How to Configure Sakai

July 19, 2005

Please direct questions or comments about this document to:
Glenn R. Golden, ggolden@umich.edu

This document covers how Sakai is configured. There are two different types of configuration in Sakai, configuration of components, and general configuration, used by tools and other parts of the system. Files in the Sakai code base initially set these configurations. Additional and override configuration values can be specified in configuration files, formatted as standard Java Properties files, located on the Sakai application server.

Component Configuration

Sakai's components are configured using the Spring bean definition property elements in the `components.xml` files. All configuration values for a component are specified in this way, and have a matching setter method to accept the values in the java class.

Sakai ships with default configuration values for all properties.

You may of course modify these before you deploy your Sakai instance by editing the `components.xml` files distributed throughout the software, but this becomes hard to maintain as the Sakai code base changes.

Instead, Sakai provides ways to override the configuration values. Files located in the Sakai home or Sakai security directories on the application server are read by Sakai to do this. These properties files can have entries that override any bean property setting in a `components.xml` file.

These entries take the form of a key with the property name, followed by the "@" character, followed by the component's bean id, fully dotted. (Read this as "property 'at' bean"). Then specify the value to use.

For example, the SMTP server used for outgoing email in Sakai can be set by including this line in a Sakai .properties file:

```
smtp@org.sakaiproject.email.api.EmailService=234.434.23.111
```

This works to override the value from the email service's entry in the `components.xml` file:

```
<bean id=" org.sakaiproject.email.api.EmailService "
class="org.sakaiproject.component.framework.email.BasicEmailService"
            init-method="init"
```

```
                destroy-method="destroy"
                singleton="true">
          <property name="smtp"><null/></property>
          <property name="logger">
                <ref bean="org….Logger"/></property>
    </bean>
```

Additional bean properties can be set in this way, but these must have corresponding setter methods in the class.

Note, the format of the keys in this file is different from the Spring format for bean configurers to accommodate our dotted bean ids.

Other Configuration

Some parts of Sakai that are not components also need configuration, such as the Portal and Tools. These make use of the `ServerConfigurationService` to find configuration values.

The `ServerConfigurationService` holds a set of configuration values, some with special getter methods, and others available with the generic `getString(name)` method. The configuration values it has available are those from the Sakai .properties files, the same set of values that are used to override components properties.

Most of the values of interest to the `ServerConfigurationService` are simple keys in the Sakai .properties files. These are keys that are just the property name that the code is asking the `ServerConfigurationService` for, not using the '@' format used to override a component property.

For example, to set the server id, enter this in a Sakai .properties file:

```
serverId=mongoose
```

Because these keys do not have the '@' character, they are not confused with the component property overrides, and can be included in the same files.

New properties can be added in this way. These will be available from the `ServerConfigurationService`.

All configuration values from the Sakai .properties files are available from the `ServerConfigurationService`. All of the configuration values can also act as placeholders.

Placeholder Configuration

Sakai also supports placeholder values in the bean definitions and override properties. These are strings looking like this:

```
${key}
```

These values are set in a Sakai .properties file, among the override values and simple `ServerConfigurationService` values. Any setting in the .properties files can be used as a placeholder value.

You can use a placeholder in the `components.xml` property definition, like this:

```
<bean …>
      <property name="checkEvery">
            <value>${session.check}</value></property>
</bean>
```

Then you can define `session.check` in one of your .properties files:

```
session.check=30
```

This value will be used anytime the placeholder `${session.check}` is found in any bean definitions.

Placeholder strings can also be used in the any Sakai .properties file, as parts of override values.

Placeholder strings cannot be used within simple `ServerConfigurationService` values – they will show up as their placeholder syntax and not be expanded.

All placeholders used in the distributed Sakai code need to be satisfied somewhere in the Sakai .properties files.

The Sakai home path is automatically available as the placeholder variable `${sakai.home}`.

Sakai Home and Security folders

To establish a Sakai home directory on your application server, you can modify the java startup command to set the system property `sakai.home`:

```
-Dsakai.home=/path/to/desired/sakai/home/
```

Specify the complete path to the folder in which Sakai configuration files and other files can be found. This should be readable and writable by the Sakai code.

If this is not done, and Sakai is running in Tomcat, then a folder called “sakai” in the tomcat root folder will be used as the Sakai home.

If for some strange reason Sakai does not detect that it is running in Tomcat, the home will be set to `/usr/local/sakai/` .

To establish a Sakai security directory on your application server, you can modify the java startup command to set the system property `sakai.security`:

```
-Dsakai.security=/full/path/to/folder
```

If this is not set, no default value is used.

.properties Files Locations

Sakai reads a number of optional files from the Sakai classpath, the Sakai home directory, and the Sakai security directory to form the set of configuration values. Files are read in a specific order; the files read in later override settings from files read earlier. These files are, in order:

- (classpath, sourced in the kernel/component module) /org/sakaiproject/config/sakai.properties
- (file) sakai.home/sakai,properties
- (file) sakai.home/local.properties
- (file) sakai.security/security.properties

Any of these files can be missing – they are all optional. The classpath set of properties is distributed with Sakai and must supply defaults for all required settings.

Usually you will set your override configuration values in the `sakai.home/sakai.properties` file.

To support common configuration between application servers in a cluster, you can give them all the same `sakai.properties`, but specify their server specific settings (usually just `serverId`) in `sakai.home/local.properties`.

To support security sensitive settings, like a database user and password, you can establish a sakai.security folder, and put these settings in the `sakai.security/security.properties` file.

Change In Practice

Sakai components used to look to the `ServerConfigurationService` in their `init()` methods to override selected configuration parameters. We no longer do this. Instead, all parameters of all components are subject to override using the .properties files.

The `components.xml` bean definitions for components in older versions of Sakai were incomplete. They are no longer so. Instead every possible value is specified, some with the `<null/>` value. This makes it clearer what properties are available for overriding.

Sakai 2.0.0 had a properties file called “placeholder.properties”, with special restrictions on its use. This file is no longer read by Sakai. Now you can mix your placeholder

definitions with all the other Sakai configuration values, in any .properties file read by Sakai.