Space Details

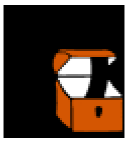
Key:	CSP
Name:	Public: Concord Software Projects
Description:	
Creator (Creation Date):	scytacki (Apr 29, 2007)
Last Modifier (Mod. Date):	scytacki (Apr 29, 2007)

Available Pages

- OTDocumentView

OTrunk

This page last changed on Jan 30, 2008 by stepheneb.



OTrunk is a Java framework developed at Concord Consortium that supports the creation and modification of rich Java component-based interactive content and the persisting of both author and learner state of this content. OTrunk uses otml, an extensible declarative xml-based language for both declarative layout and composition of content as well as for object state persistence (learner data).

OTrunk stands for Object Trunk. Trunk is used both to be a root of a tree of objects. And also to be a container of lots of objects like the trunk you would take to summer camp (or the trunk of your car!).

Start with OTrunk Introduction and OTrunk Examples and the Developers FAQ page: OTrunk HowTo. You can also download a PDF export of the OTrunk portion of this space (last updated 20080130).

Child pages sorted with the most recently updated first:

- OTrunk SDS URL Generation
- OTrunk Activity Authoring
- OTrunk Profiling
- OTrunk Schemas
- OTrunk Collections
- OTrunk Object References
- OTrunk Jackrabbit
- OTrunk Introduction for Developers
- OTrunk Reporting
- OTrunk Controllers
- OTrunk Session Forensics
- OT Rubric
- OTrunk Script Example
- OTrunk HowTo
- OTrunk Layers
- OTrunk Overlays
- OTrunk Testing
- DataGraphApplet
- OTrunk ID System
- OTrunk View Services
- Integrating Java applications into OTrunk
- OTrunk Services
- OTrunk Script
- OTrunk Viewer Properties

- Creating OTClasses
- OTrunk View System
- OTrunk Goals
- OTViews
- Creating otml files
- OTrunk Examples
- OTClasses
- OTrunk Curnit
- OTrunk Archived
- OTrunk Plan for CC
- OTrunk Introduction
- Extensions for 2.0

External resources:

- Using Concord OTrunk/DIY steps within Wise/Pas

Creating OTClasses

This page last changed on Apr 29, 2007 by scytacki.

There are currently 3 ways to make a new OTClass. They are listed in order of preference. The first is the preferred way to make an OTClass.

- make an interface that extends the OTObjectInterface
- make a class that extends DefaultOTObject
- make a class that implements OTObject

Interface Extending OTObjectInterface

Make a new interface which extends OTObjectInterface. The properties of the OTClass are defined by defining get and set methods for those properties.

The OTrunk framework will dynamically create an implementation of this interface at runtime. You do not need to implement your interface

Class Extending DefaultOTObject

DefaultOTObject implements all of the methods of OTObject. The only thing you need to do is make an interface that extends OTResourceSchema and make a constructor for your class that takes a parameter whose type is this interfaces. You add properties to your OTClass my defining get and set methods for those properties. You do not need to implement your ResourceSchema interface, the OTrunk framework will dynamically create an implementation at runtime.

The difference between this approach and the Interface approach above is you can add behavior to your OTClass. But it also means if you want to provide access to the properties of your OTClass you need to write methods in the OTClass which delegate to the resource schema that was passed in the constructor.

Class Implementing OTObject

You need to create a resource schema interface like above, and you need to implement all the methods of OTObject.

This option should only be necessary if you want to turn an existing class into an OTObject. You can subclass it and implement the OTObject methods. There are alternative ways to link existing classes with OTObjects, particularly the OTView System and the OTController System. These alternative ways are preferred because they separate the data structure from the behavior.

Comments

- What does an OTClass do?
- Can you list a few examples?
- Can you sketch an example of adding a new class and what new functionality that adds to the system?

Posted by stepheneb at Apr 30, 2007.

Creating otml files

This page last changed on Jun 23, 2008 by stepheneb.

OTrunk configurations are created by making OTML files that specify the existance and behaviour of each of the OTrunk components you want to include. OTML is an extensible declarative xml-based language for both layout and composition of content as well as for object state persistence (learner data).

The simplest OTML file must specify an OTrunk, declare a number of imports (the OTrunk components being used), and specify the location and behaviour of each of the components.

Hello World example

In this example we will create a OTrunk Viewer containing a simple text object.

You can run it here: text-label.otml

text-label.otml

```
<?xml version="1.0" encoding="UTF-8"?>
<otrunk id="33754150-b594-11d9-9669-0800200c9a66">

  <imports>
    <import class="org.concord.otrunk.OTSystem"/>
    <import class="org.concord.otrunk.view.OTViewEntry"/>
    <import class="org.concord.otrunk.view.OTViewBundle"/>
    <import class="org.concord.otrunk.ui.OTText"/>
  </imports>

  <objects>
    <OTSystem>

      <bundles>
        <OTViewBundle>
          <viewEntries>
            <OTViewEntry
              objectClass="org.concord.otrunk.ui.OTText"
              viewClass="org.concord.otrunk.ui.swing.OTTextView"/>
          </viewEntries>
        </OTViewBundle>
      </bundles>

      <root>
        <OTText name="Text Object">
          <text>
            Hello World!
          </text>
        </OTText>
      </root>

    </OTSystem>
  </objects>
</otrunk>
```

You can assume as a convention that a name beginning with an uppercase letter represents an object with a subtype of `OTObject`, and one in camel case that begins with a lowercase is a property of the enclosing object. For example, in the otml above, we can tell that "text" is a property of OTText that can be retrieved by calling `t.getText()` for some object `t` of type `OTText`.

Let's begin from the end product: we want to show "Hello World!" in the application window.

Let's regard "Hello World!" as data for the model (as in the Model/View/Controller architecture). We need to define both the model and the view for it. When otrunk process an ot object, it finds the corresponding view from the `<viewEntries>` section of the otml. (For more see OTrunk View System.)

The "root object", usually put under `/otrunk/objects/OTSystem/root/`, and its view, encompasses everything displayed in the client area. (Be careful to distinguish it from "root element of the xml", which is `<otrunk>` here).

Here we used `OTText` as our root object. When otrunk reads in the otml and runs across the `<OTText>` tag, it knows how to construct an object that implements the `OTText` interface so that the "view" code can call `getText()` on it to retrieve the text which is "Hello World!".

In order to tell otrunk what we exactly mean by `OTText` though, we need the `<import>` element in the `<imports>` section that shows the fully qualified name (`org.concord.otrunk.ui.OTText`). (Same for all the other ot classes that appear in the otml.)

Note that `OTText` is mapped to `OTTextView` in an `OTViewEntry`. If you look into the source code of `OTTextView`, you'll find the getComponent() method that returns a swing label that contains the text obtained from the `OTText` object.

Slider Data Graph Example:

Here's a slightly more advanced example where the property data producer for a `OTDataCollector` graph comes not from a sensor but instead from the movement of a slider component.

You can run it here: slider_data_graph.otml.

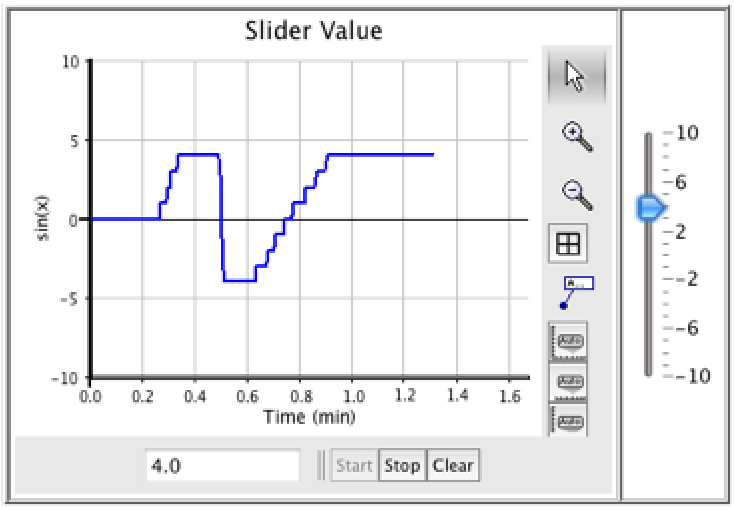
Inside the bodyText (*lines 31..40*) is a simple html table with object references in two cells. The embedded object in line 34 is a OTDataCollector (*line 44*). This type of object is most often used to collect data from probes or to allow a student to sketch a dataset however in this case the `OTPropertyDataProducer` is setup to read inputs from the slider.

The slider (*line 61*) has a very simple set of attributes:

- local_id: "slider"
- name: "Slider"
- minimum: -10
- maximum: -10

- value
  The only one a learner can change is the value which is sent as a data stream to the `OTDataCollector`. Any changes to the actual value of the slider will get persisted at the end of the session as leaner data.

Here's what a running instance of the activity looks like:



Here's what the otml (*xml*) looks like:

See OTrunk Introduction for an introduction to otrunk.

See OTrunk for advanced topics for otrunk.

Link to source code:

- OTSystem
- OTViewEntry
- OTViewBundle
- OTText
- OTTextView
- OTObject

DataGraphApplet

This page last changed on Mar 18, 2008 by stepheneb.

The OTrunk library includes a JApplet extension called: OTAppletViewer (_trac_)

A few examples can be seen here:
http://continuum.concord.org/otrunk/examples/DataGraphApplet/ot-index.html (_trac_), scroll down to the **All Files** section to find the links to the html files that embed OTrunk applets.

The multi-applet-test.html page shows how two or more applets can be embedded on the page and share the same OTrunk state.

In most of the examples above the applet uses 3 jars:

- otrunk-datagraph.jar - this is a combination of several java projects into one jar. It was built with eclipses jar export feature.
- jdom-1.0.jar - this is a 3rd party jar needed by OTrunk
- jug-1.1.2.jar - so is this

The jars referenced by the examples were last updated July 2007 and are accessed in the `output` directory relative to the codebase for the html files.

You can find newer versions of these jars in these directories.

- http://jnlp.concord.org/dev/org/concord/datagraph/
- http://jnlp.concord.org/dev/jdom/jdom
- http://jnlp.concord.org/dev/org/concord/datagraph/

An alternative to bundling the jars together is to reference each jar from our jnlp server: jnlp.concord.org Here is an example of that: nlpserver-applet-test.html.

Another interesting example is the UDL prototyping work Sam Fentress did on this page http://www.concord.org/~sfentress/Clouds2.html. This example uses older jars and is not maintained on our contninuum server. What is different is the more extensive examples of controlling OTrunk objects from Javascript. Click through to the second page and explore.

Extensions for 2.0

This page last changed on Apr 29, 2007 by scytacki.

OTrunk Next Steps

Object Templates

OTrunk Object Copies

Data Collector schema version 2

Move to EMF

More Semantic Information

Data Collector schema version 2

This page last changed on Apr 29, 2007 by scytacki.

This is just a draft. It needs more work.

More Semantic Information

This page last changed on Apr 29, 2007 by scytacki.

The current set of otml file used by teemss is missing a lot of semantic information.

For example the Unit, Investigation, Section structure is not there. Instead that is stored as a tree of PfCompoundDocs.

That information should be stored in the otml file used by the SensorPortfolio. This way the report tool could be better. Also there could be conversion to labbook at the client instead of only at the server.

One difficulty with this is making sure the style in which the semantic information can be easily changed. Currently this style is stored in xslt files. It isn't super easy to change but at least it doesn't require chaning java code and regenerating jar files.

The rendering from the semantic info to the actual display, the style, should be stored in the otml file as well. Or in an external shared file. Some thoughts on this are described on the Object Template page. However that page was thinking more along the lines of a one to one releationship between objects and their templates. But this case one piece of semantic information could be displayed in multiple places. So that one to one doesn't work out. However the templating or object copying is a requirement for doing this style work. The style could be defined in a template object. Then this could be copied or templated and the result would have the particular objects bound to it.

But you shouldn't have to explicitly bind objects to an instance or copy of style to generate a view. This should be dynamic in someway. So new sematic objects automatically result in new views.

Two cases of styled views are the navigation/folder view that is on the left of the SensorPortfolio and the document view on the right.

Move to EMF

This page last changed on Jun 19, 2007 by scytacki.

It appears to make sense to move the guts of OTrunk to EMF, eclipse modeling framework.

Why

- Better authoring system - emf can generate eclipse authoring system
- Better support for object schema specification - makes authoring easier and less error prone
- More developed serialization - I believe this includes concepts of modularizing the xmi input files.
- Easier for beginning developers to understand.

Why not?

- Introduces more dependencies
- Requires code generation, which increases the initial barrier for creating new object types.

Intermediate approach

To start with I'll try using EMF as a OTDatabase implementation. This will allow us to use the better xml format, and cross document references. It will also push the ID system used by OTrunk to be more portable.

Doing this will require using the solutions to 2 of the issues below: containment and XML properties. Inorder to phase in EMF.

Issues with moving to EMF

Containment

The issue here is that EMF requires static containment specification. This means that a feature has to set to be containing or not containing in the EObject definition. A instance of the EObject cannot decide if a particular feature should contain another instance or just reference that instance.

After a discussion on the EMF users news group. It appears the best way to deal with this is to have a common superclass (EDynamicContaimentObject) for all of the objects in the OTrunk system. And that superclass will have a single 0..N containment freature. Perhaps called dynamicContainmentContents. Objects which extend this one will use references for all of their freatures.

The InternalEObject of the EDynamicContainmentObject will modify the behavior of eObjectForURIFragmentSegment and eURIFragmentSegment so if the object is stored in the dynamicContainmentContents and it is referenced by another feature in the object then the other freatures path is used to refer to the object.

Finally the XMLSerializer will be modified so the dynamicContainmentContents is not written out if its objects are referenced by other parts of the same object. This last bit will probably be tricky especially in the case of lists.

XML Properties/Features

I saw a reference in the emf news group about handling xml elements which are not emf objects, but I didn't read it enough to know if will work here. In the short term these elements can marked to be written out as nested elements, but they will still get escaped. That will be really frustrating to look at for our xml authors.

User Data Proxies

Services (dependency injection)

Memory Management

Templating

Object Templates

This page last changed on Apr 29, 2007 by scytacki.

With the templates discussed under object copies. It should be possible to move more of the layout java code into otrunk templates. For example the data collector design could consist of separate tool bar, start/stop control, and graph objects. That can be stored in a template. And then the references too it can just be filling in parts of it.

This is something that isn't mentioned in the object copies pages. The current design of object copies has a simple concept of variables. So that a copied object can have a part of itself replaced. Currently on one variable is allowed. A more general design would allow an number of named and typed variables to be defined. And then this would become something like a new object type.

The current design makes something like this possible because the resources are separated from java interfaces. However taking the new object design and generating a schema for it won't work. The move to emf would make this even harder. At least in the way I understand it.

One question is what does student saved state look like in one of these dynamic types. Theoretically any of the values of the template could be changed by the student at run time, so those changes need to be saved somewhere.

One way to say these are not really dynamic types. Instead a object copy is made that is backed by the original object. Then the variables are filled in into the copy. Then the student is given a chance to modify this object. So the students work is saved into the form of the orginal object not the dynamic type.

I think eliminating the java code from this design is important but not crucial. It seems ok to force an author to create the "dynamic type" using the standard object defining mechanism, (which requires java code) but then they can define the mapping of that definition to the actual object with otrunk objects instead of in java code.

OTrunk Next Steps

This page last changed on Apr 29, 2007 by scytacki.

The order of these is roughly in order of importance and dependency. The first two should make the OTrunk system easier for new developers to get a handle on.

- add plugin support to views so they can be extended with sub views they don't know about.

- attempt to remove ResourceSchema inner classes, replace the ResourceSchema with an interface, and replace the current OTObject with a wrapper object, or view. This is good for a few reasons:
    - makes external manipulation of the OTrunk object easier.
    - it is a step towards an emf implementation of OTrunk
    - this will not work in all cases. There are some cases where having code in the OTClass is very convenient.
        - to address this cleanly, we need to have generated java code, or use bcel or script to extend abstract classes.
        - this is needed for service implementations, for example the view service is used to setup views, so if we try to replace resource schemas with views and interfaces this view service would have to be an exception because it can't serve itself.

- merge wrapper objects and views. these two features are really the same thing, just being used in two different ways. A wrapper object is really just an object view of a model.

- extend view service to support views implemented by OTrunk objects instead of only java objects. This will open up the possibility of view implementations using template languages (velocity), scripting languages (javascript, jruby), form languages (xform) or a custom otrunk form languague.

- implement an xml view. This could be used to pass a dom tree representation of a particular root object to an xml viewer, or editor. This could eventually be a replacement for the xmldatabase. This comes out of the following logic:
    1. we want to move to emf or something like it, which can store the otrunk state directly in java objects.
    2. this removes ability to maintain the object state in a single place outside the objects.
    3. but we want to maintain the ability to store and manipulate the objects as an xml tree or as in a relational database.
    4. the manipulation part of this is basically what a graphic view of the objects does. So the same view architecture can be used.

1. a. the idea of loading a set of otrunk objects in from an xml view is the same as the desired loading of otrunk objects from a pre constructed set of wrapped objects which are also views.

- implement a custom form view. This requires the extended view service above. The form would specify form controls for editing properties of an otrunk object. This is very similar to xforms. Basic (primitive) properties are easy, because we have controls for them. But object, list, and map, properties do not yet have controls. Probably multiple types of controls will be needed. One example is the multiple graphable view. This essentially is a control for handling lists. An object property could have its form view embedded, with the ability to find and select another object instead.
    - A first application of this would be to implement a way to author sensor data graphs directly in the OTViewer

- add support for combining multiple files together: either imports, or url based reference ids. The latter is the approach taken by EMF. There is currently support for multiple files, by making a separate database for each and then using the UUID of the database to reference the object, then the OTViewer just needs to open both files. But that isn't very easy to extend.

- implement very simple document editor that allows insert of existing objects, and creation of new objects. This is most useful after the custom form view is implemented.

- analyze the service setup, examples of services are the view service, and the interface manager. This analysis will be best informed after the plugin arch is finished.

View Updates

This page last changed on Apr 29, 2007 by scytacki.

View Entries

- otObjectClass
- otViewClass
- otAuthorEntry
- otReportEntry

the otAuthorEntry and otReportEntry are optional references to other view entries.

If one class wants to implement all 3 functions, it should make 2 inner classes one for each of the functions. And then make 2 more view entries pointing to these classes.

Alternative approachs

One way to implement the 3 functions in one class is to add 2 new interfaces: OTAuthorView and OTReportView. And the interfaces would have a getAuthorComponent and getReportComponent. This would make it easy to implement, but:

- if one classes does this and the author wants to reference one of these alternative functions directly there would need to be a special referencing mechanism invented.

OTrunk Object Copies

This page last changed on Apr 29, 2007 by scytacki.

Update

Read below for a description of the problem. The update is that this can almost be handled with the code the way it is now. These copies are really more like templates. And templates are a generalizatoin of the way student data is working. So in the same way that a student data object has an authored object. An authored object can have a template object. The current design of the object service almost makes this possible. We need to keep this in mind if we move to emf.

Original Text Below

Currently in the teemss content and in the MAC content there are objects that could be defined in one place and then referenced in many other places. However some of these object then have student state that is associated with them.
In the teemss content these are mainly data collectors. The same data collector is used in several places throughout the units.

There is a facility for doing object references in the OTrunk, however this aren't enough in this case. With an object reference there is still only one object. So any state a user creates only has one place to go. So a question that is referenced twice will be linked. The current solution is to make a literal copy of the object so there are two or more places for the state to be stored.

Currently we handle this with an extra layer. The is a one data collector saved in the database and then references
to this collector stored in the units. When the OTrunk objects are created the data collector is literally copied once for each reference.
This is bad for a few reasons.

1. It means we always need that extra layer if someone was authoring content directly with OTrunk objects they wouldn't have this extra layer and would have to do these extra copies.
2. The data is repeated so it makes the content larger than it has to be.
3. If the reports are self contained in the OTrunk then these reports won't know that these multiple data collectors
   are really all exactly the same.

So it would be better if the OTrunk had the ability to handle these object copies internally. In any solution to this problem even with the extra layer we are currently using there are 2 issues:

1. nested objects
   If the copied object has object inside of it they must be copied too. There could be cycle. Initially these cycles can be forbidden. I'm pretty sure the can be detected. It might also be possible to handle them in some automatic way.

1. references to the copied object and nested objects
   When a student creates state that is for one of these copied objects or an object inside of it, this state needs

to have a reference or be linked to this copied object. This is the state can be retrieved later either to restore the object in the same context, or for doing an external report on the state.
The current OTrunk system doesn't have a clean way to handle this. Either all the sub-objects in the copied object would need to be assigned globally unique ids, or there would need to be some transient or chained id facility in the OTrunk.
The chained id concept seems the best. These "chained" ids would be stored in some cases, and they would be used for lookups. There is already a specific chained id lookup used to find a students state for a particular object. In this case the chain consists of the particluar authored object and the students id.
If this was generalized, then when the student map contained an id to a object it might be a chained id which is an id to the copied object and the id of the original sub-object. A report would then ask the student map for the student state associated with this chain.

We will need to do this anyhow in this first version but it will be more explicit. The layer that copies the datacollector will need to also copy the sub objects and assign modified local ids to them. Then the import will need to map these local ids to global ids. And when someone wants to do a report on a datacollector they will need to construct the modified local id. First they get the local_id of the data collector then they add the id of the response where the datacollector is used. This id is then mapped to the global id of the particular data collector.
If the report needs a sub object like the data store inside the data collector. Then the report would get the global id of the copy and then either traverse it down to the data store, or it would figure out the modified local id of the data store and then the global id, and then ask for the student answers.

View Templates

This page last changed on Apr 29, 2007 by scytacki.

This is similar to Object Templates. The idea here is that some views which are currently implemented as java classes, could be implemented with otrunk objects instead.

For example there is an ot object that represents a Data Logger setup configuration (OTDataLoggingRequest). One view for this object is a button that sends the request to the current logger when pressed. This view could be implemented with OTrunk objects instead a java class.

The view would be something like

```
```

This can be implemented by copying the object inside of the template element and inserting the object being viewed at the at the insertPoint. Doing the inseration is tricky, because an id can't be used because it will not be the same after the OTSetupLogger has been copied. However the concepts of templates could be used, so the id of the "template" can be used and it will just be overridden. Fo example if you think of the template as an authored object, and the instance of the template is the user object. Then just the same way a user can set a value in a particular place, the instance of the template can have its object set in a particular place.

A further extension of this is to be able to choose arbitrary parts of the object being viewed to insert. This is more like a mapping. So in the simple example it would say put the root object at this point. The more complex case would say put this attribute of the root object in place 1, and this attribute of root in place 2.

Because viewEntries are implemented on top of the core OTrunk framework, then if the OTrunk frame suppoted the basic concept of these types of templates, then this facility could be used here. So no special code is needed.

The difficult part of this is with user data. If a particular view can create user data. Because the view was created dynamically, there is prexisting location to store the user data. However a new area can be created that stores the data in a way that is specific to the dynamic view, but it contains references to the original objects that created the dynamic view.

So if a report needs to find the information about a particular orginal object it will need to search for references to this object or its sub objects in the dynamic view user state area.

Integrating Java applications into OTrunk

This page last changed on May 21, 2007 by sfentress.

In order to create an OTrunk component from an external Java application, you need three things:

1. The application's source code and/or a compiled JAR file
2. The ability to initialize and run the application without using the application's main method
3. Access to a Java Swing component that comprises the elements of the application to be presented to the user.

While it is not strictly necessary to have access to the application's source code, correctly initializing and running the application without reading its main method becomes a tedious task of trial-and-error.

OTrunk components consist of two parts: a model and a view. Unlike a model-view-controller system, however, it is the view that must handle the initialization and running of the external application.

The simplest scenario for integrating an external Java component is to initialize the application (loading initialization data from an external URL if necessary) and display the application's Swing component. This supports embedding the component along with authored data but does not support OTrunk persistence of either the authored data or changes to the learner state data.

This level of integration is relatively simple: initializing an external application is as easy as copying the main method of the application and pasting into a "load" method in the OTrunk view. This load method is then typically called from the getComponent() method, which is itself called when the view is displayed in the OTrunk document. In many cases, however, some further tweaking of the initialization code is necessary to get the component to display correctly within the document.

After the application has been loaded, all that remains is to return a Swing component representing the application. Most Java applications display a JFrame when they are run. If this is the case, it is usually simplest to return the content pane contained within the application's JFrame, and to then hide or dispose of the frame to prevent multiple windows from opening within the OTML document.

We have currently integrated the following external applications:

Application/Applet	Integration Level
Molecular Workbench	component display
NetLogo	component display
Pedagogica	component display
PhET Circuit Construction Kit	component display
PhET Wave Interference Model	component display
PhET Sound Model	component display
PhET Faraday Model	component display
PhET Discharge Lamps Model	component display

OT Rubric

This page last changed on Dec 18, 2007 by imoncada.

This documents explains briefly the steps to follow to use the OTRubric framework to display a student report (including grading) of an activity given basic rules to calculate grades with different indicators.

Assessment and reporting process overview

In this framework, in order to assess student work and produce a quantitative report, the following steps take place:

- **Step 1**. The activity (script) logs the goal of the activity and everything the student does (all-events log).
- **Step 2**. From the general all-events log, a script interprets the relevant information and processes it, doing all necessary calculations in order to produce learning indicators that can be quantified (graded) individually. Note: Generally, these indicators are designed based on the assessment needs and they are expressed into a **rubric**, which basically contains a collection of indicators (or "relevant things to pay attention to") with a grading system that contains points to be assigned per indicator according to the different relevant possibilities that the student can create.
- **Step 3**. From the individual indicator values, a script takes the **rubric** and applies it to the student work, producing a report that contains the total grade of the student and also includes details on the grading of each indicator.

The following documentation talks about Steps 2 and 3. **Step 1, and the whole all-events log is skipped right now** since it's easier for the activity script to go Step 2 right away and generate directly the indicator values instead of going through an all-event log first (which would need to be parsed anyway by another script).

In order to create a report out of your activity, you need to implement Step 2, and you also need to design a rubric that matches your indicators. Step 3 is done automatically for you by Java classes already written in the framework. The framework also contains a basic UI to edit the rubric.

How to create a report of an activity

Design a rubric

Think of the useful learning indicators that you want to consider when grading the student work. Make a list of each indicator and all the possible values that you want to consider in each indicator.
Then, you will have to express the rubric in an otml file using the classes in the **org.concord.otrunkcapa.rubric** package.

For example, let's say we have an activity that asks a student to perform a simple sum of two numbers. In order to grade the activity, probably, you would like to know first if the student answer correctly. Maybe you also want to know how many different answers the student wrote down before he decided to submit the answer definitively. Another useful indicator might be how long did the stduent take answering the question.

So, your rubric otml file could look like this:

This sample rubric has the three indicators mentioned above. Once all the indicators have been designed in the rubric, there is a UI that can be used to change the points per indicator. In this particular example, the points total to 20 points, and 1/2 of them are derived from whether the answer is correct or not.
The first indicator (whether the answer is correct) has only two possible values: incorrect or correct. We could consider a middle ground possibility, for example, if the student is "almost" correct (for example, if the student forgot a zero, had the sign wrong, etc).

Write a script that generates a value for each indicator in your rubric

In order to be able to use a rubric, the activity script will have to generate an assessment object (**OTAssessment**) that contains one value per indicator in the rubric.
Generally, the script will add this object to the **contents** section of the **OTScriptObject**.
To follow the previous example, this would be a script code that can be written to generate indicator values for the sample rubric we wrote:

It is your job (the script's job) to perform the right calculations to determine the value of each indicator (whether the answer is correct, how long did the student take, etc). Use the indicator name as the key in the value map for the assessment object.

Display the report

In order to show a report, use the **OTAssessmentView** class as the OT view for an OTAssessment object. This view will need a view config object (OTAssessmentViewConfig) that provides the OTRubric object to use.
This is an example of the otml needed to use this view:

In this case, the view config is referencing the OTRubric object using its local id, assuming the rubric is defined in the same otml file.
If the rubric is defined in a separate otml file, you can include the external otml file using an OTInclude, and then reference the rubric using its **global id** (unique uuid) instead of the local id.

To display this report, you can use the report button which automatically pops up showing the contents section of a script object, so that will include your new OTAssessment object.

OTClasses

This page last changed on Aug 14, 2007 by scytacki.

- OTCompoundDoc
- OTDataAxis
- OTDrawingTool2
- OTFrame
- OTMultiDataGraph
- OTZoom

OTCompoundDoc

This page last changed on Apr 29, 2007 by scytacki.

This is core object used by SesnsorPortfolio. It is the way to represent a Document. It does this with a subset of xhtml. It also acts as a folder which contains other objects. These childrent objects are shown in the tree view on the left of the OTViewer application. So it has 2 parts, the document content, and the list of sub objects.

Here is the list of xhtml elements which are passed though the teemss authoring system. This is **not** a complete list of the elements supported by the OTCompoundDoc component.

In addition to xhtml it has an extension for the "a" and "object" elements. These extensions make it possible
to embed other objects inside the document ("object" element), or link to other objects ("a" element).

Links to objects either in "a" elements or in "object" elements. Can either use the objects local_id or id (global id) attribute.
If the reference is to the local id the notation should be:

If the reference is to a global id the notation should be:

If you want to embedd or link to an object that doesn't exist anywhere else, and you don't want to make it visible in the folder tree. Then use the element:

You can define new objects inside of this one, that won't be seen.

OTDataAxis

This page last changed on Apr 29, 2007 by scytacki.

You use different units for a time axis:
seconds = s
minute = min
hour = hr
day = d

The min and max number are still in seconds.

OTDrawingTool2

This page last changed on Aug 14, 2007 by scytacki.

A page to document the format for, and how to create your own drawing tool otml.

You can run a drawing tool with stamps example here:
http://rails.dev.concord.org/sds/2/offering/144/jnlp/540/view?sailotrunk.otmlurl=http://continuum.concord.org/otrun

Then go to the file menu and select "Save As..."
Open the saved file in a text editor and look for this part:

```
```

You can put in references to your images, save the file, and go back to the File menu and select "Reload Window".
Now you should see your changes.

You can use URLs relative to the location of the saved otml file, so you do not need to post the images online to test this out.
It should work with pngs, gifs, and jpeg images.

OTFrame

This page last changed on Apr 29, 2007 by scytacki.

Any object can be put into a popup window. This works when the object is referenced from compound document.

There needs to be a frame object that defines the size of the popup window. Then in the link tag the frame object is referenced by a target attribute. Just like in a web browser if two links use the same target then the same popup window will get used for both.

The href of the a link can be any otrunk object. If the object doesn't have a proper view class then there will be a error message in the popup window.

OTMultiDataGraph

This page last changed on Apr 29, 2007 by scytacki.

OTZoom

This page last changed on Aug 14, 2007 by scytacki.

You can create your own zooms by running this jnlp:
http://rails.dev.concord.org/sds/2/offering/144/jnlp/540/view?sailotrunk.otmlurl=http://continuum.concord.org/otrunk

Click on the File menu and "Save As..."
Save the file to a location you can find later.
Open the file with a text editor. If you scroll down through the file, you will find the piece of text that looks like this:

There is one transform between every two images. So there is one less transform than images.
The transforms indicate how to place the next image in the current image.
So in the first transform above, the second image is scaled by 0.5, and then located at (30,92) in the first image.
This is the reverse of how the actual zoom will work.

You can modify the file you saved, and then go to the file mean and do reload.

The iamgeBytes URLs can be local URLs that are relative to the saved otml file.

OTrunk Activity Authoring

This page last changed on Jun 19, 2008 by sfentress.

OTrunk activities may be created by

1. directly writing OTML files, either by hand or using a template system
2. creating and editing OTrunk components from within the OTrunk system and saving the resulting OTML file
3. some hybrid combination of the two

Directly authoring OTML files

OTML files may be written directly by hand, or they may be produced with the help of a template system that will create a basic OTML file and allow the author to fill in the content. Some work has been done to assist hand-writing OTML files by using schemas, which can then be used in conjunction with XML editors to produce valid OTML files.

See:

- Creating otml files
- OTrunk Schemas
- ITSI DIY for an example of web-based template authoring.

Authoring activities from within OTrunk

When an OTrunk activity is run, all changes to the OTrunk components are saved, and the resulting changes can be exported as an OTML file.

Currently there are two modes that an OTrunk activity can be run in: *Learner mode* and *Author mode*. When an activity is run in Learner mode, all changes are saved into a database which is separate from the main database containing all the OTrunk components. When the learner saves their data, the resulting OTML file represents only the changes to the components made by that learner. When an activity is run in Author mode, in contrast, all changes are reflected directly in the main database. When an author saves their data, the resulting OTML file consists of the entire activity, with their changes included. This allows an author to create and edit new OTrunk activities.

Author mode and view mode are controlled by the system property `otrunk.view.author`, which must be set to `true` for authoring.

While any changes to an object while in author mode will be saved directly into the complete OTML file generated, most default views of OTrunk components allow very few changes to be made. For example, the default view of a Data Table, OTDataTableView, allows text to be entered into the table's cells, but does not allow the author to change the table's title, or the number of columns in the table. For this reason, authoring views of most OTrunk components have been created which allow an author to change attributes of a component that would normally be inaccessible to a student.

By using authoring views, authors can change the attributes of embedded OTrunk components. However, this alone does not provide a way to embed new components into an activity, or to start an activity from scratch. Fortunately, the Document authoring view provides this functionality, which enables complete WYSYWYG authoring.

See:

- OTrunk Authoring Views
- OTrunk WYSIWYG Authoring
- OTrunk View Modes

OTrunk Authoring Views

This page last changed on Jun 18, 2008 by sfentress.

All (or almost all) views for OTrunk components provide affordances to a learner or author which allow that user to change some attribute of that object. For example, the basic drawing tool view, OTDrawingToolView, allows a user to add graphables to the tool, such as lines and stamps. These changes will then be saved either in the learner database or in the OTrunk database, depending on which mode the author is in. If the changes were made in the OTrunk database (by using the system property `otrunk.view.author=true`), then the activity could be saved as a complete OTML file with the changes included.

However, most basic views of OTrunk components allow only a small set of attributes to be edited. For example, the default view of a Data Table, OTDataTableView, allows text to be entered into the table's cells, but does not allow the author to change the table's title, or the number of columns in the table. For this reason, authoring views of most OTrunk components have been created which allow an author to change attributes of a component that would normally be inaccessible to a student.

Authoring views are no different from other views except in the number of attributes they allow a user to directly edit. Thus, while the OTDataTableView only allows text to be entered into the table's cells, the OTDataTableEditView provides further UI affordances to allow changing the number of columns, adding titles to the columns, locking individual columns from being edited, and sharing the data from the table with other components.

Many authoring views also provide a preview of what the component will look like to a learner, frequently in a tabbed pane. This preview is generated by requesting the default view from the OTViewFactory and embedding that view within the authoring view. For this to work, the authoring view expects to be defined within the ViewBundle within a specific view mode, and for the learner view to be defined as the default view within that bundle. For more on view modes, see OTrunk View System and OTrunk View Modes.

View class naming conventions

Most default view classes in OTrunk use the naming convention "<OT component name>+View"; e.g. OTDrawingTool -> OTDrawingToolView, OTDataCollector -> OTDataCollectorView.

Views that represent variations on the default view provide the name of that variation between the component name and "View". For example, the version of the OTDrawingTool view which does not allow the student to draw is OTDrawingToolNonEditableView.

Authoring views thus generally add the word "Edit" to the view name, giving OTDataCollector -> OTDataCollectorEditView, OTDataTable -> OTDataTableEditView.

Creating new authoring views

Authoring views can be created exactly as regular views. The only difference is that the UI affordances will generally allow more access to the components attributes.

An authoring view can either be a subclass to the original default view, or it can be a new view entirely. If the authoring view is subclassed, then a preview of the view can be generated simply by calling the getComponent method of the parent view. If a separate class is created, then it will need to be able to embed the view inside itself to provide a preview.

To embed the preview inside itself, the authoring view should typically extend AbstractOTJComponentContainerView. This will provide access to methods which will make embedding the default view possible. The easiest way to get the component provided by the default view mode is to use

```
getChildComponent(otDraw, null, OTViewFactory.NO_VIEW_MODE, viewContext,
jComponentViewContext)
```

This requires that the learner/preview view be defined as the default view in the ViewBundle. See OTrunk View Modes for more details.

More complex authoring

Some components are able to engage in complex interactions with other OTrunk components. For example, data may be shared between two graphs, or a component may need to know what other components exist in the activity. For these kinds of interactions the OTJComponentViewContext is very helpful as it can provide a list of all OTrunk components within the component's view context.

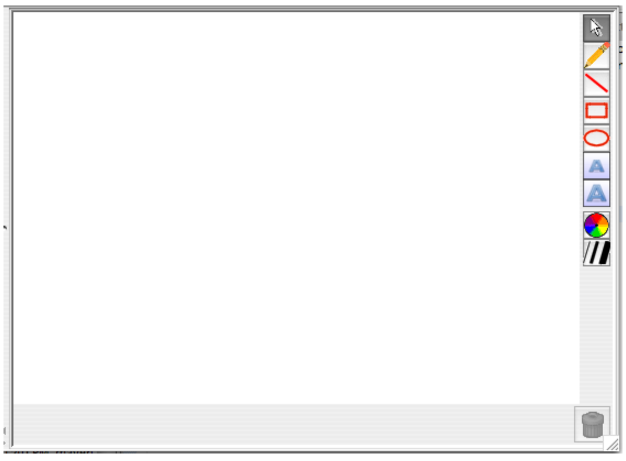
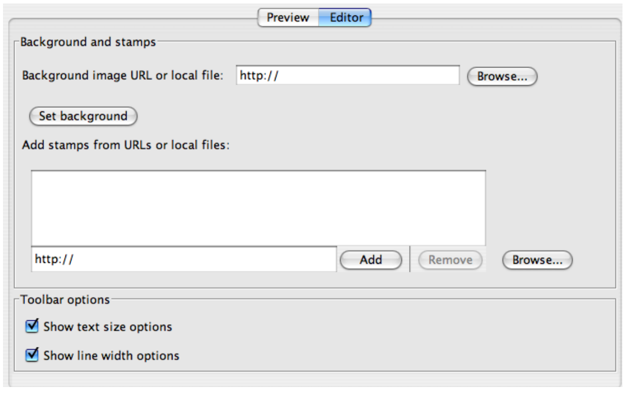
An example of using the OTJComponentViewContext to create a list of OTrunk components can be seen in the OTSnapshotButtonEditView.

Graphs, tables and other objects that use data may with the share their data between each other. For this, the OTJComponentViewContext may not be sufficient as the components may reside on different pages, in different contexts, and, further, the data cannot be named or annotated. For this reason, the OTSharingBundle and OTSharingManager were created. This is a simple way for a data-rich object to add itself or its datastore to a list during authoring. View can then request this list to find which components are sharing themselves, and can then use the data contained by those objects.

Examples of authoring views

OTDrawingTool

Default view	Authoring view

Preview Editor

Background and stamps

Background image URL or local file: http:// Browse...

Set background

Add stamps from URLs or local files:

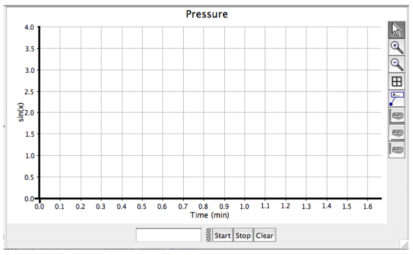
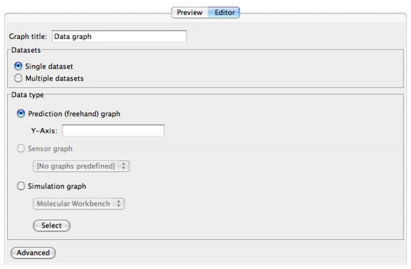
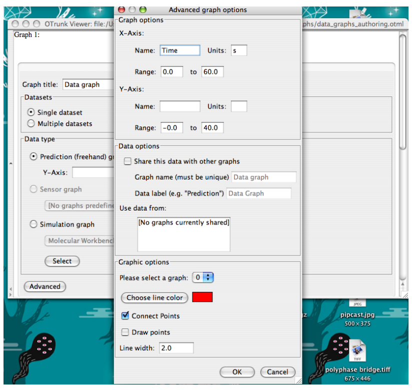
http:// Add Remove Browse...

Toolbar options

✓ Show text size options

✓ Show line width options

OTDataCollector

Default view	Authoring view	Advanced authoring view
		

Preview Editor

Graph title: Data graph

Datasets

[✓ Single dataset]

Multiple datasets

Data type

[✓ Prediction (freehand) graph]

Y-Axis:

Sensor graph

[No graphs predefined]

Simulation graph

Molecular Workbench

Select

Advanced

OTrunk Viewer: file:/U...phs/data_graphs_authoring.otml

Graph 1:

Advanced graph options

Graph options

X-Axis:

Name: Time Units: s

Range: 0.0 to 60.0

Y-Axis:

Name: Units:

Range: -0.0 to 40.0

Data options

Share this data with other graphs

Graph name (must be unique) Data graph

Data label (e.g. "Prediction") Data Graph

Use data from:

[No graphs currently shared]

Graphic options

Please select a graph: 0

Choose line color 🟥

✓ Connect Points

Draw points

Line width: 2.0

OK Cancel

See also

- OTrunk WYSIWYG Authoring
- OTrunk View Modes

OTrunk WYSIWYG Authoring

This page last changed on Jun 19, 2008 by sfentress.

What-you-see-is-what-you-get (WYSIWYG) authoring is a means of authoring OTrunk activities without needing to edit OTML directly. WYSIWYG authoring uses authoring views of OTrunk components to edit their attributes directly, which then can be exported to OTML to be run by students. In particular, WYSIWYG authoring uses an authoring view of the OTDocument component which allows an author to immediately write and edit text (like a Word document), embed OTrunk components inline into the document, and easily move components around and edit their attributes.

WYSIWYG document authoring

The authoring view for the OTDocument/OTCompoundDoc component, OTDocumentEditView, is a Word-like document editor. It uses a JEditorPane with a HTMLEditorKit for simple text writing and components from the open source project EKit for toolbar commands such as formatting text. The editor has support for creating new OTrunk objects, which will be added to the document's documentRefs and referenced within the text of the document.

To generate a list of embeddable OTrunk components, the OTViewEntry for the document's edit view must be of type OTDocumentEditViewConfig. This can provide a set of components which can be embedded. When a component is embedded, a copy of the object in the list is made before the object is added to the document's documentRefs.

WYSIWYG activity authoring

Some projects, such as the UDL project, contain multiple sections, each of which contain multiple pages. To support WYSIWYG editing, authoring views of page menus were created which allow easy creation, re-ordering and deletion of new pages and sections. Each time a page is added, a separate OTDocument is created, which can then be edited using WYSIWYG authoring. Affordances were also created to give authors the ability to designate the type of section they were creating (Pre-test, computer-based activity), as well as deeper attributes such as whether the views of pages automatically close once a student leaves a page.

Use of scripts in WYSIWYG authoring

The UDL project, which uses nearly 100% WYSIWYG authoring for creation of UDL units, makes use of scripts which greatly speeds up the process of authoring and performs some actions that would be much more difficult for an author to do without editing the OTML directly.

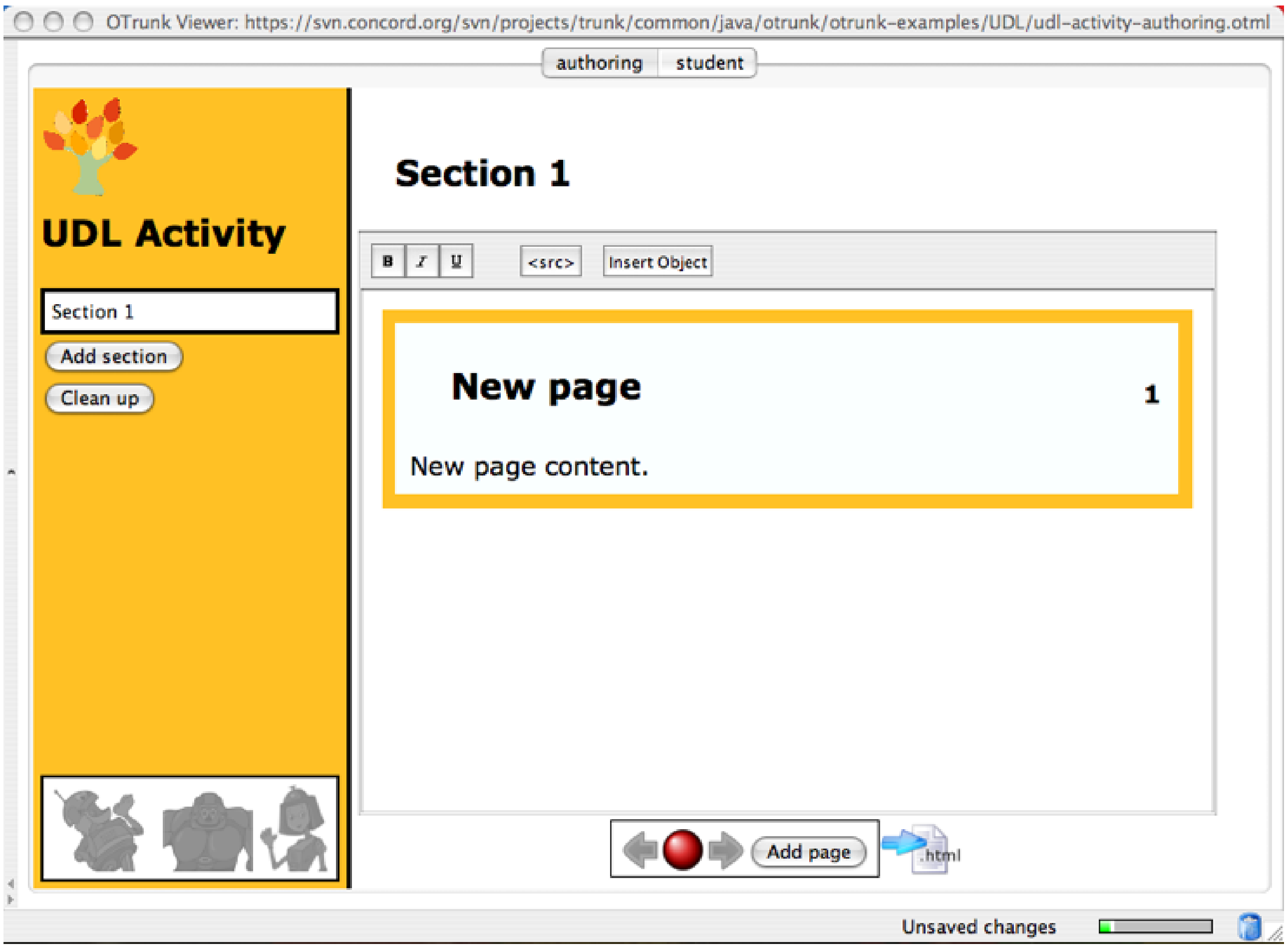
For instance, when a new section is created, the script automatically makes a new page for that section, fills in that page with some sample text, creates a new menu for navigating through the section and links the menu to the section. As another example, when an author adds a new snapshot button, the script parses the current page, finds the OTrunk component directly above the button, if any, and links the button to that component.

Examples of WYSIWYG authoring

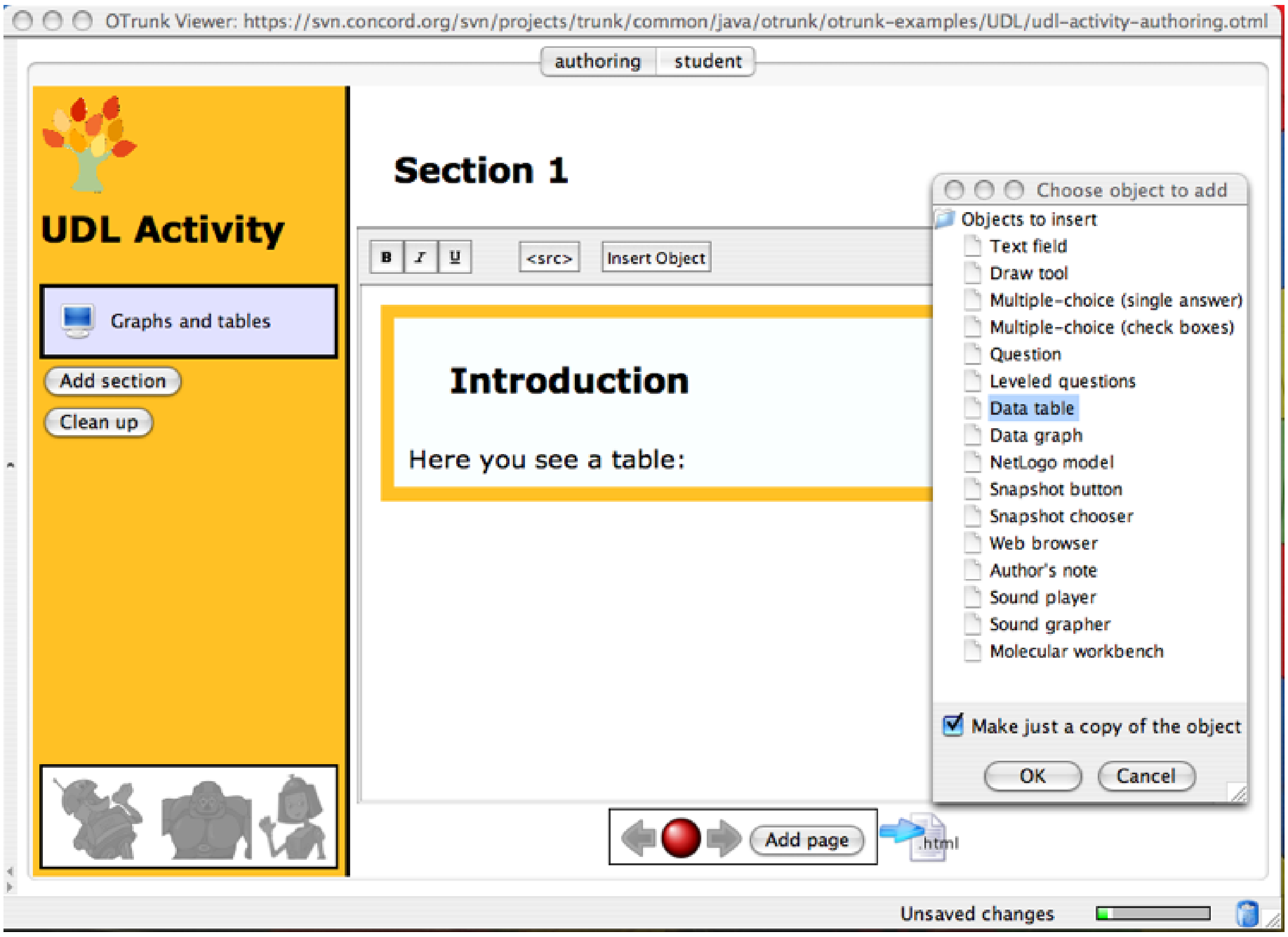
The basic document editor example can be run from OTrunk examples using this JNLP .

The examples below show document editing using the UDL authoring tool, which includes multiple pages and sections in addition to the document editor.

1. The author starts with a blank activity



2. Titles and text are added, and a new OTrunk table is embedded


OTrunk Viewer: https://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-examples/UDL/udl-activity-authoring.otml

Introduction

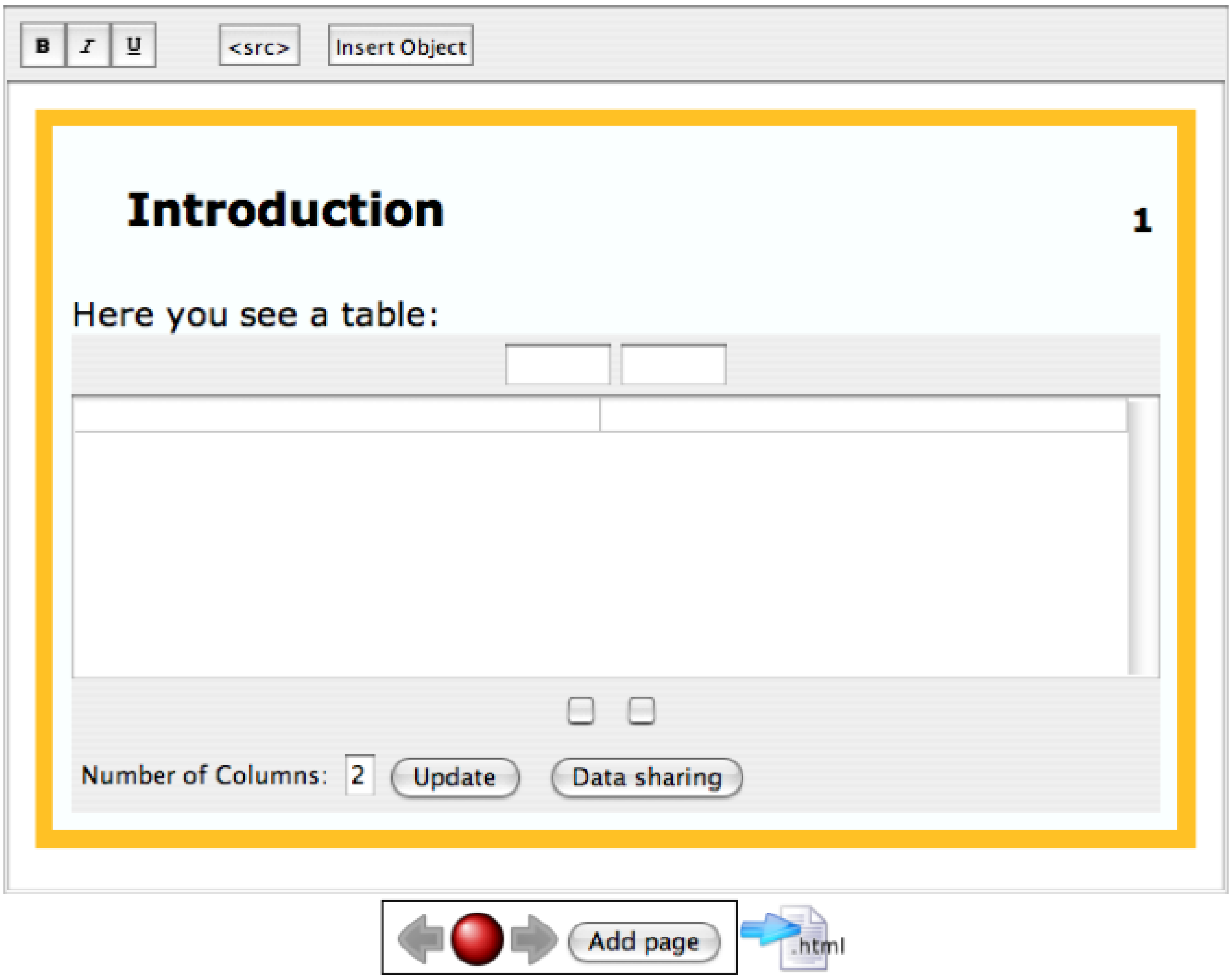
Here you see a table:

Choose object to add

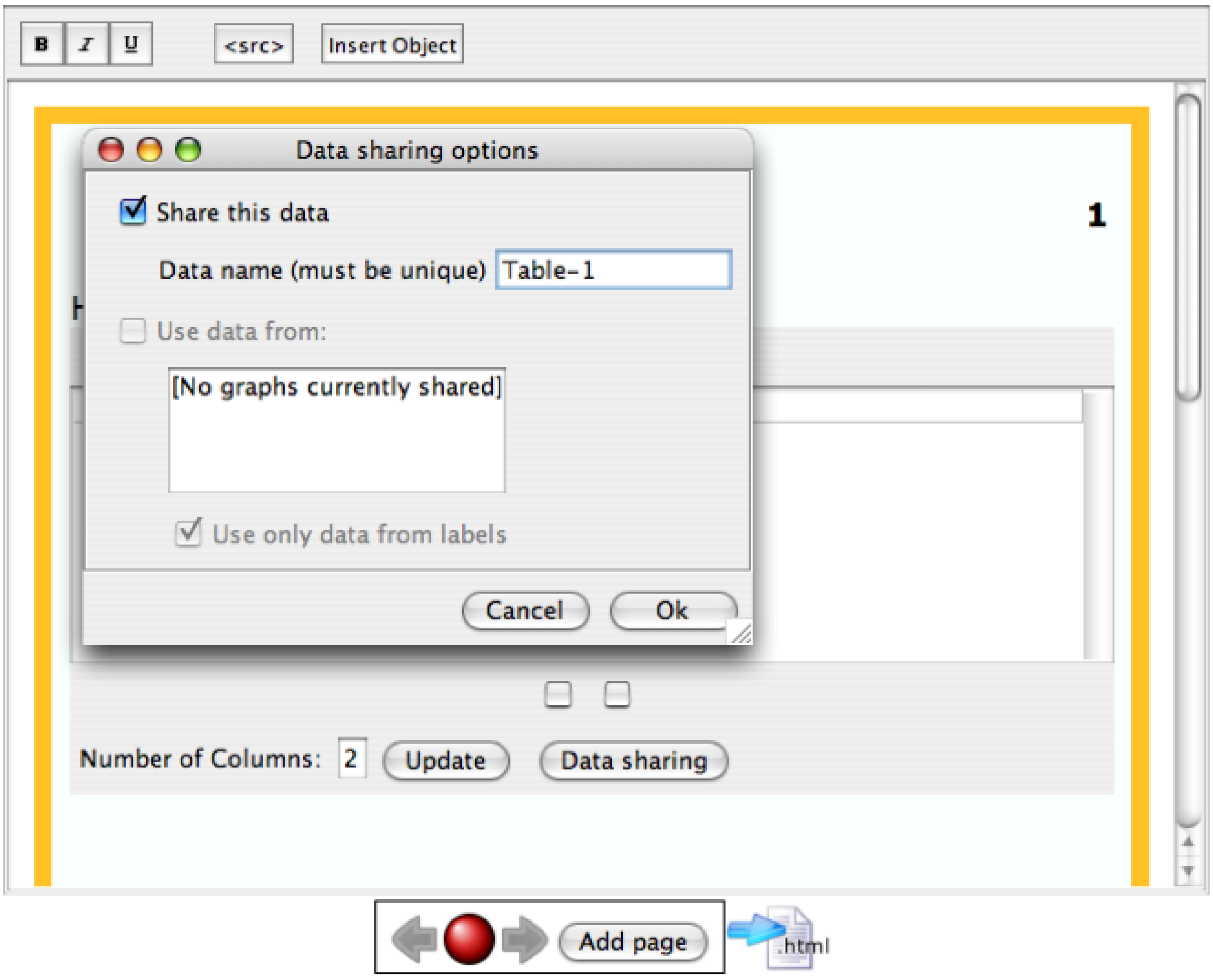
Objects to insert
- Text field
- Draw tool
- Multiple-choice (single answer)
- Multiple-choice (check boxes)
- Question
- Leveled questions
- Data table
- Data graph
- NetLogo model
- Snapshot button
- Snapshot chooser
- Web browser
- Author's note
- Sound player
- Sound grapher
- Molecular workbench

[✓] Make just a copy of the object

OK Cancel



3. The table is edited to enable it to be linked to a graph



Share this data [✓]

Data name (must be unique) Table-1

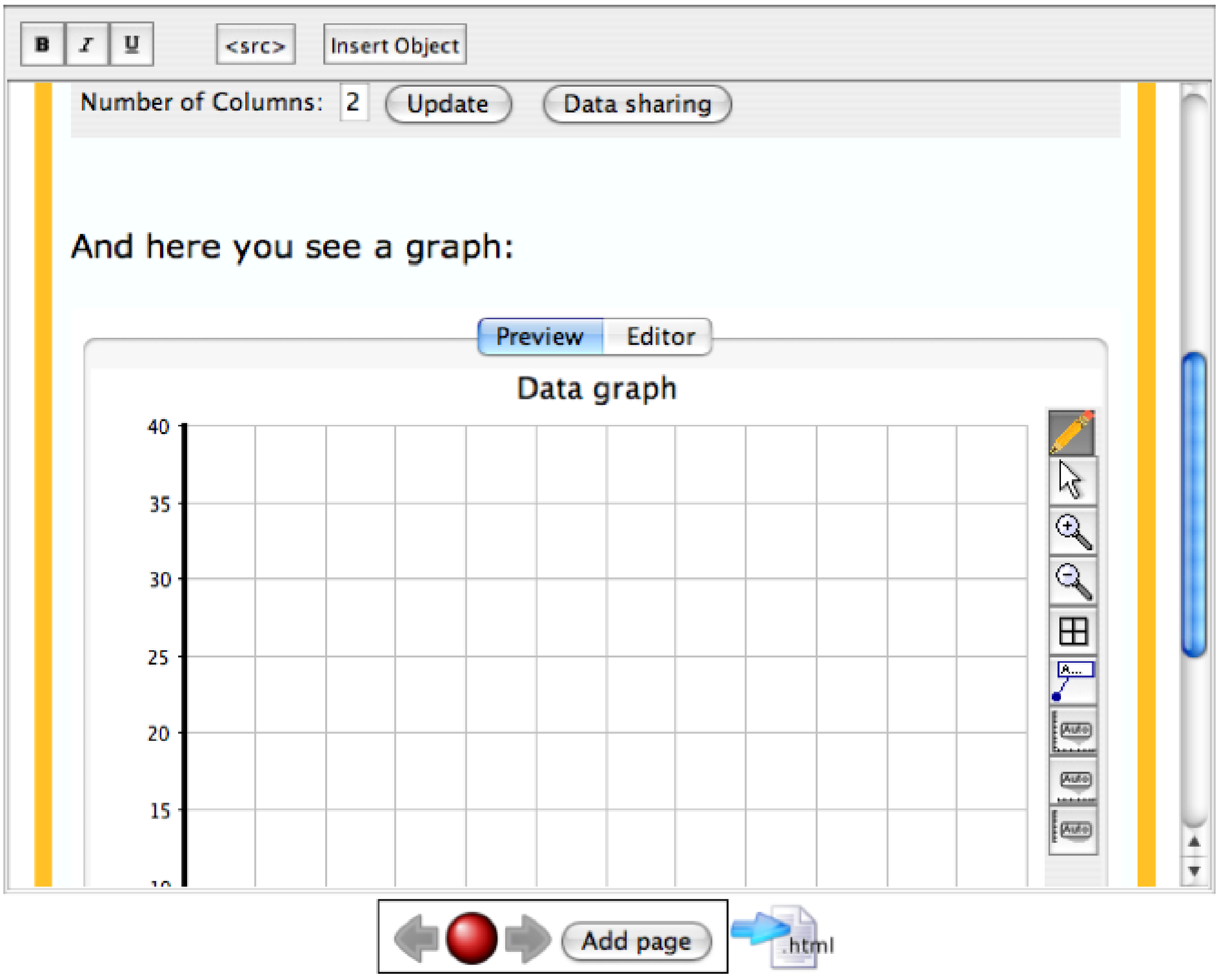
Use data from: [ ]

[No graphs currently shared]

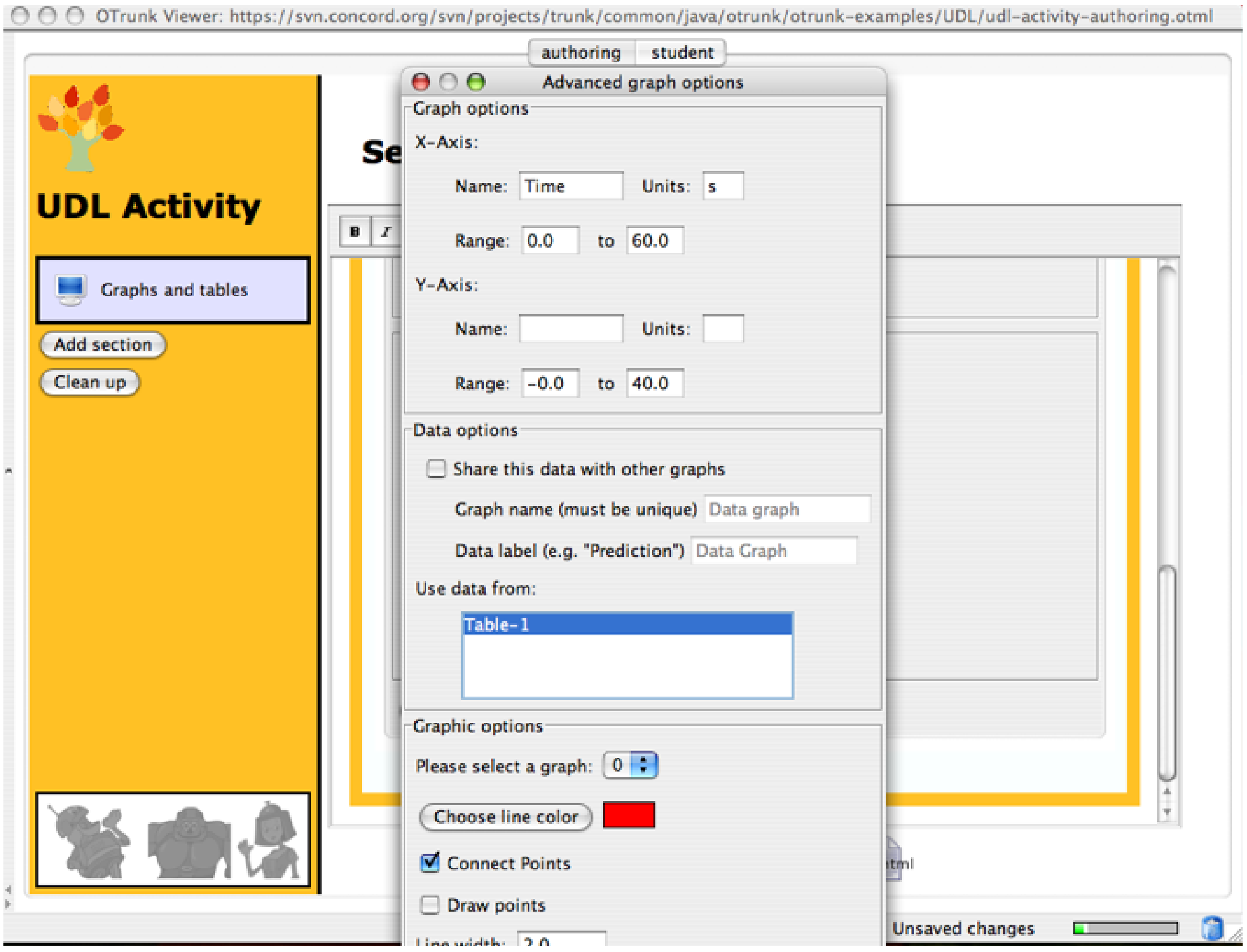
Use only data from labels [✓]

Number of Columns: 2

4. A graph is added below the table



5. Finally, the table is linked to the graph, by selecting the table from a list


OTrunk Viewer: https://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-examples/UDL/udl-activity-authoring.otml
authoring
student
UDL Activity
Graphs and tables
Add section
Clean up
Advanced graph options

Graph options

X-Axis:

Name: Time Units: s

Range: 0.0 to 60.0

Y-Axis:

Name: Units:

Range: -0.0 to 40.0

Data options

Share this data with other graphs

Graph name (must be unique) Data graph

Data label (e.g. "Prediction") Data Graph

Use data from:

Table-1

Graphic options

Please select a graph: 0

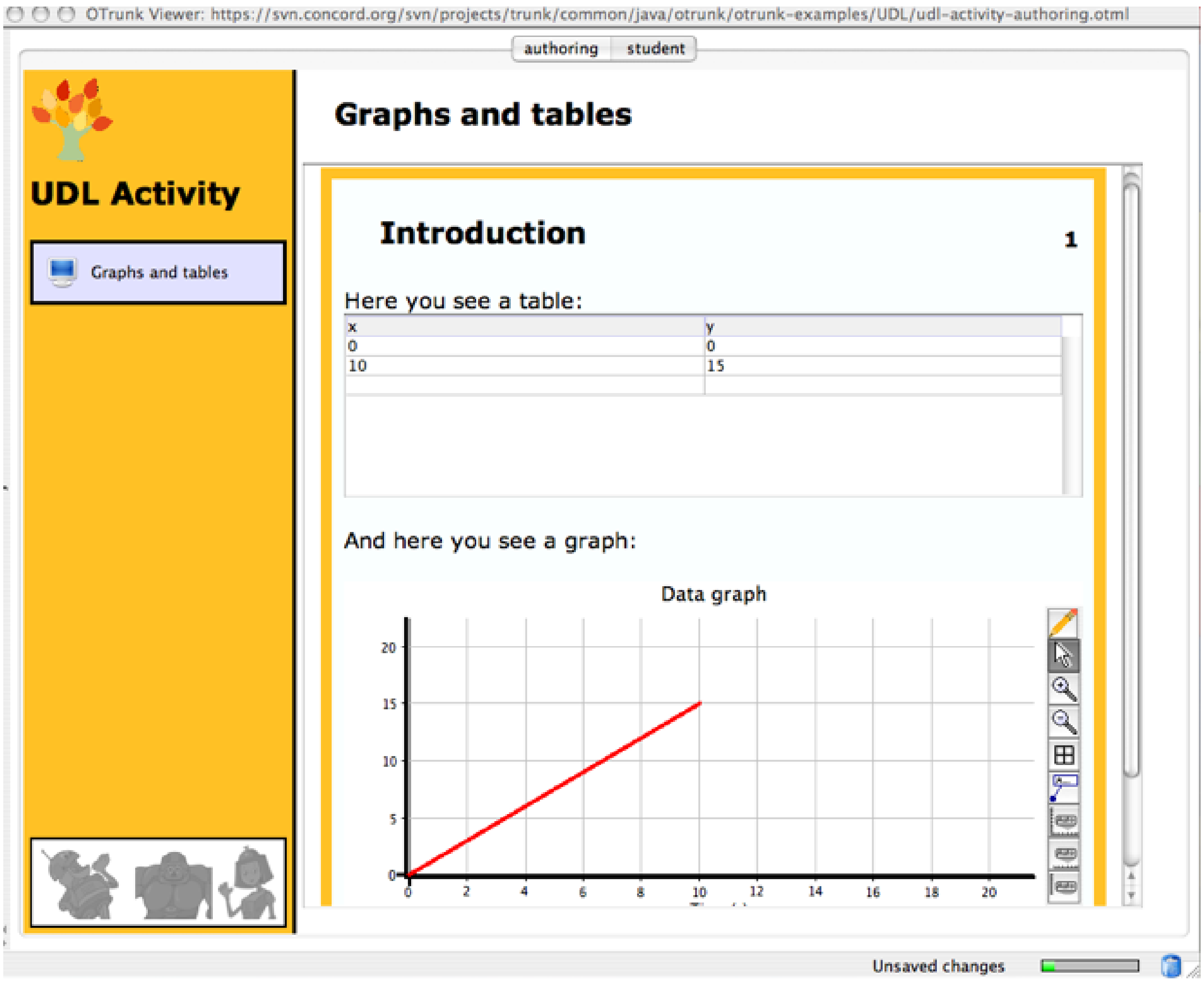
Choose line color 🟥

[✓] Connect Points

Draw points

Unsaved changes

6. Now we can preview the student's version of the activity, and see that the graph will indeed plot the values on the table



OTrunk Viewer: https://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-examples/UDL/udl-activity-authoring.otml

Here you see a table:

x	y
0	0
10	15

And here you see a graph:

See also

- OTrunk Authoring Views
- Authoring demo (JNLP)

OTrunk Archived

This page last changed on Apr 29, 2007 by scytacki.

- OTrunk Wrapped Object Analysis

OTrunk Wrapped Object Analysis

This page last changed on Apr 29, 2007 by scytacki.

This page does not describe the current code. It was a review of the code before it was refactored to the OTController design.

The idea of the these interfaces make it possible to have a generalized framework for managing wrapped objects.

createWrappedObject - is called to make the object being wrapped, a generic implementation of this interface could use
the wrappedObjectClass to create the object.

initWrappedObject - is called afterward on a second cycle through the objects. This is so it can track down any referenced objects
that are also wrapped.

registerWrappedObject

- this has been used to add listeners to the to the object
- additionally the wrapped object is added to the object service using: getOTObjectService().putWrapper(wrappedObject, this);
- all existing OTWrapperS call registerWrappedObject in createObject
- OTDataFlowing line calls putWrapper on the object service in this method, and starts the line moving
- Most OTWrappers add listners and call putWrapper in this method: OTDataGraphable, OTEraserGraphable, OTDrawingImageIcon, OTDrawingShape, OTPointTextLabel

createWrappedObject

- it is called in:
  - DataGraphManager.addGraphable - this is used to added a new graphable to the graph from a template OT object
    it copies the ot object and then calls createWrappedObject
  - DataGraphManager.initGraphables - this method sets up the graphables (wrapped objects) when the object is loaded
    from the state. each wrapped object is added to its reverse map
  - DataGraphManager.initLabels - same as above
  - DataGraphStateManager. initialize - same as above.
  - OTDrawingToolView.loadGraphable - given an ot object coming from the marshaled state this method creates the wrapped
    object and addes it to the reverse map
- the implementations all call registerWrappedObject
- most implementations initialize the wrapped object in this method: OTDataGraphable, OTEraserGraphable,
  OTDrawingImageIcon, OTDrawingShape, OTPointTextLabel
- only OTDataFlowingLine uses the initWrappedObject method to setup the object. This is because it needs to pull togther

other objects in its container.

initWrappedObject

- only OTDataFlowingLine uses the initWrappedObject method to setup the object. This is because it needs to pull togther
  other objects in its container.

saveObject

- called when the wrapped object changes. many times this is called by the wrapper itself when an event is
  fired by the wrapped object
- it is also called by the managers when lists change or get added.

Manager Objects

- Currently these classes act as a manager in one form or another:
  OTDrawingToolView, OTDataDrawingToolView, DataGraphManager, DataCollectorView and DataGraphStateManager

ObjectService.getWrapper - look up ot object (wrapper) from real object (wrapped)

- mostly called in listGraphableRemoved of the manager objects, this is so the ot object can be removed
  from ot list of the parent ot object.
- in saveObject OTDataPointLabel and OTDataPointRuler, the wrapper is looked up from associated graphable,
  and this wrapper object is added to the resources.

ObjectService.putWrapper

- only called in registerWrappedObject
- not called by every OTWrapper implementation, but most do

reverse maps - look up real object (wrapped) from ot object (wrapper)

- DataGraphManager, OTDrawingToolView
  maintain reverse maps so a graphable (wrapped object) can be looked up given an OT object (wrapper).
- DataGraphManager.otDataGraphableMap
  - provides method getDataGraphable to access the map
  - also provides getOTDataGraphable to access the map in reverse
    this should have been handled by the objectservice.getWrapper, but perhaps there was a problem with that
  - the map is used to look up the correct graphable for a label to linked to in initLabels,
    this should be moved to the initWrappedObject of the OTDataPointLabel.
  - OTDataPointLabel is setup the same way in initLabels it should also be fixed.
  - removeItem has to remove the graphable from the reverse map, this is a method of CheckedColorTreeModel

- OTDrawingToolView.graphableObjectMap
    - provides a method getWrappedObject so other wrappers can look up wrapped objects.

Keeping a reference to the wrapped object in wrapper?

- none of the existing implementations of OTWrapper keep a reference.

OTrunk Collections

This page last changed on May 08, 2008 by aperritano.

There are 4 types of collections in OTrunk:

OTResourceList, OTResourceMap, OTObjectList, OTObjectMap

In all cases you should not use a set method in your OTClass. You only add a get method. The list object will be created automatically when it is requested.

An example on how to create an OTResourceList:

1. Add a sample list to your OTML file:

1. Add your desired list to your OT interface

1. Next update its corresponding controller and process the values as you see fit

OTrunk Controllers

This page last changed on Apr 07, 2008 by sfentress.

Overview

OTrunk Controllers are used link OTrunk objects with Plain Old Java Objects (POJO). The are useful if you want to use OTrunk to configure and save data from java libraries which don't directly use OTrunk objects.

There are 3 objects involved when an controller is used. The OTObject, the OTController, and the "RealObject"(POJO).
A fourth object, OTPackage, is used in order to support plugging in new implementations of the RealObject.

Here is an example of how controllers can be used. There is a java library that supports graphing data. To supply data to the graph a DataProducer interface is implemented and the implementation is passed to the graph. For example lets say we want to provide a DataProducer that always supplies the same value: SingleValueDataProducer.

Create a OTClass

Make a new java interface which extends OTObjectInterface
Add get and set methods for the properties of the new OTClass

For example:

Create an OTController

Make a new java class which extends OTController
For example:

For each controller you need to do the following things:

Declare classes

Define the following two public static fields:

Replace MyObject.class with the class this controller is working with.
Replace OTMyObject.class with the OTClass interface created above.

Implement loading code

Implement the "loadRealObject" method. It should get properties from the otObject field, and set them on
the realObject.

Register Controller

There needs to be a OTPackage in the same java package as the OTObject. The name of the OTPackage class
is important it has to follow a convention. OTrunk Package Naming

In the initialize method add a call to register your new controller class:
for example:

OTrunk Package Naming

This page last changed on May 22, 2008 by scytacki.

The name of the package class currently has to follow a convention. This classname is based on the java package of the OTObject that it if for. Currently packages are mainly used for registering controllers. The reason for this approach is:

- java does not provide a standard way to find all the classes in a package, so you cannot just search for a OT*Package file inside of the package.
- using the same name for class in different packages is confusing for developer when they have to work with more than one.
  This is why all OT*Package classes are not called OTPackage.

If you are having problems getting your package recognized you can use the following system property:

if this is true OTrunk will print the name of each OT*Package it looks for.

If for example the className is com.example.datagraph.state.OTDataCollector
It looks for a class called: com.example.datagraph.state.OTDatagraphPackage

This is figured out by

1. Taking off the classname
   com.example.datagraph.state
2. striping off the .state (if there is one)
   com.example.datagraph
3. If the packages starts with "org.concord" then it is handle specially, otherwise:
4. taking the last element of the package name
   datagraph
5. capitalizing the first leter and adding OT to the front and package to back
   OTDatagraphPackage
6. use the original package of the imported class
   com.example.datagraph.state.OTDatagraphPackage

If the package starts with "org.concord" then there is special handling for more nested java packages.

For example: org.concord.graph.util.state.OTPoint2D

1. Steps 1 and 2 above are followed.
2. org.concord is stripped off
   graph.util
3. each dot separated string is capitalized.
   GraphUtil
4. OT is added to the front and Package to the back
   OTGraphUtilPackage
5. the original package of pre-pended:
   org.concord.graph.util.state.OTGraphUtilPackage
   Then org.concord

Future Notes

This approach for finding packages is not ideal. In the future, packages will be imported in otml files instead of classes. And each package will list its OTClasses. So the need for this approach will go away.

OTrunk Curnit

This page last changed on Apr 29, 2007 by scytacki.

An OTrunk curnit is an archive that can be "run" by the SDS. There are two attached to this page:

	Name	Size	Creator	Date	Comment	
	otrunk-curnit-external-otml.jar	1 kb	Scott Cytacki	Jan 24, 2007	external otml file	Edit | Remove
	otrunk-curnit-included-otml.jar	2 kb	Scott Cytacki	Jan 24, 2007	included otml file	Edit | Remove
	otrunk-curnit-untangled.jar	2 kb	Scott Cytacki	Feb 02, 2007	otrunk curnit for untangled sail-otrunk project	Edit | Remove
	otrunk-curnit-external-diytest.jar	1 kb	Stephen Bannasch	Feb 02, 2007		Edit | Remove

There are two ways to structure the data in an OTrunk curnit. Either the otml file can be an external link the curnit references, or the otml file can be embedded in the curnit.

Basic curnit structure

The curnit is a zip file. It contains xml files and properties files and sometimes nested zip files.

curnit.xml

It looks like this:

This file determines the starting *pod* of the curnit, the title of the curnit, and the id of the curnit.

- The title of the curnit will probably be ignored.
- the starting pod (rootPodId) should match the name of the pod file also in the archive.
- the curnitId should be unique for each curnit created. You can use uuid services to make this id or most likely you can find a library for the languague you are using.

POD_*.xml

This file setups up the OTrunk parts of the curnit. The entire file looks like:

Much of this file is boilerplate, so you don't need to mess with it.

Further details on elements inside the POD.xml file:

- **podId**

needs to match the file name of the file, and the rootPodId in the curnit.xml file.

- **authoredDataURL**
  - Specifying the otml with an URL pointing to an external resource:
-
  - Specifying the otml (and any related resources) with an URL pointing to an internal jar archive contained within the curnit archive:

    There needs to be more files in the curnit to work, see below.

- **hideTree**

podsReferenced.properties

If the otml is external this file needs to exist but can be empty.

If the otml is stored inside the curnit, this file will look like this:

This is a properties file so the key is everything before the '='.
The key is the id of the pod.
The value is the name of the file inside the zip file. When these curnits are created automatically these file names are cryptic urls. But the name you use can be any thing you want.
If the otml is stored inside the curnit then this file is required to be setup. curnit is embedded in the file

withincurnit/tmp/podarchive54815.jar

This file name can be anything, as long as that name is entered in the podsReferenced.properties file.
The file itself is a zip file. Inside of it should the otml file, the name of this otml is referenced in the POD_*.xml file. In this example that reference is:

So the name of the otml file needs to be "document-textbox.otml".

OTrunk Examples

This page last changed on Oct 18, 2007 by scytacki.

Applets

http://teemss2.concord.org/demo/applets/ - these are some applets based on older jars

OTrunk and SAIL examples:

The following OTrunk examples all use the Sail Data Service (SDS) to manage the learner data persistence.

- OTrunk All Test
  This example shows many of the Concord components in a simple OTrunk container (saving data back to the SDS is disabled).

- OTrunk examples generated on the CC's Contunuum automated build-server
  There are more than 25 different OTrunk examples here that are run using the SDS. The only ones that don't yet work are the Vehicle/ The only ones that don't yet work are the Vehicle/ examples.

The next two OTrunk-SAIL examples are hard-coded to useexamples. the otml generated by two activities Carolyn Staudt created on the TEEMSS2 DIY site. In the examples below persistence of learner data is enabled. If you run one of them more than once changes you made previously will be restored and visible. This new SAIL-based learner data persistence is being added to the development TEEMSS2 DIY site. When reliable the new functionality will be migrated to the public TEEMS DIY site.

- Greenhouse Effect
- Mixing Different Temperature Water

Understanding how the OTrunk and SAIL examples are started:

Here's the actual url used to start the *OTrunk All Test* example:

http://rails.dev.concord.org/sds/2/offering/144/jnlp/540/view?sailotrunk.otmlurl=http://continuum.concord.org/otrun

There are three parts to that url:

1.

   This request is being made to SDS Portal realm 2: *OTrunk Testing*.

2.

   Tells the SDS to return the jnlp for Offering 144 and one of the Workgroups associated with that Offering (*Workgroup 540*) and to disable saving of data back to the SDS. You can use this url to see the Offering in the SDS: http://rails.dev.concord.org/sds/2/offering/144.

3.

    These are extra parameters passed to the SDS when requesting a jnlp and are made available as properties in the jnlp webstart application. These extra parameters are required to run this OTrunk example correctly however they are optional when requesting the SDS to generate a jnlp. The SDS includes these attributes as properties in the config fle it generates as part of the SAIL application startup process.
    In this case there are two extra parameters:

    - sailotrunk.otmlurl
      This is set to http://continuum.concord.org/otrunk/examples/Everything/everything.otml.
      When set the OTrunk appliation will ignore the otml embedded in the curnit and instead use the otml supplied by this url.
    - sailotrunk.hidetree
      This is set to false so the object tree navigation panel on the left will be visible.

References:

- OTrunk
- OTrunk Curnit
- REST protocol for SAIL Data Services (SDS): Adding additional portal-specific config parameters

Updating OTrunk Examples

This page last changed on Apr 29, 2007 by scytacki.

These are the steps to update the jars and otml files on the following page:
http://continuum.concord.org/otrunk/examples/example-index.html

1. Go to http://continuum.concord.org
2. make sure the projects (jars) you've updated have been built successfully
    - the projects are built once an hour, so if you've checked in your changes a while ago they should be built.
3. you need to build the MavenJnlp project which updates this site: http://jnlp.concord.org/jnlp/
4. then finally you have to build the OTrunkExamples project which updates this page: http://continuum.concord.org/otrunk/examples/example-index.html

Notes

You might not need to do all of these steps each time. If you only changed the jar files, you just have to do the first 3 steps. If you only changed the otml file then you only have to do the last step.

There is currently a delay from when you check in a change to CVS before it is available to continuum. This delay is between 1 and 5 mins. Sometimes it can be longer. Each build in continuum tells you the files that have changed, so you can see if your changes have been picked up by continuum.

Freezing Jars in an otrunk-examples folder

This page last changed on Oct 17, 2007 by scytacki.

Look at the bottom of the page for the folder on the website. For example:
http://continuum.concord.org/otrunk/examples/CAPA/ot-index.html

It has some text:

```
The jnlp urls were constructed using the following template:
http://rails.dev.concord.org/sds/2/offering/15941/jnlp/14902/view?sailotrunk.otmlurl=%otml_url%&sailotrunk.h
You can change this string by putting it in a file named: jnlp_url.tmpl in this directory
```

If you go to the first part of the url:
http://rails.dev.concord.org/sds/2/offering/15941

Then you can see which jnlp is being used for that folder. You do this by clicking on the link after the "Jnlp:" at the top of the page.

If you want to change the jnlp being used:

- go back to the offering page:
  http://rails.dev.concord.org/sds/2/offering/15941
- click the edit link
- select a different jnlp. If there isn't a jnlp for what you want then you need to make one.

By default all the folders in otrunk-examples use the same offering so if you change the jnlp for that common offering it will change it for every folder in otrunk-examples. See below for information about
creating a new offering and making the folder in otrunk-examples use that new offering.

Updating DIY JNLP

This page last changed on Jun 27, 2008 by scytacki.

Find the jnlp resource of the diy you want to update.

Go to
http://rails.dev.concord.org/application-info/
Find the diy url in the first column of the table.
Find the SDS URL and SDS JNLP in the last column of the table
Combine them like this:
SDS URL/jnlp/*SDS JNLP*
If you go to that address you'll see a web page for the jnlp resource for the diy.

Here is an example url:
http://saildataservice.concord.org/8/jnlp/273
That is the jnlp resource for the ccdiy

Figure out the new url for the jnlp

On the jnlp resource web page is a Url: for the actual jnlp file.
From the example above this url is:
http://jnlp.concord.org/dev/org/concord/maven-jnlp/all-otrunk-snapshot/all-otrunk-snapshot.jnlp

To see other available versions of that jnlp look at the directory that jnlp file is in.
So from the example above look at:
http://jnlp.concord.org/dev/org/concord/maven-jnlp/all-otrunk-snapshot/

All jnlp that end with a timestamps do not change. For example:
all-otrunk-snapshot-0.1.0-20080111.153223.jnlp

These are in effect frozen. jnlps that don't end in a timestamp will get updated each time the jnlp project is build on continuum. For example:
all-otrunk-snapshot.jnlp

Update the jnlp resource

Back on the jnlp resource page of the sds, click the edit link.
Change the url field in form an click save.

Updating just a few jars in a jnlp

The only way to do this right now is to make a custom jnlp. Here is how it is done:
ssh maven@jnlp.concord.org
cd /var/tomcat/jnlp_tomcat/webapps/dev/org/concord/maven-jnlp/capa-otrunk

```
cp capa-otrunk-0.1.0-20080215.201427.jnlp capa-otrunk-0.1.0-20080215.201427-custom.jnlp
```

edit capa-otrunk-0.1.0-20080215.201427-custom.jnlp so it has a reference to the version of the jars needed.
update the sds jnlp like you do to freeze the jar so it points to the -custom.jnlp url.

OTrunk Goals

This page last changed on Apr 29, 2007 by scytacki.

Intro by Scott

OTrunk is an instance of a type of framework I've been try to create for sometime now. I believe this type of framework can be widely useful. And it seems that other people are creating parts of this framework, but I haven't yet seen a fully integrated one like OTrunk is trying to become.

The goals below are the goals that OTrunk is trying to achieve, they can be seen as a set of characteristics of this type of framework. Hopefully, these characteristics can be broken in to smaller self contained pieces. There are already frameworks that can be used for some of this functionality.

Goals

Object Level

- Create new object types or classes
  - there is little reduntant information in the type definition
  - it can be done without programming
  - the types can be created in multiple ways
  - a collection of object types are analagous to a database table schema or a xml schema

- Load and Save objects using xml
  - this way the objects can configured by programs and the xml can be maually edited.
  - also the xml can be edited using xml editors.
  - the xml can be edited by loading the objects using their views to modify them and then save them again.

- Use these objects in programming languages
  - Java is the primary language
  - Scripting languages are next: Ruby, Javascript, PHP and Python are at the top of the list

- Import or convert the object types and object instances into standard systems
  - xml schema
  - relational databases
  - XPCOM components
  - object databases

- Represent modifications to objects as differences
  - a form is a set of objects, when the form is filled the original form objects are used as "prototypes" of new objects. Those new objects are the ones actually modified when the form is filled it.
  - support multiple modified objects at the same time. For example a report of multiple filled out forms needs to have all the modified objects available.

- Support mapping object types to existing classes in thirdparty libraries
    - For example you find a cool widget that you want to embed in your document, it should be easy to make a new object type that can be used to configure that widget, and by updated when the widge updates.
    - the existing classes can be seen as a kind of view of the object type

- Support services for objects defined by other objects
    - this allows multiple objects to share resources and configuration
      without modification to the underlying framework
    - this might be movable to the view layer

View Level

- Create multiple views for the object types
    - different views can be used at the same time: author, runtime, and report views

- Create views using different renderers
    - report views using Java Swing objects
    - support for views using XHTML/CSS

- Nest views inside of views
    - For example a document view can have an embedded graph view inside of it

- Create views out of existing objects and views
    - this a allows making a new view out of existing views.
    - it is useful for new object types that can be rendered using existing views.

Existing Frameworks

EMF

EMF the eciplse modeling framework supports many of the goals in the Object level.

Spring

supports configuring objects with xml as well as passing them services, but it doesn't support saving the modifications of the objects back to the xml.

Modello

Is used by maven it also configures objects with xml.

This page last changed on May 14, 2008 by stepheneb.

OTrunk Developers FAQ

How to...

Hiding left side view in OTViewer
Q. The OTViewer always displays that left side panel with a tree view, but I don't want to show it. How do I take it out?
A. There are a few ways to do that. If you just want to temporarily take it out when you are running your application from Eclipse, you can set the otrunk.view.hide_tree property to true on the Eclipse launcher. For more information, look at OTrunk Viewer Properties.
If you want to get rid of the left panel altogether, you can set that in your otml file. There is a property for that on the OTViewBundle object (the one that contains all your view entries). Just set it to false, like this:

```
```

For more information on how to make your own otml file, look at Creating otml files.

Creating new OT objects from code
Q. I need to create a new OT object (instance) from scratch from my code. The keyword *new* doesn't seem to work (specially when the ot object is just an interface). How do I do that?
A. Use OTObjectService to create a new ot object, using the method createObject() and passing the class you need. OTObjectService also provides other services like making copies of ot objects. In order to get an instance of OTObjectService, you can use an existing OTObject an then call the getOTObjectService() method on it. If you are a writing a script, you can use the variable "otObjectService" directly from your code.

Listening to OT changes
Q. How do I add an OTChangeListener to an OTObject
A. You can add change listeners OTChangeNotifying objects. Most OTObjects implement OTChangeNotifying. If they do not, ask their author why and if they can fix it.

Getting real objects
Q. How do I get a "real object" out of an OT object?
A. You need an instance of OTControllerService (see next question). Then you can call the method getRealObject() on it.

Getting a controller service
Q. How do I get an OTControllerService object?
A. There are multiple ways:

- If you are in a script, you can just use the variable "controllerService".
- If you are a in a OTJComponentView and you are extending from AbstractOTJComponentView, then you can use the method createControllerService() to obtain an instance of a controller service that will work with your view context.
- If you are in an OTView that extends AbstractOTView, you can use this snippet of code:

```
```

- If you are in an OTView that doesn't extend AbstractOTView, you should extends OTViewContextAware and use this snippet of code:
- If you are in an OTController and you are extending DefaultOTController, you can use the protected field "controllerService", which is set in the initialize method.
- If you are not in an OTView or an OTController you can use an instance of OTObjectService and then call the method createControllerService() on it. You can obtain an OTObjectService object out of your OTObject, calling the method getOTObjectService(). This approach should be used as a last option.

Sharing a real objects between views
Q. I want to share a single real object between 2 views how can I do this?
A. The OTController which is responsible for that real object needs to return *true" for the isRealObjectSharable(OTObject otObject, Object realObject) method. If it does so and both views have a common top level container, then the real object will be shared. The current top level containers are: OTViewer, OTUDLSidebarView, new frames created by the OTFrameManager

OT object IDs
Q. When I reference an ot object, I do <object refid="${local_id}"/>. What is that dollar sign for?
A. The dollar sign and the curly brackets mean that the ID is a local id. If you are using a global id, you don't put the $ or the {}.

OT object IDs - Refer to an object by a local_id instead of its existing global id
Q. I want to reference an object or view by a local_id, but it already has/needs to have a global id. Can I do that?
A. Yes, you can use an idMap, which maps global ids to local_ids. At the same level as the <imports> in the otml file, add

Now wherever you would use <object refid="23cc14f0-c44f-11dc-95ff-0800200c9a66"/> you can use <object refid="${my-id-1}"/>

Getting views from OT objects
Q. I have an ot object, but I need an instance of its view. How do I get that?
A. You need a JComponentViewContext object, and this context has to be in the same context as the view you want to get.
If you are a view and you are extending AbstractOTJComponentView, then you can use this snippet of code:

If you are in a script, you need to give access to the view to the OTScriptObject in the otml file. To do that, add an OTScriptVariableView entry to the variables section of the OTScriptObject. Then, you can use the view as a normal variable from your script.

Writing otml files - Using maps
Q. I want to use a map but I don't know how to write the otml for that.
An OTObjectMap allows you to have a table with strings as keys and OTObjects as values. This is a sample code. Say there is a class called OTExample with an OTObjectMap property called mapProperty. Then the otml would look something like this:

An OTResourceMap allows you to have a table with strings as keys and primitives as values. This is a sample code. Say there is a class called OTExample with an OTResourceMap property called mapProperty. Then the otml would look something like this:

Writing otml files - Including external otml files
Q. I want to reference an object that is defined on a separate otml file that the one I'm writing. Can I do that?
A. Yes, you can include the external otml file using an OTInclude object in the **includes** subsection of the OTSystem section on your otml file. Then you can reference the object normally, using its **global id**.

Writing otml files - Directly use the root object from an external otml file
Q. I want to directly reference an external without having to first import it and then refer to the id of the object. Can I do that?
A. Yes, there is a short-cut to the above how-to which directly references the root object of an external otml file.

Use:

where you would use

with the viewClass org.concord.otrunk.OTIncludeRootObjectView

Viewing and editing learner data directly
Q. I want to edit a student's otml data by hand and post it to the SDS as a new bundle.
A. First you need to find the student's data in the SDS. To do this, you must navigate to the SDS offering associated with a specific activity offered by a DIY. e.g. http://saildataservice.concord.org/13/offering/7086. Then you copy the url referenced by the Run link next to the student's name, e.g. http://saildataservice.concord.org/13/offering/7086/jnlp/19407. Then you append the session property "sailotrunk.learnerotml.edit=true" to that url:

```
http://saildataservice.concord.org/13/offering/7086/jnlp/19407?sailotrunk.learnerotml.edit=true
```

This will launch the SailOTViewer and show you a text representation of the learner's otml. You can then edit this otml by hand, and upload it as a new bundle by closing the window (you will be asked to confirm upload). Note that no xml checking is done at this point, so there are no guarantees on what will happen if the xml is invalid.

Common error messages and solutions

Errors when loading an otml file
Q. I'm getting an error that says can't find handler for: .... What does that mean?
A. It means that it cannot find the class you are referring to. If it is an OTObject, you need to add it to the imports section of your otml file.

Errors when loading an otml file

Q. I'm importing a class in the imports section, but I'm still getting an error that says Error importing class: ... this class was listed as an import in the otml file. I checked the spelling of the package and the class and looks right. What's wrong?
A. All the classes that you put in the imports section have to be OTObject. If the class you want to import is indeed an OTObject, then this error probably means that the class needed is not in the classpath. The solution depends how are you running the otml file. If you are using a launcher from Eclipse, you just have to add the appropriate project or jar file to the launcher. If you are launching from otrunk-examples, make sure the build system has the latest jar files and that the project has the correct dependencies.
If the class you are trying to import is not an OTObject, then you don't need to import it. For example, if it's an OTView, you should not import it, but just reference it from the otml file within a view entry.

Errors while loading a realObject
Q. I'm trying to load in a real Object and I get the following error Can't find a controller for this otObject: ....
A. The controller needs to be properly registered. Follow the instructions here: Registering a Controller If you are having trouble with the packages, you can set the system property: otrunk.trace.packages=true

OTrunk Scripting
Q. I followed all the steps for creating an otml file with a script on it (JavaScript), but I'm getting a bunch of errors when I try to run it. Here are some of the errors I've gotten:

```
Exception in thread "AWT-EventQueue-0" java.lang.NoClassDefFoundError:
org/apache/bsf/BSFManager
Exception in thread "AWT-EventQueue-0" java.lang.NoClassDefFoundError:
org/apache/commons/logging/LogFactory at org.apache.bsf.BSFManager.<init>
org.apache.bsf.BSFManager loadScriptingEngine SEVERE: Exception:
java.lang.NoClassDefFoundError: org/mozilla/javascript/Scriptable
unable to load language: javascript-db: java.lang.NoClassDefFoundError:
org/mozilla/javascript/Scriptable
java.lang.RuntimeException: org.apache.bsf.BSFException: unable to load language: javascript-db
```

A. You need to add the necessary jar files for the JavaScript engine. The jar files are located in the thirdparty project in cvs. For JavaScript, the files needed are: bsf-2.4.0.jar, js-1.6R5.jar and commons-logging.jar. Add these to your classpath and it should solve the problem.

OTrunk ID System

This page last changed on Jul 03, 2007 by scytacki.

OTrunk uses ids to look up and store references to objects. The root class of this system is an empty interface called OTID. This interface is implemented by:
OTPathID, OTRelativeID, and OTUUID.

A user of the OTrunk system shouldn't have to worry about these different types of IDs, but there are a few places where they show up.
The current xml representation of OTrunk otml, uses something called a local_id which is the most widely used way to reference objects. The original intention was for local_id to only be valid within a particualar otml file. And global_ids all be UUIDs and to be valid across multiple files. However as the system has grown this has been blurred a bit.

In the current system most people have used local_ids in their otml files. Most of these ids are handled when the document is loaded. They are replaced with: (otml UUID)!/(local_id). So for example a object with a local_id like this:

While get converted and when it is saved again it will look like this:

The UUID was taken from the otrunk element of the initial document.

local_ids are referenced again using a syntax:

```
${(local_id)}
```

OTUUID - provides a OTID implementation that uses a UUID for the id of the object. Currently when a new OTObject instance is created in the Java code a new OTUUID is created for that object. OTUUID's are strongly encouraged for each otml file which is created, this id is specified in the otrunk element at the root of the document. Something like this:

OTRelativeID - provides a way to reference an id relative to another id. This class has a rootId and a relativeId. It is typically used to store the local_id style ids above. So the local_id is relative to the root id of the otml file. When this id is written out the "!" character is placed between the root and relative ids.

OTPathID - this id is used by the current xml representation to store an id based on the path to the object in the document. So if you create an OTObject in a otml file which doesn't have a local or global id then a path id is used. The path id is based on the location of this OTObject in the otml file. An example of a PathID is:

The path is divided by the "/" characters. This id illustrates several parts of the path ids. The first element is an object with an id attribute set to a UUID. The next element says this points to the first service in the services feature/property of the first element. The next points to the second mode in the modes feature/property of the service. The next points to the map entry which has the key:

```
${compound_doc_view}
```

Adding a new OTDatabase with a different id system

A new OTDatabase might work best with a different ID design.
Currently the main hinderence to this is the method:
OTIDFactory.createOTID(String idStr)

The problem with this is that it means the OTIDFactory needs to know about this new ID system. That method has been hidden behind 2 layers of abstraction, so it should be possible to work with those layers to support multiple ID system. However the OTIDFactory.createOTID method is still called in 11 places, however all of those places are internal to the OTrunk system so they are under our control.

The first abstracted way of getting an id is to use the OTrunk method getOTID(String id).
That method is used only in 3 places:

- the object service impl which is the next abstraction layer
- 2 times in the sync4j code which is not used anymore.
  The sync4j code can be hacked to fit whatever we come up with.

The second abstraction is the OTObjectService method getOTID(String id). This is used in 4 places:

- 2 in the DefaultOTObject code to implement the methods getReferencedId and getReferencedObject
- 1 in the OTrunkUtil method getObjectFromMapWithIdKeys, this is a helper when you need to create maps that have keys were are the ids of objects.
- 1 in AbstractOTDocumentView to implement getReferencedObject

So the issue that needs to be dealt with is how the OT system can identify which data storage library should convert an id string to an actually OTID. It might work to say that an OTObjectService has to be associated with a particular data storage library. The OTObjectService already had 2 databases the creationDb and the mainDb. Each of these could be a different type of database. It seems the safest approach would be to require that the call of the method needs to also pass in the OTObject that is making this string based id reference. Then the database of that object can be looked up and it can be used to create the id. Even this could still be a problem in the case of overlayed object, like the user object. The overlaying object might be in one type of the database and the underlying object in another type. In this case simplying having the OTObject is not enough. The system would need to know the property of the object, to see which database that property comes from for this particular object.

Another alternative is to remove ids entirely from being used as strings within properties. There are 2 cases that need to be dealt with here:

- documents with references and links
  - this can be handled by turning all the references into indexed references from a list inside of the document. For backward compatibility this can be done
    when the document is being loaded. And if it becomes a pain to uses lists like these then the objects can be remapped to ids automatically as they are written out. This approach would probably work the best if the way the ids were referenced in the document was with a string not an index. This will make it easier to maintain. But this will make referencing an object harder in a document. If the document loader and writer could do it automatically that would be the best, but that makes it more difficult to write crossplatform libraries for processing this

    stuff. If someone adds a new type that does this type of thing, all of the libraries for processing it will have to be updated to handle that type.
    - perhaps an approach is to separate the authored format from the portable format. When authoring you use the easy to write approach, and then before commiting it to a repository, you convert it to the portable format so it can be processed.
    - another alternative is to have a way for the portable schema to specify a pattern for finding references to other objects. I think that is probably not a good approach.
    - The best seems to be the object map with an automatic conversion process for now.
- maps which use the id as a key.
    - the maps can be special cases so there is a special type of mape that works with objects instead of strings.

If we do all of these changes then it will simplify the copying code, and database implementations. The OTID string issue will go away.
OTrunk Plan to remove id strings

The tough case here is when copying from one data store to another. Currently this type of copying happens below the level of the OTObjectService.

OTrunk Plan to remove id strings

This page last changed on Jul 09, 2007 by scytacki.

ID strings are used in a few places in the OTrunk code. These cause problems when adding new database types, and when copying objects. However using indrect references can be pretty annoying when writting otml directly.

It is possible to have the data layer automatically convert id strings. To do this it would need to know the pattern to look for these strings. and then when it finds them, it would need to add those references to standard map which a view of the object could later use to look up the actual object. If a portible version of the otml it should be processed and rewritten out.

From a scripting point of view it makes sense for references to objects in the document have local string tags. Then these can be used so the resulting components can be referenced by those tags. Otherwise the script has to lookup the components using object lookups which are tedious, hard to maintain, and limit the number of views of a single object to one.

The ideal solution would be to have the text of the document be a first class citizen of the OT system. Then elements in the document can be individually changed and those updates will be saved, as well as the abillity to embed objects directly in the text of the document. This is currently not possible with the otml we are using. But this should be possible with the move to emf.

So the plan is that we modify the existing otml parser to automatically convert references to objects in documents to use an indirect references. This will break the reporting tool. But it can be fixed. The otml written out will then have these modified references, and copying will work without special casing documents.

Then with the emf code we will attempt to add full support for xhtml inside of the emf document. Then in this case objects can be embedded directly inside of the document text. And references will be real references.

OTrunk Introduction

This page last changed on Apr 29, 2008 by scytacki.

OTrunk is a framework for configuring and running a set of components. While running it provides a way save changes made to those component configurations. Additionally, it provides mechanisms for the components:

- registering services
- giving the components access to services
- multiple mechanisms to connect components to views through controllers.

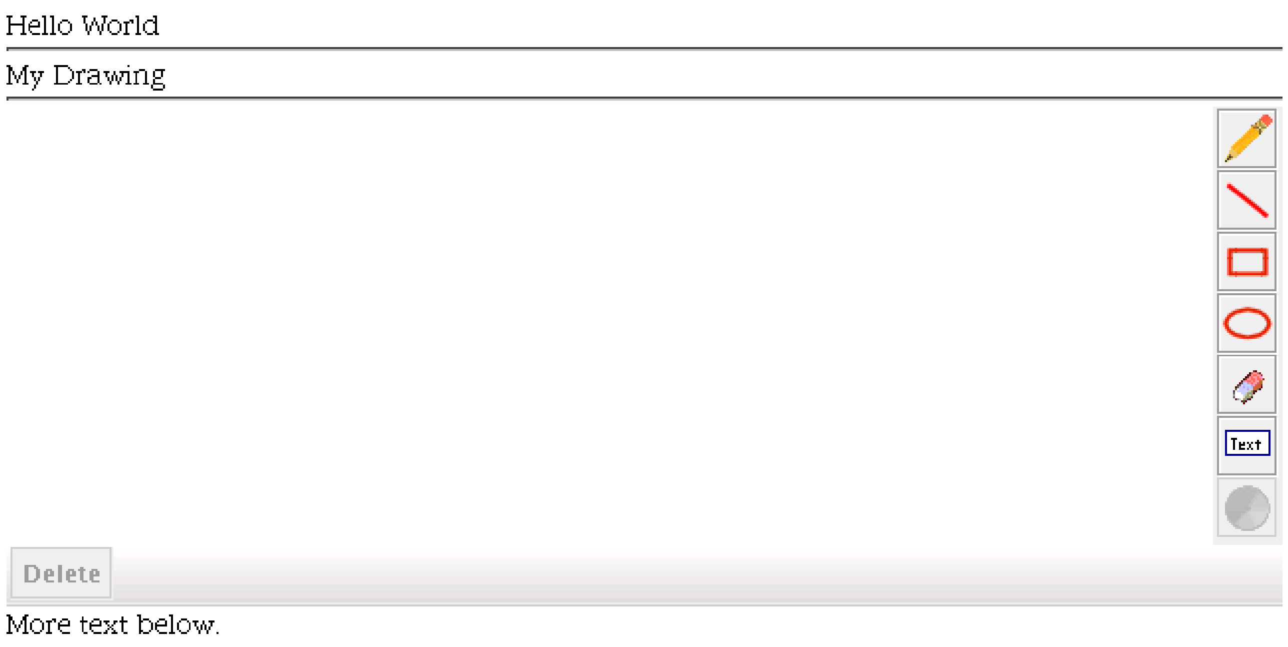
A component can either correspond to something visual or non-visual (that is, seen by the user or not).

Examples of of components available for OTrunk include:

- html document (visual)
- image (visual)
- text entry box (visual)
- drawing tool (visual)
- graphing tool (visual)
- data set (non-visual)
- data producer (non-visual)
- sensor data producer (non-visual)

There is not yet a authoring tool to make OTrunk configurations from scratch. To get started you use an xml format called otml. Here is an example of part of an otml document which represents a html page with a drawing tool in it.

Here is what it looks like when it is rendered:



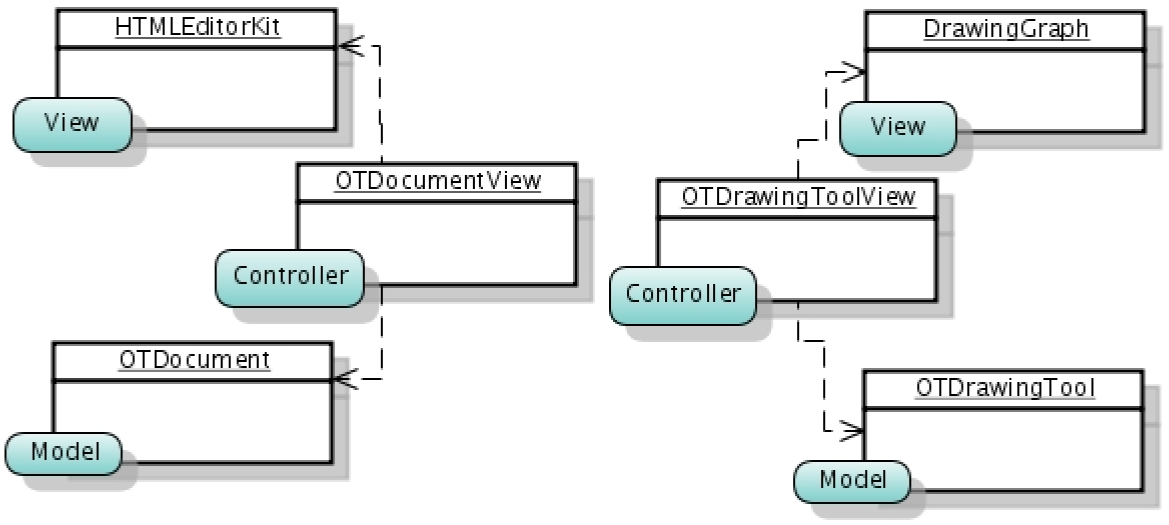
Hello World

My Drawing

More text below.

This is rendered by using views of the OTObjects in this case the OTDocument and OTDrawingTool. These views are configured in the otml file like this:

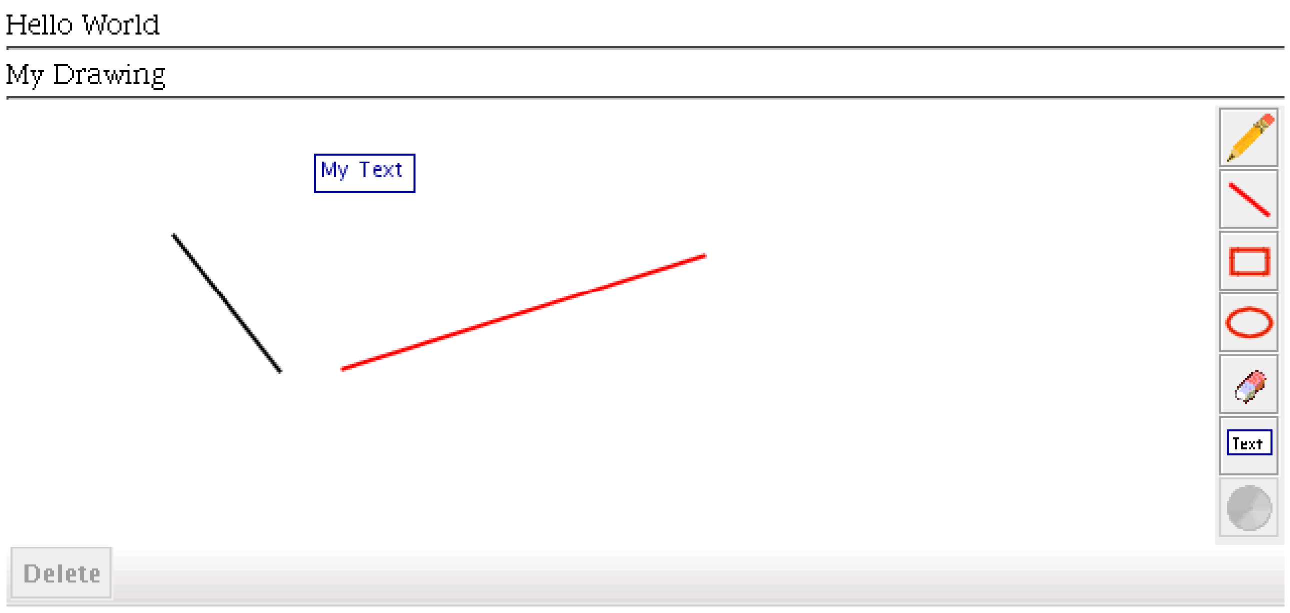
Diagram of View relationship



There is a view service used by the OTDocumentView. This is how it looks up the appropriate view for the OTDrawingTool.

Data Saving

In this example the runtime OTDocument does not provide anything the user can change. However embedded inside of it the OTDrawingTool can be manipulated. The actual visual component manipulated is the DrawingGraph, and as it changes the OTDrawingToolView updates the OTDrawingTool. Then when the document is saved these changes to the OTDrawingTool are saved. For example, if the drawing tool is manipulated to look like this:


Hello World

My Drawing

More text below

If this is saved, then the resulting otml will look like this:

OTrunk Introduction for Developers

This page last changed on Apr 29, 2008 by scytacki.

There is another page OTrunk Introduction which covers similar concepts. This particular page is more targeted at developers.

OTrunk is library or framework that tries to make it easy to author, run, and analyze rich content applications.

Rich Content Application

A rich content application is one where an end user can explore the application by changing parts of it, and these changes are saved. The opposite of a rich content application is a static web page. Something in between is a web page with an applet on it which the user can interact with. Typically the applet does not save user data, but at least the user can change how the applet looks by interacting with it.

Key Concepts

OTrunk is built on a few key concepts:

- as much as possible should be configurable without programming.
- the above concept translates into: everything should be a plugin or component.
- authoring, running, and reporting can all be built from the same components.
- the saved format of the content of an application should be as easy to read as html.
- the same content should be viewable in multiple forms with various levels of functionality: java appliction, html, pdf, stripped down handheld version

Functionality

OTrunk provides:

- a serialization/marshalling mechanism for saving objects
- a deserialization/unmarshalling mechanism for loading objects
- one or more contexts with service contexts so it can more easily be used as a "Inversion of Control container" (IoC container).
- 2 model-vew-controller systems for separating models from views.
- a COW (copy on write) mode that saves just the changes made by a user in a particular context.
- a data modeling approach so it is easy create and update the data model.

This concept of loading in objects, modifying them, and saving them again is similar to EMF and modello. XMLEncoder can do this but it is not good at it.
OTrunk has Multiple change sets can be loaded in at the same time. This results in multiple contexts. Then a reporting aware view can display each of the changes.

similar libraries:

- . In this respect it is similar to XMLEncoder, JAXB, or EMF.
- Spring has an IoC container

OTrunk Jackrabbit

This page last changed on May 30, 2008 by scytacki.

Links

- OTrunk Jackrabbit Evaluation
- \~skim:Jackrabbit
- http://www.encorewiki.org/display/encore/CMS+and+Jackrabbit
- http://www.telscenter.org/confluence/display/SAIL/Jackrabbit+Usage+Research
- http://www.telscenter.org/confluence/display/SAIL/Proposal+to+use+Jackrabbit+for+PRP

Source code

http://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-jackrabbit

Next steps

Setup a learner data test

- setup an jackrabbit server that can be access over webdav
- test the data saving into this server
- setup a sds config that tells SailOTrunk to use this for the learner data
- setup a test diy activity, and switch its config to be the one above

Import sds otml learner data

- setup a test import to see how the references to the original objects are maintained
- find a useful query or report that could be run on the imported data
- make sure this query or report will work on the imported data
- write a script or java program for doing the import

Repository representation improvements

Here a list of things that might need to be improved in the repository representation:

- xml text is stored as escaped strings instead of nodes or binary blobs
- there is no support for binary resources yet. In OTrunk these are url's or byte arrays.
- OTrunk lists and maps have a verbose representation which will be hard to query.
- the logic to navigate the structure is complex because an OTrunk property can be either a:
    - child node
    - reference property
    - string property which is a OTrunk id pointing to an object outside of the repository

- consider making everything a reference and using a standard multi-valued "ot:child" node for the children. This won't handle external references.

Round tripping otml

It seems best if developers can continue to use otml to create and update content in the repository, this requires a few things:

- the format of the original otml (whitespace, comments, local_ids) should be maintained so diffing works.
- a merging tool will need to be created so developers can merge their otml into the repository.
- otrunk objects that don't have ids will need to be matched with existing objects in the repository. Repository paths can be used for this.

Result of copying in an otml file

This example is helpful to see how the OTrunk objects are represented in the repository. The way they are represented is probably going to change because it needs to make querying the structure easy. The current represent was the quickest way to get something working.

If you look at the export from the repository you can see that the jcr:uuid attributes are used to reference objects in the repository. This allows the queries to take advantage of jackrabbit's built in reference handling.

Original file

the imports, and bundles have been removed to make it a little easier to read

Export of repository after these objects are copied in

The bundles have been removed to make it easier to read

OTrunk Jackrabbit Evaluation

This page last changed on Jul 23, 2008 by scytacki.

Overview

We are looking at using jackrabbit to help solve a few different issues:

- automatic incremental upload and download of learner data.
- on demand syncing of subsets of the learner data
- report query performance
- replication

Automatic incremental upload and download of learner data

Benefits

This would support new features

- near realtime collaboration
- near realtime reporting
- more learner data because the upload is spread out over time

This would make the system more robust:

- more data would be captures in the case of computer failure
- with a spread out upload the upload at the end of the session would be unnoticed.
- network failures can be detected in near real time.

Requirements

- http based protocol must be used so this works well through school firewalls
- should not impact the performance of the client software.
- client components do not have to know about remote persistence.
- in the case of a failure the set of objects on the server should be internally consistent so they can be reloaded at a later time.
- if the network goes down the learner data should still be saved to a local disk a regular intervals in case the program crashes.

Basic design

At regular intervals send all changes up to the server which haven't been sent before. This should be done asynchronously, so the learner doesn't have to wait for this operation to complete.

Jackrabbit supports

Jackrabbit has beta support for remoting jcr over webdav. This is done through the SPI stack.
One way to use this is:
client -> otrunk-jackrabbit -> jcr -> jcr2spi -> spi2dav -> internet -> jcr-server -> jackrabbit repository

JCR uses the concept of transient objects. So code using JCR can create a set of transient objects and then only store them when they are complete. This is done by calling save on a object or on the entire session. The jcr2spi -> spi2dav layer sends the objects over the network when save is called.

It is not clear in the jcr2spi -> spi2dav how the JCR read methods work. Do they update all the time, or do they only update on changes, or do they only update once. I saw tests for what looked like different versions of this but haven't explored it yet.

Options and analysis

Using the basic stack described above **cannot meet the requirements**. This is for performance and consitency reasons. Instead a local copy of each object needs to be maintained and then these are asynchronously written into the jcr nodes and then saved.

Basic Stack Call save at regular intervals

This would result in all the unsaved transient objects to be saved over the network. This doesn't work because save is not thread safe so all write calls need to stop during the save. If a user is in the middle of drawing, or writing they would have to wait for this to complete. If the save is done in another thread then the stored objects might not be internally consistent, and some object might be marked saved when really they have been modified.

Basic Stack Call save as each change is made

This would result in consistent data, but the performance is too slow since each key stroke or object movement has to wait for the save to complete.

Local copy

Implementing this seems like the only option. It can be done a couple of ways:

1. using a local jackrabbit repository that is synchronized/replicated with a remote repository.
2. using a local OTrunk Database that is synchronized/replicated with jcr objects.

The first option would solve two problems at the same time, but it might be too heavy weight. The second option is lighter weight but would duplicate synchronization/replication effort

One difficult design choice is how to deal with reading the data:

- Should all read calls pull down information from the repository.
- should only the first read call pull down information and and later calls use that info, this could be combined with a refresh/reload option.

Amount of effort for number 1
This depends on the existing synchronization/replication options available for jackrabbit. If none of them would work, then it might be possible to reuse parts of the "update" and "merge" code that is already part of jackrabbit which handles synchronizing 2 workspaces within a single repository. If versioning is ignored then this should be pretty straight forward so I'd estimate 2 weeks. If versioning is added in it gets more complex so then I'd estimate a month.

Amount of effort for number 2
There is no built in support for versioning, so that could be ignored in this case.

Report query performance

Replication

Run Local Jackrabbit Demo

This page last changed on Jun 10, 2008 by scytacki.

Description

This demo loads in user data from your local jackrabbit repository using jcr-spi-jcr-rmi.
It only saves data back to the repository when save is chosen from the file menu or save is chosen when OTViewer is exited.
It should handle multipe users.
Setting up the local jackrabbit server is current pretty involved.
A quick version could be created by putting together a stripped jetty with a preinstalled jackrabbit-webapp. This would probably be around 15mb in total.
The current demo requires setting up a local sds, because the necessary config files are not in the development sds yet.

Directions

1. Setup OTrunk Jackrabbit WebDav Server - this will actually be used with the following stack jcr-spi -> spi-jcr -> jcr-rmi -> jcr-server
2. setup local sds. simple shell script for this
3. run jruby -S rake sds_config:setup_jackrabbit_config_versions
4. setup a test jnlp, user, workgroup, offering, dummy curnit (not needed but sds wants an offering to have curnit)
    a. jnlp should point to: http://jnlp.concord.org/dev/org/concord/maven-jnlp/otrunk-jackrabbit-test/otrunk-jackrabbit-test.jnlp
5. run the jnlp with the extra url param: sailotrunk.otmlurl=http://continuum.concord.org/otrunk/examples/LOOPS/motion-graph-multi-activity1.otml

Next Steps

- get jackrabbit config onto dev sds
- document issues with webdav jackrabbit, and jcr-rmi direct to server
- setup a diy activity that uses this config file.
- setup a jackrabbit webdav server on a concord server so the demo can be run without a local server.
- import sds learner data

Completed Steps

- ✓ document issues with realtime or near realtime learner data saving.
- ✓ create sail config file which setups up SailOTViewer to load in user data from jcr. This uses the workgroup uuid from the sds
  The following configurations are supported:
    - jcr-spi -> spi2dav -> jcr-server
    - jcr-rmi -> jcr-server
    - jcr-spi -> spi-jcr -> jcr-rmi -> jcr-server
- ✓ added this new config file format to the sds. - this isn't checked out on the dev sds yet.

Setup OTrunk Jackrabbit WebDav Server

This page last changed on Jun 13, 2008 by scytacki.

Setup

Do the basic server setup

This page has a script under the "Automated Script" heading for setting up the basic server:
http://wiki.apache.org/jackrabbit/HowtoSpi2Dav

Start the server and setup an empty repository

Now you need to start the server which will show an exception in the log. This is because there
is no repository yet. You setup the repository by going to:
http://localhost:8080/jackrabbit-1.5/

That will show you a "Content Repository Setup".
Click the "Create Content Repository" button. This will make a jackrabbit folder inside of the toplevel folder
of your servlet container.

If you want your server to be remotely available with rmi then you may have to set the following system property to point to your servers ip address:

```
java.rmi.server.hostname
```

I haven't looked into this much so any more details on it would be helpful.
What I do know is that if you go to your servers remote access page:

```
[your server]/jackrabbit/remote.jsp
```

you can find a url to download the rmi config file. That file is binary but inside is the ip address to the RMI server. In my case it was 127.0.0.1 which wouldn't work remotely. Once I set the system property the file had the correct ip address in it.

Shutdown the server

Shut down the server.

Add the OTrunk specific type information to the repository

Run the ConfigureRepository class in otrunk-jackrabbit/src/main/java/org.concord.otrunk.jackrabbit.test
you need to pass it a command line argument of the location of the newly created jackrabbit folder. This

folder
was created by the "Create Content Repository". It should be inside of the toplevel directory of your servlet container.

There are a few ways to get the correct classpath to run this Class.

Running from Eclipse

- install the m2eclipse plugin
- check out the otrunk-jackrabbit project
- launch the ConfigureRepository class (there is not a launcher checked in for this)

Running from the shell with maven

- check out the otrunk-jackrabbit project
  - svn co http://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-jackrabbit/
- build the project
  - mvn compile
- create classpath
  - mvn dependency:build-classpath.
  - this will print out the classpath as the last message in the build log
- run the command with the classpath plus the projects compiled classes
  - java -cp "paste classpath here:target/classes" org.concord.otrunk.jackrabbit.test.ConfigureRepository "repository home directory"

Running from the shell using jnlp file

We have a few ways for turning a jnlp file into classpath which can be used on the command line.
These can be pointed at the jnlp file:
http://jnlp.concord.org/dev/org/concord/maven-jnlp/otrunk-jackrabbit-test/otrunk-jackrabbit-test.jnlp
which includes all the necessary jars (plus many more) needed to run the ConfigureRepository class.
The main class of that jnlp is not ConfigureRepository so it will need to be modified.
And remember to pass in the repository home directory argument

- the jnlp2shell.jar can be used build these classpaths
- stephen has a ruby program for doing the same thing

Start the server.

Start the server

Viewing Content

You can view basic content through a web browser by going to:
http://localhost:8080/jackrabbit-1.5/server
This doesn't show you a nice view of the child nodes of each node. You have to know the sturcture and

type in the urls.

You can use the eclipse jcr plugin and connect to the repository using the rmi method.

Dav explorer allows you to browse the repository over webdav.
http://www.davexplorer.org/
However, I haven't figured out an easy way to look at the values of "properties" using davexplorer.

Of the 3 options above the eclipse jcr plugin is the best because it knows the jcr structure.

OTrunk Layers

This page last changed on Aug 06, 2007 by scytacki.

This came from an email by Stephen with comments by Scott.
The more technical page on this is:
OTrunk Overlays

> Stephen wrote:
> At this point OTrunk layers are defined with otml. There is work
> going on to support defining OTrunk layers with EMF. A layer
> specifies objects, collections of objects, and their attributes.
>
> An OTrunk instance always has at least one layer which is the base
> layer. Multiple layers can be defined on top of the base layer and
> each in effect stacks onto the layers beneath it. The result is a
> union of all the objects defined in the layers. If the identical
> object is specified in multiple layers then the object in topmost
> layer shadows the object instances in any lower layers and any
> attribute changes are made to only this object. An object that is
> shadowed can't be changed.
That isn't quite accurate. But perhaps it is a good place to start people with.

First some terminology.

This system is very similar to COW (copy on write). COW is used by most virtual machines to maintain a base file system and a delta file system which only contains the changes that a user made. Referencing COW will help some people get a handle on this.

The objects which "shadow" lower objects are called delta objects. The virtual object which is actually used by the view is a composite object.
The composite object looks like a single object to the view. The composite object knows about the base object, any middle layer objects, and possibly a top layer object. A view sees this is a single object which it can read and write properties to. The base object of a composite object could actually be in any layer. This will happen when layer defines a new object which isn't shadowing anything below it.

Now for the inaccurate parts:
Only the top layer can be changed. So a base object, or object in a middle layer, cannot be changed. Saying "an object that is shadowed can't be changed" makes it sound like there has to be a delta object in the top layer otherwise a base object will be changed.

>
> OTrunk has often been used with just two layers: the base layer is
> the activity definition and defines the initial state of the
> activity; the layer on top of the base layer is for Learner data.
> Initially this layer is empty. As a Learner navigates through an
> activity, interacting with rich components, and entering data in
> assessments of various types attributes are changed for many objects.
> When a Learner changes the attributes of an object for the first time
> the original object in the base layer is cloned and the copy is put
> into the Learner layer. Changes the Learner makes are to the object

> in the Learner layer. Objects in the Author layer are not changed.
> When the Learner layer is persisted objects that have been copied
> from the Author layer are saved in a special form that just describes
> attributes that have been changed.

The object is not entiredly copied when a Learner makes a the first change to the object. Instead here is what is happen:

1. An empty delta object is created in the Learner layer
2. The property that the learner changed is copied into this new delta object
3. The change made by the learner is then applied to this copy.

Any later change to the object follows these steps:

1. If the property already exists in learner delta object skip to step 4
2. If the property does not exist in the learner delta object continue
3. Copy the property from the base object to the delta object
4. The property in the delta object is modified.

>
> For example a Question object might contain two attributes: an object
> representing the question and an object representing the answer. In
> the Author layer the answer attribute could be null or have a default
> prompt like:
>
> "enter your answer here ...".
>
> When a Learner enters text into the answer and persists this work the
> difference between the original Question object in the Author layer
> and the new copy in the Learner layer is saved.
>
> The differences are saved in a map (equivalent to a Ruby hash) that
> uses a key to identify the original object and then a list of the
> attributes that have been changed.

Not quite. Yes, the key identifies the original object. The value is the "delta object". The "delta object" is another instance of the same type as the base object. The delta object only has the properties set which are different than the base object. I use the term properties instead of attributes, because attributes have a special meaning in xml. However "properties" have a special meaning in Java. In EMF terms these would be called "features".

>
> This Learner layer has often been referred to as the difference layer
> and this is how our activities have worked. OTrunk does not constrain
> additional layer to only contain differences. A new layer could
> contain entirely new objects that didn't exist in the Author layer.

OTrunk Object References

This page last changed on Jul 30, 2008 by stepheneb.

See also OTrunk ID System

In otrunk you can store references to other objects.
For this example lets say you two OTClasses: OTHouse OTDoor

You want your OTHouse to have a reference to a OTDoor.

To do this you modify your OTHouse interface or resource schema like this:

There are a few ways this can be represented as in the otml.

OTDoor with uuid id and attribute reference in OTHouse

OTDoor with uuid id and element reference in OTHouse

OTDoor with local_id and attribute reference in OTHouse

OTDoor with local_id and element reference in OTHouse

OTrunk Overlays

This page last changed on Jul 28, 2007 by scytacki.

Overview and terms

Overlays are used to create alternative versions of activities. And they are used to store modifications made by a user as they go through an activity.

An Overlay contains a set of **delta** objects and a set of **non-delta** objects.
The delta objects add modifications to **base** objects. The non-delta objects are complete objects which are contained by the overlay.

Within an overlay there can only be one delta object for any base object. Multiple overlays can be layered to combine a multiple delta objects for one base object. The combination of one or more delta objects with a base object is a **composite** object.

The views or controllers which are using OTrunk model objects work with composite objects. They do not need to know if the object is a composite object or a simple complete object.

Editing

When editing an overlay the same procedure is used wether you are a student going through an activity, or an author creating an alternative version of the activity. An author might have more affordances, so they could control which modifications go into the overlay and which modifications are to the original document. The code works by supplying a OverlayObjectService for editing the overlay. When an object is requested from this OverlayObjectService a **composite** object is created that has a reference to the **base** object and the overlay object if it exists already.

To provide affordances for editing the overlay and the base object, a view can get both the OverlayObjectService and the OTObjectSerivce for the base objects. The default object service for the view will be the object service which is managing the view's object. So to edit an overlay without having special views, the top level view just needs to be created with an object from the OverlayObjectService and then everything will propigate down from that.

Notes

The current system uses overlays to manage user data ontop of author data. 3 OTDatabases are used to acomplish this.

- Author database
- Learner database
- Combination database

The combination database has the author database and differencing database. Whenever an object is requested from the combination database it returns a combination object which has a pointer to the authored object from the author database and a pointer to the learner object from the learner database.

If there is no learner object yet then that will be null.

When a property is requested from the combination object, it first looks in the learner object, and then in the author object.
When a property is set on the combination object, it is always set in the learner object.

This approach is used a lot in virtual machine file systems now and is refered to as COW or Copy On Write.

There are a few interesting cases. If a new object is created by the learner a new combination object will be created to go on top of it. This is because this new object might have a reference to an existing authored object. And when that reference is requested from the new object a combination object needs to be returned so the COW can be maintained.

Doing it this way also leads to complications with the IDs of the object. When an object is requested with an author ID, the actual object returned will have a different ID. This has caused the most problems when doing lookup tables with object IDs. For example a technical hint uses a look up table to find the correct technical hint based on the current selected hardward device. The current selected hardward device is an OTObject. So the technical hint code, gets this object and asks for its ID and then trys to find the appropriate technical hint in its map. Doing this the obvious way fails because the ID returned by the current seleced hardware device is not its authored ID, but instead it is the ID of the combination object. The workaround for this particular case is to use a utility method in OTrunkUtil called getObjectFromMapWithIdKeys. What that method does is go through all the keys in the map, get their object and then see if it equals the object being looked up.

It seems it would be better to move this code out of the database level and into the OTObjectService. This should simply the database level some, and also possibly remove the need for the combination object altogether. At the OTObjectService dynamic OTObject instances are created and these instances could keep track of their authored and learner object. As well as many more overlay objects in between.

One difficultly with this approach is the object introspection and copying code. Currently this code works on the database level. This would mean that copying a learner object which is overlayed on a authored object would not do the right thing. The copy code gets the DataObject underneath the OTObject and copies that. So either the copy code needs to be revised to deal correctly with overlays, or it needs to work with the OTObjects instead of the OTDataObjects.

One question with the copy code is what should happen when an object that is actually stored as multiple layers is copied. Should all the layers be collapsed? Should each layer be copied? Copying each layer would require existing writable overlays the the layers could be stored in. That doesn't fit with the current model, so it seems they need to be collapsed.

OTrunk Plan for CC

This page last changed on Apr 29, 2007 by scytacki.

- TELS
- UDL
- ITEST

TELS

- Need to integrate reporting using pods and rims with reporting using otrunk objects.
    - instead of trying to figure out how to fit otrunk into rims, it would be easier to figure out an abstraction that works for both and code the reporting system to that abstraction.
- model step updates
    - logging need to add logging to existing model steps, in a OTrunk model, this means converting existing model steps into OTrunk. And then adding logging. The dribble log would be easy with a properly converted OTrunk object. The more concise report is harder. It would be best done with script to start like edward did in airbags and chemical reactions. A more advanced approach would be to use a rule language like xslt to declare paterns to match in the dribble log and then run a script when that pattern matches. In the case of the concise report the script is just to generate more xml so it would be perfect for xslt.

How to make xslt doable by authors.

Next steps for OTrunk usage for TELS model authoring tool:

- Choose an existing tels model step make it configured by otrunk
    - I don't want to choose the steps Edward just converted to pedagogica becuase it is duplicate work.

OTrunk Profiling

This page last changed on Jun 11, 2008 by stepheneb.

Methods for and results from profiling OTrunk.

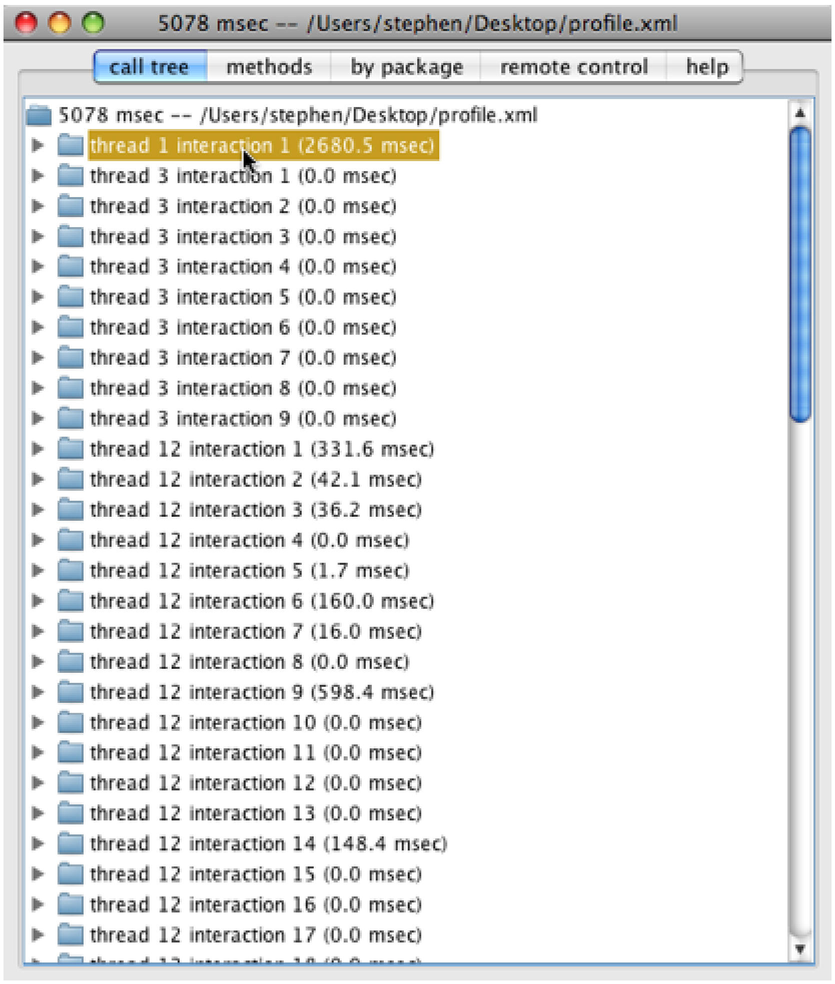
JIP

See: Java Profiling with JIP for instructions for setting up JIP and using it with Eclipse.

Profiling by thread and call tree

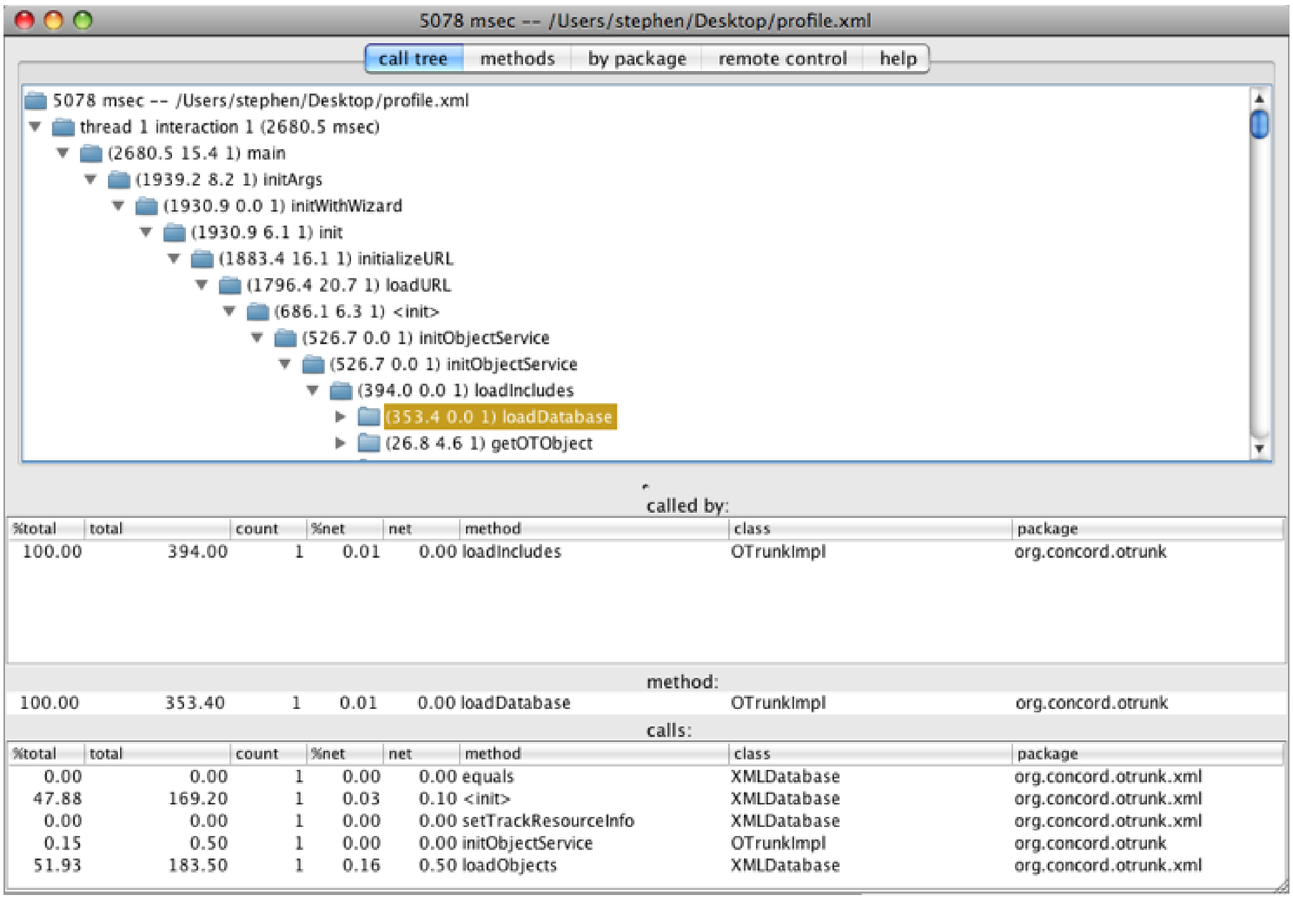
I profiled the OTrunk LOOPS demo activity: motion-graph-multi-activity1 using JIP.

JIP collected data on all the threads *(screen captures from jipViewer)*:



```
5078 msec -- /Users/stephen/Desktop/profile.xml
  thread 1 interaction 1 (2680.5 msec)
  thread 3 interaction 1 (0.0 msec)
  thread 3 interaction 2 (0.0 msec)
  thread 3 interaction 3 (0.0 msec)
  thread 3 interaction 4 (0.0 msec)
  thread 3 interaction 5 (0.0 msec)
  thread 3 interaction 6 (0.0 msec)
  thread 3 interaction 7 (0.0 msec)
  thread 3 interaction 8 (0.0 msec)
  thread 3 interaction 9 (0.0 msec)
  thread 12 interaction 1 (331.6 msec)
  thread 12 interaction 2 (42.1 msec)
  thread 12 interaction 3 (36.2 msec)
  thread 12 interaction 4 (0.0 msec)
  thread 12 interaction 5 (1.7 msec)
  thread 12 interaction 6 (160.0 msec)
  thread 12 interaction 7 (16.0 msec)
  thread 12 interaction 8 (0.0 msec)
  thread 12 interaction 9 (598.4 msec)
  thread 12 interaction 10 (0.0 msec)
  thread 12 interaction 11 (0.0 msec)
  thread 12 interaction 12 (0.0 msec)
  thread 12 interaction 13 (0.0 msec)
  thread 12 interaction 14 (148.4 msec)
  thread 12 interaction 15 (0.0 msec)
  thread 12 interaction 16 (0.0 msec)
  thread 12 interaction 17 (0.0 msec)
```

Here's a partial display of the call tree for thread 1:



```
5078 msec -- /Users/stephen/Desktop/profile.xml
▼ thread 1 interaction 1 (2680.5 msec)
  ▼ (2680.5 15.4 1) main
    ▼ (1939.2 8.2 1) initArgs
      ▼ (1930.9 0.0 1) initWithWizard
        ▼ (1930.9 6.1 1) init
          ▼ (1883.4 16.1 1) initializeURL
            ▼ (1796.4 20.7 1) loadURL
              ▼ (686.1 6.3 1) <init>
                ▼ (526.7 0.0 1) initObjectService
                  ▼ (526.7 0.0 1) initObjectService
                    ▼ (394.0 0.0 1) loadIncludes
                      ▶ (353.4 0.0 1) loadDatabase
                      ▶ (26.8 4.6 1) getOTObject
```

called by:

%total	total	count	%net	net	method	class	package
100.00	394.00	1	0.01	0.00	loadIncludes	OTrunkImpl	org.concord.otrunk

method:

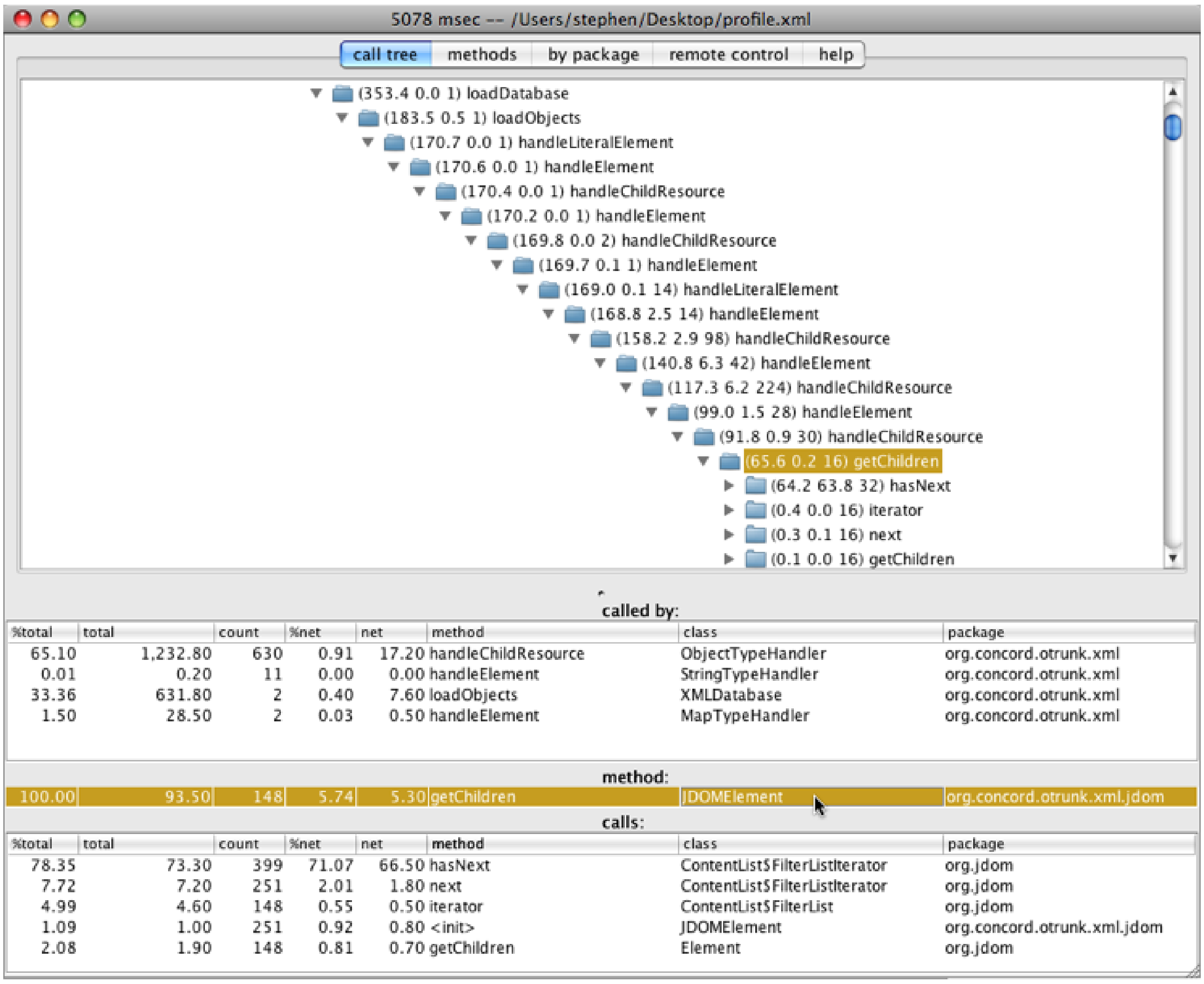
100.00	353.40	1	0.01	0.00	loadDatabase	OTrunkImpl	org.concord.otrunk

calls:

%total	total	count	%net	net	method	class	package
0.00	0.00	1	0.00	0.00	equals	XMLDatabase	org.concord.otrunk.xml
47.88	169.20	1	0.03	0.10	<init>	XMLDatabase	org.concord.otrunk.xml
0.00	0.00	1	0.00	0.00	setTrackResourceInfo	XMLDatabase	org.concord.otrunk.xml
0.15	0.50	1	0.00	0.00	initObjectService	OTrunkImpl	org.concord.otrunk
51.93	183.50	1	0.16	0.50	loadObjects	XMLDatabase	org.concord.otrunk.xml

Executing Main in thread 1 takes almost 2.7s and 0.35s of that was spent in loadData.

Taking a closer look at why loadData was taking so much time you can see that a portion is spent in jdoms getChildren method:


5078 msec -- /Users/stephen/Desktop/profile.xml

```
▼ (353.4 0.0 1) loadDatabase
  ▼ (183.5 0.5 1) loadObjects
    ▼ (170.7 0.0 1) handleLiteralElement
      ▼ (170.6 0.0 1) handleElement
        ▼ (170.4 0.0 1) handleChildResource
          ▼ (170.2 0.0 1) handleElement
            ▼ (169.8 0.0 2) handleChildResource
              ▼ (169.7 0.1 1) handleElement
                ▼ (169.0 0.1 14) handleLiteralElement
                  ▼ (168.8 2.5 14) handleElement
                    ▼ (158.2 2.9 98) handleChildResource
                      ▼ (140.8 6.3 42) handleElement
                        ▼ (117.3 6.2 224) handleChildResource
                          ▼ (99.0 1.5 28) handleElement
                            ▼ (91.8 0.9 30) handleChildResource
                              ▼ (65.6 0.2 16) getChildren
                                ▶ (64.2 63.8 32) hasNext
                                ▶ (0.4 0.0 16) iterator
                                ▶ (0.3 0.1 16) next
                                ▶ (0.1 0.0 16) getChildren
```

called by:

%total	total	count	%net	net	method	class	package
65.10	1,232.80	630	0.91	17.20	handleChildResource	ObjectTypeHandler	org.concord.otrunk.xml
0.01	0.20	11	0.00	0.00	handleElement	StringTypeHandler	org.concord.otrunk.xml
33.36	631.80	2	0.40	7.60	loadObjects	XMLDatabase	org.concord.otrunk.xml
1.50	28.50	2	0.03	0.50	handleElement	MapTypeHandler	org.concord.otrunk.xml

method:

%total	total	count	%net	net	method	class	package
100.00	93.50	148	5.74	5.30	getChildren	JDOMElement	org.concord.otrunk.xml.jdom

calls:

%total	total	count	%net	net	method	class	package
78.35	73.30	399	71.07	66.50	hasNext	ContentList$FilterListIterator	org.jdom
7.72	7.20	251	2.01	1.80	next	ContentList$FilterListIterator	org.jdom
4.99	4.60	148	0.55	0.50	iterator	ContentList$FilterList	org.jdom
1.09	1.00	251	0.92	0.80	<init>	JDOMElement	org.concord.otrunk.xml.jdom
2.08	1.90	148	0.81	0.70	getChildren	Element	org.jdom

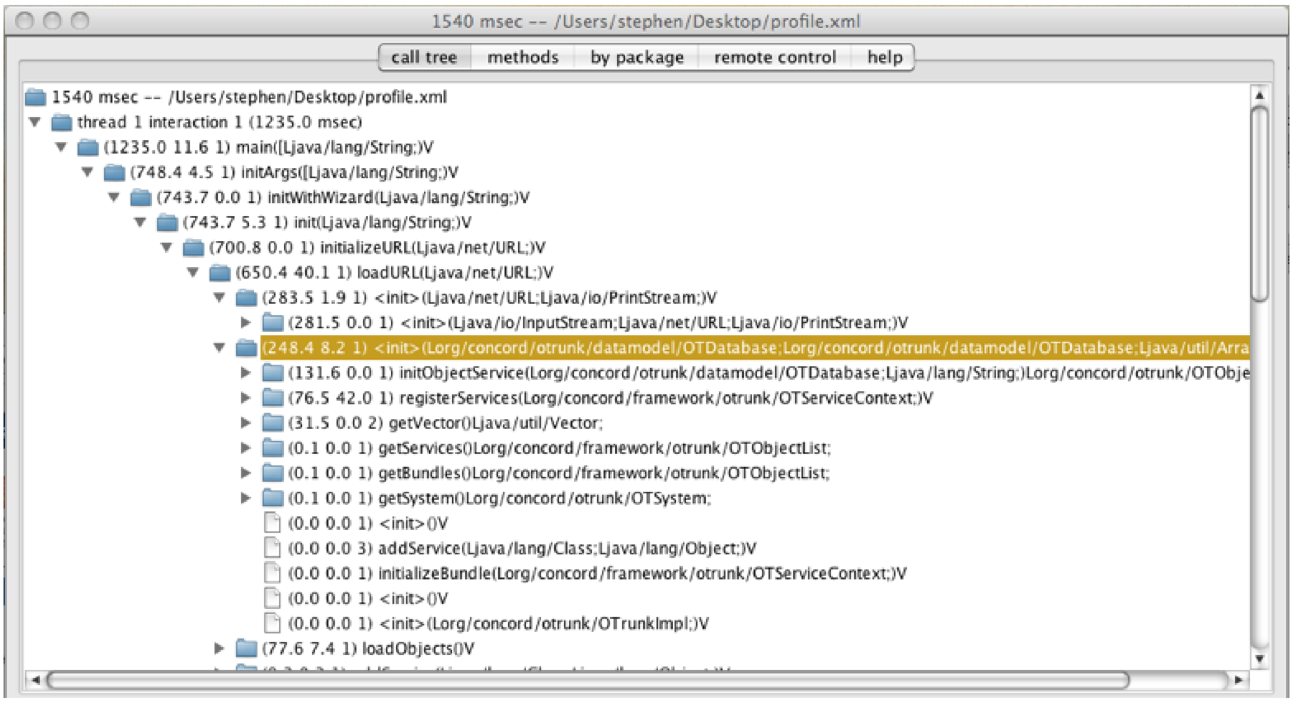
Adding method signatures

I added this parameter to my profile.properties file:

```

```

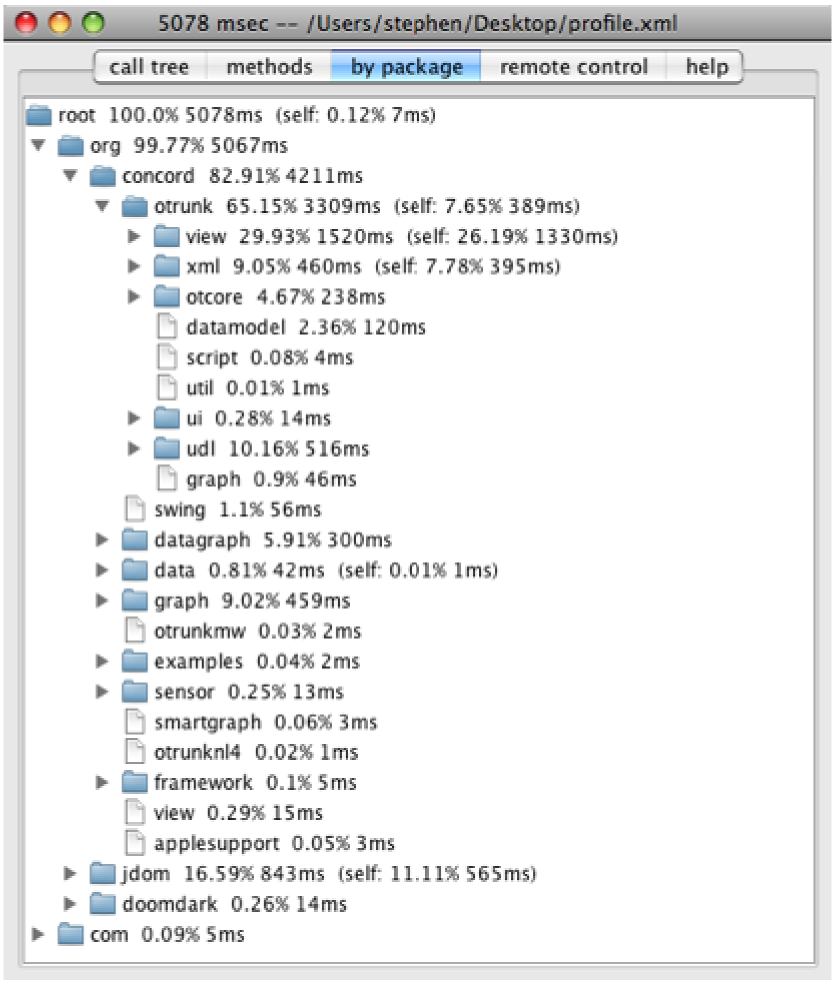
and created a profile for one of the simplest OTrunk activities: Basic Text Edit. The result below shows how now the two calls to <init> by loadURL can now be distinguished by their method signatures *(and also that this simple activity starts up in about 1/2 the time of the more complex activity)*:



```
1540 msec -- /Users/stephen/Desktop/profile.xml
1540 msec -- /Users/stephen/Desktop/profile.xml
▼ thread 1 interaction 1 (1235.0 msec)
  ▼ (1235.0 11.6 1) main([Ljava/lang/String;)V
    ▼ (748.4 4.5 1) initArgs([Ljava/lang/String;)V
      ▼ (743.7 0.0 1) initWithWizard(Ljava/lang/String;)V
        ▼ (743.7 5.3 1) init(Ljava/lang/String;)V
          ▼ (700.8 0.0 1) initializeURL(Ljava/net/URL;)V
            ▼ (650.4 40.1 1) loadURL(Ljava/net/URL;)V
              ▼ (283.5 1.9 1) <init>(Ljava/net/URL;Ljava/io/PrintStream;)V
                ▶ (281.5 0.0 1) <init>(Ljava/io/InputStream;Ljava/net/URL;Ljava/io/PrintStream;)V
              ▼ (248.4 8.2 1) <init>(Lorg/concord/otrunk/datamodel/OTDatabase;Lorg/concord/otrunk/datamodel/OTDatabase;Ljava/util/Arra
                ▶ (131.6 0.0 1) initObjectService(Lorg/concord/otrunk/datamodel/OTDatabase;Ljava/lang/String;)Lorg/concord/otrunk/OTObje
                ▶ (76.5 42.0 1) registerServices(Lorg/concord/framework/otrunk/OTServiceContext;)V
                ▶ (31.5 0.0 2) getVector()Ljava/util/Vector;
                ▶ (0.1 0.0 1) getServices()Lorg/concord/framework/otrunk/OTObjectList;
                ▶ (0.1 0.0 1) getBundles()Lorg/concord/framework/otrunk/OTObjectList;
                ▶ (0.1 0.0 1) getSystem()Lorg/concord/otrunk/OTSystem;
                  (0.0 0.0 1) <init>()V
                  (0.0 0.0 3) addService(Ljava/lang/Class;Ljava/lang/Object;)V
                  (0.0 0.0 1) initializeBundle(Lorg/concord/framework/otrunk/OTServiceContext;)V
                  (0.0 0.0 1) <init>()V
                  (0.0 0.0 1) <init>(Lorg/concord/otrunk/OTrunkImpl;)V
              ▶ (77.6 7.4 1) loadObjects()V
```

Viewing profile results by package

You can see in the display below that during the course of the OTrunk activity almost 17% was spent in **jdom** while 30% was spent in the **view** package.



```
root  100.0% 5078ms  (self: 0.12% 7ms)
▼ org  99.77% 5067ms
  ▼ concord  82.91% 4211ms
    ▼ otrunk  65.15% 3309ms  (self: 7.65% 389ms)
      ▶ view  29.93% 1520ms  (self: 26.19% 1330ms)
      ▶ xml  9.05% 460ms  (self: 7.78% 395ms)
      ▶ otcore  4.67% 238ms
        datamodel  2.36% 120ms
        script  0.08% 4ms
        util  0.01% 1ms
      ▶ ui  0.28% 14ms
      ▶ udl  10.16% 516ms
        graph  0.9% 46ms
      swing  1.1% 56ms
    ▶ datagraph  5.91% 300ms
    ▶ data  0.81% 42ms  (self: 0.01% 1ms)
    ▶ graph  9.02% 459ms
      otrunkmw  0.03% 2ms
    ▶ examples  0.04% 2ms
    ▶ sensor  0.25% 13ms
      smartgraph  0.06% 3ms
      otrunknl4  0.02% 1ms
    ▶ framework  0.1% 5ms
      view  0.29% 15ms
      applesupport  0.05% 3ms
  ▶ jdom  16.59% 843ms  (self: 11.11% 565ms)
  ▶ doomdark  0.26% 14ms
▶ com  0.09% 5ms
```

Profiling a Timeline

JIP alternatively supports collecting a timeline of actions. Timeline profiling only outputs in a text file format and is not viewable using JipViewer. The timeline data shows events from different threads in sequence.

Here's a short excerpt showing one thread ending and another starting near the 2s mark:

These data are useful when investigating concurrency issues. The timeline file size for an OTrunk activity start and immediate stop is 22MB. Analyzing this type of data is more practical when collected from a more constrained period of program execution. This can be done with a remote program or programatically inside OTrunk by using JIP classes.

Profile data from several OTrunk startups

The results are attached in these files:

- JIP profile of LOOPS multi-page motion and velocity demo *(method signatures turned off)*: profile_otrunk_20080611.xml *(2MB)*

- JIP profile of much simpler OTrunk activity: Basic Text Edit *(method signatures turned on)*: profile_otrunk_viewer_basic_text_edit.xml *(264kB)*

If you have JIP installed you can download these profile and view the results using **jipViewer**.

OTrunk Reporting

This page last changed on Feb 27, 2008 by scytacki.

We are working on a integrated system with the DIY to support multi user OTrunk Reporting.

Originally we planned for a system where any *ReportTemplate* could be applied to any compatible *AuthoredContent* and group of users. However the OTrunk system isn't flexible enough for this approach yet. So instead we need to restrict the *ReportTemplate* to one *AuthoredContent*. See the "Possible Improvement" below so this restriction can be relaxed.

Current Design

ReportTemplate is bound to a particular *AuthoredContent*. The DIY needs to fill in the group of users. And the url to the *AuthoredContent*. The url to the *AuthoredContent* is filled in so the DIY can serve the *AuthoredContent* either directly from the source or from its cache.

Here is an example of the final result:

```
<?xml version="1.0" encoding="UTF-8"?>
<otrunk>
  <imports>
    <import class="org.concord.otrunk.OTSystem"/>
    <import class="org.concord.otrunk.OTInclude"/>
    <import class="org.concord.otrunk.view.OTViewBundle"/>
    <import class="org.concord.otrunk.view.OTViewEntry"/>
    <import class="org.concord.otrunk.view.OTMultiUserRoot"/>
    <import class="org.concord.otrunk.view.OTUserDatabaseRef"/>
    <import class="org.concord.otrunkcapa.OTCAPAReportTemplate"/>
  </imports>
  <idMap>
    <idMapping local_id="script_object"
id="9ddf1254-a4da-11dc-96ee-0155390dc58c!/activity_script"/>
  </idMap>
  <objects>
    <OTSystem local_id="system">
      <bundles>
        <OTViewBundle>
          <views>
            <OTViewEntry objectClass="org.concord.otrunk.view.OTMultiUserRoot"
viewClass="org.concord.otrunk.view.OTMultiUserRootView"/>
            <OTViewEntry objectClass="org.concord.otrunkcapa.OTCAPAReportTemplate"
viewClass="org.concord.otrunkcapa.OTCAPAReportTemplateView"/>
          </views>
        </OTViewBundle>
      </bundles>
      <includes>
        <OTInclude href="capa_labview_activity.otml"/>
      </includes>
      <root>
        <OTMultiUserRoot local_id="multi_user_root">
          <userDatabases>
            <OTUserDatabaseRef url="file:///Users/skim/proj/loops/micah.otml"/>
            <OTUserDatabaseRef url="file:///Users/skim/proj/loops/james.otml"/>
            <OTUserDatabaseRef url="file:///Users/skim/proj/loops/nicholas.otml"/>
            <OTUserDatabaseRef url="file:///Users/skim/proj/loops/glen.otml"/>
          </userDatabases>
          <reportTemplate>
            <OTCAPAReportTemplate script="${script_object}"/>
          </reportTemplate>
        </OTMultiUserRoot>
      </root>
    </OTSystem>
  </objects>
```

```
</otrunk>
```

This would be generated from a "template" where the OTMultiUserRoot looks like this:

The code for modifying the otml file should look for the OTMultiUserRoot element with a local_id attribute of multi_user_root and add the userDatabases element from above:

The template would have the following element in it:

It is quite likely that the report template will include other otml files. So the other <OTInclude> elements inside of <includes> should be preserved.

Possible Improvement

There is definitely a lot of places where this approach could be improved, but this seems the most practical for right now. There is one improvement that seems practical but since I'm not doing the work I don't know. If this improvement is not clear, or doesn't fit with how the DIY should be heading, then it isn't worth doing. It is more important to get something flexible, working quickly, even it is a little cumbersome.

The *ReportTemplate* otml will probably be used for several activities. The only change will be the references to the objects in the *AuthoredContent*. These changes can be encapsulated in the <idMap> element, like they are above. So parts of the idMap could be generated by the DIY this would eliminate the need for the mulitple copies. And we could get closer to our original design.

This could be done by adding a list of "bindings" to the relation between an DIY activity and *ReportTemplate*. These "bindings" would be of the form:

```
report_template_local_id, diy_activity_global_id
```

Both items are just strings.

These would be added to the <idMap> element using this form:

It is possible that the report template will have other <idMapping> elements inside of the <idMap> these should be preserved.

These binds should be added to the "relation" between the DIY Activity and the *ReportTemplate* so that multiple report templates can be connected to a single DIY Activity. And each report template might require a different set of bindings.

OTrunk Schemas

This page last changed on May 30, 2008 by stepheneb.

This work is experimental.

In the otrunk project is a folder called schema. In there is an Eclipse launch configuration called RNGSchemaGenerator.launch .

Running RNGSchemaGenerator will produce this Relax NG schema for the existing OTrunk objects: otrunk_all.rng.

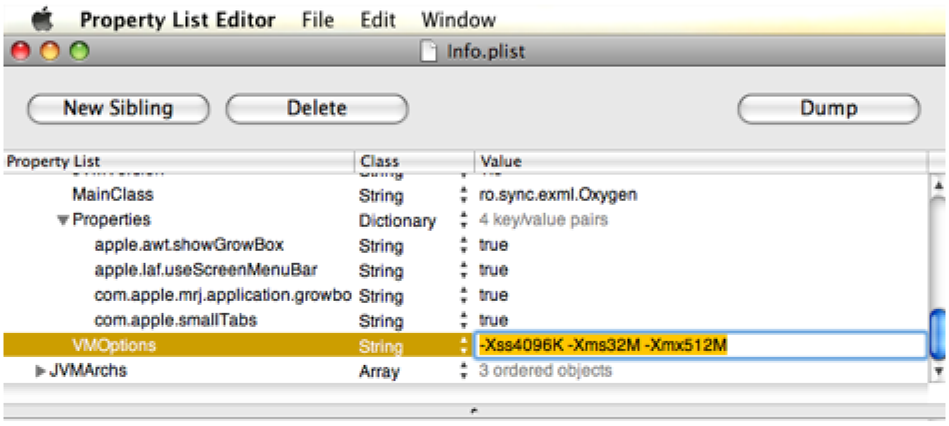
I've done a bit of testing and it appears that this schema can be used with an otml document and an XML editor like Oxygen.

I'm using Oxygen 9.2 on MacOS X and I needed to increase Oxygen's heap space to load the schema.

In the Terminal:

```
open /Applications/oxygen/Oxygen.app/Contents/Info.plist
```

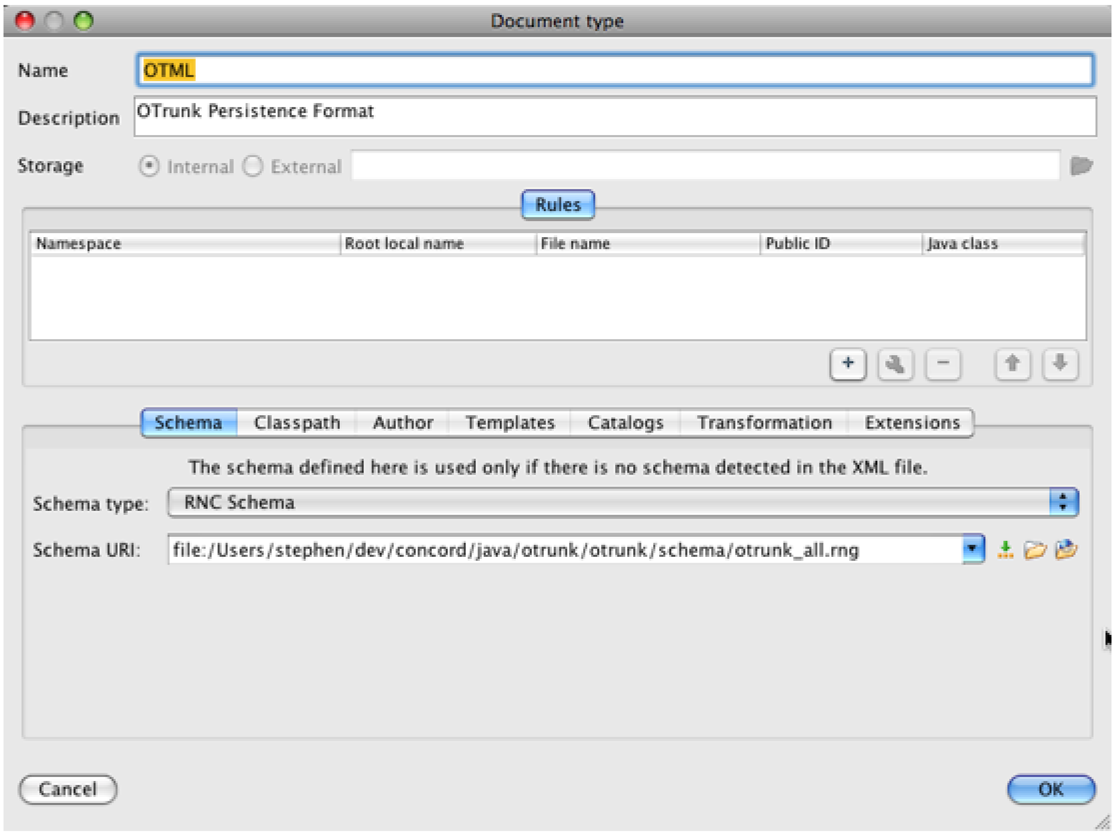
and increase the heap and memory sizes Java will use when running Oxygen:



Property List	Class	Value
MainClass	String	ro.sync.exml.Oxygen
▼ Properties	Dictionary	4 key/value pairs
apple.awt.showGrowBox	String	true
apple.laf.useScreenMenuBar	String	true
com.apple.mrj.application.growbo	String	true
com.apple.smallTabs	String	true
VMOptions	String	-Xss4096K -Xms32M -Xmx512M
▶ JVMArchs	Array	3 ordered objects

Here's an example of using the **`otrunk_all.rng`** schema while editing an existing otml file: motion-graph-multi-activity1.otml

First make sure you are associating the schema with OTML files in Preferences:Document Types:


Document type

Name: OTML

Description: OTrunk Persistence Format

Storage: [✓ Internal] External

Rules

Namespace	Root local name	File name	Public ID	Java class

Schema | Classpath | Author | Templates | Catalogs | Transformation | Extensions

The schema defined here is used only if there is no schema detected in the XML file.

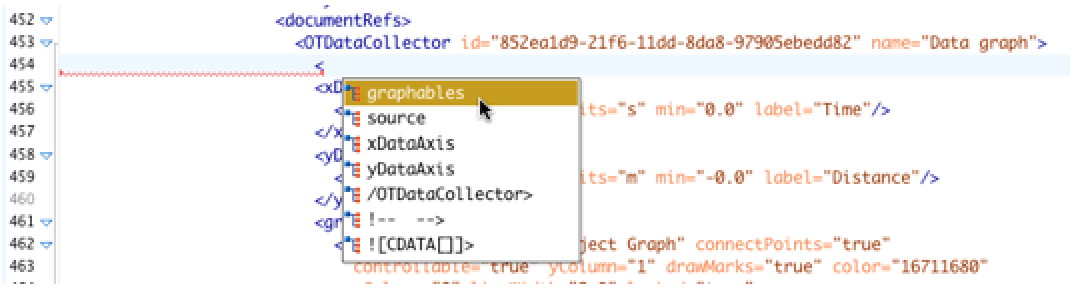
Schema type: RNC Schema

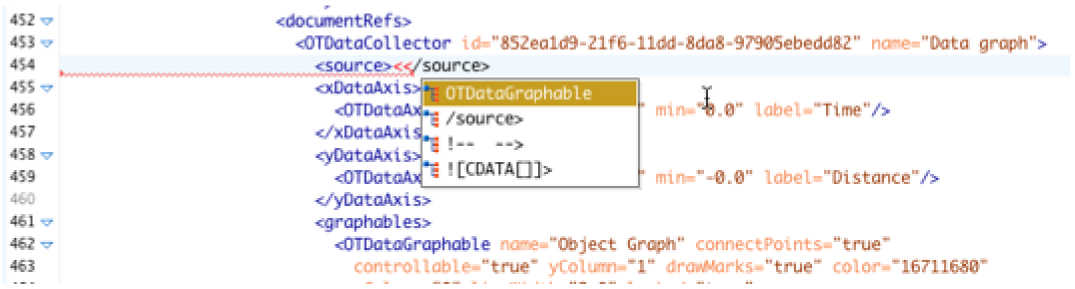
Schema URI: file:/Users/stephen/dev/concord/java/otrunk/otrunk/schema/otrunk_all.rng

Cancel | OK

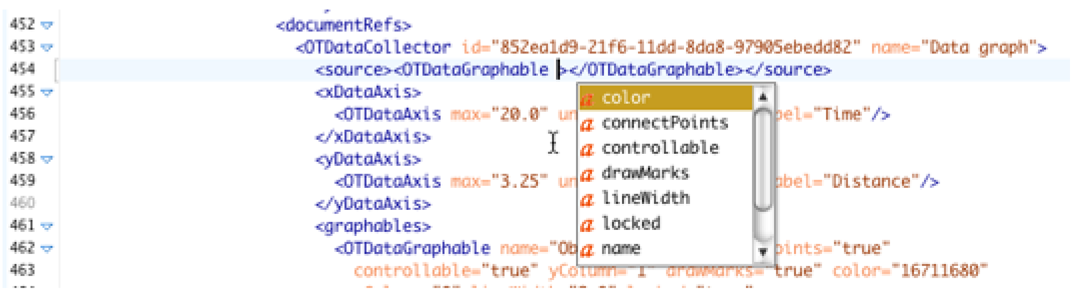
Now when you open an otml file it will be associated with the **`otrunk_all.rng`** schema.

Oxygen will display appropriate completions after you type an opening "<".


```
<documentRefs>
  <OTDataCollector id="852ea1d9-21f6-11dd-8da8-97905ebedd82" name="Data graph">
    <
    <xD
      <                                   its="s" min="0.0" label="Time"/>
    </x
    <yD
                                          its="m" min="-0.0" label="Distance"/>
    </y
    <gr
      <                                   ject Graph" connectPoints="true"
        controllable="true" yColumn="1" drawMarks="true" color="16711680"
```


```
<documentRefs>
  <OTDataCollector id="852ea1d9-21f6-11dd-8da8-97905ebedd82" name="Data graph">
    <source><</source>
    <xDataAxis>
      <OTDataAx                           min="0.0" label="Time"/>
    </xDataAxis
    <yDataAxis>
      <OTDataAx                           min="-0.0" label="Distance"/>
    </yDataAxis>
    <graphables>
      <OTDataGraphable name="Object Graph" connectPoints="true"
        controllable="true" yColumn="1" drawMarks="true" color="16711680"
```

Oxygen will display completions for attributes after typing a space character:



```
                    <documentRefs>
                      <OTDataCollector id="852ea1d9-21f6-11dd-8da8-97905ebedd82" name="Data graph">
                        <source><OTDataGraphable ></OTDataGraphable></source>
                        <xDataAxis>
                          <OTDataAxis max="20.0" un                    bel="Time"/>
                        </xDataAxis>
                        <yDataAxis>
                          <OTDataAxis max="3.25" un                    abel="Distance"/>
                        </yDataAxis>
                        <graphables>
                          <OTDataGraphable name="Ob                    oints="true"
                            controllable="true" yColumn="1" drawMarks="true" color="16711680"
```

OTrunk Script

This page last changed on May 06, 2007 by scytacki.

First the word script is not accurate, because some "scripts" will be one or more java classes. But the general idea is that these are "scripts" for adding new behavior to existing objects or components.

There are multiple types of scripting in OTrunk. To start with we'll make a scripting framework that makes it easy to explicity add to scripts objects, and then execute them in the views or controllers.

OTrunk Script Notes

Script Types

ot-view implementations

This type of scripting would be used to create a new view of an existing java object. This will become more important for reporting and logging. But right now it is easy enough to use Java for this.

ot-object lifecycle listeners

This type of scripting can respond to state changes in an ot-object, and then update another ot-object. This type of scripting will become more important as it is easier to build ot-classes which track the actual view classes. It requries making ot-classes with properties for every part of a component that needs to be scripted. And then keeping those properties synchronized with the component as it changes. When ot-view implementation scripting is feasible then this will be the preferred way of adding new behavior. It will be prefered because it will ensure that everything you want to respond to with scripts can be recorded for logging or reporting.

ot-view lifecycle listeners

This will be the first type of script to be implemented. Scripts of this type will have access to the ot-object being viewed, other views in a page or container, and the actual java objects used to render those views. This will make it easier to create any type of new behavior using existing ot-objects and ot-views. It will not ensure that the scriptable events will be logged however so it is not a good long term solution.
OTView Scripting

OTrunk Script Notes

This page last changed on May 06, 2007 by scytacki.

These need to be checked with the java scripting api.

The lifecycle methods of the script will be defined with a java interface. This will make it easy to call the script from java, and make it possible to implement it in Java (instead of script).

The context of the script is implemented as a concrete object with methods and get and set properties. In the script these methods and properties will be available directly without specifying an object. If the script is written in java, the context will be passed to it through the constructor.

OTView Scripting

This page last changed on Oct 03, 2007 by scytacki.

This page is out of date. The ViewHost is replaced with a OTViewContext which provides this functionality.

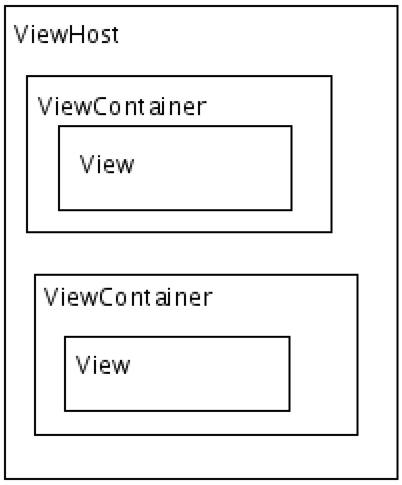
It is currently difficult to make a simple page that has a model and a button to take a snapshot of the model. The problem is because the button does not have access to the real java object which is the model. The button can easily access the ot-object that represents the model, but to actually take the snapshot it needs the jcomponent of the model.

So to enable scripting like this there needs to be a way for views to get the actual components of sibilings located on a page with them.

The first approach is to create something called a **ViewHost** all views can access their viewHost. The class will be
OTViewHost
getViewByTag(String tag)
getViewByObject(OTObject obj)

The next issue will be that the button needs to known when a view is refreshed. However to just get started we will add the view host concept.

The ViewHost represents an object above the view-container. A view container holds a single view, it can be used by a view to replace itself with a different view. A ViewHost can have many views. This diagram might help:



The view host can't be made available through the viewserviceprovider because that currently comes from the viewFactory, so it wouldn't beable to make a new one for each container.

OTrunk Script Example

This page last changed on Oct 04, 2007 by imoncada.

This page explains how to create a script in the OTrunk environment. I will use a javascript for this example, although you can write script in other languages like JRuby.

In general, the idea is that you set up the UI components in an otml file. Then, you can add script objects to the otml as well. These script objects are useful to control the user interface experience and to add interactivity, feedback, logging, etc.
The script objects are OTObjects as well, so they have to be added somewhere in the otml file in order to be loaded. The general rule would be that you add them in the same container where the objects that you want to control are located.
One script object can have access to more than one OTObject in your otml page.
You can have multiple script objects in an otml file, even next to each other in the same container, and mixed with other OTObjects. The order in which the scripts will be loaded might not be deterministic.
When you write a script, you can either write it directly in the otml file, or you can write it in a separate file and then reference it from the otml file. The first option is what we call an "embedded" script, but is mostly used only for quick testing. The latter option is recommended because that way you can keep your UI and you logic separate, and also, it might be easier to edit a script in a separate page, using an editor of your choice.

In this example, I'm going to create a script that writes something in the screen into an OTText object. It's a really simple and probably not very useful but my goal is to illustrate the steps to take.

Writing a new otml file with the visual components

Lay out your visual components in your otml file either manually or using some authoring tool. Just think about the UI without the script first. In this example, I'm going to use the following otml:

The otml above just has a document as the root. The document has some text and it's also pointing to an OTText object with an id "text_object".
The script will manipulate this text_object and it will change its text inside (which is currently empty).

Adding a new empty OTScriptObject to the otml

Now, you can add a new OT script object to your otml. Since I like using the library, I'm going to add it to the library section. For now, I'll keep the script embedded in the otml file, to make the example simple.
My new OT script object will be something like this:

The snippet above is to be added to the library section of the otml file.

Now, since I want my script to have access to my OTText object, I need to "pass" the object to it.
In order for your script to have access to OT objects defined in the otml file, you need to use the "variables" property of the OTScriptObject.
This is how it works for my example:

Now the script will have access to the object defined with an id of "text_object" in the otml, and within the script I can refer to it simply as "myTextVariable".

Adding the script to a container so it can be run

So far, the script object is defined, but if we run the otml file, it will not be loaded or run. That is because the root object on the otml file is a document, so the OTViewer will just display the document and that's it. It doesn't care about the rest of the otml file contents unless it needs to display it.
In this case, since we have a compound document in our root, and it is itself a container, we can add the script to the document directly. In a compound document, you can add other objects by adding a reference to them into the "bodyText" property, just like we added the OTText object to the document.
So, we have to edit the otml file adding the script object, with its local_id, which is "script_object":

Ok, now the script will be loaded because it is part of the root document.
We still need an extra step. When the OTViewer tries to load the OTScript object, it will look for a "view" to display in the screen next to the text object. If it doesn't find a view, it will fail to display, load and run the script.
So, we need to specify which view we want to use. The default view for OT script objects is the "OTScriptObjectView". With this view, you don't have to worry about *seeing* the script in the screen; it will be "invisible".
To specify the view to be used, add this view entry to the views section of the OTViewBundle in the otml:

Now the script will run when the document is displayed!

Adding necessary imports and engine entries

We cannot forget another step: in order to specify that we want to use the scripting engine included in OTrunk, we need to add an OT engine bundle to the OTSystem bundles. Since we're using JavaScript, we need to add this snippet to the bundles section of the OTSystem section:

We're almost ready to write the actual JavaScript. But before we can run the otml file, we need to add the necessary imports for all the new script objects we used:

Editing the script

Ok, finally! Up to this point, the otml can be loaded and run with the OTViewer. But the script won't do anything, since it's empty. So, we can now add our actual script, which will just set the text into our OTText object.
Replace the comment "//Here is where I will write my script" with this:

In this example, the script is accessing an OTText object directly, so it's using the setText() method on it.

Running the otml file with the script in it

That's it! Your otml is ready to be loaded and your script to be run. Just make sure that when you run your application, you add the jar files necessary for the JavaScript engine to your classpath. In case you need to manually add them, the jar files are located in the thirdparty project in cvs. For JavaScript, the files needed are: bsf-2.4.0.jar, js-1.6R5.jar and commons-logging.jar.

Running and looking at the full example online

You can download the otml file from svn at otrunk-examples/Script/script_basic_example.otml.

Look at the full example here

Run the full example here

Moving the script to an external file

Like it was mentioned before, it is recommended to write the script into a separate file. If it's JavaScript, you might want to give it the standard .js extension.

To specify that your script is in an external file, the OTScriptObject will look like this:

OTrunk SDS URL Generation

This page last changed on Jul 28, 2008 by stepheneb.

SAIL/OTrunk SDS URL Generation

Table of Contents

Five Parts of an SDS URL

There are five parts to a SDS url which can be used to run a SAIL/OTrunk session:

1. **SDS Jnlp URL** *(required)*
2. **SAIL/OTrunk OTML URL** *(required)*
3. **Jnlp Properties** *(optional)*
4. **Jnlp Filename** *(optional)*
5. **SAIL Session Properties** *(optional)*

[**SDS Jnlp URL**] ? sailotrunk.otmlurl = [**SAIL/OTrunk OTML URL**] & *jnlp_filename* = *[**Jnlp Filename**]* & *jnlp_properties* = *[**Jnlp Properties**]* & *[**SAIL Session Properties**]*

Example:

1. SDS Jnlp URL

(required)

This is the url of the jnlp for a valid workgroup and offering in an SDS portal realm.

Example:
`http://rails.dev.concord.org/sds/2/offering/144/jnlp/540/view`

2. SAIL/OTrunk OTML URL

(required)

The value of the **`sailotrunk.otmlurl`** parameter references the otml that will be used by this SAIL/OTrunk session.

The **`sailotrunk.otmlurl`** parameter and value are returned as part of the `<argument>` element of the jnlp rendered by the SDS and are passed to the SDS a second time where it is used to specify values in the config.

The value of the sailotrunk.otmlurl parameter **must** be escaped.

Example:

- **encoded in the jnlp**: `?sailotrunk.otmlurl=http%3A%2F%2Fcontinuum.concord.org%2Fotrunk%2Fexamples%2FUDL%2Fudl-clouds-`
- **decoded value**: `http://continuum.concord.org/otrunk/examples/UDL/udl-clouds-34.otml`

This is a representation of how the SAIL/OTrunk OTML URL is encoded as an element of the complete config url in the **`<argument>`** element in the jnlp: *(linebreaks added for clarity)*

The url itself then appears in the 5. SAIL Session Properties section of the SDS config file like this:

Right now the SDS requires this property be present even if you are also passing the otml url as a Jnlp Property.

Note: this part of the url is just a special case of a SAIL Session property see 5. SAIL Session Properties below.

3. Jnlp Properties

(optional)

The properties passed here are added as system properties in the first <resource> element in the jnlp itself.

The value of the **`jnlp_properties`** parameter is an escaped list of properties and values in url parameter form: `<key>=<value>`. If you are passing multiple parameters concatenate them with "&" characters".

The entire value of the **`jnlp_properties`** parameter **must** be escaped.

Example 1:

- encoded in the jnlp: **`&jnlp_properties=sailotrunk.hidetree%3Dfalse`**
- decoded value: **`sailotrunk.hidetree=false`**

Example2 :

- encoded in the jnlp: **`&jnlp_properties=otrunk.view.mode%3Dreporting%26otrunk.view.frame_title%3DUniversal%20Design%2`**
- decoded value: **`otrunk.view.mode=reporting&otrunk.view.frame_title=Universal Design in Science Education`**

Here's how Example 1 would appear in the jnlp:

Here's how Example 2 would appear in the jnlp:

4. Jnlp Filename

(optional)

This value will be used by the SDS to suggest a filename to the browser when it downloads and saves the jnlp.

The value of the **`jnlp_filename`** parameter **must** be escaped if it includes spaces (or any character that is used to parse urls).

Example:

- parameter in jnlp: **`&jnlp_filename=udl_grades_3_4_udl_clouds_sample_student_data_vernier_goio.jnlp`**
- suggested filename: **`udl_grades_3_4_udl_clouds_sample_student_data_vernier_goio.jnlp`**

Here are the headers that would be generated by the SDS:

5. SAIL Session Properties

(optional)

SAIL session properties are passed to the SAIL session in the XMLDecoder config file generated by the SDS. These properties may be used by the SAIL session operationally. In addition they are returned as key, value attribute pairs in `<launchProperties>` elements contained in the `<sessionBundles>` element returned by the SAIL session. See: The actual learner session sessionBundles sent for an example of `<launchProperties>` elements in an previous `<sessionBundles>` being sent to a SAIL session loads learner data.

The SDS always adds these SAIL session properties to the config:

- **`sds_time`**
- **`sailotrunk.otmlurl`**

- **`jnlp_properties`**

Any additional SAIL session properties you add will also be included.

When SAIL session properties are returned to the SDS in a `<sessionBundles>` element they will appear as key, value attribute pairs in `<launchProperties>` elements.

In addition to all of the SAIL session properties present in the config the **`maven.jnlp.version`** property of the jnlp is also included as a `<launchProperties>`.

The properties passed here are added as SAIL session properties in the config file.

The values of these parameter **must** be escaped if they contain characters that delimit properties in urls or are invalid in XML.

Values passed here should be in url parameter form: `<key>=<value>`. If you are passing multiple parameters concatenate them with "&" characters"

Example 1:

- appended to the end of the jnlp: **`&extra_property=1`**

Here's how Example 1 would appear in the config: *(note the property was added to the three default SAIL session properties)*

Example 2:

- appended to the end of the jnlp: **`&extra_property=1&sailotrunk.learnerotml.edit=true`**

Here's how Example 2 would appear in the config: *(note the property was added to the three default SAIL session properties)*

Practical Details

Escaping characters that delimit properties or are invalid in XML

When properties (`<key>=<value>` pairs) are passed as values of one of the standard properties the '&' and '=' characters need to be escaped. In general bare "&" characters should be escaped when rendered in XML. This is not a problem for the properties passed just to the SDS but when the SDS embeds the **`sailotrunk.otmlurl`** url in the jnlp **`<argument>`** element "&" characters must be escaped as **&** enitities *("&" + "amp;")*.

A Ruby script for OTrunk SDS URL generation

The following Ruby script will do the proper url escaping and construction:

Enter the Interactive Ruby console with the shell command: **`irb`** and paste the Ruby script above into the **`irb`**
console and press return. This defines a method on the main object called: **`make_sds_url`**.

Here are some examples of it's use.

Which constructs this url:

```
"http://rails.dev.concord.org/sds/2/offering/144/jnlp/540/view?sailotrunk.otmlurl=http%3A%2F%2F
continuum.concord.org%2Fotrunk%2Fexamples%2FBasicExamples%2Fdocument-edit.otml&jnlp_filename=
document-edit&jnlp_properties=sailotrunk.hidetree%3Dfalse"
```

Which constructs this url:

```
"http://rails.dev.concord.org/sds/2/offering/45268/jnlp/87346?sailotrunk.otmlurl=http%3A%2F%2Fwww.telscente
confluence%2Fdownload%2Fattachments%2F20047%2FHydrogen_Cars.otml&jnlp_properties=sailotrunk.otmlurl%3Dhttp%
www.telscenter.org%2Fconfluence%2Fdownload%2Fattachments%2F20047%2FHydrogen_Cars.otml"
```

Session Example

Requesting a Jnlp Session from the DIY

The DIY returns a redirect to the sds:

Asking the SDS for the actual Jnlp

The returned Jnlp

Because of confluences inability to wrap content in a block here is the url in the jnlp argument above

Using the Jnlp argument to request the Config

The actual Config file

The config file is an XMLDecoder Java serialization:

Requesting the learner session bundle

The previous learner session bundle is requested with urlString in the the portfolioUrlProvider:

The actual learner session sessionBundles sent

And the bundle itself is returned as an EMF xml serialization:

OTrunk Services

This page last changed on Feb 14, 2008 by sfentress.

There are 2 layers where services are used in OTrunk. At the object/model layer and at the view/controller layer.
This page focuses on the services for the object/model layer of OTrunk. Look at the OTrunk View Services page for information about the view/controller layer.
These services are stored in the OTSystem object, and are managed by the OTrunkImpl object

Accessing services

There are 2 ways to access these services:

- call the instance method OTrunk.getService(Class serviceInterface)
- add the service interface as a param to the constructor of a concrete OTClass. for example:

Adding services

There are 2 ways to add services to the system so they are available.

- Make OTSystem be the root object in the otml file, and add an OTObject which implements OTBundle in the bundles property of the OTSystem. There is also a deprecated services property in the OTSystem object. If you are making new otml files use bundles instead of services.
- When creating the OTrunkImpl object, an array of Objects and array of Classes can be passed which are then added to the service map.

Creating a new service

The best way to add a new service at this time is to make a new class that extends DefaultOTObject and implements OTBundle. Well call this a new bundle. The new bundle should create and add the services it provides during the registerServices call.

Current service usage

OTrunk Services being used:

- OTViewFactory - this is provided by the OTViewService bundle
  - OTViewer
  - OTViewerHelper
  - PfViewPrinter
  - OTReportViewer
- SensorDataManager - this is provided by the OTInterfaceManager bundle
  - OTLoggerImporter
  - OTLoggerRecordChooser

    - OTSensorDataProxy
    - OTSeuptLogger
- OTScriptManager - this is provided by the OTScriptEngineBundle
    - OTScriptButtonView
    - OTScriptObjectView
- UserMessageHandler - this added through the OTrunkImpl constructor
    - OTInterfaceManager - this is the class implementing SensorDataManager
- OTSharingManager - this is provided by the OTSharingBundle

OTrunk Session Forensics

This page last changed on Jan 30, 2008 by stepheneb.

A set of pages showing OTrunk sessions for forensic analysis:

- OTrunk Session UDL 20080130

OTrunk Session UDL 20080130

This page last changed on Jan 31, 2008 by stepheneb.

A session from California running on the local UDL server:

See also attached Excel spreadsheet: udl_session_20080130-2.xls showing http logs for a similar session conducted with the main CC UDl Portal on a high latency (satellite) medium-speed network connection.

After the workshop had been going on for about 1:20 I asked the vice principal Dan Meyer to test whether the "Install Embedded Flash Support" link on the sign in page of the udl portal was working. This link runs the mozswing.jnlp whch in turn runs the mozswing_xulrunner.jnlp which is what actually installs the native libraries that allow Java to embed Mozilla components.

He went to a computer that had ben used in the workshop. Every time he tried the link he got a java web start exception. The problem was caused by an error in the actual jnlp on the server. The confusing part here was that I had already corrected this error and tested the correction. He was using the Firefox browser and he even quit and restarted and had the same problem. He needed to take more drastic action to clear the cache. We also need to look at the HTTP headers we use to see if we can more forcefully explain to Firefox not to cache this resource.

I asked Dan to try on another computer that hadn't run the "Install Embedded Flash Support" link. He went down the hall to his laptop and tried it there. The IP address for his computer is: 172.24.109.42.

This is an annotated log from the web server that shows all the web requests. Because all the request are proxied through Apache tis log shows requests to he PHP Portal, the Rails DIY and SDS, and the Tomcat jnlp-jar server.

Dan already had the portal open on his computer and to get to the "Install Embedded Flash Support" link he had to first sign-out. I don't think it makes much sense to have to sign out just to get to the "Install Embedded Flash Support" link

```
172.24.109.42 - - [30/Jan/2008:19:56:37 -0500] "GET /signin/?signout HTTP/1.1" 200 4458
"http://172.24.108.13/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"}}
```

The next 12 connections render the UDL Portal sign-in page:

```
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/reset-fonts-grids.css HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/utilities.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/yahoo-dom-event.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

```
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/datasource-beta-min.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/datatable-beta-min.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/jquery-1.2.min.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/jquery.tablesorter.pack.js HTTP/1.1" 304
- "http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /css/udl.css HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /scripts/portal.js HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /images/udl-soft-tree.jpg HTTP/1.1" 304 -
"http://172.24.108.13/css/udl.css" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /images/students-using-portal.jpg HTTP/1.1" 304
- "http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /images/udl-logo-150.gif HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:38 -0500] "GET /images/cc_logo_gray_text.gif HTTP/1.1" 304 -
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

The first request for the initial jnlp: mozswing.jnlp, is made by the browser and immediately followed by the Javaplugin requesting the same file again:

```
172.24.109.42 - - [30/Jan/2008:19:56:43 -0500] "GET /jnlp/mozswing/mozswing.jnlp HTTP/1.1" 200
1075 "http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:56:45 -0500] "GET /jnlp/mozswing/mozswing.jnlp HTTP/1.1" 200
1075 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
```

The first thing Java does after receiving mozswing.jnlp is to also request the jnlp extension installer: mozswing_xulrunner.jnlp.

```
172.24.109.42 - - [30/Jan/2008:19:56:46 -0500] "GET /jnlp/mozswing/mozswing_xulrunner.jnlp
HTTP/1.1" 200 1439 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
```

Eight of the next ten requests are loading the Java jar files for both the mozswing and mozswing_xulrunner jnlps. What's strange is that two more requests for mozswing_xulrunner.jnlp are sandwiched in this set also.

```
172.24.109.42 - - [30/Jan/2008:19:56:46 -0500] "GET /jnlp/mozswing/mozswing.jar HTTP/1.1" 200
304178 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:56:47 -0500] "GET /jnlp/mozswing/commons-logging-1.1.jar
HTTP/1.1" 200 57757 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:56:48 -0500] "GET /jnlp/mozswing/mozdom4java.jar HTTP/1.1" 200
807787 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:56:53 -0500] "GET /jnlp/mozswing/MozillaInterfaces-1.8.1.4.jar
HTTP/1.1" 200 671787 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:56:57 -0500] "GET /jnlp/mozswing/mozswing-xulrunner-win32.jar
HTTP/1.1" 200 6656490 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:57:16 -0500] "GET /jnlp/mozswing/commons-logging-1.1.jar
HTTP/1.1" 200 23788 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:57:17 -0500] "GET /jnlp/mozswing/mozswing_xulrunner.jnlp
HTTP/1.1" 200 1439 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:57:17 -0500] "GET /jnlp/mozswing/mozswing_xulrunner.jnlp
HTTP/1.1" 200 1439 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:57:17 -0500] "GET /jnlp/mozswing/mozswing-xulrunner-win32.jar
HTTP/1.1" 200 23786 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:57:17 -0500] "GET /jnlp/mozswing/commons-logging-1.1.jar
HTTP/1.1" 200 23788 "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
```

34 seconds elasped from when Dan clicked on the initial "Install Embedded Flash Support" link to when all the jars had been delivered. I am assuming that the time in these logs indicates when the request was successfully completed. It could be when the request was received.

An additional 40s elapsed from when all the jars were loaded to when Dan clicked on the sign-in link. We can't tell from the logs how much time it took the Java program to complete after all the jars were loaded. Presumably less than 40s.

```
172.24.109.42 - - [30/Jan/2008:19:57:30 -0500] "POST /signin/process/ HTTP/1.1" 302 291
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

I don't have an explanation for this request which appears to sign him out – I don't think it actually did:

```
172.24.109.42 - - [30/Jan/2008:19:57:30 -0500] "GET / HTTP/1.1" 200 5251
"http://172.24.108.13/signin/?signout" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

The next three requests render the UDL Portal page for Dan as a teacher and displays activities for him to preview.

```
172.24.109.42 - - [30/Jan/2008:19:57:31 -0500] "GET /images/icons/picture.png HTTP/1.1" 200 606
"http://172.24.108.13/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:57:31 -0500] "GET /images/icons/page_white_copy.png HTTP/1.1"
```

```
200 309 "http://172.24.108.13/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
172.24.109.42 - - [30/Jan/2008:19:58:03 -0500] "GET /activities/ HTTP/1.1" 200 12059
"http://172.24.108.13/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

This request is to the UDL Portal asking for the grade 3-4 Plants activity.

```
172.24.109.42 - - [30/Jan/2008:19:58:06 -0500] "GET /2diy/view/24/ HTTP/1.1" 302 552
"http://172.24.108.13/activities/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11"
```

The UDL Portal responds with a re-direct to Dan's browser and Don's browser is now requesting that the DIY return the jnlp file for running the UDL 3-4 plants activity

```
172.24.109.42 - - [30/Jan/2008:19:58:06 -0500] "GET
/diy/external_otrunk_activities/b895f8f0-c443-11dc-923b-0014c2c34555/sail_jnlp/11/6/view HTTP/1.1"
302 299 "http://172.24.108.13/activities/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11"
```

The log is actually out of order here. The next request is from the SDS requesting the template jnlp for all UDL activities. I do not know why the second HEAD request was made for the same resource. The first request is from the Ruby net/http library. The user-agent is "Ruby" – this is actually a patch in the library I created to deal with a bug in the Tomcat web server which responds with a 500 error to a GET request of this type without a user-agent in the HTTP request. The second request almost certainly is from the SDS Ruby process also but not the part I patched.

```
127.0.0.1 - - [30/Jan/2008:19:58:07 -0500] "GET
/jnlp/org/concord/maven-jnlp/udl-otrunk/udl-otrunk.jnlp HTTP/1.1" 200 8010 "-" "Ruby"
127.0.0.1 - - [30/Jan/2008:19:58:07 -0500] "HEAD
/jnlp/org/concord/maven-jnlp/udl-otrunk/udl-otrunk.jnlp HTTP/1.1" 200 - "-" ""
```

The following request actually occurred before the previous two requests. The DIY doesn't actually generate the jnlp it instead return's another re-direct url to Dan's browser which then calls the SDS with a bunch of parameters. The SDS uses both parameters in the path of the url and parameters passed as a CGI parameter string (after the '?' character) and combines these with the template jnlp to create a custom jnlp for Dan. In this url the presence of the string 'view' just before the '?' indicates to the applications performing these services that no learner data will be saved in this session.

```
172.24.109.42 - - [30/Jan/2008:19:58:06 -0500] "GET
/sds/1/offering/87/jnlp/94/view?sailotrunk.otmlurl=http://172.24.108.13/diy/cache/b895f8f0-c443-11dc-923b-0014c
HTTP/1.1" 304 - "http://172.24.108.13/activities/" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11"
```

Normally what would follow if this was Dan's first time running this program would be about 40MB of Java jar files which would be stored in Dan's web start cache. In this case he had run a UDL activity earlier in the day and because all of the jars referenced in the jnlp had the same version as those already stored in his cache none of these requests were made.

The next request comes from the Java web start code (javaws) and requests the image icon that is associated with this application.

```
172.24.109.42 - - [30/Jan/2008:19:58:10 -0500] "GET /sds/images/udl-tree-64x64-transparent.png
HTTP/1.1" 304 - "-" "JNLP/6.0 javaws/1.6.0_03 (b05) Java/1.6.0_03"
```

This time Java doesn't request the jnlp twice. The next request is for the config file for the SAIL learner session. This tells the Java application where to load and save Dan's learner data for this activity, where the curnit (curriculum unit) is located as well as describing other services and parameters. In this instance Dan is just previewing the activity so no data will be sent back.

```
172.24.109.42 - - [30/Jan/2008:19:58:12 -0500] "GET
/sds/1/offering/87/config/94/1/view?sailotrunk.otmlurl=http%3A%2F%2F172.24.108.13%2Fdiy%2Fcache%2Fb895f8f
HTTP/1.1" 200 1845 "-" "Java/1.6.0_03"
```

The next request is for the curnit. In other SAIL implementations the curnit can hold all the resources used in an activity. In this case the curnit is really just a stub because the activity otml will be provided by the DIY.

```
172.24.109.42 - - [30/Jan/2008:19:58:12 -0500] "GET
/sds/cache/1/curnits/1/otrunk-curnit-external-diytest.jar HTTP/1.1" 200 1531 "-" "Java/1.6.0_03"
```

This request is for any previous learner data.

```
172.24.109.42 - - [30/Jan/2008:19:58:13 -0500] "GET /sds/1/offering/87/bundle/94/1 HTTP/1.1" 200
158 "-" "Java/1.6.0_03"
```

All of the following requests that start with the path: /diy/cache are for static resources that have been copied to the local UDL server.

The next two requests are for the otml that structure the activity. The first is the main body of the activity and the second is for a template for sensor graphs.

```
172.24.109.42 - - [30/Jan/2008:19:58:13 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/b895f8f0-c443-11dc-923b-0014c2c34555.otml
HTTP/1.1" 200 425300 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:15 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/global-imports/predefined-sensor-graphs.otml
```

```
HTTP/1.1" 200 20202 "-" "Java/1.6.0_03"
```

This section of the requests are for resources needed by the running UDL activity. There is some obvious in-efficiencies visible that can presumably be easly corrected. In the next 25s many resources are requested and delivered over and over.

Some examples:

```
    resource                times requested
----------------------------------------------------
  css/otrunk-34-color.css        29
  css/otrunk-normal-font.css     27
  css/otrunk-normal-layout.css   27
  glossary/common-glossary.txt   14
  glossary/plants-glossary.txt   14
```

It didn't appear that requesting these resources over and over slowed down Dan's learner run very much however on a slower network with latency issues this would dramatically slow the program down. When I open this activity from home it takes just a bit more than one minute to process the approximately 50 requests processed to render the first page (39 of these requests are to just five objects):

This is definitely a bug in OTrunk that should be fixed.

```
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/scripts/authoring_script.js HTTP/1.1" 200 6871 "-"
"Java/1.6.0_03"
```

These two requests are made by embedded Java components for resources that are not actually needed. Both of these requests are made from third-party code that was not created by Concord.

```
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/jnlp/org/concord/external/phet/phetcck/jmock-1.1.0.jar HTTP/1.1" 404 952 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET /jnlp/org/concord/nlogo/netlogo4/NetLogoLite.jar
HTTP/1.1" 404 952 "-" "Java/1.6.0_03"
```

The next 102 requests are often for the same resources over and over. Dan sequenced through the two pages of the pre-test. The first page of the next section and then immediately jumped to the Food-o-meter section to test the Flash embedding:

```
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
```

```
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:16 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/blah-head.png
HTTP/1.1" 200 56711 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/blah-head.png
HTTP/1.1" 200 56711 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/clunk-head.png
HTTP/1.1" 200 27671 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/clunk-head.png
HTTP/1.1" 200 27671 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/roxy-head.png
HTTP/1.1" 200 7333 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/roxy-head.png
HTTP/1.1" 200 7333 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/roxy-head.png
HTTP/1.1" 200 7333 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:17 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/udl-tree-only.png HTTP/1.1"
200 27331 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/clunk-head.png
HTTP/1.1" 200 27671 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/share/shared-images/blah-head.png
HTTP/1.1" 200 56711 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
```

```
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:18 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
```

```
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:19 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:21 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/scripts/change_page.js HTTP/1.1" 200 1477 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/pretest_q3a.png
HTTP/1.1" 200 58038 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/jnlp/org/concord/external/phet/phetcck/jmock-1.1.0.jar HTTP/1.1" 404 952 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET /jnlp/org/concord/nlogo/netlogo4/NetLogoLite.jar
HTTP/1.1" 404 952 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:24 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/pretest_q3b.png
HTTP/1.1" 200 169530 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
```

```
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/answerA.png
HTTP/1.1" 200 32229 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/answerB.png
HTTP/1.1" 200 29612 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/answerC.png
HTTP/1.1" 200 29071 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:25 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/answerD.png
HTTP/1.1" 200 31554 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:26 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/pretest/answerE.png
HTTP/1.1" 200 31724 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
```

```
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:30 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/intro/pl_34_intro.png
HTTP/1.1" 200 59693 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:31 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/intro/pl_34_intro.png
HTTP/1.1" 200 59693 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
```

```
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:34 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/observe_garden/pl_34_observ
HTTP/1.1" 200 59736 "-" "Java/1.6.0_03"
```

Dan's now viewing pages 1..3 of the Food-o-Meter section:

```
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
```

```
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 27080 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:38 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
```

```
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 5852 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 7335 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 7373 "-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:40 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 19387 "-" "Java/1.6.0_03"
```

The next set of requests are when the food-o-meter model should have been loaded in page 4.

The otml for doing that looks like this:

```
<OTMozSwing id="908edbdf-cf5a-11dc-8ff1-37cf223ff3c1" name="Web browser"
url="udl.concord.org/share/Plants/food-o-meter/index.htm" showToolbar="false" showStatusBar="false"
preferredHeight="430" />
```

But the resource index.htm never appears in the http log:

```
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/common-glossary.txt HTTP/1.1" 200 1418
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/glossary/plants-glossary.txt HTTP/1.1" 200 139 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-font.css HTTP/1.1" 200 581 "-"
"Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-34-color.css HTTP/1.1" 200 1038 "-"
"Java/1.6.0_03"
```

```
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/css/otrunk-normal-layout.css HTTP/1.1" 200 726
"-" "Java/1.6.0_03"
172.24.109.42 - - [30/Jan/2008:19:58:41 -0500] "GET
/diy/cache/b895f8f0-c443-11dc-923b-0014c2c34555/udl.concord.org/artwork/plant_34/food-o-meter-screenshots/pl_
HTTP/1.1" 200 4865 "-" "Java/1.6.0_03"
```

OTrunk Testing

This page last changed on Jul 25, 2007 by scytacki.

BasicExamples

This page last changed on Jul 24, 2007 by bcard.

Items in red are bugs and were entered into jira

text-label

- text is centered on left hand side
- text is not selectable
- text is not editable

choice

- mouse and keyboard functionality
- multiple items not selectable via ctrl or shift

text-edit

- text is selectable
- text is editable
- tab and return work
- non alphanumeric characters work

choice-authoring

- items can be added and removed
- multiple items not selected via shift or ctrl
- duplicate names cannot be created
- scroll bar appears when items fill screen

snapshot_album_example [https://jira.concord.org/browse/OTK-18]

- picture links broken, unable to test

otobject_chooser [https://jira.concord.org/browse/OTK-19]

- linked files (should) work

basic_drawing [https://jira.concord.org/browse/OTK-20]

- tooltips work
- each of the objects can be created and moved
- vector objects can have different colors
- text background can be changed

css-example

- graphs and drawing areas work
- text is displayed directly

document-text

- label text in not selectable
- edit text is editable and accepts non alphanumeric characters
- Edit text does not accept formating like tab and return, might be intended.

choice-radiobuttons

- all choices can be selected
- 2+3 give feedback
- 1 and three have the mirrored buttons

layout-example

- card layout displays the object of the button pressed

- placement layout has the objects staggered
- border has them in a frame up top

Biologica Examples

This page last changed on Jul 24, 2007 by bcard.

items in red are bugs and were entered into jira

biologica-static-organism-screen

- all possible traits tested, not all possible pairs of traits.

[biologica-meiosis-screen|http://jira.concord.org/browse/OTK-21]

- nothing loads but a replay button, console complains about calling method "getWorld"

biologica-multiple-pages

- meiosis loads and works fine, it probably needs to get passed the information from static-organism-screen to run properly
- tested all of the simulations
- tested the 'challenges'

Calculator

This page last changed on Jul 24, 2007 by bcard.

items in red have bugs and are on jira

calculator https://jira.concord.org/browse/OTK-22

- on ExpressionList, list doesn't accept non alphanumeric characters
- checked the edit button: if the input box is blank the edit is effectively canceled, it checks for invalid formulas.
- ExpressionSymbolList, ExpressionConstantList, and Programmable Calc provide no interactive functionality.
- Add measurement and calc test buttons don't work, tested adding and deleting from the text fields.
- Text fields accept non alphanumeric characters and formatting (tab, return)

calculator_cck

- test dragging the objects, splitting connections, changing the resistance of resistors, adding things from the grab bag, checking voltages.
- clicking in empty space and on objects, confirming voltage drops.
- calculator functionality?

OTrunk View Services

This page last changed on Feb 14, 2008 by sfentress.

There are 2 layers of services in OTrunk. The object/model layer and controller/view layer. This page deals with the controller/view layer for information on the object/model layer look at this page: OTrunk Services

Accessing Services

If your view is a subclass of AbstractOTView then services can be accessed by calling the method:
getViewService(Class serviceClass);
The serviceClass should be the class or interface of the service you are looking for.

If you view doesn't subclass AbstractOTView, then it needs to implement the plumbing so it can look up services. This is done by:
implementing OTViewContextAware - this requires your class to have a setViewContext method.
then when you want a service you call getViewSerivce on the passed in OTViewContext.

Adding Services

There are 2 ways to add view services.

1. using the OTBundle mechanism
2. using the OTViewContext object

Create a new bundle or modify an existing bundle. See OTrunk Services for information about bundles. Then in the initializeBundle method of your bundle

get the viewFactory from the serviceContext and add your new view service. For example:

```
```

Add the service to the view context.

If your view gets its own OTViewContext and uses the addViewService method then that service will be available to your view and all of its siblings. The service will not be availabe to any parents of this view. There are 3 cases where this might be used:

1. a child view wants sibling views to have access to a service it provides. In this case, the child will get its view context as described in "Accessing Services" above. Then call the addViewService method.
2. a parent view wants to provide a service to its children. In this case the parent view will have created a OTViewFactory inorder to create its child views. The viewContext that will be passed to these children can be accessed using the getViewContext on the otViewFactory instance. Then the addViewService can used to add the service to that.
3. modifying the top level startup objects: OTViewer, OTViewerHelper, and OTReportViewer In this

case, you going to add your new service to the top level viewFactory. Here is the snippet from OTViewer where these services are being setup:

```
```

Current View Services

OTrunk - The OTrunk object itself is added as a view service
OTFrameManager - is added so views can make new frames
OTJComponentServiceFactory - is added so views get JComponents from other views. There is some setup needed when accessing the JComponent from another view and this service takes care of that for you.
OTScriptManager - this allows a view to execute scripts.
OTControllerServiceFactory - this allows a view to use the OTController system. This system helps connect POJOs to their OTObject through controllers.
OTSharingManager - this allows objects to be shared across other obejcts

Nested OTViewFactories

Multiple OTViewFactories are created. This is used a few ways.

1. Each view that has children views creates a nested OTViewFactory to create those children views. This allows specific services to be added to just those views, and it allows those views to look up each other.
2. It is also used by the OTReportViewer. It allows it to show an aggregate view of the object in one place and single user view in another.

As far as services are concerned, first the service is searched for in the current view factory and if it has a parent then the service is looked for in the parent, and if it has a parent it is looked for there.

OTrunk notes for refactoring view services

This page last changed on Jul 16, 2007 by scytacki.

The current view service system has been getting hacked on additions for a few months now, and it is to the point where it should be refactored. It seems like there are several places where there are redundant references. And the mechanisms by which the view services get setup is looking fragile.

These notes are based on a slightly refactored version of the code which has not been checked in yet.

New Design

Each view should have:

- a viewContainer which allows the view replace itself with another view.
- a viewContext which allows:
    - access its sibling views
    - method to create a viewFactory which can be used to make children views
    - access to view services. This will replace the viewServiceProvider.

- if it creates a viewFactory
- its own viewFactory for creating sub views of referenced objects.
    - this viewFactory will maintain references to these sub views so it can provide a viewContext to them.

Those three items can all be services or provided by services:

- viewContext can directly be a service, it will need to be created whether it is used or not.
- viewFactory can be created on demand from a viewFactoryFactory (or something like that)
- viewContainer can directly be a service,

The current JComponent specific versions of this same functionality should be modified to use the more abstract version.

Notes

Currently the view services are maintained by instances of OTViewFactoryImpl
OTViewFactoryImplS can have a parent OTViewFactoryImpl.
In this case the child first looks for services in itself and then in the parent.

View services are looked up using an OTViewServiceProvider interface which a view can access by implementing the OTViewServiceProviderAware interface.

The OTViewServiceProvider interface is implemented by an OTViewServiceProviderImpl class inside of OTViewFactoryImpl.

The OTViewFactoryImpl supplies the viewServiceProvider during the initView method.

Which is in turn called from the getView methods of the OTViewFactoryImpl class.

The main OTViewFactoryImpl is only instanciated in the registerServices call of the OTViewBundle class. All other instances are created by the createChildViewFactory method.

The viewFactory itself is added as a service to itself.

So views can access their viewFactory through the their viewServiceProvider.
this is a confusing design - it would be better if the viewFactory didn't manage the viewServiceProvider. Then the viewFactory could be seen as just another service views can use.

The main use of the viewFactory is behind the OTViewContainerPanel. This is the class that most compound views use to display their children views.

The setCurrentObject method in OTViewContainerPanel indirectly calls getView on its viewFactory.

The OTViewContainerPanel now has a notion of being a topLevel container. In this case it maintains a shared controllerService for its view. The shared controller service is setup when the setOTViewFactory method is called on the OTViewContainerPanel.
In topLevel mode it also adds a new OTJComponentServiceFactory service which wraps the passed in viewFactory. Because this OTJComponentServiceFactory is how most compound views get their sub views this service needs to duplicated so the subviews have the correct viewFactory and hence the correct set of services. **that could be fixed by requiring the user to pass in the viewFactory when calling OTJComponentServiceFactory.createJComponentService**

the setOTViewFactory is called in 7 places. In several places the viewFactory set is from getChildViewFactory. This is how the caller can setup a nested set of services.

In the OTUDLSidebar case this nested factory is inside of another view so the service nesting matches the view containment hirearchy.
In the other cases the factory is used for a real top level container.

The OTViewContainerPanel also has the concept of parent containers. This is to support the new additions to the OTViewContainer interface that allows a view to look up the nearest "updatable" container. An updatable container is one whose current object can be changed or updated. This type of nesting of the containers is setup by the OTJComponentContainerHelper which knows about the current container and is used by objects to make sub containers.

Another key element is how the OTJComponentServiceFactory is used. Objects that need to make sub views use its createJComponentService method to make a "sub" component service. For one thing this alows the sub views to share a common OTJComponentViewContext so they can access their peers.

OTrunk View System

This page last changed on Apr 29, 2007 by scytacki.

Disclamer

This system is in flux. What is describe here is the way it exists right now. I'm pretty sure it will have to change to deal with how we are going to use it.

Terminology

I used the dash notation so these terms aren't confused with classes. Some of them have classes or interfaces but not all of them.

- **ot-object** the core building block of OTrunk.
- **ot-class** this is the class of an ot-object. Currently it is declared as a java interface, but it is intended to be platform and language independent.
- **ot-view** this is a "view" of an ot-object. Call it a "view" is a bit confusing, because it more like a controller in the model-view-controller pattern. An ot-view returns an rendered-object which is what is actually displayed. ot-views are created using the view-factory by calling the getView method. There is an OTView interface in java which marks objects which are ot-views.
- **rendered-object** the object or string the ot-view returns for display. Currently this can be a JComponent or a string of xhtml text.
- **view-requester** the object calling getView. Typically this object is another ot-view that contains other ot-views. For example the ot-view which displays compound documents (OTCompoundDocView, OTrunkViewer, OTrunkAppletViewer, OTMultipleChoice, OTGraph can ask for plugins) is a view-requester. This term is not used in the code, but it helps for describing the system.
- **view-factory** this object provides views to view-requesters. The java interface OTViewFactory defines the methods a view-factory must implement.
- **view-entry** this is how a ot-view is defined and configured in an otml file. It has the class name of the ot-view object and the class name of the ot-object it can be a view for. An ot-view can access its view-entry view-entry throw the OTViewEntryAware interface. The view-entry can be extended so further configuration options can be passed to the ot-view. view-entries can also be referenced directly, so the same ot-class can have multiple views.
- **view-mode** this is used so same view can be rendered in multiple way. Examples of view-modes are "author", "author-embed", and "report". A view mode is defined in the otml file and contains a default view-entry, and a set of mapped view-entries. A view mode is used by passing its name to the view-factory getView method. The view-factory then looks up the view-entry using the other parameters of teh getView method and then looks in the view-mode object for the cooresponding view-entry.
- **view-service** this is service which can be used by a view. It is added to the view-factory, and then available to all the views created by that view factory.

View Lookup

When a view-requester asks for a ot-view of an ot-object which ot-view does it get?

View Factory Context

There can be multiple view factories in a hierachy. This is currently used by the report tool. A few new view entries are added to the view-factory, and a user-list service is also added. These new view-entries override the default entries. For example a multiple choice view is rendered as a table showing each student as a row and the choices as columns.

a root to store particular view documentation under:
OTViews

a review of an older version of the system
OTrunk View System Analysis

Making an OTView Editable

This page last changed on Feb 16, 2008 by stepheneb.

Inorder to support the reuse and referencing of the same OTObject multiple ways to view it are necessary.
A simple version of this is an non editable view, and an editable view.

Because of the way views are managed, you need to define 2 OTViewEntry object one for the non editable view and one for the editable view. Then you need to reference the correct viewEntry when you are displaying the object.

Create necessary Java class

Currently you need to make Java classes to support this unless there are already classes that do what you need.

Custom OTViewEntry

In this case you make a new interface that extends OTViewEntry. In your new interface you define properties which make the view editable or not.
An example of this is OTUnitValueView.

Second OTView class

In this case you make a sub class of your original OTView class. If the original class was editable then your sub class makes it non editable or vice versa. An example of this is: OTDrawingToolView and OTDrawingToolNonEditableView

Defining and Referencing the OTViewEntry objects

Define 2 viewEntries and use viewid attribute

The 2 viewEntries can be defined in the viewEntries element of OTViewBundle. However they will point to the same OTClass, so the first one in the list will always be used. So it is more clear to put the default one in the viewEntries list, and put the secondary one in the OTSystem.library or some where else that is appropriate. The secondary one needs an local_id (or id), so you can reference it later.

Then in the bodyText of an OTCompoundDoc object, you use the viewid attribute to reference your particular view. Something like:

```
```

If you need to reference the object and you aren't using a OTCompoundDoc, then you can try using the OTViewChild object. You can put this inside of many of the container views and they will use the viewid specified.

TODO
need example of this and list of containers which support it

Define 2 viewEnties and use the ViewMode system

You can also use OTViewMode objects to group a set of views together. In this case you define one OTViewEntry in the OTViewBundle.viewEntries element. And then define an OTViewMode in the OTViewBundle.modes element. Inside the OTViewMode.map element you need to put an "entry" element with a "key" attribute with the value of the default OTViewEntry. Then inside of the entry element put the OTViewEntry which should replace that one in the mode.

Now you can set the mode of a particular view, either as a attribute in the OTCompoundDoc or through a custom OTViewEntry. The OTCompoundDoc does this with the OTDocumentView which uses a OTDocumentViewConfig to set the view mode.

Define the view entry in Java

Another option is to create the OTViewEntry object directly in java. And then passing that to the appropriate getComponent or createComonent method. This is currently the easiest way for a parent view to specify which child view should be used to display one of its child views. This approach makes it hard to customize though.

Examples

UDL WYSIWYG Authoring

The otml for these examples authoring system can be found here

- svn: http://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-examples/UDL
- trac: http://trac.cosmos.concord.org/projects/browser/trunk/common/java/otrunk/otrunk-examples/UDL

I recommend first looking at the blank authoring template

This references several other files using statements like the following:

```
```

The udl-view-bundle.otml import is a good place to look for examples of how we are using modes. You will see the same otml object mapped to a number of different view classes using OTViewModes:

- student
- authoring
- reporting
- printing

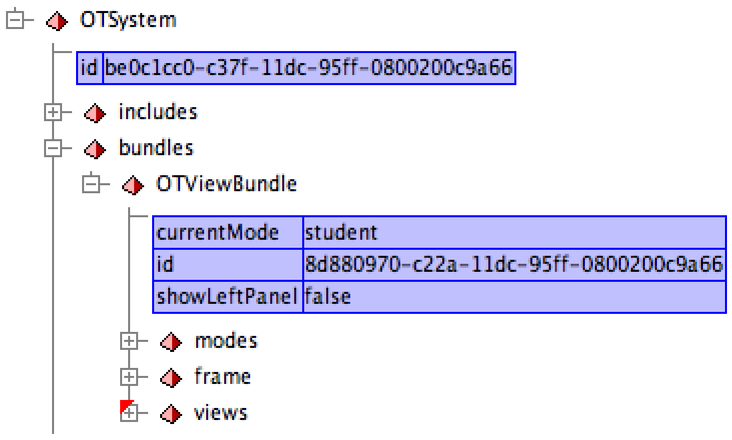
Data Graph Authoring (see this thread)

- svn: http://svn.concord.org/svn/projects/trunk/common/java/otrunk/otrunk-examples/DataGraphs/data_graphs_auth
- trac: http://trac.cosmos.concord.org/projects/browser/trunk/common/java/otrunk/otrunk-examples/DataGraphs/data_

I recommend taking a look at these otml files in a structural xml editor.

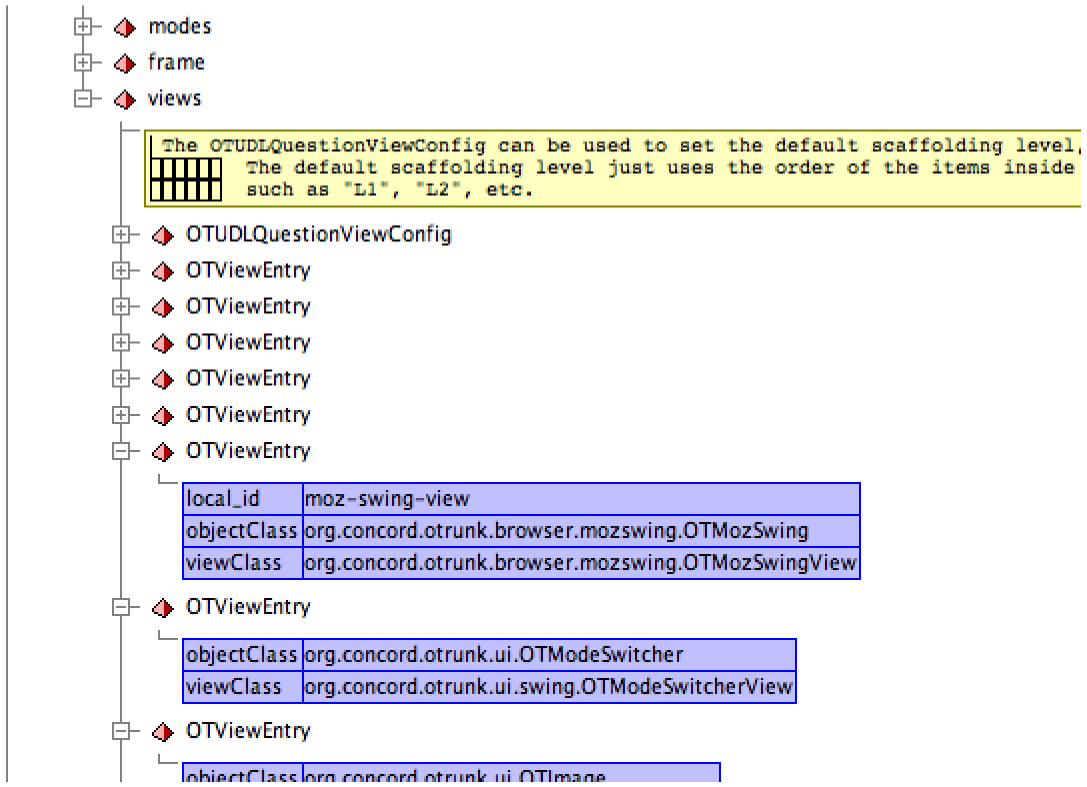
Here are some screen shots using the free xml editor xmlmind:

The OTSystem/bundles element includes the following elements: **modes**, **frame**, and **views**.



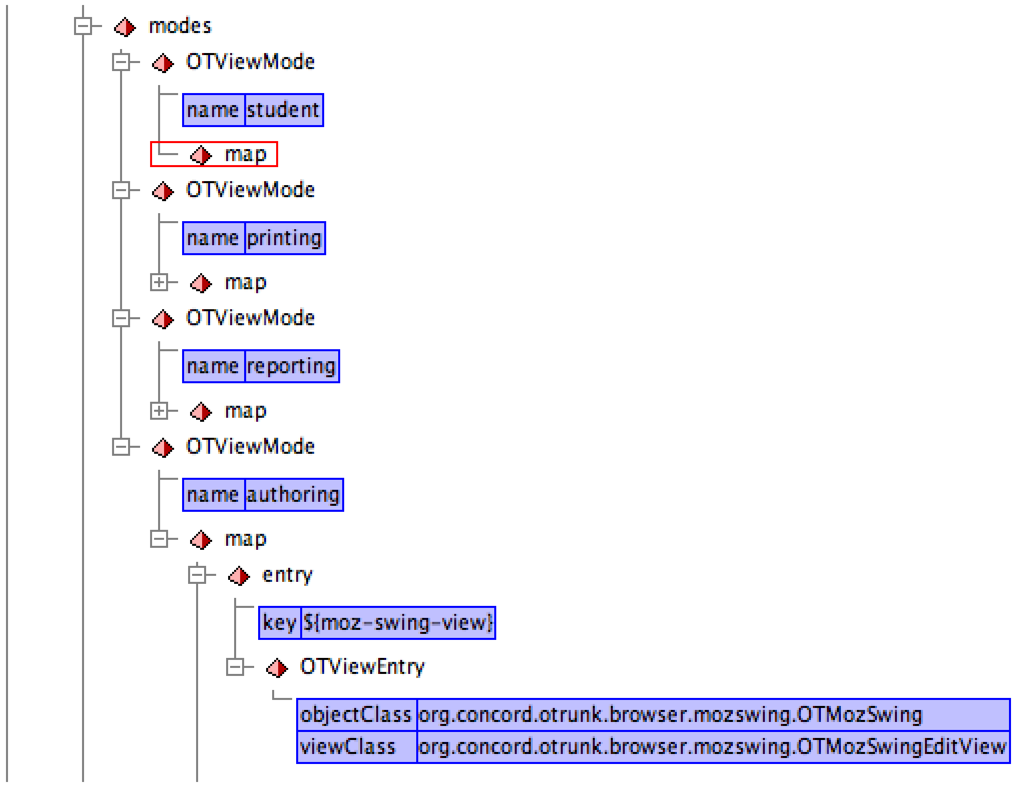
```
OTSystem
    id  be0c1cc0-c37f-11dc-95ff-0800200c9a66
    includes
    bundles
        OTViewBundle
            currentMode    student
            id             8d880970-c22a-11dc-95ff-0800200c9a66
            showLeftPanel  false
            modes
            frame
            views
```

The views element has a series of OTViews that match object and view classes. These are the default matching of otrunk objects and view classes used when another active view mode isn't in effect for that specific otrunk object class. Below you can see that the default mapping (used for student mode) maps the object class OTMozSwing with the view class OTMozSwingView.



```
modes
frame
views
  The OTUDLQuestionViewConfig can be used to set the default scaffolding level,
      The default scaffolding level just uses the order of the items inside
      such as "L1", "L2", etc.
  OTUDLQuestionViewConfig
  OTViewEntry
  OTViewEntry
  OTViewEntry
  OTViewEntry
  OTViewEntry
  OTViewEntry
    local_id     moz-swing-view
    objectClass  org.concord.otrunk.browser.mozswing.OTMozSwing
    viewClass    org.concord.otrunk.browser.mozswing.OTMozSwingView
  OTViewEntry
    objectClass  org.concord.otrunk.ui.OTModeSwitcher
    viewClass    org.concord.otrunk.ui.swing.OTModeSwitcherView
  OTViewEntry
    objectClass  org.concord.otrunk.ui.OTImage
```

The modes element of the OTViewBundle includes a series of OTViewModes. Each one consists of a name and a map where a key is matched with an altername OTViewEntry. In the example below you can see that in the authoring view mode the object class OTMozSwing is matched with the OTMOzSwingEditView view class.



```
modes
  OTViewMode
    name student
    map
  OTViewMode
    name printing
    map
  OTViewMode
    name reporting
    map
  OTViewMode
    name authoring
    map
      entry
        key ${moz-swing-view}
        OTViewEntry
          objectClass  org.concord.otrunk.browser.mozswing.OTMozSwing
          viewClass    org.concord.otrunk.browser.mozswing.OTMozSwingEditView
```

Future Changes

I'm going to be working on the view system and these mechanisms will hopefully get easier to use. They currently are not flexible enough, and hinder programmers from easily creating and customizing activities.

OTrunk Old View System Analysis

This page last changed on Apr 30, 2007 by scytacki.

The view/controller system in OTrunk is a bit tangled up right now. Here is a list of its different parts and how they are used. This is excluding the OTController classes which are really similar in functionality to these classes but they haven't been merged yet.

Problems

- views are really controllers. So most of these classes which start with OTView really should be OTController.
- **swing specific** - many of these classes are specific to using swing JComponents. This framework should support swt and xhtml based views.
- overlap between the OTViewContainer and OTViewFactory.
- **view lookup** - looking up views too restrictive once there are several views for the same object.

Interfaces

- OTViewContainer - an interface to the container of a view.
- OTViewFactory - creates views from objects given the type of view.
- OTViewService - the ot object used to configure a view factory in the otml file.
- OTViewContainerListener - notifies when the current object of a container changes
- OTView - parent of all views
- OTObjectView - creates and synchronizes a JComponent from an ot object
- OTXHTMLView - generates xhtml from an ot object
- OTMultiUserView - provides a special display of an ot object which aggregates multiple users
- OTViewFactoryAware - used by views which want access to the view factory
- OTXHTMLHelper - provided to OTXHTMLViews so they can create bitmaps of JComponents and ot objects.
- OTFrame - represents a window in the display.

OTViewContainer

The idea of the OTViewContainer is to provide a service to view objects. It is supposed to be used for:

- so the view can replace itself in its container.
- so the view can get views of objects it references.
- so the view can open a new frame with a particular object.

- implemented by
  - OTDocumentObjectView
  - OTViewContainerPanel
  - PfListCellRenderer
- not used by OTLayout but it should be: OTLayout makes a new component and embeds it into the document without implementing a OTViewContainer to give to it.

setCurrentObject

This method changes the object displayed in the current container if the otFrame is null. If otFrame is not null then it replaces the object displayed in that frame. The only time the frame functionality is currently used is in hyperlink event handling.

- Views that call it:
  - OTDocumentObjectView implements OTViewContainer and delegates setCurrentObject to the parent container if the otFrame is not null. This might be able to use OTViewContainerPanel.
  - OTDocumentView calls setCurrentObject when a hyperlink is clicked.
  - PfDocumentViewXHTML calls setCurrentObject when a hyperlink is clicked.
- A number of classes use the OTViewContainerPanel, and call the setCurrentObject method on it. These classes are not technically view objects so it is a bit confusing that they call a method on this interface. However they all the need the same functionality.
  - OTAbstractAppletViewer
  - OTViewer
  - OTViewerHelper
  - OTReportViewer

Problems

- The setCurrentObject method does not specify a view.
- It is implemented in three places slightly differently.

Fixes

- there is a OTFrameManager interface now, so that can be used by the views with hyper links, and the otFrame can be removed from setCurrentObject.
- there should only be one swing implementation of this.

getCurrentObject

- this is not used by a view, the places where it is used could be fixed so it isnt used. But it seems useful to keep this symmetry.

getComponent

This method gets a component from the view container it could be fully replaced by methods in the viewfactory.
It has an editable flag which is not general enough. Just like the setCurrentObject above it doesn't allow specifying the type of view which should be used.

General Fixes

OTViewContainer is currently passed to a view though OTObjectView.initialize(). In most of these views it is just bloat because they never use it. it is referenced 105 times Instead there should be a OTViewContainerAware interface which views implement that need to access the view container. Currently the only views that need this are the hyperlink views. The getComponent method will be removed.

The unsolved problem is how to let a caller of setCurrentObject specify a particular view.

OTViewFactory

There is only one implementation of the view factory: OTViewFactoryImpl

createChildViewFactory

This is currently only used by the reportviewer. It is used so a special child view factory can be setup which has a set of users. Which will then make the viewFactory try to create a OTMultiUserView instead of a OTObjectView. It is a child of the main view factory so it can inherit all the view entries of the parent.

setUserList

This is only used by the reportviewer.

getComponent

This is used by the 3 implementations of OTViewContainer and by the PfViewPrinter. PfViewPrinter is an application which takes an otml file and generates a set of images from it. getComponent is called to get a component to be printed.

getView

This is most configurable way to get a view. It takes an otObject and a viewInterface. It is used by:

- OTMLtoXMLConverter to get a view to be turned into an image. It first looks for a view that implements OTPrintDimensions. And if not then it looks for an OTObjectView.
- OTViewFactoryImpl uses it to implement getObjectView by looking for an OTObjectView and then calling initialize.
- OTDocumentView uses it to look for an XHTML view of embedded objects. If an object has this kind of view that is used instead of the JComponent view.
- PfResponseTextService also looks for an XHTML view of the response object it is working with. This is because responses have xhtml views, and can have embedded objects like the OTDocumentView

getObjectView

- OTViewFactoryImpl uses this to implement getComponent.
- OTViewContainerPanel uses this to implement setCurrentObject
- OTMLToXHTMLConverter uses it to get the main view which it will be saving, it doesn't use getView for some reason. it also uses getObjectView to get the component view to be turned into an image. It has to use both getView and getObjectView because otherwise initialize is not called.

addViewEntry

- OTViewService uses this to setup the viewfactory with viewEntries

Problems

- initialize being called by getObjectView makes it unclear which method to use when.

OTObjectView

Drawing Tool Revision

This page last changed on Apr 29, 2007 by scytacki.

Currently the drawing tool design is all in java code. With the new view system it could moved to be defined in otml instead.

To start with:

- tool bar would be setup with otml:
    - icons for tools
    - shape tools, stamp tools
    - attributes for stamp tools

ViewFactory issues

This page last changed on Apr 29, 2007 by scytacki.

Issues with ViewFactories:

- main view factory should not be swing specific
- a object that is using a view factory should be able to ask for a particular type of view currently:
    - jcomponent
    - xhtml
- the view factory needs to do specific types of initialization for different view types currently:
    - jcomponent
        - needs to handle ViewFactoryAware, ViewContainerAware, FrameManagerAware
    - xhtml (has no initialization needs)
        - might want to have screen resolution if it generating images.
    - these initializations need inputs from the environment:
        - viewContainer, frameManager, screen resolution

- a object using the view factory that asks for a JComponent from an object or viewEntry that is XHTML should
  get back a JComponent that wraps that, so the object doesn't need to care.
- same with an object wanting xhtml, if it asks for that from a JComponent view the viewFactory should hand back
  a wrapper that returns a chunk of xhtml with an image of the component.

- viewFactories should be nestable, so different contexts can return different types of views. An example is the report
  tool. In one place it uses multi-user views, in the other it uses single user views.

Because of these requirements there cannot be a single view factory, or lots of subclassed view factories. Because an object that asks for a SwingViewFactory will want it to handle xhtml views too.

So the viewFactory needs to have "plugins", of sorts.
It could work two ways.

Way 1
An object using the view factory would do:
Object viewFactory.getViewInstance(Object otObject, Class viewType, ViewEntry viewEntry);
The view factory would then lookup the plugin for handling the viewType, and that plugin would need to be initialized with the appropriate stuff.

However this won't fit the "viewContainer" approach. Each object that request a JComponent should provide
a view container, so the returned object can change itself.

Way 2
An object using the view factory would do:
OTViewHandler viewFactory.getViewHandler(Class viewType);
then the object would have to cast that the viewHandler that they want. Something like:
a SwingViewHandler.

And then SwingViewHandler would have methods:
JComponent getJComponnet(otObject, viewContainer);

Way 3
An object use a view factory would mark itself as:
SwingViewFactory/Handler aware.
Then this would be set when it is started up. But this would still be connected to the master viewFactory so the views can be setup in one place but objects can use them through different view specific ways.

Interesting Cases

Choice Object

The choice object has two views:

- PfChoiceView, which is a JComponent
- PfChoiceMultiUserView, which is a XHTML view

When displaying a PfChoice directly on the left, a JComponent view is requested by calling getComponent on the view factory.
When displaying a PfChoice inside of a PfQuery view the JComponent is requested because it has been special cased.
When displaying a PfChoice directly using a reference to it from the bodyText of an OTDocument, then the OTDocument requests the OTXHTML view first, and finds one. But when it tries to use this view, it fails because, it isn't in multi user mode. Even if this worked it would be the wrong thing. What is needed here is the JComponent view, but how is it supposed to be setup for this?

Options:

- the reference to the PfChoice object could explicitly reference the JComponent view
- the mechanism for looking up views needs to have more options. In this case some control should be given to the PfChoice views, so they can say: when I'm being requested in a OTDocument, single user, and student mode. Then return the JComponent. Student Mode can be specified as the default.
    - if the OTDocument had its own viewFactory which could be plugged into, then the PfChoice could plug into it and specify that the JComponent view is better than the XHTML view for choices.

Text Object

The text object should have lots of views:

- multiple sized text boxes that can be used to edit the text object
- labels for read only text objects
- xhtml views for read only xhtml rendering.

If the view of the object is specified when it is used then these will not be a problem. Perhaps that is the best solution for the above problem too. If it turns out that this is too cumbersome, a custom OTClass could be created which has a default view of one of the above, then objects of that custom OTClass can be embedded and they will be rendered properly. This approach has the benifit of not tying the content directly to a particular view, instead it just has the OTClass which then figures out the correct view. It isn't clear this will solve the problem though, we'll need to work through cases to figure it out.

OTrunk View Modes

This page last changed on Jun 17, 2008 by scytacki.

OTrunk view modes allow you to override a default view of an OTObject. This form of overriding is used to provide different "views" of parts of the application. For example it is used to provide a authoring view of some objects.

Here is a simple example of configuring the OTViewBundle with a authoring view mode for a OTText object.

In this case you define the default OTViewEntry in the OTViewBundle.viewEntries element. And then define an OTViewMode in the OTViewBundle.modes element. Inside the OTViewMode.map element you need to put an
"entry" element with a "key" attribute pointing to the id of the default OTViewEntry.
Then inside of the entry element put the OTViewEntry which should replace the default OTViewEntry when that particular OTViewMode is activated.

So you can use view modes to group a set of overriding views together. And then turn them activate them all on at once.

OTrunk View System Analysis

This page last changed on Oct 24, 2007 by scytacki.

The view system was refactored after it was analyzed on this page:OTrunk Old View System Analysis

It still has some problems

Problems

- OTJComponentService was created to provide a standard way for views to create new OTJComponentViews. However this service is not being used by the main view creator OTViewContainerPanel. This is because it is doing some fancy stuff to automatically handle xhtml views. That code which wraps the xhtml view in a jcomponent should be moved into the OTJComponentService.
- The way the OTDocumentView decides which view of an object to use is contorted. It uses a combination of the existance of a viewEntry, and the existence of a OTXHTML view to decide which to use. And it also factors in whether the view has a view mode or not.

Classes

View System Class Summary

Logic for deciding which view to use

Starting with the OTDocumentView here is the current logic.

Initial text parsing

1. if the viewEntry is not null (view-id attribute on object element), call:
2. if the viewEntry is null request a view implementing the OTXHTMLView interface:
   a. if that returns null and the viewMode is not null then request a view implementing the JComponentView interface:
3. if the resulting view is an instance of the OTXHTMLView then get its text and insert it in the document
4. otherwise leave the object tag in for the next stage

Component loading by HTMLEditorKit

If there are object elements in the text then the editor kit has been modified to ask the OTDocumentObjectView object for the Component to display. This view uses a OTViewContainerPanel interface to do this. It sets the object, entry, and mode of the viewcontainerpanel. The view container panel implements the logic of which view to use with the createJComponent method:

1. if the viewEntry is not null call getView on the view factory with the object, viewEntry, and mode
2. otherwise call getView on the view factory asking for a JComponentView implementation with the mode.
    - This causes problems because the mode of a JComponentView implementation might result in a XHTMLView. This should actually have been handled in the initial text parsing stage.
3. if the resulting view is OTJComponentView then that is saved as the currentView
4. otherwise a OTXHTMLView implementation is requested and wrapped with a OTXHTMLViewWrapper which is basically a OTDocumentView.
5. the getComponent method on currentView is used to get a component for the current view

OTrunk View System Analysis Accessible Objects

This page last changed on Oct 24, 2007 by scytacki.

A OTJComponent view has several objects available to it or that it can create. These can be accessed in several different ways.

- OTViewContext
- OTViewFactory
- OTViewContainer
- OTJComponentViewContext
- OTJComponentService
- OTJComponentServiceFactory
- OTViewEntry

The OTViewContext and OTJComponentViewContext are filling the same role, but the JComponent version has a getComponentByObject method that
cannot be available in the OTViewContext.

The OTViewFactory and OTJComponentService fill the same role, but the OTJComponentService works with OTJComponentView objects instead of generic OTView objects.

The OTJComponentService and OTJComponentViewContext might be mergeable with each other.

View System Class Summary

This page last changed on Oct 23, 2007 by scytacki.

OTViewContext

A view has an OTViewContext. One OTViewContext is shared between sibling views.
It provides methods for:

- looking up a view service
- adding a view service
- unimplemented: getting a view by object
- creating a child view factory.

OTViewFactory

Parent views(views that have children views), should create a OTViewFactory instance to create their children views. A parent view should use the same OTViewFactory instance for all of its children views, this
way all the children will have the same OTViewContext.
A parent view uses a OTViewFactory instance by

- extending AbstractOTJComponentContainerView and calling one of the createSubViewComponent methods
- using OTViewContext.createChildViewFactory()
- calling OTJComponentService.getViewFactory()

OTJComponentViewContext

provides methods for:

- finding a view by object
- finding a component by object
  This is used by a lot of the scripts and java code so one view can control another view without going through the OTObjects.
  It can be obtained through 2 ways:
- implementing OTJComponentViewContextAware
- calling OTJComponentService.getJComponentViewContext()

OTJComponentService

provides methods to instanciate OTJComponentView. These views require additional initialization over the basic OTView, so this service takes care of that initialization.
it can be accessed:

- extending AbstractOTJComponentView and calling getJComponentService();

- getting the OTJComponentServiceFactory which is registered as a view service, and calling createOTJComponentService, a view factory has to be passed in.

OTJComponentViewContextAware

aware interface used to pass the OTJComponentViewContext to a OTJComponentView.
This api is redundant with the method getJComponentViewContext on the OTJComponentService interface.
A functional difference is that it saves the creation of the OTJComponentServiceImpl,
just to access the OTJComponentViewContext.

OTJComponentServiceFactory

a factory service provided so OTJComponentService objects can be created by parent views and used to create their
children views. This was created because each parent view needs its own OTJComponentService instance. If
AbstractOTJComponentView extended then a parent view doesn't need to worry about this interface.
Currently this is used in only 7 framework classes. It is not used by any views besides the top level OTViewer.

OTViewContainerPanel

This is a JPanel which contains, the JComponent returned by an OTJComponentView. In many cases it is unnecessary,
originally it contained the different accessor methods for things a view needed. But now views have the view context
and view factory for that. The one functionality that provides is the OTViewContainer functionality. To be a
proper OTViewContainer an implementer needs to be able to replace the object it is viewing with another one. In
most parent view cases this should be possible without maintaining the OTViewContainerPanel, but that needs to be
verified.

OTViewChild

This is a way to modify the view entry and scroll bars of a child view.
It it can be used in many OTObjects that represent a container view. Like OTPlacementContainer or OTBorderContainer.
It has drawbacks. It cannot be used by an container object that expects a specific child OTObject.
When the child object is requested from the container object the OTViewChild will be returned instead of the child inside.

In OTCompoundDoc objects the viewid attribute can be used on the object reference to modify the viewid of a particular
object reference.

The plan for the future is to support annotations on objects that will allow the view id and scroll bar settings to be specified.
Another option is to use css style selectors.

AbstractOTJComponentContainerView

provides methods for OTJComponentView objects which are parents of other views. It provides convenience methods
for getting components from the children objects.

AbstractOTJComponentView

provides basic functionality for OTJComponentViews,

- reloading views when the view entry changes, this uses the viewContainer, so this is affected by the "updateable" container
code.
- creating controller services, so the real object system can be used
- getting child components, these methods don't property setup up the view container so they should be used with caution
- storing the viewContainer.

OTViewContainer

interaface which provides methods:

- changing the current object,
- getting the parent container
- getting the updatable container
- reloading the view.

An OTViewContainer instance can be accessed:

- by implementing OTViewContainerAware
- extending AbstractOTJComponentView and calling getViewContainer()

OTViewContainerAware

a view uses this interface for gaining access to the view container of a view.

OTView

Marker interface that all views should implement, typically views implement this indirectly.

AbstractOTView

provides a few useful methods and fields for views.
implements OTViewContextAware, provides getViewContext so subclasses can access it.
creates and caches the viewFactory instance

OTJComponentView

Interface for views that provide JComponents. It currently has two methods:
getComponent(OTObject otObject);
viewClosed();
Classes that implement this method are created by the OTViewFactoryImpl. This is mainly
called done by OTJComponentServiceImpl

OTRequestedViewEntryAware

Views can implement this interface so they know what the request viewEntry was. The actual
view entry can be different because the viewMode was set. The requested viewEntry can
be specified by the user explicitly or it could have been determined by matching the OTClass
to the objectClass of the viewEntry.
Currently this is only used by OTAuthorEmbedDefaultView

OTViewEntry

This is an OTClass that is used in a few cases:

- it is used to register a mapping between a OTClass and viewClass, with the top level OTViewFactory created by
  the OTViewBundle
- it is passed as an argument to the OTViewFactory so its viewClass is used for an object
- it can be extended and new properties added so a particular view can be configured. Examples of this are:
    - OTChoiceViewConfig
    - OTImageViewConfig

OTViewContextAware

Aware interface so views can access the OTViewContext, the Abstract* classes take care of this.

OTViewEntryAware

Aware interface so views can access the OTViewEntry

OTXHTMLView

Extension of OTView that has one method:
getXHTMLText(OTObject obj)
this returns a string of xhtml, which is the "view" of the object.

OTrunk Viewer Properties

This page last changed on May 14, 2008 by stepheneb.

There are few properties that can be used to configure the OTViewer which is the main class which displays otml files.

The properties can be passed as:

- system properties
- sds runtime properties (look at SailOTViewer in sail-otrunk project in tels svn)
- properties of the Pod setting up the otml file. (look at SailOTViewer and BuildOTrunkCurnit in sail-otrunk project in tels svn)

Which properties can be passed where is pretty random.

Here is a link to the current list of properties:
JavaDoc constants

This is the code snippet from OTViewerHelper

```
```

These are some properties in OTViewer

```
```

OTViews

This page last changed on Apr 29, 2007 by scytacki.

OTCompoundDocEditView

This page last changed on Apr 29, 2007 by scytacki.

An example of this view can be found here:

http://continuum.concord.org/otrunk/examples/example-index.html
CompoundDocument/document_edit.otml

If you select the Compound Doc item on the left, you will see a basic editing view for a document. It allows you to insert 4 different objects: text box, button, and 2 images.

If you look at the otml file you can see these objects are configured in the otml file. Here is the snippet that configures them:

OTDocumentView

This page last changed on Apr 29, 2007 by scytacki.

This is the view used to display compound documents of markedup text with components embedded inside.

New Features

- the external links, ones that go to real web pages, should use a service to go to them. This will allow us to support both running jnlp where a new window is opened. And in a standalone mode where jdic is available to also open a window.
- Another option is to open a browser directly embedded in the OTViewer frame, this will be hard because it will be a different behavior than the other two so it won't be consistent to author.

- Replacing the current Java Swing based htmleditorkit, with a native mozilla component. Using jdic is not an option here because we cannot embedd java components inside of a jdic component on all platforms. And those were we can embedded the components they are in another virtual machine. The current goal is to use XPCOM to do this embedding:

Found 19 search result(s) for XPCOM

XPCOM Overview (Public: Technology Reviews)
XPCOM stands for cross platform component object model. It is the framework used by mozilla/firefox for extending and providing services to the browser and web applications. It has bindings for most of the popular languagues http://developer.mozilla.org/en/docs/XPCOM:LanguageBindings : Java, Mono ...
Sep 24, 2006
Embedding XPCOM in Linux (Public: Technology Reviews)
an issue with initializing the gtk when using xpcom in linux: https://bugzilla.mozilla.org/showbug.cgi?id=342691 The end of this bug refers to some things to make the gtk work: https://bugzilla.mozilla.org/showbug.cgi?id=335696#c16
Mar 14, 2007
Embedding Mozilla in Java with XPCOM (Public: Technology Reviews)
appears to be the best list to monitor this: http://www.nabble.com/MozillaOJIf6677.html However their seems to be more information bugs in the mozilla bugzilla, and eclipse bugzilla. Here is a post with code which does this. It looks really easy: http://permalink.gmane.org/gmane.comp.mozilla.devel.java/2247 Probably the devil ...
Mar 18, 2007
Embedding XPCOM in OSX (Public: Technology Reviews)
now two project a project in CC&apos;s cvs for this: Projects/XPCOM/Xul18Browser Projects/XPCOM/Xul19Browser Those contains some code found online for embedding the browser. I changed it to try it out on mac. It should be refactored so it can be tried on all the platforms ...
Mar 18, 2007
Embedding XPCOM in Windows (Public: Technology Reviews)
thread provides some useful info: http://www.nabble.com/JavaXPCOMandHandletf1318510.html It has a reference to a web site with an example implementation: http://nithril.free.fr/Xul/ The I uncompressed and recompressed the sample.rar so it is xulsample.zip Given that the code is all taken originally from ...
Mar 15, 2007
Mono Browser plugins (Public: Technology Reviews)
References to mono support for XPCOM in the mozilla source tree:

http://lists.ximian.com/pipermail/monodevellist/2005March/011161.html
http://lists.ximian.com/pipermail/monodevellist/2004June/006152.html
http://lxr.mozilla.org/mozilla/source/extensions/mono/README It looks like these extensions to mozilla would be enough ...
Apr 29, 2007
Browser Support (Projects: TELS)
page should be moved to the TELS confluence. Create tests which break our current JDIC browser: popup window freezing unsupported plugins, behavior slow loading pages cookies available to popups multiple instances on one page. Embedded ...
Feb 13, 2007
Re: Embedding XPCOM in OSX (Public: Technology Reviews > Embedding XPCOM in OSX)

So... is the most interesting part the embedding XPCOM in OSX or the code above to integrate canvas support in OS X? Are you looking at using Canvas for some drawing/plotting functionality. I took a look at a lot of the sample code for canvas a while ...
Mar 14, 2007 - All Comments
Home (Public: Technology Reviews)

home page for the Public: Technology Reviews space. This space is intended for reviews of current technologies investigated for Concord Consortium projects. If you are adding new pages to this please take the time to reoraganize the section you are adding them to. There should be a page for the section ...
Jul 21, 2008
Plan for current projects (CC Software Development)
4 current projects that have upcoming deadlines which are all using the same software framework. I&apos;m trying to collect a plan of how the software will evolve for these projects. Certainly we will discover new things along the way so this will change. ITSI ...
Aug 06, 2007
Schedule 2007-05-09 (CC Software Development)
Rough schedule for Sam, Aaron, Shengyao, Ingrid, and I on UDL, ITSI, CAPA, and TELS. TELS researcher reporting Aaron is working 100% on this critical draw tool fixes drawing tool editor Convert MW steps from pedagogica to otrunk ...
May 09, 2007
Layout of svn repository (Public: Concord Software Projects)
Directory structure common java core applesupport(AppleSupport), framework(Framework), frameworkview(FrameworkView), loader(Loader), swing(Swing) graph graph(Graph), graphfunction(GraphFunction), graphutil(GraphUtil) data data(Data), datagraph(DataGraph) multimedia flashswf(FlashSWF ...
Nov 02, 2007
General Schedule 2007-05-17 (CC Software Development)
Rough schedule for Sam, Aaron, Shengyao, Ingrid, and I on UDL, ITSI, CAPA, and TELS. Some tasks have been removed from the eariler version of this list because they are not critical in the short term. TELS researcher reporting Aaron is waiting for feedback ...
May 18, 2007
Sams Tasks (CC Software Development)
First Steps (/) install Eclipse setup Java source path (/) install subclipse (/) install command line maven (/) check out all of concord code (/) Overview of Current Concord Technology run teemss run MW run Pedagogica wisedev ...
Aug 20, 2007
General Schedule 2007-05-24 (CC Software Development)
Rough schedule for Sam, Aaron, Shengyao, Ingrid, and I on UDL, ITSI, CAPA, and TELS. Some tasks have been removed from the eariler version of this list because they are not critical in the short term. () critical not being done yet. (!) critical, and somone ...
May 24, 2007
3rd Junior Programmer (Junior Programmer)
Here is the old junior programmer posting:

http://www.concord.org/about/opportunities/archive/juniorprogrammer.html Here is some modified text starting from the description I made up for Evan: Junior Programmer GENERAL DESCRIPTION The Concord Consortium (CC) is a fastpaced ...
Jul 17, 2007
OTrunk Goals (Public: Concord Software Projects)
Intro by Scott OTrunk is an instance of a type of framework I&apos;ve been try to create for sometime now. I believe this type of framework can be widely useful. And it seems that other people are creating parts of this framework, but I haven&apos;t yet seen ...
Apr 29, 2007
CC TELS Technical Tasks (Projects: TELS)
Component Organization TASK Description Person This is just a line of text. Component TASK Organization Description Person Sail Data Service CC Maintenance: fixing bugs, work with Berkeley to solve deployment issues Aaron and Stephen Sail Data Service ...
May 09, 2007
CSP-OTrunk.pdf (Public: Concord Software Projects > OTrunk)
Space Details Key: CSP Name: Public: Concord Software Projects Description: Creator (Creation Date): scytacki (Apr 29, 2007) Last Modifier (Mod. Date): scytacki (Apr 29, 2007) Available Pages • OTrunk • Creating OTClasses ...
PDF Document - 559 kb - Jan 30, 2008 - Download - All Attachments