Eclipse Unit Tests

Back to DRLVM Test Tracking

Summary

Mission and Status

Test Harmony with Eclipse Unit Tests (version 3.3) to achieve 98% pass rate for all available suites.

EUT is currently demonstrates 99.5+% pass rate on both Windows (2003 Server) and Linux (SLES10) x86 platforms. Please see the testing results at automated testing report page which is:

```
http://people.apache.org/~mloenko/snapshot_testing/script/snapshots_summary.html
```

Back to Summary

How to run EUT

1. Prepare Common Environment

1.1 Download Eclipse SDK and Eclipse Unit Tests 3.3 archives from:

```
http://archive.eclipse.org/eclipse/downloads/drops/R-3.3-200706251500
```

where the tests archive name is:

```
eclipse-Automated-Tests-3.3.zip
```

and depending on platform the Eclipse SDK archive names are:

```
eclipse-SDK-3.3-win32.zip
eclipse-SDK-3.3-linux-gtk.tar.gz
eclipse-SDK-3.3-linux-gtk-x86_64.tar.gz
```

1.2 Choose Reference Java to launch Eclipse

EUT architecture allows to use two runtimes - one to launch Eclipse and parse EUT setup files and second one is to run tests themselves. Using Harmony as both runtimes is the subject of future work. For now `HotSpot VM` should be used as the Eclipse launcher - just add this java to path like:

```
export PATH=<path to HostSpot Server VM 1.5.0>
```

1.3. Patch SWT suite or set the stable proxy

If you are running EUT from outside the firewall or do not run `swt` suite then just skipt this step.

Recently it was discovered (HARMONY-5306) that `swt` suite accesses web pages and it hangs if such a page is unavailable (regardless to the runtime being used). So one needs to setup a stable proxy (it looks like that `swt` takes it from Mozilla or IExplorer settings) or rebuild `swt` test suite classes and data to replace external URLs to any internal one which does not require a proxy to be accessed. The patching is described in 2. Running EUT within Build & Tests infrastructure section below.

1.3*. Patch jdtdebug suite to run on x86_64

If you run the `jdtdebug` suite (separately or in whole EUT suites pack) you need to patch `jdtdebug` suite. The patching is described in 2. Running EUT within Build & Tests infrastructure section below.

1.4. Prepare Windows environment

1.4.1 On Windows the EUT must be run from `Command Prompt` (running from `Cygwin` is not tested).

1.4.2 EUT requires `unzip` command to be available, its documentation recommends using *Info-Zip* one (`http://www.info-zip.org`).

1.4.3 There is a path length restriction on Windows for EUT, so the safest way to do not break it is to choose the working directory as close to root as possible (like `c:\tmp-eut33`).

1.5 Prepare Linux environment

1.5.1 EUT is a GUI application. You need to care about correctly configuration and starting X-server/connection if you run EUT on remote host.

1.5.2 As a temporary workaround for HARMONY-2914 set open files limit to 16256 as:

```
$ulimit -n 16256
$ulimit -n
16256
```

You may need to have the root rights to increase the hard limit before setting the soft one as written above.

1.6 Setup cvs server

If you are running `teamcvs` suite (or just the whole EUT pack) you need to configure cvs server and add the cvs-related settings to configuration files (the files path depends on the way you choose to run EUT). The following values should be specified:

```
user  - account name used to connect to the cvs repository
password - the account password
host - the name of cvs server
root - the repository path
```

for example:

```
cvs_user=user1
cvs_password=verylongpassword
cvs_host=mycvshost.com
cvs_root=/localdisk/users/cvsroot
```

Note: the remained steps of environment preparation depend on a way of EUT runnning which are described below.

2. Running EUT within Build & Tests infrastructure

There is a Cruise Control based automated testing infrastructure. In particular it contains the ant-based script to run EUT. It is the safest way to run EUT because this script does most of preparation work - you need just to download the archives, provide the path to Harmony JDK and run the script.

2.1 Checkout Build & Tests infrastructure (BTI):

```
svn checkout -r HEAD https://svn.apache.org/repos/asf/harmony/enhanced/buildtest/trunk/tests/eut eut33
```

Enter the resulted `eut33` directory as the main work directory for the steps below.

To proceed using the checked-out scripts you need to add ant1.6.5 (or later) to your environment like:

```
export ANT_HOME=<path to ant 1.6.5 or later>
export PATH=$ANT_HOME/bin:$PATH
```

You may also need to care about proxy settings for ANT if you work from inside the firewall:

```
export ANT_OPTS=-Dhttp.proxyHost=<your proxy host> -Dhttp.proxyPort=<your proxy port>
```

2.2 Place the downloaded archives into checkout directory like (for Linux x86_64):

```
cp <eclipse-Automated-Tests-3.3.zip> .
mkdir -p eclipse-testing
cp <eclipse-SDK-3.3-linux-gtk-x86_64.tar.gz> eclipse-testing/.
```

If you do not have these archives downloaded yet then next step with gets them.

2.3 Build utilities classes

```
ant setup
```

This builds a set of utility classes required for EUT results processing. Also this downloads and places accordingly the EUT archives if they are not available yet - like described in #1.1 and #2.2 above.

2.4 Patch EUT3.3 data

One needs to patch the `swt` suites data if one runs EUT from inside the unstable firewall (see HARMONY-5306 for details). Also `jdtdebug` is to be patched if you run on x86_64 arch. The patching process is the same for all suites, so the steps below described `swt` patching for example.

Pick the local site you want to access from EUT (say, `http://localhost:8080`) and execute the steps below:

```
cd extra/eut.3.3.swt.patching
```

Copy already downloaded archives to avoid one more downloading from web:

```
cp ../../eclipse-Automated-Tests-3.3.zip .
cp ../../eclipse-testing/eclipse-SDK-3.3-win32.zip .
```

Edit 'properties' file to specify your local URL:

```
cat properties | sed -e 's@^swt.*@swt.local.url=http://localhost:8080@' > X
mv X properties
```

Do the tests patching by simply executing:

```
ant
```

Save the patched archive for future use, copy it to `eut33` root directory at least:

```
cp ./patched/eclipse-Automated-Tests-3.3.zip ../../.
```

2.5 Download ant contrib archive

Download ant contrib archive to `eut33` root directory from:

```
http://ant-contrib.sourceforge.net
```

In the later steps it is supposed that `ant-contrib-1.0b3.jar` is downloaded and used.

2.6 Run EUT with BTI scripts

Now you are ready to run EUT. Ok, since running requires `ant contrib` each run command example below should also contain the following proprty set:

```
-Dext.ant-contrib.location=ant-contrib-1.0b3.jar
```

For example to run a single suite - `jdtdebug` - execute the following command from eut33 directory:

```
ant -Dtest.jre.home=<path to harmony jdk> -Dvm.options="-Xmx1024m" -Dtests=jdtdebug
```

You can run several suites like:

```
ant -Dtest.jre.home=<path to harmony jdk> -Dvm.options="-Xmx1024m" -Dtests="jdtdebug,compare,ant"
```

Or all available suites by simply omitting "tests" setting:

```
ant -Dtest.jre.home=<path to harmony jdk> -Dvm.options="-Xmx1024m"
```

You may define these properties also in `eut.properties` file.

Note: there is no need to clean-up EUT files between runs - the latest version of EUT/BTI cares about such a cleaning before each suite run.

2.7 Understanding EUT-from-BTI results

If there is no internal errors in BTI scripts then the `results/latest/index.html` files is created which contains the summary of the results of recent run. This file contains the following information:

- relative summary - the relative pass rate for the suites you have just run, so it can be 100% if you run a single suite and all its tests passed
- absolute summary - it shows the absolute pass rate calculated based on total tests number in EUT which is over 40'000
- link to output.txt - contains EUT run output (both standard and error)
- link to report.txt - contains unexpected failures/errors details (like stack traces) collected from correspondent xml reports, recently the number of failures/errors details was limited to reduce report.txt size.
- link to eut.efl file - there is no eXclude list for EUT, all of the tests are run, still the results are filtered based on Expected Failures List (EFL)
- Suites Details information - the following rules are used to create this table:
    - if the suite was not run then it is gray in the table
    - if the suite ended normally (without crash or hang) and related html report was generated then the suite record is linked to this html file
    - if the suite crashed or hanged (and killed by timeout) then it is highlighted with red
    - if the suite contains no unexpected errors/failures then no errors/failures are given for it in this summary table

Back to Summary

3. Running EUT using its own scripts

Suppose that you've successfully done all steps from the section #1 above, and EUT related archives are placed in the current directory... Then:

3.1 Unpack tests and move SDK to 'eclipse-testing':

```
unzip -qq eclipse-Automated-Tests-3.3.zip
cd eclipse-testing
mv ../eclipse-SDK-3.3-linux-gtk-x86_64.tar.gz .
```

You entered the resulted `eclipse-testing` directory which is the main work directory for the steps below.

3.2 Prepare properties file:

Create any property file (say, `eut.run.properties`) with the content like:

```
J2SE-5.0=<your Harmony JDK Home>/bin/java
jvm=<your Harmony JDK Home>/bin/java
extraVMargs=-showversion -Xmx1024M -Duser.home=<your home> -Djava.io.tmpdir=<your temp>
runtimeArchive=eclipse-SDK-3.3-linux-gtk-x86_64.tar.gz
user.home=<your home>
java.io.tmpdir=<your temp>
```

You may miss the `user.home` and `java.io.tmpdir` redefinition still it is recommended to do this settings as well as removing all data from these directories before each run of suite.

If you are running `teamcvs` suite you need to configure cvs server (this subject is out of this document scope) and add the cvs-related settings to properties file like:

```
cvs_user=<account name used to connect to the cvs repository>
cvs_password=<the account password>
cvs_host=<the name of cvs server>
cvs_root=<the repository path>
```

3.3 Patch `runtests` script (Linux only):

You need to comment out the DISPLAY setting in the `runtests` script. For example:

```
cat runtests | sed -e 's@^DISPLAY@#DISPLAY@' > X
mv X runtests
chmod a+x runtests
```

3.4 Run EUT with its own scripts

Now you are ready to run EUT. To run all of the suites on SLES10 x86_64 execute the command:

```
./runtests -properties eut.run.properties -os linux -ws gtk -arch x86_64
```

Execute the same command appended by particular suite name to run this suite only like:

```
./runtests -properties eut.run.properties -os linux -ws gtk -arch x86_64 jdtdebug
```

You need to clean all of data created by Eclipse/EUT to insure the accurate run of next suite. The safest way to do this is to repeat all steps starting from 3.1 above before each suite run.

3.4 Understanding EUT-run-by-own-scripts results

The results/logs files are located at `results` directory. For example, to get the result of `jdtdebug` suite run first of all see the file:

```
results/html/org.eclipse.jdt.debug.tests_linux.gtk.x86_64.html
```

Back to Summary

4. Running EUT testcase with java command

Suppose that you've successfully done all steps from the section #1 above, and EUT related archives are placed in the current directory... Then:

4.1 Unpack tests and SDK:

For example, for SLES10 x86_64 execute the following:

```
unzip -qq eclipse-Automated-Tests-3.3.zip
cd eclipse-testing
unzip -qq eclipse-junit-tests-I20070625-1500.zip
tar xzf ../eclipse-SDK-3.3-linux-gtk-x86_64.tar.gz
```

In case of `eclipse-SDK-3.3-win32.zip` unzipping you need to allow overiding to `All` files.

You entered the resulted `eclipse-testing` directory which is the main work directory for the steps below.

4.2 Detect command line parameters for suite

First, get the failed suite class name, for example from main page of EUT report on automated testing page (see URL above). An example of such a name is `org.eclipse.jdt.debug.tests.AutomatedSuite`.

Then you can get the plugin name based on package name of the class you found from `eclipse-testing/test.xml` from the structure like:

```
<target name="jdtdebug">
        <runTests testPlugin="org.eclipse.jdt.debug.tests" />
</target>
```

Note, that sometimes one plugin keeps several suites, so there may be no 1-to-1 correspondance - be creative to derive the plugin name.

Finally you need to detect if the suite is `uitestapplication` or `coretestapplication`. Inspect the `test.xml` of related plugin to get this like:

```
eclipse/plugins/org.eclipse.jdt.debug.tests_3.1.0/test.xml
```

and look for `ant target=` pattern in it like:

```
    <ant target="ui-test" antfile="${library-file}" dir="${eclipse-home}">
```

So, if the target is `ui-test` then you get `uitestapplication`, otherwise it is `coretestapplication`.

Another way to get this data is to just follow the "Properties>>" link at the right bottom part of suite report (in case such a report is available) like:

```
http://people.apache.org/~smishura/r603534/Linux_x86_64/eut33/results/html/org.eclipse.jdt.debug.tests_linux.
gtk.x86_64.html
```

and inspect the `eclipse.commands` value (to long to be listed here). For example for `jdtdebug` suite you may see:

```
-application org.eclipse.test.uitestapplication
-testPluginName org.eclipse.jdt.debug.tests
-className org.eclipse.jdt.debug.tests.AutomatedSuite
```

The simplest way is to use the table below in the EUT3.3 information table section.

4.3 Run EUT with java command

You are ready to run EUT suite with java command. For example, for jdtdebug suite on Linux x86_64 the command looks like:

```
export JAVA_HOME=<HARMONY_JDK_HOME>
export PATH=$JAVA_HOME/bin:$PATH

${JAVA_HOME}/bin/java \
        -showversion \
        -jar eclipse/plugins/org.eclipse.equinox.launcher_1.0.0.v20070606.jar \
        -application org.eclipse.test.uitestapplication \
        -dev bin \
        -data workspace \
        formatter=org.apache.tools.ant.taskdefs.optional.junit.XMLJUnitResultFormatter,log.xml \
        -testpluginname org.eclipse.jdt.debug.tests \
        -classname org.eclipse.jdt.debug.tests.AutomatedSuite \
        2>&1 | tee log.txt
```

If you need to specify more properties to this run you need to use the standard way to pass them like:

```
       ...
       -Duser.home=<your home> \
       -Djava.io.tmpdir=<your temp> \
       -jar eclipse/plugins/org.eclipse.equinox.launcher_1.0.0.v20070606.jar \
       ...
```

4.4 Understanding EUT-run-with-java-command results

The EUT/suite results are located at `log.xml` (because this `log.xml` was specified for `formatter` value in the command above). For example, one of the first lines gives the status of suite run like:

```
<testsuite errors="0" failures="3" name="AutomatedSuite" package="org.eclipse.jdt.debug.tests" tests="598"
time="517.00">
```

Also the process standard/error output was redirected to `log.txt` - you may want to see it also.

5. Running EUT under Eclipse GUI

Finally you may run EUT as JUnit Plug-in Tests in Eclipse project (which is very helpful for single test debugging).

You need to unpack tests and SDK as described in 4.1 section above. After that you need to do the following steps:

- Run `eclipse` from `<path to your eclipse-testing>/eclipse` directory
- Select new workspace directory - make sure you control this location to clean up it before next run of Eclipse
- Select menu `"File"->"Import"`

- In the newly appeared "Import" dialog expand "Plug-in Development" and select "Plug-ins and Fragments", click "Next" button
- In the "Plug-ins and Fragments" dialog:
  - Uncheck the checkbox "The target platform (as specified in the Preferences)" and make sure that "Plug-in Location" is set to <path to your eclipse-testing>/eclipse
  - Click "Source Code Locations..." button - in the opened "Preferences" dialog go to "Source code locations" tab, click "Add..." button and select the following location:

```
<path to your eclipse-testing>/eclipse/plugins/org.eclipse.sdk.tests.source_3.2.0.v20070607/src
```

- Click "Ok" to submit your selection, then click "Ok" to close "Preferences" dialog
- In the "Import As" pane select "Projects with source folders" radio button and click "Next"

- In "Selection" dialog select the test plugin name from the list (like org.eclipse.jdt.debug.tests), click "Add -->" button and "Finish" button.

Now the test suite is imported as an Eclipse project and should be built (check "Build Automatically" in the "Project" menu if unchecked). When it shows no errors, do the following:

- Select the project in "Package Explorer" view (Alt+Shift+Q,P)
- Go to menu "Run"->"Open Run Dialog..."
- Select "JUnit Plug-in Test" here and click a "New" icon in the left top corner of the dialog window
- In the right pane of this "Run" dialog:
  - Choose the "Test" tab, check "Run a single test" and specify here:
    - plugin name like org.eclipse.jdt.debug.tests as "Project:"
    - test class name like org.eclipse.jdt.debug.tests.AutomatedSuite as "Test class:"
  - Then go to the "Main" tab, select "Run an application:" radio button and choose "org.eclipse.ui.ide.workbench" from the drop-down-list.
  - In the same tab click to "Installed JREs" button and specify the path to Harmony JRE, then select it as "Runtime JRE"
  - Finally click the "Run" button.

In a minute the suite will be launched and you will have "Junit" view opened with test and testcases tree. You can use this view to re-run one test class (or testcase) if necessary.

Back to Summary

EUT3.3 information table

Suite name	Plugin name	Application type	Test Class	Tests	Platform
ant	org.eclipse.ant.tests.core	core	org.eclipse.ant.tests.core.AutomatedSuite	85	all
antui	org.eclipse.ant.tests.ui	ui	org.eclipse.ant.tests.ui.testplugin.AntUITests	171	all
compare	org.eclipse.compare.tests	ui	org.eclipse.compare.tests.AllTests	60	all
coreexpressions	org.eclipse.core.expressions.tests	ui	org.eclipse.core.internal.expressions.tests.AllTests	94	all
filebuffers	org.eclipse.core.filebuffers.tests	core	org.eclipse.core.filebuffers.tests.FileBuffersTestSuite	362	all
coreresources	org.eclipse.core.tests.resources	core	org.eclipse.core.tests.resources.AutomatedTests	862	all
coreruntime	org.eclipse.core.tests.runtime	core	org.eclipse.core.tests.runtime.AutomatedTests	360	all
coretestsnet	org.eclipse.core.tests.net	core	org.eclipse.core.tests.net.AllNetTests	6	all
jdtapt	org.eclipse.jdt.apt.tests	ui	org.eclipse.jdt.apt.tests.TestAll	109	all
jdtcorebuilder	org.eclipse.jdt.core.tests.builder	core	org.eclipse.jdt.core.tests.builder.BuilderTests	175	all
jdtcorecompiler	org.eclipse.jdt.core.tests.compiler	core	org.eclipse.jdt.core.tests.compiler.parser.TestAll	5763	all
core	org.eclipse.jdt.core.tests.compiler.regression.TestAll	8213	all		
core	org.eclipse.jdt.core.tests.eval.TestAll	531	win32		
jdtcoremodel	org.eclipse.jdt.core.tests.model	core	org.eclipse.jdt.core.tests.RunFormatterTests	758	all
core	org.eclipse.jdt.core.tests.dom.RunAllTests	2917	all		
core	org.eclipse.jdt.core.tests.model.AllJavaModelTests	5328	all		
jdtcoreperf	org.eclipse.jdt.core.tests.performance	core	org.eclipse.jdt.core.tests.performance.OneTest	1	all
jdtdebug	org.eclipse.jdt.debug.tests	ui	org.eclipse.jdt.debug.tests.AutomatedSuite	598	all
jdttext	org.eclipse.jdt.text.tests	ui	org.eclipse.jdt.text.tests.JdtTextTestSuite	515	all
jdtuirefactoring	org.eclipse.jdt.ui.tests.refactoring	ui	org.eclipse.jdt.ui.tests.refactoring.all.AllAllRefactoringTests	3449	all
jdtui	org.eclipse.jdt.ui.tests	ui	org.eclipse.jdt.ui.tests.AutomatedSuite	1488	all
ui	org.eclipse.jdt.ui.tests.LeakTestSuite	9	win32		
jfacedatabinding	org.eclipse.jface.tests.databinding	core	org.eclipse.jface.tests.databinding.BindingTestSuite	730	all
jface	org.eclipse.jface.text.tests	core	org.eclipse.jface.text.tests.JFaceTextTestSuite	138	all
ltkcorerefactoringtests	org.eclipse.ltk.core.refactoring.tests	ui	org.eclipse.ltk.core.refactoring.tests.AllTests	1	all
ltkuirefactoringtests	org.eclipse.ltk.ui.refactoring.tests	ui	org.eclipse.ltk.ui.refactoring.tests.AllTests	1	all
osgi	org.eclipse.osgi.tests	core	org.eclipse.osgi.tests.AutomatedTests	198	all
pdeui	org.eclipse.pde.ui.tests	ui	org.eclipse.pde.ui.tests.AllPDETests	310	all
relEng	org.eclipse.releng.tests	core	org.eclipse.releng.tests.BuildTests	5	all
swt	org.eclipse.swt.tests	core	org.eclipse.swt.tests.junit.AllTests	5297	win32
core	org.eclipse.swt.tests.junit.AllGtkTests	5276	linux		

teamcore	org.eclipse.team.tests.core	core	org.eclipse.team.tests.core.AllTeamTests	15	all
ui	org.eclipse.team.tests.core.AllTeamUITests	1	all		
teamcvs	org.eclipse.team.tests.cvs.core	ui	org.eclipse.team.tests.ccvs.core.AllCoreTests	41	all
ui	org.eclipse.team.tests.ccvs.core.AllTests	155	all		
text	org.eclipse.text.tests	core	org.eclipse.text.tests.EclipseTextTestSuite	441	all
ua	org.eclipse.ua.tests	ui	org.eclipse.ua.tests.AllTests	228	all
uieditors	org.eclipse.ui.editors.tests	ui	org.eclipse.ui.editors.tests.EditorsTestSuite	11	all
uinavigator	org.eclipse.ui.tests.navigator	ui	org.eclipse.ui.tests.navigator.NavigatorTestSuite	12	all
uircp	org.eclipse.ui.tests.rcp	core	org.eclipse.ui.tests.rcp.RcpTestSuite	29	all
uiviews	org.eclipse.ui.tests.views.properties.tabbed	ui	org.eclipse.ui.tests.views.properties.tabbed.AllTests	6	all
ui	org.eclipse.ui.tests	core	org.eclipse.jface.tests.AllTests	674	all
core	org.eclipse.ui.parts.tests.PartsReferencesTestSuite	12	all		
ui	org.eclipse.ui.tests.UiTestSuite	1234	all		
ui	org.eclipse.ui.tests.session.SessionTests	25	all		
uiworkbenchtexteditor	org.eclipse.ui.workbench.texteditor.tests	ui	org.eclipse.ui.workbench.texteditor.tests.WorkbenchTextEditorTestSuite	56	all
update	org.eclipse.update.tests.core	core	org.eclipse.update.tests.AllTests	130	all

Back to Summary

Back to DRLVM Test Tracking