FAQ

JT Harness FAQ

Common questions about the JT harness project are answered here.

About the JT Harness Project

- What are the licensing terms for JT harness?
  A: The JT harness project is licensed under the GNU Public License, version 2, with the Classpath Exception. See the licensing section of the download page for more information.Questions

- What is the classpath exception?
  A: The classpath exception was developed by the Free Software Foundation's GNU/Classpath Project (see http://www.gnu.org/software/classpath/license.html). It allows you to link an application available under any license to a library that is part of software licensed under GPL v2, without that application being subject to the GPL's requirement to be itself offered to the public under the GPL.

- Why do you need the classpath exception?
  A: If the JT harness was distributed under GPL v2, that application could be subject to the requirements of the GPL that all code that is shipped as part of a "work based on the [GPL] program" also be GPL licensed. Accordingly, a GPL license exception is needed that specifically excludes from this licensing requirement any application that links to the GPL implementation. The classpath exception accomplishes this.

- What is the relationship between the JT harness project and the Mobile & Embedded community?
  A: The cqME project is the open source home for technologies used to test the Java Platform, Micro Edition (Java ME platform). The Mobile and Embedded community includes the cqME project. These ME-specific test technologies are built around the JT harness, and the relationship between the JT harness project and the cqME project is a close one. In fact, both projects share the same governance model and licensing requirements.

- How can I submit or suggest changes to the JT harness?
  A: We encourage you to contribute source code to the JT harness project. In accordance with the JT harness governance process, you can submit code to the project leader either as a user, a developer, or as a committer. If your code is accepted to be integrated into the source tree, you will be asked to sign a Contributor's Agreement similar to what Apache Software Foundation requires. For more information about the governance process and the various roles for members in the community, please review Mobile & Embedded Community Governance.

  Note that the JT harness plug-in architecture enables you to create custom implementations of most of the core functionality. Many customizations to the JT harness can be accomplished by that method, and not by changing core functionality.

- What documentation is available for developers?
  A: The layout of the JT harness repository is described in JT Harness Repository Structure.

  The JT Harness Developer's Guide describes how the source code is organized and provides important information for developers who want to modify the code.

  The JT Harness Source Code Description describes how the JT harness source code is organized.

  The JT Harness Build Instructions describes how to build the harness.

- Can I change the project documentation?
  A: The JT harness online help is in the code repository under `trunk/code/src` and is released under GPL version 2. The JavaTest harness product documentation hosted on java.sun.com is provided solely for your convenience and is not being released under the GPL version 2 as part of this project. These documents are part of the JavaTest harness commercial product and cannot be modified.

- Do you have any plans to translate the project into any other languages?
  A: We do not have any plans or resources to do this, although the code is written so that translation could occur with minimal or no changes to the code. Resource bundles are used extensively throughout the harness code. If you would like to attempt a translation, you must locate each of the resource bundles in each source package, and identify the strings associated with messages to be translated. You should also consider translating the HTML-based JavaHelp online help files associated with the online documentation.

- Do you take into account assistive device technologies? Can I use a screen reader?
  A: The code is written attempting to fill in all the fields necessary for the use of accessibility technologies. If you would like to test these capabilities, we will be happy to assist you if you encounter any problems.

How to Run the JT Harness

- How do release numbers of releases on the binary download page relate to the repository or the trunk of the repository?

A: During development of a release, the trunk is used to create the binary development (dev) images. Developers add to the trunk during their development process, so it is a changing body of code. When a release is completed, that final body of code is copied into the `branches/code/` directory in the SVN repository and marked with the appropriate version number. That code corresponds to the milestone release you will see on the download page.

If you ever have trouble finding something, please post a question in the JT harness forum. One of the developers will explain the current status.

- In the tutorial, the metadata to describe a test is located in the test source `.java` files. Is this required? What if my test suite does not contain `.java` files?
  A: This is just one way that a test suite can present the metadata. Another popular method is to use HTML files for this purpose. Different TestFinders are included in the project for this purpose. You can also write a custom TestFinder that extracts this information in a way that works best for your test suite. See the *JavaTest Architect's Guide* and Javadoc generated documentation for information about this.

- The `jtharness.zip` file does not have a top-level directory to encapsulate the contents being unpacked. This makes a mess in my current directory if I forget to unpack it in a separate directory. Is this intentional?
  A: Yes, it is created this way because the product is usually packaged within another deliverable (for example, a test suite product). Because the harness is often imported from the Zip bundle. we do not create a top-level directory because it creates an unnecessary directory. We realize this is not optimal for a person previewing the JT harness and will consider changing the Zip layout for future releases.

- Is there a way to pause or resume a test run?
  A: This functionality is not provided by JT harness directly, but it can be provided by a test framework. For example, the ME Framework allows users to pause a test for an unlimited time.

- Is the ordering of items on the command line significant?
  A: Yes. See the *JavaTest User's Guide - Command-Line User Interface* available from java.sun.com or the JT harness online help for more information.

- We have a test suite and run it as part of the nightly builds. How can I avoid having to re-answer the configuration interview questions each day?
  A: You can save the interview to a `.jti` file and specify that file when you run the tests in batch mode. You can change the answers as required on the batch command line.

- My entire organization uses basically the same configuration settings for running the test suite. Do we all need to answer the entire configuration interview and keep track of our own interview files?
  A: No. An "administrator" can set up a central repository of interview templates that answer all the questions that the organization has in common. Users only have to answer questions that vary from individual to individual. The templates can be created using bookmarks so that only the questions that require change are displayed.

- Are interviews forward and backwards compatible?
  A: If the configuration interview changes and a set of answers is loaded, question-answer pairs that were saved from the old interview are still valid in the new interview. However, depending on the changes, the new interview might be incomplete.

- I run tests using a status filter, and the tests seem to disappear from the display when their result status changes to "pass". Also, the progress bar indicates that the tests passed, failed, and so on, but they do not appear in statistics that way. What's going on?
  A: Most likely the tests moved into the "filtered out" category because they no longer fit the selection criteria defined in the current configuration, which is what the tree and its statistics show by default. You can change your view filter to change this effect. Try using the "All Tests" or "Last Test Run" filters. This is a side-effect of maintaining a GUI state that is always consistent. It is important to remember that the tree and its statistics represent the contents of the entire work directory. The progress bar and progress monitor dialog represent only the last or current test run. Understanding that is vital to understanding the numbers in the GUI display.

- The progress monitor dialog box (or any dialog box) insists on staying on top of the main JT harness window. Is there anything I can do about that?
  A: Unfortunately, the JT harness does not have complete control over window ordering. This is a function that your operating system window manager controls. The implementation of this functionality might be changed in a future release to allow the main window to cover the dialog box.

- When I exit the JT harness GUI are my settings saved?
  A: All GUI settings are saved to your desktop unless you set the preference that prevents them from being saved. If you restart the JT harness, your settings are reinstated. However, if you start the JT harness on the command line with explicit configuration flags or with the `-newDesktop` flag, the settings are not reinstated. A few options are saved in the preferences subsystem, that survive beyond the saved "desktop".

  Note: Beginning in version 4.3 of the harness, settings are no longer saved by default. You must specify that you want them saved using the Preferences dialog box. If you use version 4.3 or later on a work directory created by an earlier version, the default settings used by the earlier version are retained.

- Why doesn't the log viewer show any output?
  A: The harness itself generates minimal output to the logging subsystem. Most of the output to the logging subsystem is generated by the test suite itself. Some test suites create more output than others. Developers should attempt to use the built-in logging system whenever possible, instead of sending output to `System.err` and `System.out`.

- I'm testing my app in an ME emulator. Can the harness set up my emulation environment for me?
  A: Yes, starting with version 4.2 of the harness, an ME emulator can be started using the service management facility. Emulators (as well as other test services) can be specified as services to be started by the harness before a test run. The built-in service management facility can identify this at runtime and then request that service to start. The request to start the service is done through the use of Ant, which allows the architect or tester to configure the operations required to activate a particular service.

  For more information about using service management see the *JavaTest Harness 4.2 Architect's Guide*. Note that earlier versions of this guide do not include this information.

- Can I exchange configurations (`.jti` files) across different locales?

A: Yes. In harness versions prior to 4.2 it should normally work, but unintended behavior might occur in cases where questions store numeric information. The interview stores incorrect answers as strings and saves them for later use. If the user later returns to that question in the Configuration Editor, the invalid answer appears for editing. For example, if a user enters 1,4 for a floating point value in the "en,US" (US English) locale it is invalid. However, if that JTI is transferred to the "fr,FR" (France French) locale, the number becomes valid because that locale uses the comma as the floating point separator.

Beginning with version 4.2, invalid values are no longer stored. In addition, the original and current locales are taken into account when reading floating point and integer answers from the JTI file. Invalid answers now stay invalid or blank, and valid answers remain valid.

If the JTI file is written out again, it is done so using the current locale. There is currently no way to save the results in a locale different from the one you are using. As a workaround, you can use your operating system settings and run the harness in a different locale.

- Where does the harness store information on my computer? How can I delete it or use it to troubleshoot?
  A: The harness writes information to only one location that the user does not explicitly specify — the `.javatest` folder. The `.javatest` folder is created in the directory specified by the user.home JVM system property. This folder primarily contains your preference settings and stored desktop information. If you want to restore your system to the state it was in before you ever ran JT harness, you can delete this folder. It is not usually necessary to manipulate this folder. All other data is written to locations specified by the user in the harness GUI.

  The work directory contains multiple log files that can sometimes be useful in sorting out problems. Feel free to contact the developers via the forum on java.net if you want help interpreting the output in those files.

- My test suite needs some separate processes for the tests to run, can the harness start/stop them for me?
  A: Yes, there are two general ways to do this. In the past, this was usually accomplished by a wrapper script that started and stopped the harness. This is a reasonable solution, but does not allow for more integrated behavior such as restarting services during the test run for example, capturing of log output within the harness and starting services on-demand.

  The harness provides two ways to start and stop services. First is by monitoring the observer messages from the harness that indicate the start and completion of a test run. See the API documentation for the TestSuite and Harness.Observer interfaces for information. determined Beginning with version 4.2.1 of the harness, an integrated service management subsystem exists that allows the starting and stopping of services on a per-test run or as-needed basis. These services can be programatically specified, through a XML based configuration file or by a combination of those. See Chapter 11 in the *JavaTest Architect's Guide* for more information.

- With services, how are the binaries located on the machine under test?
  A: This is configured by the test suite. Consult the test suite documentation for details. These paths are usually set by answering a question in the configuration interview or by changing a value in an XML file.

- Are any port (number) management capabilities provided?
  A: This is not available yet, but would be useful for dynamically finding an available port on which to run a service — for example, if you want to run two test runs simultaneously and both tests need an HTTP and HTTPS server. The test suite needs to find four available ports to start the services, and having this managed by the harness might be useful.

  If you would like to help implement a post/resource management system, contact the developers in the JT harness forum on java.net.

Reporting Test Results

- Can I generate a report in a custom format? For example, can I generate a report as an XML file for a given schema?
  A: The test suite architect (not testers) can provide plug-in code to add to the available report types. The custom report API provides access to the entire set of results, including all test output and the current environment (configuration). All this can be used to generate or execute whatever the architect needs. GUI panels can also be provided so that the tester can configure the options for that report type in the Create Report dialog box. Access to the custom report generation functionality is also available from the command line.

  See `com.sun.javatest.exec.ContextManager` and `com.sun.javatest.report.CustomReport` in the API documentation. Please ask questions in the JT harness forum if you need more information.

- How can I use the known failures list (KFL)?**
  A: This feature is useful if you generate reports and repetitively do identical test runs. It enables you to more easily locate tests that change from one known state to another. For example, the KFL reporting feature can help you see if a test has been failing for the past month but suddenly passes today.

- How are test cases supported inside the KFL report?**
  A: The harness has not previously supported test cases in past releases. Starting in release 4.4, test cases are supported in the KFL report. In the 4.4 release, test cases are only identified in one format — in an output section named "out1" inside the test result. Each test case is identified by a string matching the following general form:

  ```
  (sometext): (Status string)
  ```

  Where (sometext) is the test case name, specified without any internal whitespace, followed by a colon. This is followed by a string that begins with a status string from the `com.sun.javatest.Status` object (Failed, Passed, Error, Not Run). The following is an example of the output:

  ```
  MyTest0001: Passed. Security test 10001.
  ```

- What is the format of the known failures list (KFL)?**
  A: The general format is:

```
testname[testcases,...] [bugid,bugid]
```

Please see the *JavaTest User Guide* for more information. The file format is currently the same as the exclude list. In future versions of the harness a revised format may be introduced. If this happens, you can identify it by a version string on a comment line at the top of the file.

Using the JT Harness to Create a Test Suite

- What do I have to do to run my test suite using JT harness?
  A: Because the JT harness can run any type of test, you should not have to modify the tests themselves. However, you must take some steps so that the JT harness can find the tests and set up the appropriate execution environment. Using the JT harness plug-in architecture, you can customize the harness so that it can find the tests and execute them properly. You can also modify the test suite so that it runs with the default harness components. This functionality is discussed in detail in the *JavaTest Architect's Guide* that you can download from java.sun.com.

- What types of tests can I execute with JT harness?
  A: The JT harness can execute any type of tests including conformance tests, performance tests, unit tests, and integration tests.

- Can I execute native code in tests?
  A: Yes. You have to make sure the libraries that contain the native code are available when the tests are run.

- Can I execute "distributed" tests that consist of multiple components requiring synchronization?
  A: Yes. The JT harness does not provide distributed test libraries with the core package. This functionality is available with extensions such as the ME Framework.

- What factors affect initial startup performance and test suite loading time?
  A: Factors include the following:

  - Size of the test suite.
  - Whether the test suite is run in the JT harness GUI, or using batch mode. The GUI has higher overhead and is slower.
  - Use of the binary test finder. The binary test finder uses an optimized test description format that significantly improves the startup performance of large test suites.
  - Amount of physical memory available on the machine (if it is in limited supply).
  - Location of the test suite being loaded. If the test suite is unpacked to a local file system, performance is probably better than if it is accessed over a network.

- What factors affect how fast the test suite runs?
  A: Factors include the following:

  - Test framework overhead.
  - Proper scheduling of tasks. The JT harness (and your test framework if you use one) can be configured for optimal performance in your test environment.
  - Concurrent test execution. This functionality can be provided by either by the harness or the test framework.
  - The speed of the communication channel between the JT harness host and the test platform affects performance.
  - In some cases, the performance of the file system on which the work directory is stored can affect throughput.

- What is an agent?
  A: It is a small application that communicates with the JT harness to get and execute tests and send back results. It is used in situations where the JT harness is not run directly on the test platform.

- Do I have to write my own agent?
  A: Situations could arise to make this necessary, but it is unlikely. This situation might arise if your test platform uses a communication protocol such as UDP, which is only guaranteed to be available on the CDC platform. Another example is if the platform requires a specific application model like the PBP Xlet. Custom agents are available for those and other situations from the ME Framework.

- I have an existing test suite (for example, JUnit tests). What is required to run these tests in JT harness?
  A: The first step is to read the *JavaTest Harness 4.2 Architect's Guide* to get a general understanding of the harness architecture and terminology. If you are running JUnit tests, the guide covers the procedure for running those types of tests with little or no modification.

  If you are running tests of a different format, you must generally create your own test finder and test runner classes. The harness uses these classes to find and run your tests. Both classes are subclassed from existing library classes. They can be passed to the harness by specifying them in your `testsuite.jtt` file. For more information consult the *JavaTest Hareness 4.2 Architect's Guide* and ask questions on the JT harness forum.

Using JT Harness with JUnit Tests

- What additional work is needed/planned for JT harness JUnit support?
  A: There are some ways that the use of classpath and diagnostics can be made easier to use. JUnit also provides an additional API for capturing test output that could be integrated with similar support in JT harness. In the future, we hope to add more information to the *JavaTest Architect's Guide* section that covers JUnit integration.

- Can any JUnit test suite be integrated into the harness?

A: JUnit test suites are relatively free-form, which can be an advantage, but does make it difficult to make a generalized method for integrating those tests into a harness. It depends on the test's style of execution and the configuration/parameter requirements of the test suite. The amount of work to integrate the test can be minimal, or may require an understanding of the JT harness APIs and may require you to write some framework code.

- Where should I start my conversion/creation of a JUnit test suite within the JT harness?
  A: If you have not used the JT harness before, you should first run the tutorial — Chapter 2 in the *JavaTest Architect's Guide*. You will find a link to the *JavaTest Architect's Guide* can find on the JT Harness Documentation page. The tutorial will familiarize you with how the harness behaves with a working test suite so you will know what the goal is when creating your JUnit suite. The *JavaTest User's Guide* will also help you understand how the harness works.

  After you familiarize yourself with the harness, read the JUnit integration section of the *JavaTest Architect's Guide* (Chapter 12). To fully understand that section, you should also read at least the introduction to Chapter 6 (Configuration Interview) and 9 (Test Finder), parts of Chapter 4 (Creating a Test Suite) all of Chapters 3 and 8. This should give you enough background to start playing with files.