Kepler - Bug #5095

test kepler and wrp for memory leaks

07/14/2010 03:56 PM - Matt Jones

Status:	In Progress	**Start date:**	07/14/2010
Priority:	Immediate	**Due date:**	
Assignee:	jianwu jianwu	**% Done:**	0%
Category:	general	**Estimated time:**	0.00 hour
Target version:	reporting-2.X.Y	**Spent time:**	0.00 hour
Bugzilla-Id:	5095		

Description

Oliver Soong reported having difficulties with memory leaks. There are two specific bugs about this, which I have set to block this testing bug. In addition, testing may reveal additional leaks, which should be fixed before 2.1 is released. Here's Oliver's synopsis of the issues:

I think this is limited to the wrp suite, but Kepler's performance degrades significantly over time. Provenance recording can become prohibitively slow, and there is no native in-Kepler fix. There is a large memory leak somewhere, and many components are quite memory-intensive regardless. Given the intention to record executions and the large number of analyses scientists perform, I suspect any dedicated user of Kepler will quickly encounter data management problems. In my case, I stopped using local repositories and began closing Kepler after running any large workflows.

Related issues:

Blocks Kepler - Bug #4642: memory usage & slowdowns	**New**	**12/18/2009**
Blocks Kepler - Bug #4801: out of memory	**In Progress**	**02/17/2010**
Blocks Kepler - Bug #5033: Workflow Run Manager - Allow multiple instances of...	**Resolved**	**05/27/2010**
Blocks Kepler - Bug #5249: test kepler for memory leaks	**In Progress**	**11/30/2010**
Blocks Kepler - Bug #5254: provenance database connections keep growing when ...	**Resolved**	**12/09/2010**

History

#1 - 07/16/2010 08:44 AM - Christopher Brooks

How I've fixed memory leaks in the past is by writing Java test cases that
illustrate the bug and then using commercial tools like JProfile to track
down the bug. Eclipse has a memory leak analyzer at http://www.eclipse.org/mat/.

It is also easier to track down leaks in non-gui code first. Gui code
tends to leak because of underlying issues in the JDK. For example, applets
were leaking under JDK1.5 on the Mac because of an internal Apple class
kept a reference to something.

For information about a leak from last year, see
http://chess.eecs.berkeley.edu/ptolemy/listinfo/ptolemy/2009-June/011591.html

Brian Hudson recently pointed out a leak in the MoMLFilter, which I'm taking
a look at.

Tracking down memory leaks can by time consuming, I'm putting this at 100.0
hours.

#2 - 07/16/2010 12:22 PM - Ilkay Altintas

Derik will ask Sean for his experience with it and update the time estimate.

#3 - 08/13/2010 05:11 PM - jianwu jianwu

After digging a while using JConsole and Eclipse MAT plugin, I didn't found obvious memory leaks. As expected, hsqldb takes a lot of memory (100 M for reporting suite).

One way we can try is to start hsqldb using cached option (http://radioae6rt.wordpress.com/2007/09/01/hsqldb-memory-leak/)

#4 - 08/13/2010 06:18 PM - ben leinfelder

Jianwu,
WHen you say hsqldb - is that for provenance, data, or both? I think one of the things the TPC workflows exercised where many many data tables

(EML datasource actors) and it's probably important to have test workflows that do cache quite a bit of data.

#5 - 08/13/2010 06:44 PM - jianwu jianwu

ben, I think it is for provenance since my current tests are just normal workflows, not those TPC workflows.

Thanks for your info. I'll test those TPC workflows soon.

#6 - 08/13/2010 06:48 PM - ben leinfelder

We will get your 'jianwu' account (o=SDSC) set up so that you can read the data used in those workflows. Please don't distribute the data, not that you would.
I'll see if I can't find the other missing data that referenced and was supposed to exist of the development server.

#7 - 08/13/2010 08:25 PM - ben leinfelder

The data referenced on DEV is there - I was mistaken earlier about it being missing.
http://dev.nceas.ucsb.edu/knb/metacat/soong.4.14/default
Once you are added to the TPC ldap group, you'll also have access to those data files. I'll let you know when that's set up.

#8 - 08/13/2010 08:28 PM - jianwu jianwu

Ben, cool. Thanks.

#9 - 09/21/2010 05:54 PM - jianwu jianwu

Which changes in kepler trunk r25888 and ptolemy trunk r59134, mainly switching to Weak References in singleton classes and adding cleanup when a window is closed, now the memory used by Kepler suite will not increase much after opening and closing big workflows.

#10 - 09/21/2010 05:59 PM - jianwu jianwu

To verify the changes, you can look for the instance number of ptolemy.gui.Top after opening and closing workflows. The number should be the opened workflow windows plus 1 (the extra one is for welcomewindow). It used to keep increasing even a window is closed.

p.s.: typo of the first word of the last comment: 'which' should be 'with'

#11 - 09/22/2010 03:09 PM - jianwu jianwu

as discussed with derik, one leak exists in org.kepler.workflowrunmanager.WorkflowRunManagerManager class and the class needs re-design to fix it.

#12 - 10/27/2010 05:55 PM - Derik Barseghian

The WorkflowRunManager no longer keeps a tableauFrame as a member variable,
and the WorkflowRunManagerManager now extends KeplerGraphFrameUpdater so that on dispose (window closed) a tableauFrame=>WRM mapping is released. Let me know if this doesn't fix the leak you saw here.

(In reply to comment #11)

> as discussed with derik, one leak exists in
> org.kepler.workflowrunmanager.WorkflowRunManagerManager class and the class
> needs re-design to fix it.

#13 - 11/03/2010 04:56 PM - Derik Barseghian

In r26239 I've changed the ObjectManager's _namedObjs WeakHashMap back to a HashMap, see bug#5200 for details on some problems the WeakHashMap caused. If the HashMap is causing memory issues, we could explicitly remove items from it as necessary. Or if we want to move to a WeakHashMap, we'll have to solve the issues in bug#5200 an alternate way, i.e. changing the kar system / kar entry handlers. Note the ObjectManager is used in many places throughout Kepler, so a change back to WeakHashMap will also require a lot of testing for issues similar to 5200: attempts to look up items from OM that've been discarded.

#14 - 11/09/2010 01:55 PM - Derik Barseghian

In Reporting, if you save a workflow that has a report design, modify the workflow and save again, the report layout will not get saved and will disappear from the gui. This is because after r25851, KeplerGraphFrame dispose() calls removeNamedObj(this.getModel()), which actually disposes of ObjectManager's copy of the newest version of the model (and leaves behind in ObjectManager an old version). During the 2nd save after model modifiation, ReportLayoutKAREntryHander.save fails to get the model from OM using its newest lsid, and so doesn't save the layout.

ObjectManager complicates and confuses things by offering methods that don't do what they purport to, and some that don't do what you probably expect (I'll be checking in some cleanup soon). One thing to look out for is the difference between what's actually kept in the _namedObjs map (lsids are not changed) and the objects and lsids the methods actually utilize (often the NamedObjId for a NamedObj, different from the original unchanging lsid in the map).

I don't see a quick fix. Many classes depend on the idiosyncratic behavior of the OM, and so care needs to be taken with refactoring it or manipulation of the object it holds. The need for a KAR/workflow Manager that can keep things clearly differentiated by frames has come up a number of times, so one idea is to implement that and begin migrating things to using it, relying less on OM.

Until a solution is identified, I'm going to comment out the r25851 change. This will fix this report layout bug, but means ObjectManager is going to leak memory again.

#15 - 11/30/2010 02:53 PM - jianwu jianwu

change the milestone to be wrp-modules-2.X.Y, since reporting 2.2 release is separate with kepler 2.2. bug 5249 is for Kepler 2.2.

#16 - 12/03/2010 02:00 PM - jianwu jianwu

new change is made at version 26414 for the problem at http://bugzilla.ecoinformatics.org/show_bug.cgi?id=5095#c14. Now when a kar is saved, old frame is closed before new frame is open. So KeplerGraphFrame.dispose() can be correctly invoked to clean up ObjectManager._namedObjs map during old frame closing and new namedObj can be added correctly during new frame opening.

#17 - 12/08/2010 10:37 AM - jianwu jianwu

in r26414, I changed the kar saving steps. It used to open a new window for saved kar before closed the old one. Now it closes the old window before open a new one. This change will make sure KeplerGraphFrame.dispose() can get rid of old version of the model in ObjectManager (_namedObjs map) when closing old window, and have the new version loaded when opening a new window.

After discussing with Derik, We think this solution will cause risks.

First, 'close then open' is more scary than 'open then close'. If we export a workflow to xml, it act as 'open then close'.

Second, new window open might fail, which will cause bigger problem.

Feedback on this issue is welcome.

#18 - 12/08/2010 04:13 PM - jianwu jianwu

in r26444 and r26445, Kepler kar saving is back to open first before closing.

This solution is borrowed from KeplerGraphFrame._saveAs(String) where effigy.setContainer(null) is also used there.

Also it will not have version problem http://bugzilla.ecoinformatics.org/show_bug.cgi?id=5095#c14, since KeplerGraphFrame.dispose() is not called.

This way will also not cause memory leak because the opened workflow will be cleanup when its window is closed. It is done by calling KeplerGraphFrame.dipose().

#19 - 12/17/2010 03:21 PM - jianwu jianwu

Now in Kepler suite, window close will release memory. But it is not the case in Reporting suite. By debugging, I found that org.kepler.reporting.gui.ReportViewerPanel is not properly removed from KeplerGraphFrame._updaterSet when a window is closed. The class's function initializeTab() has a line for the removing:
KeplerGraphFrame.removeUpdater(keplerGraphFrameUpdater). But it doesn't work well.

The current behavior is that when the first window (Unamed1 before re-naming) is closed, its ReportViewerPanel instance will not be removed. Other windows work correctly. org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel has a similar implementation, yet it works very well. I haven't figured out why.

Some debug info is at below.

```
_updaterSet's size before removing: 9

_updaterSet's elements before removing: [org.kepler.workflowrunmanager.WorkflowRunManagerManager@62a34b91,
org.kepler.module.provenance.Initialize@6b0cc9b4, org.kepler.module.gui.Initialize@2dabcea,
org.kepler.module.workflowrunmanager.Initialize@5d4fa79d, org.kepler.module.reporting.Initialize@78d2883b,
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,invalid,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,invalid,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSize=,preferredSize=]]

updater to be removed:
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,invalid,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSize=,preferredSize=]

_updaterSet's size after removing: 9
```

```
_updaterSet's elements after removing: [org.kepler.workflowrunmanager.WorkflowRunManagerManager@62a34b91,
org.kepler.module.provenance.Initialize@6b0cc9b4, org.kepler.module.gui.Initialize@2dabcea,
org.kepler.module.workflowrunmanager.Initialize@5d4fa79d, org.kepler.module.reporting.Initialize@78d2883b,
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,invalid,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flag
s=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,invalid,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0
,border=,flags=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,ma
ximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border
=,flags=9,maximumSize=,minimumSize=,preferredSize=]]

_updaterSet's size before removing: 9

_updaterSet's elements before removing: [org.kepler.workflowrunmanager.WorkflowRunManagerManager@62a34b91,
org.kepler.module.provenance.Initialize@6b0cc9b4, org.kepler.module.gui.Initialize@2dabcea,
org.kepler.module.workflowrunmanager.Initialize@5d4fa79d, org.kepler.module.reporting.Initialize@78d2883b,
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,invalid,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flag
s=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,invalid,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0
,border=,flags=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,ma
ximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border
=,flags=9,maximumSize=,minimumSize=,preferredSize=]]

updater to be removed:
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,invalid,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0
,border=,flags=9,maximumSize=,minimumSize=,preferredSize=]

_updaterSet's size after removing: 8

_updaterSet's elements after removing: [org.kepler.workflowrunmanager.WorkflowRunManagerManager@62a34b91,
org.kepler.module.provenance.Initialize@6b0cc9b4, org.kepler.module.gui.Initialize@2dabcea,
org.kepler.module.workflowrunmanager.Initialize@5d4fa79d, org.kepler.module.reporting.Initialize@78d2883b,
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,invalid,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flag
s=9,maximumSize=,minimumSize=,preferredSize=],
org.kepler.reporting.gui.ReportViewerPanel[,2,25,800x412,hidden,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,ma
ximumSize=,minimumSize=,preferredSize=],
org.kepler.workflowrunmanager.gui.WorkflowRunManagerPanel[,2,25,800x198,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border
=,flags=9,maximumSize=,minimumSize=,preferredSize=]]
```

#20 - 12/17/2010 04:05 PM - jianwu jianwu

the problem in #19 is fixed at version 26554.

#21 - 01/05/2011 02:40 PM - jianwu jianwu

status update: after the check in yesterday at version 26623, now there is no big memory leak for window open and close in Kepler and Reporting suite. My tests show Kepler memory usage is 42MB after starting (with GC) for reporting suite. Memory usage changed to 65MB after open a big workflow (with GC and open from xml). After closing the workflow, memory usage is back to 44MB (with GC).

Yet workflow open and close is just one basic function for Kepler. Many other functions still need to be tested, such as workflow closing after execution, reporting view, run manager and etc. My tests today also found memory leaks.

Also my tests for bug http://bugzilla.ecoinformatics.org/show_bug.cgi?id=4801 still show memory leaks. Kepler memory usage is around 100MB after the tpc01-buffalo-tb.kar is open and closed. Two possible reasons are 1) tens of data connection threads are created in the beginning; 2) some actors in the workflow got initialized during the opening but not released during closing.

#22 - 01/06/2011 04:37 PM - jianwu jianwu

My tests show Kepler has memory leaks for "workflow open, execution and close" and "workflow open, edit and close". More detailed information can be found at
https://kepler-project.org/developers/teams/build/dealing-with-memory-leak-in-kepler

From the tests, I doubt some memory leaks might be from Ptolemy also. So I tested Ptolemy itself today. Here is what I did:

1) Start Ptolemy
2) Open a new 'Graph Editor', whose title is "Unamed".
3) Open a sample workflow (constant actor + display actor in SDF)
4) Run the sample workflow
4) Close the window (now only the "Unamed" window left)

From the jprofiler (other tools should also be good to see it), I can see there is one instance for class "ptolemy.actor.lib.Const" and "ptolemy.actor.lib.gui.Display" after garbage collection. Clearly the objects for these two actors are not released. I also saw 4 instances for "ptolemy.gui.Top$1", which should be 2 if there is one window left, and 8 instances for "ptolemy.kernel.util.Workspace".

From "Heap Walker" in JProfiler, I didn't see obvious wrong references in the GC path of "ptolemy.actor.lib.Const" and other instances.

Christopher Brooks, do you want to verify the phenomenon?

#23 - 01/07/2011 10:20 AM - Christopher Brooks

I'm not that surprised that there is a memory leak in the GUI when
a simple model is run just in Ptolemy, outside of Kepler.

I can look at this eventually, but have other tasks ahead of this
task. It could be several weeks before I get to this.

As I've said, I'd focus first on testing without using the GUI
and then try the same model in the GUI.

#24 - 01/07/2011 04:43 PM - Aaron Aaron

Coming up to speed on this.

I've added a step-by-step example of using jvisualvm to find memory leaks. You can find it in the "Example of analyzing memory usage/leaks using Java VisualVM" at the bottom of the "Dealing with memory leak in Kepler" page here:
https://kepler-project.org/developers/teams/build/dealing-with-memory-leak-in-kepler

#25 - 01/10/2011 05:02 PM - jianwu jianwu

based on the comments 22 and 23, I think it is really hard to fix all memory leaks in Kepler by the code freeze date (1/15/2010). So I suggest leaving the current Kepler and Ptolemy status for Kepler 2.2. The leaks left can be solved in the next versions. Hopefully, we can find good time schedule for both Kepler and Ptolemy on this issue.

Feedbacks are welcome.

#26 - 02/17/2011 08:13 PM - Aaron Aaron

I have discovered that there is a thread that is waiting even after a workflow is closed. I suspect that it is this thread which is holding references to the Tableau and associated objects. One can see in the Java VisualVM "Threads" view while profiling the Kepler application that by simply doing a File->Open operation on a workflow starts several new threads. Most of which finish after some short time, except for one which continues waiting indefinitely, even after closing the window. My first guess was that this was due to the MenuMapper which implements the Runnable interface, but no such luck yet proving that this is the cause for the waiting thread...

#27 - 02/18/2011 04:08 PM - Aaron Aaron

That thread turns out to be started by the batik SVG jar when KeplerXMLIcon calls SVGDocumentFactory.createSVGDocument in the _batikCreatePaintedList method. I upgraded to batik 1.7 and that thread still exists but is now labeled the Batik Cleaner Thread. I observed no effect on the existence of Top references after window close when upgrading to batik 1.7. :(

#28 - 03/01/2011 04:50 PM - Aaron Aaron

I found what might have been a leak, the _modelToFrameMap that was previously in KeplerGraphFrame seemed to not be losing it's reference to closed KeplerGraphFrames. I simplified the Map by moving it into a singleton class to assure the references were getting removed. change was made at revision 27223

#29 - 03/04/2011 04:43 PM - Aaron Aaron

Many ActionListeners hold references to the main Top frame, especially in the menus. Our menu system is pretty funky since it is laid over the top of ptolemy. Many menuitems are laying around in memory not doing anything (because they are generated by ptolemy but ignored by kepler) and they have ActionListeners that Swing holds onto in it's memory unless you specifically remove them. Systematically removing actionListeners when the window is closed seems to help but still hasn't freed up the Top JFrame from memory yet.

Also, ptolemy.gui.Top calls it's private method _populateHistory which generates a menu item that has a HistoryMenuListener. That listener never gets garbage collected and I can't figure out how to get to it to remove it so I plan to just override that method in KeplerGraphFrame and have it do nothing. Then that listener (which kepler isn't using anyway) will never be created in the first place.

Also, getContentPane().removeAll() is never called anywhere in ptolemy, I've read that it is a good thing to call after closing a window for memory sake. (but I tried it and it still doesn't free up the main Top frame)

Have to clean up all my code before checking it in...

#30 - 03/30/2011 07:51 PM - Aaron Aaron

Continuing to make progress on this bug.

There are 3 levels I am working at, the ptolemy level, the kepler core level and the sensor-view level. The ultimate goal is to free up as many resources as possible whenever windows are closed so memory usage does not build up over time as windows are repeatedly opened and closed.

Unfortunately it is difficult to declare that a class is free from leaks since it can easily depend on other classes in different situations. For example, I could say KeplerGraphFrame was free from leaks because when you open a new workflow then immediately close it all the memory for the frame gets garbage collected. However, in a more complex scenario, if I open a KeplerGraphFrame and build a workflow containing a Display actor, run it, then close the window, KeplerGraphFrame will not be garbage collected, so the Display actor leak causes KeplerGraphFrame to leak (and the leak in the Display actor may in turn come from a leak in the TextEditor class, and so on). So my point is that characterizing leaks can be difficult (not to mention there are over 300 actors in our library that can contribute to leaky graph frames).

Memory improvements have been made on leaks found in TextEditor, Display actor, WindowPropertiesAttribute, ActorGraphFrame (and all super classes), EditParametersDialog, FigureAction
At the Kepler level improvements have been made in KeplerGraphFrame
At sensor-view level improvements have been made in WorkflowManager

As leaks are discovered and addressed at the lower levels it becomes easier to identify and address leaks at the higher levels.

#31 - 04/21/2011 03:19 PM - Aaron Aaron

A standard test I've been using to measure leakiness in the main KeplerGraphFrame class is to open 101 blank frames then close them all and measure the difference in memory usage after garbage collections. Here are steps for performing this test.

1.) Start Kepler
2.) Start Java VisualVM
3.) Begin monitoring the running org.kepler.Kepler instance in VisualVM
4.) Perform GC on the Monitor tab in VisualVM
5.) Measure the lowest used heap amount just after performing the GC operation by dragging the cursor over the Heap usage graph (rounded to nearest MB)
6.) Back in Kepler open 101 frames using Ctrl-N
7.) Perform GC on the Monitor tab in VisualVM
8.) Close 100 of the 101 open frames in kepler using the X button in the upper right corner of the frame
9.) Select Help->About from the menu and click on the splash screen that comes up
10.) press Ctrl-O and cancel the open frame (step 9 and 10 are just to clear frame references from focus managers)
9.) Perform GC on the Monitor tab in VisualVM
10.) Measure the lowest used heap amount just after performing the GC operation by dragging the cursor over the Heap usage graph (rounded to nearest MB)

I then take a snapshot of the graph and name it using the following naming convention
<Suite name>-<OS>-<test name>-<description>-<min>to<max>MB.jpg
where
<Suite name> The name of the suite of Kepler modules that was used for the test (or Vergil if plain old ptolemy was used)
<OS> The operating system the test was run on, including 64bit or 32bit designator
<test name> A descriptive name for the test, this is 101_blanks for the test described here
<description> Information about the significance of this particular test. Such as a svn revision number or specific issue being tested
<min>to<max>MB are the two measurements of concern, for the 101_blanks test the <min> comes from step 5 and the <max> comes from step 10

Attached are several of these graphs

#40 - 04/22/2011 02:52 PM - Aaron Aaron

Found a leak in runtimemonitor (although it only manifested when running the sensor view suite).

org.kepler.plotting.Plot has a static list of all open plots. All new plots were being added to the list but never removed. Removing the plot from the list using a windowClosed listener on the frame solved the leak.

#41 - 03/27/2013 02:29 PM - Redmine Admin

Original Bugzilla ID was 5095

Files

Kepler-Win7_64bit-101_blanks-rev_27517_without_ResultTreeRoot_leak_fixed-23to32MB.JPG	88.3 KB	04/21/2011	Aaron Aaron
Kepler-Win7_64bit-101_blanks-rev_27518_with_ResultTreeRoot_leak_fixed-23to26MB.JPG	93.8 KB	04/21/2011	Aaron Aaron
Sensor_View-Win7_64bit-101_blanks-rev_27536-32to137MB.JPG	130 KB	04/21/2011	Aaron Aaron
Tagging-Win7_64bit-101_blanks-rev27540-33to33MB.JPG	94.3 KB	04/22/2011	Aaron Aaron
Provenance-Win7_64bit-101_blanks-rev27540-33to35MB.JPG	96.3 KB	04/22/2011	Aaron Aaron
Workflow_Run_Manager-Win7_64bit-101_blanks-rev27540-37to46MB.JPG	93 KB	04/22/2011	Aaron Aaron
Reporting-Win7_64bit-101_blanks-rev27540-41to46MB.JPG	93.6 KB	04/22/2011	Aaron Aaron
Sensor_View-Win7_64bit-101_blanks-rev27540_double_checking-39to143MB.JPG	104 KB	04/22/2011	Aaron Aaron
Sensor_View-Win7_64bit-101_blanks-rev27542_with_static_plots_leak_fixed-39to43MB.JPG	102 KB	04/22/2011	Aaron Aaron