# Sakai Kernel Bundle Service Manual - Tools

Date: November 9, 2005
Version: 2

## Table of Contents

# 1 Overview

This document describes the Sakai Tool Manager and related functions. Requirements for this service are defined in *Sakai Kernel Bundle Requirements* [1]. This document is based, in part, on an earlier description of tools by Glenn Golden [5].

## 1.1 Tool Definition

A *tool* is a piece of software that generates a user interface and responds to requests from users. This response could be an HTML fragment, or it might be a binary stream (in file downloads, for example). Sakai tools are usually intended to be embedded in a larger interface, such as a portal.

## 1.2 Helper Tools

A *helper tool* is a tool intended to manage some well-defined part of a user interface, such as the date picker. Helper tools can be mapped to part of the containing tool's URL space.

## 1.3 Tools and Web Applications

Although a Sakai tool can be set up as a web application, in most cases the incoming HTTP request is handled by a Tomcat Filter like the RequestFilter or the FacesServlet. It is possible to write your own filter, but most of the typical request cases are covered.

## 1.4 The Current Tool

The Tool Manager maintains a reference to the current tool associated with the current request. This reference is created when the request is received.

## 1.5 Tool Placement

The same tool can appear in more than one place in a portal or other managed web environment. To distinguish one instance of a tool from another, a *tool placement* is used. The tool placement is a string that can be used to build a URL designed to be processed by a request handler.

## 1.6 Active Tools

The Active Tool Manager extends Tool Manager to provide extended support for tools that might want to include, forward, or help. Active tools extend Sakai to include support for applications based on servlets.

# 2 Application Programming Interface

All interfaces and objects described here can be found in the *kernel* module and the *tool* and *active-tool* Module-Parts.

The following APIs are associated with the Tool Manager:

- ToolManager
- Tool
- Placement
- ActiveToolManager
- ActiveTool
- ToolURLManager
- ToolURL

## 2.1 The Tool Manager API

**Register a Tool given a Tool Object**

| | |
|---|---|
| void register(Tool tool); | Add this tool to the registry of known tools. |

**Register a Tool given its XML document**

| | |
|---|---|
| void register(Document toolXml); | Add all of the tools in this parsed XML document (based on the DOM) to the registery of known tools using the Tool XML schema. [Is this a published schema?] |

**Register a Tool given its XML file**

| | |
|---|---|
| void register(File toolXmlFile); | Add all of the tools in this unparsed XML file to the registry of known tools. The format of this file must conform to the Sakai Tool XML schema. |

**Register a Tool given its XML input stream**

| | |
|---|---|
| void register(InputStream toolXmlStream); | Add all of the tools in this XML input stream to the registry of known tools. The format of this input stream must conform to the Sakai Tool XML Schema. |

**Get a Tool given its well-known ID string**

| | |
|---|---|
| Tool getTool(String id); | Find and return the tool associated with the well-known ID string passed. Null is returned if the tool is not found. The well-known ID is defined in the tool registration file in the ID parameter of the <tool> tag. |

**Find the set of Tools that match categories and keywords**

| | |
|---|---|
| Set findTools(Set categories, Set keywords); | Find and return the set of tools (Tool objects) that match the categories and keywords provided. The category or keywords set can be null, which indicates that all categories or keywords match (wildcard). If both are missing, then all registered tools are returned. If a category and a keyword are provided, the tool must match both. If a set of categories and/or a set of keywords are supplied, then the tool must match one of the categories and one of the keywords. If no tools are found, null is returned. |

**Get the current Tool**

| | |
|---|---|
| Tool getCurrentTool(); | The tool associated with the current request or thread is returned or null if there is none. |

**Get the current Tool placement**

| | |
|---|---|
| Placement getCurrentPlacement(); | Returns the tool placement associated with the current request or thread or null if there is none. |

## 2.2 The Tool API

The Tool interface models a Sakai Tool that produces a user interface. Once created, the parameters associated with a tool object may not be altered.

**Get the Tool ID**

| | |
|---|---|
| String getId(); | Get the well-known ID of this tool. |

**Get the Tool title**

| | |
|---|---|
| String getTitle(); | Get the title of this tool. |

**Get the Tool Description**

| | |
|---|---|
| String getDescription(); | Get the description of this tool. |

**Get the configuration properties for this Tool**

| | |
|---|---|
| Properties getRegisteredConfig(); | Get the configuration properties defined in the tool registration file. Access to these properties is read only. There is no limit set on the number of properties that can be defined. |

**Get the keywords for this Tool**

| | |
|---|---|
| Set getKeywords(); | Get the set of keywords defined for this tool. Keywords may not be altered at runtime. |

**Get the Categories for This Tool**

| | |
|---|---|
| Set getCategories(); | Get the set of categories defined for this tool. Categories may not be altered at runtime. |

## 2.2.1 Tool Attributes

The following static constants are associated with the Tool interface to define standard request attributes.

| Constant | Keyword | Meaning |
|---|---|---|
| FRAGMENT | sakai.fragment | The request attribute name whose value if "true" requests producing a document fragment rather than a full document. |
| PORTLET | sakai.portlet | The request attribute name whose value if "true" requests producing a document suitable for aggregation in a portal. |
| TOOL | sakai.tool | The request attribute name containing the Tool definition for the current request. |
| TOOL_SESSION | sakai.tool.session | The request attribute name containing the ToolSession for the current request. |
| NATIVE_URL | sakai.request.native.url | The request attribute name if present causes our wrapped requests to report the native URL rather than the Sakai setup URL information. |
| PLACEMENT | sakai.tool.placement | The request attribute name containing the Tool placement for the current request. |
| PLACEMENT_ID | sakai.tool.placement.id | The request attribute / URL parameter name containing the Tool placement ID for the current request. |

| | | |
|---|---|---|
| HELPER_DONE_URL | sakai.tool.helper.done.url | Standard session attribute shared between client and helper: URL to redirect to when helper is done. |
| HELPER_MESSAGE | sakai.tool.helper.message | Standard session attribute shared between client and helper: end user message. |

## 2.3 The Placement API

Tool Placement models how we place and manage tools within Sakai. Placement provides the context for a tool in a worksite (portal location). Tool placement objects can be modified and saved back out to a database.

Tool placements may be specified as parameters than can be loaded, or they can be determined by a portal and its URL conventions.

**Get the configuration properties (includes registration properties)**

| | |
|---|---|
| Properties getConfig(); | Get the configuration properties, combined from placement and registration, for the tool placement. Placement values override registration. Access is read only. |

**Get the placement context**

| | |
|---|---|
| String getContext(); | Get the context associated with this tool placement. [This is probably a context ID string, but needs to be verified.] |

**Get the place ID string**

| | |
|---|---|
| String getId(); | Get the tool place ID string. |

**Get the placement configuration properties (excludes registration properties)**

| | |
|---|---|
| Properties getPlacementConfig(); | Get the configuration properties for this tool placement – not including those from the tool registration. |

**Get the placement Title**

| | |
|---|---|
| String getTitle(); | Get the tool placement title. |

**Get the Tool placed**

| | |
|---|---|
| Tool getTool(); | Get the tool placed by this placement object. |

**Set the placement Title**

| | |
|---|---|
| void setTitle(String title); | Set the title for this tool placement. Non-null values override the tool registration title. |

#### Set the Tool to be Placed

| | |
|---|---|
| void setTool(Tool tool); | Set the tool for this tool placement. |

#### Save Changes to This Placement

| | |
|---|---|
| void save(); | Save any changes made to this placement. |

## 2.4 The Active Tool Manager API

The Tool Manager API is extended to include support for Servlet API-specific activity.

#### Add a Tool to the Registry

| | |
|---|---|
| void register(Tool tool, ServletContext config); | Add this tool to the registry. |

#### Add the Tools in this XML document to the Registry

| | |
|---|---|
| void register(Document toolXml, ServletContext config); | Add the tools in this XML document to the registry, using the Sakai Tool XML schema. |

#### Add the Tools in this XML file to the Registry

| | |
|---|---|
| void register(File toolXmlFile, ServletContext config); | Add the tools in this XML file to the registry, using the Sakai Tool XML schema. |

#### Add the Tools in this XML input stream to the Registry

| | |
|---|---|
| void register(InputStream toolXmlStream, ServletContext config); | Add the tools in this input stream to the registry using the Sakai Tool XML schema. |

#### Get the active Tool with the given well-known ID string

| | |
|---|---|
| ActiveTool getActiveTool(String id); | Get the active tool with the given well-known ID string in the tool registry. |

## 2.5 The Active Tool API

Active Tool is an extension to the Tool API designed to introduce Servlet API-specific tool activity.

#### Invoke a Tool to handle a complete request

| | |
|---|---|
| void forward(HttpServletRequest req, HttpServletResponse res, Placement placement, String toolContext, String toolPath); | Invoke the tool to handle the complete request given placement, context, and tool path. |

#### Invoke a Tool to produce a fragment

| | |
|---|---|
| void include(HttpServletRequest req, HttpServletResponse res, Placement placement, String toolContext, String toolPath); | Invoke the tool to handle the request by producing a fragment given placement, context, and tool path. |

**Invoke a Tool as a helper**

| | |
|---|---|
| void help(HttpServletRequest req, HttpServletResponse res, String toolContext, String toolPath); | Invoke the tool to handle the complete request as a helper. Note: the placement is shared between invoker and invoked. |

## 2.6 The Tool URL Manager

The ToolURLManager interface allows creation of ToolURLs that reference the portlet itself.

Sakai tools assume the Servlet APIs as the basis for generating markup and getting parameters, yet for presentation inside different portals, the URL encoding API javax.servlet.http.HttpServletResponse#encodeURL is not sufficient. This is because the Servlet API treats all URLs the same (hence a single encodeURL method), whereas portlet technologies such as JSR-168 and WSRP differentiate between URLs based on what they represent.

There are primarily three different URL types as distinguished by WSRP (JSR 168 has 2, which is a subset of the three types distinguished by WSRP). The only reasonable way to allow tools to generate markup that can be presented in a portlet is to have the tools differentiate the URLs that are embedded in the markup. Some of this can be done automatically if the URLs are generated using macros or other APIs that allow for this differentiation to be plugged in underneath.

For instance, most velocity-based tools used different macros for different URL types, so it is possible to plug the appropriate URL encoding underneath the macros when the tool is being rendered as a portlet. However, tools that directly access Servlet APIs to generate markup must use these APIs directly.

Using these APIs is simple. Instead of creating a String URL with the parameters, you create a ToolURL object. You must specify the URL type (render, action, or resource) to create a ToolURL object. You can then set the request path and request parameters by using methods in ToolURL. Finally, to include it in the generated markup, you convert the ToolURL to a String using the ToolURL#toString method.

**Create a URL that is a link back to this Tool**

| | |
|---|---|
| ToolURL createRenderURL(); | Create a URL that is a hyperlink back to this tool. HTTP GET requests initiated by simple &lt;a href&gt; constructs fall in this category. |

**Create a URL that is an action performed on this Tool**

| | |
|---|---|
| ToolURL createActionURL(); | Create a URL that is an action performed on this tool. HTML Form actions that initiate an HTTP POST back to the tool fall in this category. |

**Create a URL that is a resource for this Tool**

| | |
|---|---|
| ToolURL createResourceURL(); | Create a URL for a resource related to the tool, but not necessarily pointing back to the tool. Image files, CSS files, JS files etc. are examples of resource URLs. Paths for resource URLs may have to be relative to the server, as opposed to being relative to the tool. |

## 2.7 The Tool URL API

A ToolURL is used to create a URL and encode it appropriately to the context and placement of the tool.

**Set the path for this URL**

| | |
|---|---|
| void setPath(String path); | Set the path for this URL. The path can be either absolute with respect to the server or relative to the servlet context in which this tool is placed. |

**Set a URL parameter**

| | |
|---|---|
| void setParameter(String name, String value); | Sets the given String parameter associated with this URL. This method replaces all parameters with the given key. An implementation of this interface may prefix the attribute names internally in order to preserve a unique namespace for the tool. |

**Set an array of URL parameter values**

| | |
|---|---|
| void setParameter(String name, String[] values); | Sets the given String array parameter to this URL. This method replaces all parameters with the given key. An implementation of this interface may prefix the attribute names internally in order to preserve a unique namespace for the tool. |

**Set URL parameters given a map**

| | |
|---|---|
| void setParameters(Map parameters); | Sets a parameter map for this URL. All previously set parameters are cleared. An implementation of this interface may prefix the attribute names internally in order to preserve a unique namespace for the tool. The Map contains parameters as name/value pairs. The names in the parameter map must be of type String. The values in the parameter map must be of type String array (String[]). |

**Return a URL string representation**

| | |
|---|---|
| String toString(); | Returns the URL string representation to be embedded in the markup. Note that the returned String may not be a valid URL, as it may be rewritten by the portal/portlet-container before returning the markup to the client. [Does this mean validation is necessary?] |

## 2.7.1 URL Tool Request Parameters

Additional request parameters for URL Tools.

| Constant | Keyword | Meaning |
|---|---|---|
| MANAGER | tool.url.manager | Property name to retrieve an instance of ToolURLManager from an HttpServletRequest. |
| HTTP_SERVLET _REQUEST | org.sakaiproject.util .RequestFilter.http_ request | Property name to set the HttpServletRequest in a given thread context, to default to when it is not available to the caller. This allows calling ToolURLManager's create&lt;Type&gt;URL with a null HttpRequestServlet, if one has been set in the current thread context.<br><br>We use the same attribute name as set in org.sakaiproject.util.RequestFilter to prevent having to depend on RequestFilter class only to get this attribute. |

## 3 Default Sakai Implementation

The Tool, Active Tool, and Tool URL Managers are implemented as components in kernel/tool-component and kernel/active-tool-component. Tool and Placement are implemented as utilities objects in kernel/tool.

## 4 Utility Objects

Tool and Placement implementations are coded as utility objects.

The Tool Listener is implemented as a ServletContextListener in the kerne/tool-registration module-part.

# 5 Recommended Practices

## 5.1 Accessing the Tool Manager

The Sakai Tool Manager can be accessed via its cover or through the Component Manager.

### 5.1.1 Access Using  the Tool Manager Cover

Use this code fragment to get to the Tool Manager using its cover:

```
ToolManager tm =
    org.sakaiproject.api.kernel.tool.cover.ToolManager.getInstance();
Tool current = tm.getCurrentTool();
```

The main trick to using the cover is to import the ToolManager, and use its cover to get an instance of it. Note the difference in package name for the cover.

### 5.1.2 Access Using the Component Manager

Alternatively, the Tool Manager can be accessed using the Component Manager:

```
ComponentManager cm =
    Org.sakaiproject.api.kernel.component.cover.ComponentManager.getInstance();
ToolManager tm =
    (ToolManager) cm.get("org.sakaiproject.api.kernel.tool.ToolManager");
```

The two methods are very similar. In fact, the Tool Manager cover is implemented by accessing the Component Manager cover.

## 5.2 Tool Registration

Tools are registered with the Tool Manger by creating a tool description document that follows the Sakai Tool XML schema with a name corresponding to its ID. In this case the file would be called "sakai.sample.tools.dash-tool-mgr.xml".

```
<registration>
        <tool
                id="sakai.sample.tools.dash-tool-mgr"
                title="Tool Manager Dashboard"
                description="An application to test and show registered tool.">

                <configuration name="integration.property.size" value="[size here]" />
                <category name="sakai.sample" />
                <keyword name="dashboard" />

        </tool>
</registration>
```

The well-known ID of this tool is provided in the ID parameter, followed by a title and description. Configuration properties can be included as shown. Finally, a category and keywords can be provided to group this tool with others like it.

In addition to this document, the ToolListener must be included in the web.xml file:

```
<listener>
   <listener-class>org.sakaiproject.util.ToolListener</listener-class>
</listener>
```

## 5.3 Finding the Current Tool

See Accessing the Tool Manager above. Once you have a tool manager reference, you can use it to find the current tool:

```
ToolManager tm =
    org.sakaiproject.api.kernel.tool.cover.ToolManager.getInstance();
Tool current = tm.getCurrentTool();
```

## 5.4 Working With Tool Placements

Tools are often associated with a work site. Since each tool can be used by many such sites, it is necessary to indicate the existence of a tool in a site. This is handled by Tool Placement. Each placement acts as if it were a different tool and maintains configuration information and sessions for each active user.

Tool placement can be defined in the web.xml file as a configuration parameter:

```
<filter>
   <filter-name>sakai.request</filter-name>
   <filter-class>org.sakaiproject.util.RequestFilter</filter-class>
   <init-param>
      <param-name>tool.placement</param-name>
      <param-value>tooldash</param-value>
   </init-param>
</filter>
```

## 5.5 Access to Resources

Resources can be included in a tool WAR using xml:

```
<build>
         <sourceDirectory>src/java</sourceDirectory>
         <resources>
                  <resource>
                           <directory>${basedir}/src/bundle</directory>
                           <includes>
```

```
                    <include>**/*.properties</include>
               </includes>
          </resource>
     </resources>
</build>
```

In this case, a set of .properties file is included from the "../src/bundle" directory. Once the resource bundle is included in a component it can be accessed using:

getClass().getClassLoader().getResourceAsStream("name.properties");
[This needs to be verified.]

## 5.6 User Interfaces

As a design rule, Sakai encourages the separation of user interface declaration from the application code that manages it. This allows certain kinds of adjustments to the UI (skinning, etc.) to happen without recompiling the code. The following user interface approaches are currently supported:

- JavaServer Faces – Sun reference implementation
- JavaServer Faces – MyFaces implementation
- Servlet – HTML response

### 5.6.1 JavaServer Faces

To use JavaServer Faces to present your user interface to a user, you need to set up the FacesServlet to handle incoming requests. The following web.xml file has commentary highlighted in blue:
[Verify this against Glenn's document.]

```
<servlet>  The faces servlets is declared.
     <servlet-name>Faces Servlet</servlet-name>
     <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
     <load-on-startup> 2 </load-on-startup>
</servlet>

<!-- Faces Servlet Mapping -->
This mapping identifies a jsp page as having JSF content. If a
 request comes to the server for foo.jsf, the container will
 send the request to the FacesServlet, which will expect a
 corresponding foo.jsp page to exist containing the content.

<servlet-mapping>  Pages ending in .faces.
     <servlet-name>Faces Servlet</servlet-name>
     <url-pattern>*.faces</url-pattern>
</servlet-mapping>

<servlet-mapping>  Pages ending in .jsf.
     <servlet-name>Faces Servlet</servlet-name>
```

```
        <url-pattern>*.jsf</url-pattern>
</servlet-mapping>

<welcome-file-list>
        This defines two default index files.
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsf</welcome-file>
</welcome-file-list>
```

In addition to the web.xml file, the faces-config.xml file is used to define faces pages and how they transition from one to another.

```
<faces-config>
        <managed-bean>
                <description>
                        Example backing bean for multiple example pages
                </description>
                <managed-bean-name>examplebean</managed-bean-name>
                <managed-bean-class>example.ExampleBean</managed-bean-class>
                <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>

        <navigation-rule>
                <navigation-case>
                        <from-outcome>index</from-outcome>
                        <to-view-id>/index.jsp</to-view-id>
                        <redirect />
                </navigation-case>
                <navigation-case>
                        <from-outcome>alphaIndex</from-outcome>
                        <to-view-id>/alphaIndex.jsp</to-view-id>
                        <redirect />
                </navigation-case>
                <navigation-case>
                        <from-outcome>anchorReference</from-outcome>
                        <to-view-id>/anchorReference.jsp</to-view-id>
                        <redirect />
                </navigation-case>

                ...
        </navigation-rule>
</faces-config>
```

An ExampleBean is set up as a managed bean to handle its events. A few navigation rules are defined, though these are optional if you follow the return string as page name convention.

JavaServer Faces (and MyFaces) is a powerful web user interface system. Several good books are available on JSF which will provide considerably more detail on how to write JSF applications.

### 5.6.2 MyFaces

### 5.6.3 Servlet (HTML Response)

# 6 References

[1.1] Sakai Kernel Bundle – Overview, Mark J. Norton, Sept. 2005 [URL Here]

[1.2] Sakai Kernel Bundle Requirements, Mark J. Norton, Sept. 2005 [URL Here]

[1.3] SKB Manual – Components, v1, Oct. 9, 2005

[1.4] SKB Manual – Tools, [this document]

[1.5] SKB Manual – Sessions, [to be written]

[1.6] SKB Manual – Requests, [to be written]

[1.7] SKB Manual – Configuration, [to be written]

[1.8] SKB Manual - Context and Thread Safety, [to be written]

[1.9] SKB Manual – Logging and Debugging, [to be written]

[2.1] Sakai's Component Manager API, Component Packaging, and the Underlying Spring Implementation, Glenn R. Golden, March 23, 2005 [URL here]

[2.2] Sakai Tools, Glenn R. Golden, July 12, 2005 [URL here]

[2.3] Sakai Request Processing, Glenn R. Golden, July 12, 2005 [URL here]

[2.4] How to Configure Sakai, Glenn R. Golden, July 19, 2005 [URL here]

[2.5] Sakai Sessions, Glenn R. Golden, March 23, 2005 [URL here]

Update all citation numbers to reflect these changes.

# 7 Document History

This document was created by Mark Norton. It represents research into the Component Manager (and related) software developed by Glenn Golden of the University of Michigan and includes many of the concepts initially described by him in [1].

| Date | Version | Who | Work |
|---|---|---|---|
| Oct. 10, 2005 | 1 | Mark Norton | Initial version of the document. Overview and introduction. Object and API definitions. Implementation description. Utility objects described. System configuration properties. Initial recommended practices. |
| Nov. 2, 2005 | 2 | Mark Norton | Edited by ABC. Terminology changes. |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |