Octane Devel Notes

uCompass Octane Development Notes

First Contract includes:

- Setup of Sakai 2.8.0
- Turn on BLTI
- Configure BLTI to access remote Octate content
- Modify Resource tool to inject Octane tags
- Demonstrate at Sakai Conference at BOF

Checked out sources from https://source.sakaiproject.org/svn/sakai/tags/sakai-2.8.0/

Updated JAVA_OPTS to:

```
        -Xms512m -Xmx1024m -XX:PermSize=64m -XX:MaxPermSize=128m -XX:NewSize=192m -Djava.awt.headless=true -
Dhttp.agent=Sakai -Dorg.apache.compiler.Parser.STRICT_QUOTE_ESCAPING=false -Dsun.lang.ClassLoader.
allowArraySyntax=true
```

Some of these options are new to 2.8.0, I believe.

Subversion 1.6+ is recommended. I'm at 1.4.2, currently. Perhaps it is time to update.
It's a bit weird, but svn under Cygwin comes up as 1.4.2, while in the DOS shell it comes up as 1.6.17. Whatever.
Hmm. There is a SUBVERSION_HOME env variable now.

I am up to date with Maven at 2.2.1.
Checked the Maven settings.xml file. All seems to be up to date.

The install directions are recommending Tomcat 5.5.33. I am at 5.5.30. While there have been a couple of security issues fixed, I think that I'm close enough so that I don't need to worry.

Copied a fresh version of Tomcat-5.5.30 to /dev. Changed the port to 8085 for this version of Sakai.
Verified that Tomcat will come up on http://localhost:8085/

I just had a look at the downloaded Sakai code. It is substantially smaller. Samigo seems to be missing, for example. Perhaps less used packages have been stripped out. I don't see BLTI. I might have to add that separately. We'll see once Sakai is up.

I was a bit uneasy about using the old sakai.properties file, so I grapped a copy of the default for 2.8.0.

- Changed content.html.forcedownload to be false.
- Copied the database settings from 2.7.1
- Changed the sakai database name to skai281
- Changed database username and password back to sakai and ironchef. New is sa/(none).
- Set imsblti.producer.enabled=true
- Enabled all BLTI tools
- Set secret to "top-secret-key"
- Allow all BLTI web services
- Set content.upload.max=100
- Set bodyPath@org.sakaiproject.content.api.ContentHostingService = /sakaiContent/
- Set bodyVolumes@org.sakaiproject.content.api.ContentHostingService =vol1,vol2,vol3
- Set webservices.allowlogin=true

That should do it for sakai.properties.
Next, clear out the maven repository by renaming the old one to repositor-2.7.1.

Time to build.
It seems the build process has changed a bit. You have to start in sakai/master and run "mvn clean install" to get the basic stuff first.
The master build succeeded.
However, the top-level build is failing to find a /rights folder. I will download it from the subversion repos.
Downloaded code. Changed maven verison to 2.8.0 in four pom files.

It may be that my initial checkout failed. Let's try subversion again.
There are more packages now. Hopefully it will build.

About five hours to compile. I had to re-start it several times - likely due to memory limitations. Successful build.

Jun. 7, 2011

Created a MySQL databased for the new sakai install called sakai280.
Started up sakai. Took quite a while to start up - many minutes - created all databases, so first time is expected to be slow.
Created a test course site - SMPL101.
BLTI tool is present and labeled as "Octane Content".

Created a static test page at: file:///C:/Documents%20and%20Settings/Mark/My%20Documents/_work/uCompass/steam-engine.htm

Talked to Tom Petz on the phone.
My domain needs to be registered. He is registering "localhost" for me.
He will also add some Octate tools to demonstrate.

Jun. 8, 2011

In correspondence with Mary Miles, I have asked her to get me a slot at the demo night. There is a bit of conflict there, since I'm also representing SoftChalk - though not at the demo night.

Tom is going to work with me today to get static hosted content working.
I created a page with an Octate tag in it and uploaded it to my web site at:

http://wwww.nolaria.com/os_car.htm

I configured a BLTI tool instance to link to this document remotely from my laptop. It's working just fine.

So at this point, we are close to having two out of the three goals for the Sakai Confernece next week. Next, I need to create a way to injected the Octane tag into an HTML document as it is displayed.

There are a couple of ways I can make this work:

1. Modify the existing Resource tool - a true hack.
2. Create a standalone Sakai tool that uses CHS to show HTLM files injected with Octane.
3. Create a CHS Content Handler - the right way to do it, but documentation is obscure.

These are in increasing order of appropriateness. Unfortunately, the may also be in increasing order of difficulty, too.

Subsequent research indicates that the content handler is for remote mounting of repositories. Ian Boston sent me a link to some examples from Cambridge. There is also some documentation in Sakai Confluence > Project:Resources > Content Hosting Handler. This is not really what I need, though I do see that SCORM is handled this way. Maybe this is the right way to do it. Regardless, it's going to be a lot of work to figure it out then implement it, and then inject it correctly into the Resource tool.

Sadly, I think I'm gonna go for option 1: hacking the Resource Tool.

There is also a way to register custom actions. In theory, I could add an "Open with Octane" action.
In ResourcesAction, there is a line of code, "pipe.getRevisedContentStream();" This suggests that it is possible to create an action that revises content on the fly - as it were. That is pretty close to what we want.

I didn't see any action in ResourcesAction to display or launch a Resource.

I found a file called ResourcesItem. It contains content and a resource type. It's possible that I could intercept the request to getContent() and inject the Octane tag if the type is HTML.

This is worth an experiment or two.
Added a single printout:

```
        /**
         * @return the content
         */
        public byte[] getContent()
        {
                System.out.println ("ResourcesItem.getContent() - mjn - content type is: "+contentType);
                return this.content;
        }
```

Let's recompile it and see what happens when we try to display a content item.

Darn. It didn't get called. This is one thing I don't like about the Resource tool - six ways to do everything.

According to the tool web.xml file, ResourcesAction is the main handler for this tool. Naturally, it's also one of the largest files in the whole collection, too.

Search on these:

```
        public static final String TYPE_HTML = MIME_TYPE_DOCUMENT_HTML;
        public static final String TYPE_TEXT = MIME_TYPE_DOCUMENT_PLAINTEXT;
        CONTENT_READ_ACTIONS.add(ActionType.VIEW_CONTENT);
```

Searches on MIME_TYPE_DOCUMENT_HTML:

```
        Line 1178:  resourceProperties.addProperty(ResourceProperties.PROP_CONTENT_ENCODING, UTF_8_ENCODING);
```

None of the above searches yeilded much. Strangely, I don't see any UI code in this file - or anywhere for that matter. There are some ".vm" files, which I think are Velocity files, but they don't have much in them.

```
        2119:  getContent()
        7743:  getContentAdvisor() ???
```

Added a prinf at line 2120: System.out.println ("ResourcesAction.getEditItem() - mjn - item type is: "+itemType);
This is in getEditItem(), which doesn't seem likely for a view content event, but worth a try since I'm not finding anything else.
No help. Doesn't come up on view, edit, or edit details.

I downloaded the 2.2.1 kernel and rendered the JavaDoc for it. I also did the JavaDoc for the content package. These might give me some insight into writing a content handler or extension plugin.

ResourcesAction extends VmServlet, which is a Velocity based servlet. It is possible that there are actions hidden up in the parent classes that handle things like viewing/launching/etc. There is a velocity package.

ResourceToolAction in the CHS in the kernel seems to be a way to define custom actions on specific media types.

Jun. 9, 2011

UCompass made some changes on their end and now the static content works in Sakai.

I downloaded the SCORM tool from contrib. It is an example of how the extenion mechanisms in the Resources tool can be used.

Jun. 10, 2011

Edward indicated that bochures are not available. Printed up 20 copies of the Octate web page as handouts.

Jun. 30, 2011

Signed the contract with UCompass yesterday and got it into today's mail. The Sakai development portion reads:

The Sakai course management system is a large, complex framework and application set that runs in a Java/Tomcat container. It is in use by hundreds of universities world-wide and by over a million students.

Sakai tools (plugins, modules, etc.) work with a collection of objects provided by the Sakai Kernel Services. Of specific interest to uCompass is the Resources tool, which provides content management for Sakai instances. The Resources tool uses the Content Hosting Services (CHS) which are part of the Sakai kernel. The hosting services can be extended by a plugin mechanism to provide additional functionality. I propose to develop a new content type handler for documents of type "text/html". This handler will intercept requests for Sakai HTML files and inject the Octane Fuel Cell into its header element, thus enabling Octane tools to be accessed when viewed in a browser.

Although conceptually simple, this particular plugin is technically intricate and not well documented. Development will require some research and experimentation. Fortunately, I am on good terms with the people who developed and maintain this part of the Sakai system.

Delivery will consist of the plugin code and documentation on how to install it. I recommend that uCompass post this to the Sakai contrib branch of its Subversion source management system so that anyone can use it. Access to Octane will be controlled via a license key to be kept in the Sakai system configuration properties file.

Some preliminary word was done prior to the Sakai Conference in LA to demonstrate Octane-enabled content in the Sakai environment. This was done using two methods:

- BLTI wrapped remote content
- Sakai resident content with Octane added manually

Work was done to explore injecting the Octane fuel cell into content dynamically, but this proved to be more complex that could be accomplished in the limited amount of time available prior to the conference. That work resumes today and is the primary topic of these development notes.

Research

The first phase of the development effort is to figure out how create an inject a media handler into the Resource tool that will call a handler class when a document of type "text/html" is requested using the Resource Tool. The Resource tool is Sakai's user application to manage documents and make them available on a per-course-site basis. It's a fairly complicated tool with many features. Fortunately, it can be extended via media handlers (etc), which enables features like Octane to be added to Sakai.

Work done prior to the Sakai conference indicated that support for media handlers is actually part of the support services used by the Resources tool. These services are collectively referred to as the Content Hosting Services (CHS), resident in the Sakai kernel component as of Sakai 2.6.x. Fortunately, the actual classes and methods are moderately well documented using JavaDoc and I have compiled the Kernel-1.2.2 documentation for use on this project. Additionally, there are some Sakai wiki pages that discuss media handlers. Ian Boston (Ian is a Sakai architect) and I exchanged some email on this topic as follows:

Mark Norton

Hopefully I caught you before leaving for the US. I'm trying to write a ContentHostingHandler for a new project I'm working on. Do you have any quick links to how the handler is registered and perhaps an existing example?

Ian Boston

Here is a DSpace one based on LNI, a file one and a JCR one

https://saffron.caret.cam.ac.uk/svn/projects/Content/trunk/contenthostinghandlers/content-chh/impl/src/java/org/sakaiproject/content/chh/

IIRC registration is via Spring, but it might be via injection in init. the clue should be in the above code.

The CHS has an interface called SiteContentAdvisor and SiteContentAdvisorProvider, the latter having a single method, getContentAdvisor(Site site). SiteContentProviders are registered via SiteContentAdvisorTypeRegistry, which has a single method called registerSiteContentAdvisorProvidor(SiteContentAdvisorProvider advisor, String type). This doesn't seem like the right way to extend the Resources tool to do what I want, event though the advisors are registered by content type. Rather, these seem to be focused on retraction time (the point in time when content is made invisible to students in a course site).

However, there is a different interface called ContentHostingHandler which does seem more promising. It has methods like getResourceBody() that returns the bytes in a content item and streamResourceBody() that returns the content data as an InputStream. This is a much more promising avenue of exploration. There is a method called ContentHostingHandlerResolver as well. Both of these were written by Ian Boston.

In addition to the above, there is also ResourceToolAction, which might be a bit more focused than ContentHostingHandler. An excerpt from the documentation:

> ResourceToolAction defines the way in which actions are described in a resource-type registration. Each action should have its own ResourceToolAction defined in the registration. If the action requires user interaction by a helper, the resource-type registration should include an action definition that implements the InteractionAction interface. If an action requires action by another webapp but does not involve user interaction (i.e. it does not delegate the user interaction to a helper), the resource-type registration should include an action definition that implements the ServiceLevelAction interface. Most actions for new resources types are likely to be similar to familiar actions on resources (e.g. "create", "revise", "delete", etc) and permissions for these actions are handled by the Content Hosting Service.
>
> If an action requires custom permissions, the definition of that action implements the CustomToolAction interface to provide a way for the Resources tool to determine whether to show the action as an option in a particular context to a particular user. A ResourceToolAction deinition should implement at least one of those subinterfaces, and it may implement all three. If a ResourceToolAction implements both InteractionAction and ServiceLevelAction, the activity of the helper defined by InteractionAction will occur before the service-level activity defined by ServiceLevelAction.

Another promising lead.

One angle of attack on this problem would be to look at contributed applications that use these kinds of plugins and extensions. One that I know of is one of the contributed SCORM plugins. There is a syllabus tool that also uses this approach, I believe. Others likely exist.

The Resource tool had higher level documentation in Resource Tool Project. Interesting sub-pages include:

- Content Hosting Handler - adding a handler
- Citations Helper - uses extension mechanisms

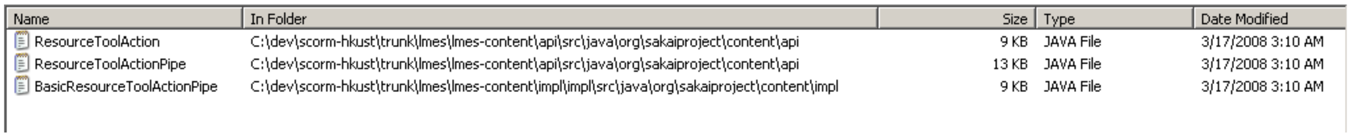
24 sub-pages, most of which are no help.

Other interesting pages:

- Content Hosting - written by me, no less.
- Content Hosing Service - largely about migration to JCR, which was implemented but never quite went all the way. Got derailed by Sakai-3 (OAE).

I was poking around a bit. I seemed to recall an implementation of a SCORM player that used ResourceToolAction. It might be the Icodeon player, which I downloaded from https://source.sakaiproject.org/contrib/rsmart/icodeon/.
Hmm. A search on ResourceToolAction in the Icodeon files turns up nothing. That's a bit of a surpris.

I downloaded the HK-UST SCORM player and ran the same search. Better results this time:



Name	In Folder	Size	Type	Date Modified
ResourceToolAction	C:\dev\scorm-hkust\trunk\lmes\lmes-content\api\src\java\org\sakaiproject\content\api	9 KB	JAVA File	3/17/2008 3:10 AM
ResourceToolActionPipe	C:\dev\scorm-hkust\trunk\lmes\lmes-content\api\src\java\org\sakaiproject\content\api	13 KB	JAVA File	3/17/2008 3:10 AM
BasicResourceToolActionPipe	C:\dev\scorm-hkust\trunk\lmes\lmes-content\impl\impl\src\java\org\sakaiproject\content\impl	9 KB	JAVA File	3/17/2008 3:10 AM

Strangely, the source code for these interfaces was included in the tool. That's a bit of a no-no, since it forks the code. Still, here is a comment from ResourceActionTool.java:

```
/**
 * ResourceToolAction defines the way in which actions are described in a resource-type registration.
 * Each action should have its own ResourceToolAction defined in the registration.
 *
 * If the action requires user interaction by a helper, the resource-type registration should include
 * an action definition that implements the InteractionAction interface.
 *
 * If an action requires action by another webapp but does not involve user interaction (i.e. it does
 * not delegate the user interaction to a helper), the resource-type registration should include an
 * action definition that implements the ServiceLevelAction interface.
 *
 * Most actions for new resources types are likely to be similar to familiar actions on resources (e.g.
 * "create", "revise", "delete", etc) and permissions for these actions are handled by the Content Hosting
 * Service.  If an action requires custom permissions, the definition of that action implements the
 * CustomToolAction interface to provide a way for the Resources tool to determine whether to show
 * the action as an option in a particular context to a particular user.
 *
 * A ResourceToolAction deinition should implement at least one of those subinterfaces, and it may
 * implement all three.  If a ResourceToolAction implements both InteractionAction and ServiceLevelAction,
 * the activity of the helper defined by InteractionAction will occur before the service-level activity
 * defined by ServiceLevelAction.
 *
 * @see org.sakaiproject.content.api.ResourceType
 */
```

This from ResourceTooActionPipe:

```
/**
 * ResourceToolActionPipe provides a conduit through which ResourcesAction and an
 * unknown helper may communicate about the execution of ResourceToolActions in which
 * the registered action specifies that some part of the action is handled by a helper.
 *
 * ResourceToolActionPipe has a set of methods through which ResourcesAction can pass
 * information to a helper about the current state of the entity (or entities?) involved
 * in an action ("setContent", "getContent", "setMimeType", "getMimeType", etc).  It has
 * another set of methods through which the helper passes back values that may or may
 * not have been updated as a result of the action ("setRevisedContent", "getRevisedContent",
 * "setRevisedMimeType", "getRevisedMimeType", etc). If a value is not changed by the action,
 * the helper should use an appropriate setter to indicate the revised value is the same
 * as the original value.  For example:
 *
 *   pipe.setRevisedMimeType( pipe.getMimeType() );
 *
 * Otherwise, the getter for the revised value will return null, and ResourcesAction will
 * set unset the property. ResourceToolActionPipe also has a few methods through which the
 * helper can report whether the action was canceled or an error was encountered.
 *
 * @see org.sakaiproject.content.api.ResourceTypeRegistry
 * @see org.sakaiproject.content.api.ResourceToolAction
 * @see org.sakaiproject.content.api.InteractionAction
 */
```

BasicResourceToolPipe looks like a very simple implementation of the ResourceToolPipe interface. However, I don't see it referencing any Scorm code, which is also strange.

Oy. This code is so old it's using Maven 1.x.

So a ResourceActionPipe is probably fairly easy to implement (though there may be hidden land mines). The real question is how to register the action with the Resource tool?

Message from Anthony Whyte:

I've not checked the code closely but below is a list core projects that use ResourceToolAction.

Citations tool: has a couple of helpers that leverage ResourceToolAction
OSP common: uses ResourceToolAction to filter allowed actions

On Jun 30, 2011, at 1:19 PM, Mark Norton wrote:

> Do either of you know of a Sakai or Contrib project that uses ResourceToolAction to extend the functionality of the Resource tool? I need to write one of these and it would help greatly to have an example to look at.
>
> Thanks,
> Mark Norton

Citations Tool is at least up to date w.r.t Sakai. It is a core tool at least. Let's have a look at that.
My search didn't turn up anything.

The Resources tool should have some references. See ../content/content-types/src/java/org/sakaiproject/content/types/ContentTypeRegistryBean.java, and others. I note that there is already an HTML document type. That could complicate things. OTOH, it might as simple as replacing a file or two in the Resource tool.

I went looking for a class that was associated with HtmlDocumentType. I didn't turn up anything on search. However, there is an inner class called HtmlDcoumentReplaceAction in HtmlDocumentType. I hate inner classes - for just this reason. Things are not where you expect them to be, and (as usual) this is PUBLIC inner class, which means it usable from anywhere.

The more I look at this code, the more it looks like the place to tinker with injecting the Octane fuel cell. The method HtmlDocumentType() sets up the following action handlers:

```
        actions.put(ResourceToolAction.CREATE, new HtmlDocumentCreateAction());
        //actions.put(ResourceToolAction.ACCESS_CONTENT, new HtmlDocumentAccessAction());
        actions.put(ResourceToolAction.REVISE_CONTENT, new HtmlDocumentReviseAction());
        actions.put(ResourceToolAction.REPLACE_CONTENT, new HtmlDocumentReplaceAction());
        actions.put(ResourceToolAction.ACCESS_PROPERTIES, new HtmlDocumentViewPropertiesAction());
        actions.put(ResourceToolAction.REVISE_METADATA, new HtmlDocumentPropertiesAction());
        actions.put(ResourceToolAction.DUPLICATE, new HtmlDocumentDuplicateAction());
        actions.put(ResourceToolAction.COPY, new HtmlDocumentCopyAction());
        actions.put(ResourceToolAction.MOVE, new HtmlDocumentMoveAction());
        actions.put(ResourceToolAction.DELETE, new HtmlDocumentDeleteAction());
```

Note that ResourceToolAction.ACCESS_CONTENT is commented out. That could mean one of several things:

1. this is being handle somewhere else.
2. there is no need to special case the access.
3. using this handler caused problems.

Jul. 21, 2011

11:00a - 12:00n

When last we looked at the Resource tool, we determined ContentTypeRegistryBean.java was a good place to look.There is an HTML document type, which might complicate things or could be used as a hook. Looked for HtmlDocumentType. There is an inner class called HtmlDcoumentReplaceAction in HtmlDocumentType. It seems like a likely place to inject the Octate fuel cell.

The way to test this to put some print statements into the code and see what happens when we try to access an HTML document managed by the Resource tool.

Better than HtmlDcoumentReplaceAction might be the HtmlDocumentAccessAction, if it is in fact registered to handle this kind of action. There is no guarantee. It implements org.sakaiproject.content.api.InteractionAction, which is probalby in the kernel bundle. Sigh.\

The kernel javadoc says;

An InteractionAction defines a kind of ResourceToolAction which involves user interaction to complete the action. The Resources tool will invoke a helper to render an html page (or possibly a series of pages), process the response(s) and turn control back to the Resources tool when done. Before invoking the helper, ResourcesAction will call initializeAction() supplying a Reference onject as a parameter. Implementations of this interface may do whatever is necessary to prepare for invocation of the helper and they may return an identifier for that initialization. After starting the helper and getting back control from the helper, ResourcesAction will call either finalizeAction or cancelAction to indicate that the user either finalized the action or canceled it. The registrant may do whatever is necessary to commit any changes in persistant storage (other than changes to the referenced resource in ContentHosting) or reverse them.

There is no intercept of getBytes() here. Perhaps this is not the place after all.

Added a display hook to C:\dev\sakai-2.8.0\content\content-tool\tool\src\java\org\sakaiproject\content\tool\ResourcesItem.java,
So let's compile this code and bring up Sakai to experiment a bit. Unfortunately, viewing the content doesn't trigger the hook message.

2:00p - 5:00p

This hit or miss approach to the problem doesn't seem to be working for me. I figure that someone, somewhere HAS to call getContent() to get the bytes associated with a web page for display.

I generated a list of all files in the content tool. Most of these files are not relevant to my search, but being thorough is important at this stage.

Here is something that might be worth following up from the web.xml file:

```
    <servlet>
        <servlet-name>
           sakai.resource.type.helper
        </servlet-name>
        <servlet-class>
            org.sakaiproject.content.tool.ResourcesHelperAction
        </servlet-class>
        <load-on-startup>1</load-on-startup>
     </servlet>
```

After analyzing all of the files in the content tool, I have found the following references to getContent():

- FilePickerAction.java - extends PagedResourceHelperAction, one instance of getContent() ****
- ListItem.java - one instance of getContent() ****
- ResourcesAction.java - extends PagedResourceHelperAction, one instance of getContent() ****
- ResourcesEditItem.java - extends ResourcesBrowseItem, one instance of getContent() ****
- ResourcesItem.java - three instances of getContent() ****

So the next step is to put hook messages in every one of these places and see if they get triggered when content is displayed.
Hooks are now in place.

None of the triggers displayed messages in the console. That's very frustrating. It means that some lower level class - probably in the kernel - is being used to get the content to be displayed.

If I could register a ResourcesToolActionPipe with the content tool, I might be able to intercept the content. One way to do this is to register a new content type such as "text/mjn" and associate it with an extension of ".mjn". In reality, these would be HTML files, but it lets me fool the content tool long enough to experiment with a new media type without worrying about stepping on the handling of existing media types, of which HTML is one.

ResourcesItem and ResourcesAction both make reference to pipes.
These two classes seem to be the high level implementaton of ResourceToolActionPipes, but they are not currently being invoked - as near as I can tell.

So if I could figure out how to register a ResourceToolAction helper, I might be able to intercept display events.

Oct. 13, 2011

Looking back at my notes on the Sakai Confluence site, I note that I haven't done any work on Octane since July. That's longer than I thought.

It occurred to me last night that perhaps I should try to understand the underlying mechanisms of the Resource tool before trying to hack it's user interface. Much of the functionality of the RT has been moved into the kernel. It might be possible to create a small JSP servlet that calls out to the Sakai kernel. If so, it would allow me to play with the functionality there.

The idea is to create a small application that displays a list of files from a well-known directory. If files are of type text/html, allow them to be opened and inject the Octane powercell. Display them in an i-frame.

I've created a folder in c:/dev called octane and added folder in it called octtool, which will be the experimental octane tool.

The kernel is up to version 1.2.6. I have 1.2.2 on my laptop.
I checked out 1.2.6 from the subversion repos.
I am attempting to build it using maven.

Kernel built without error.

Edited files to create an app (now called) oct. It just has a test page. Compliled and deployed. Works using the following URL:

- http://localhost:8084/oct/test.jsp

Next step is to find the kernel JAR and bundle it into app.

Built JavaDoc using "javadoc:javadoc", which is deprecated but seems to still work. The comment mentioned "javadoc:aggregate".

JavaDoc got built to file:///C:/dev-sakai/kernel-1.2.6/target/site/apidocs/index.html

Copied the following JARS into the lib section of oct:

- sakai-kernel-api-1.2.6
- sakai-kernel-impl-1.2.6

Hopefully, this will provide full access to the kernel. We shall see.

The first step is to grab a hold of the ContentHostingService. From there, all things are possible.

Let's see if we can bring this into Eclipse.

Seems like the static cover for ContentHostingService is now deprecated. They recommend that I use the ComponentManager, but that might lead to trouble. We shall see.

Problems right off the bat. The object, "org.sakaiproject.component.cover.ComponentManager" can't be resolved.
Sigh. It's documented in the JavaDoc. It's where I remember it to be, but Eclipse can't find it.

Hmm. When I look in "C:\dev-sakai\kernel-1.2.6\api\src\main\java\org\sakaiproject\component\cover", there are no objects.

This is where it lives: "C:\dev-sakai\kernel-1.2.6\component-manager\src\main\java\org\sakaiproject\component\cover".

So it exists, why can't I resolve the reference?
It just doesn't seem to be included in the compiled JARs.

From the looks of the m2 local maven repository, the component manager is built into a separate JAR. I found a sakai-component-manager-1.2.6.JAR in C:\Documents and Settings\Mark\.m2\repository\org\sakaiproject\kernel\sakai-component-manager\1.2.6. I'll bundle this into the oct tool and see if that helps.

Yes, that looks like it worked. Kinda makes me wonder what else didn't get included.

So, while Eclipse is seeing the ComponentManager, and ant compiles it, I get a runtime error saying it's not there.
It looks like the JARs were not deployed.
I deployed them by hand.

Ah, now we are going to run into the kernel dependencies:

java.lang.ClassNotFoundException: org.springframework.context.ConfigurableApplicationContext

There are a BUNCH of these, as you might expect.
A total of eight spring JARS alone.
Well, that helped. App is coming up now. I'm getting the following in the console:

```
Could not resolve placeholder 'sakai.security' in [file:${sakai.security}security.properties] as system
property: neither system property nor environment variable found
java.lang.Exception: traceback
        at org.sakaiproject.util.NoisierDefaultListableBeanFactory.destroySingle
```

I think this is the result of the various Sakai property files not being found.
Oh, and I'm going to need a database, I suspect.

I think I can create a Java system property called "sakai.security". I assume it points to the directory where the security.properties file lives, but I don't have an example of what that file looks like.

I created a (mostly) empty file at c:/security.properties and created a system property to refer to it.
It no longer complains about the security properties, but still fails. I think that Spring is failing to initialize, likely due to some missing piece or other.

```
java.lang.Exception: traceback
        at org.sakaiproject.util.NoisierDefaultListableBeanFactory.destroySingletons
(NoisierDefaultListableBeanFactory.java:96)
        at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.
java:388)
        at org.sakaiproject.component.impl.SpringCompMgr.init(SpringCompMgr.java:154)
        at org.sakaiproject.component.cover.ComponentManager.getInstance(ComponentManager.java:98)
        at org.sakaiproject.component.cover.ComponentManager.get(ComponentManager.java:111)
        at com.nolaria.oct.LocalContent.test(Unknown Source)
```

Short of debugging the whole initialization logic, I'm not sure it makes much sense to continue this line of inquiry. Better, I think, to build the whole Sakai environment and make a tool that runs within it to analyze the ContentHostingService.

Oct. 24, 2011

A bit over a week a ago, I started experimenting with a simple, JSP-based application that would allow me to explore support for the Resources tool in the Sakai kernel. I started running into kernel dependencies that made further development (in isolation) difficult. Since then, it occurred to me that if I just embed it in a Sakai environment, I'm very likely to get that simple app working. So let's bundle it up and move it over to my laptop and work on it there.

Naturally, things will build and startup more slowly here, but I might be able to take advantage of the hot-swapping support in Tomcat, since I'm not tinkering with the underlying libraries.

Started up tomcat and tested the app as it currently exists using:

http://localhost:8085/oct/test.jsp

Message indicates that the ContentHostingService was obtained.

Notes from June 21 of this year indicate that if I could register a ResourceToolAction helper, I might be able to intercept display events.

I created an OctaneToolAction that implements (so far) ResourceToolAction.
I had to make some guesses on on values to be returned by methods that implement this API, but I have a simple class now.
There are a couple of interfaces that can be implemented to expand what is handled by this class:

- CustomToolAction - must be implemented if the ActionType is CUSTOM_TOOL_ACTION. Doesn't apply.
- InteractionAction - implemented if action involves user interaction to complete the action. May not be needed.
- ServiceLevelAction - (no documentation - idiots) Cancel, finalize, initialize, and isMultipleAction. Not clear if needed.

So we got a basic tool action helper. The question now is, how is this registered?
There is a ResourceTypeRegistry, which includes a register(ResourceType) method.
ResourceType has a set of actions associated with it, which are ResourceToolActions.

So let's do this:

1. ResourceTypeRegistry.getType("ResourceType.HTML");
2. Examine it's actions.

It may be possible to retrieve the standard definition of the HTML action and substitute a modified version at run time. It seems likely to me that these types are all in-memory anyways.

I added code to grab the ResourceTypeRegistry using the ComponentManager. It seems like a good place to get it from.
Re-compiled - hot deploy to Tomcat worked, so I don't have to cycle Sakai.

Successfully got the registry.

Unable to get a ResourceType of HTML. I have a suspicion about this. The Resource tool may be doing all of the registrations when it is started up. Brought up Sakai and re-tried it. No luck.

There is a registry method to getTypes() that should return all registered types.
This method returned the following types (given by their labels):

- Citation List
- File Upload
- Folder
- Web Link (URL)
- HTML Page

- Simple Text Document
- Form Item

Also interesting is the fact that the id's are given as class names. The id of the HTML page is "org.sakaiproject.content.Types. HtmlDocumentType. I examined these class back in June of this year. It has several tool actions defined for it, including an ACCESS_CONTEST action that has been either deprecated or no longer used. Perhaps I can replace it with a new one.

Oct. 25, 2011

Yesterday I managed to list out all of the registered resource types. Today, I'm going to focus on the HTML type.
Verified that I can get the HTML Resource type and view it's associated actions.

I see that the HtmlDocumentType is in the Sakai Content code. Structures for resource types and resource tool actions are all part of the Sakai kernel, which is what you'd expect of good separation of data vs. structure. This opens the possibility of modifying operation of the Resource tool without having to modify the kernel (which would be difficult).

The finalize() method of a ResourceToolAction presents some possibilities. It is passed a reference to the resource, so I could modify it on demand, but it would be a permanant modification, I believe - not what I really want. What I want to do is to set up a way to intercept a call to getBytes() so that I can inject the Octane power cell.

It seems like ResourceToolActionPipe is the thing I want to take control of. It has a getContent() method that presumably can be used to modify code on the fly.

ResourceToolActionPipe provides a conduit through which ResourcesAction and an unknown helper may communicate about the execution of ResourceToolActions in which the registered action specifies that some part of the action is handled by a helper.

Looking through ResourcesAction, I note that the ResourceToolActionPipe is stored in a tool session attribute. I think it is created when an action is started, persisted as a runtime attribute, then deleted when done. ResourceToolActionPipe objects are not created in ResourcesAction. So where are they created?

A ResourceToolActionPipe can be passed into the creation of a ResourcesItem in the Resource tool, but I have console flags in place that should signal when this happens. It didn't in previous tests.

Just to see if I might gain access via an action, I uncommented support for ACCESS_CONTENT in HtmlDocumentType and added an Octane hook alert in HtmlDocumentAccessAction.finalizeAction(). Didn't get any alerts or messages. Darn.