server. The Quartz jobs are described below:

- SuperHelpdeskUserSync: This ensures that the user 'helpdesk' with role 'helpdesk' is in every site in Sakai so that those users that are superhelpdesk users can access every site when accessing Sakai via SakaiAdminX's single sign-on.
- ClientAdminAndClientHelpdeskUserSync: This job gets the list of client configuration sites (identified by /site/config.CLIENTID) and the list of clientadmin and clienthelpdesk users in each and ensures that these users have either 'admin' or 'helpdesk' access to each site that that is attached to these clients.

### Installation

The quartz jobs build using maven 2 and register themselves into your Sakai Quartz Jobscheduler.
Navigate to:

```
SAKAIADMINX-SRC/
```

and copy the folder 'quartz' to a directory in your SAKAI-SRC

```
cp -pvr quartz SAKAI-SRC/sakaiadminx-quartz
```

Now build the quartz jobs using Maven by navigating to:

```
SAKAI-SRC/sakaiadminx-quartz
```

and building :

```
mvn clean install sakai:deploy
```

These jobs require a few extra sakai.properties to be set, but you only need them if you have different admin users/helpdesk users role etc. The properties and their defaults are listed below.

```
sakaiadminx.admin.user = admin
sakaiadminx.helpdesk.user = helpdesk
sakaiadminx.helpdesk.role = helpdesk
sakaiadminx.special.sites = !admin, PortfolioAdmin, citationsAdmin
sakaiadminx.client.admin.role = clientadmin
sakaiadminx.client.helpdesk.role = clienthelpdesk
sakaiadminx.site.admin.role = admin
sakaiadminx.site.helpdesk.role = helpdesk
```

If you have not changed any of the roles then these should all be fine and can be omitted or added to sakai.properties as is. One point to note is that the list of special sites are the sites that you do not want the SakaiAdminX Quartz Jobs to ever touch. You could expand this to include other sites if you wish.

After restarting Sakai, the SakaiAdminX quartz jobs will be in the list of jobs found in the **Admin Workspace** > **Job Scheduler** > Jobs list.

### Configuration of the jobs in the Sakai Jobscheduler

- Click on the Job Scheduler tool (in the left nav tool list).
- Click on Jobs (at the top of the pane).
- Click on New Job.
- Enter a Job Name.
- Select a job from the pop up list.
- Click the "Post" button.
- In the new job listing, click on the Triggers link.
- Click on "New Trigger".
- Create a Trigger Name.
- Create a Cron Expression. Click the "help" link next to this field for cron expression information. (This "0 0 0/1 * * ?" means fire every hour.)
- Click on "Post".
- Click on "Event Log" to watch for your jobs to fire or choose "Run Job Now" to fire your job immediately.

Output from these jobs is written to catalina.out however only warnings or errors will normally be present unless you change your Sakai log level to include INFO messages.

Do this by adding this to your sakai.properties (amend if you already have one of these blocks):

```
log.config.count=1
log.config.1=INFO.uk.ac.lancs.e_science.sakaiadminx.jobs
```

You absolutely need to configure the SuperHelpdeskUserSync job to run every few hours at least, otherwise the users that you add as Super Helpdesk users will have a large delay in getting access.

You will probably need the Client Admin / Client Helpdesk job, depending on your requirements.

To determine if you need the second Quartz job, answer these questions:
1. Do you have more than one client?
2. Do you use the client admin and client helpdesk features of SakaiAdminX?
3. Do you intend do do any of the above in the future?

If you answered YES to any of these questions, you need the second one configured.

## Additional Sakai modifications

There are a number of simple modifications you should perform to your Sakai environment to make it more tightly integrated with SakaiAdminX. For a detailed guide, see here Additional customisations for a complete integration

Please see the rest of the SakaiAdminX guides for more information.