Jtest Tracer Client

This topic covers the Jtest Tracer Client tool, which is used to generate JUnit tests based on the test actions it monitors.

Sections include:

Understanding Jtest Tracer Client

If you have Jtest Tracer (available with Parasoft Jtest) listening to your Java application or web service, the Jtest Tracer tool in SOAtest can be used to control when to start and stop recording method invocations, what classes or packages' methods will be monitored, and where the recorded data gets routed.

Tracer provides a fast and easy way to create realistic functional JUnit test cases that capture the functionality covered by your SOAtest test cases. Using Tracer, you can trace the execution of Java applications at the JVM level (without the need to change any code or to recompile), and in the context of a larger integrated system. As your SOAtest test cases execute, Tracer monitors all the objects that are created, all the data that comes in and goes out.

The trace results are then used to generate contextual JUnit test cases that replay the same actions in isolation, on the developer's desktop, without the need to access all the application dependencies. This means that you can use a single machine to reproduce the behavior of a complicated system during your verification procedure.

Since the generated unit tests directly correlate tests to source code, this improves error identification and diagnosis, and allows developers to run these test cases without having to depend on access to the production environment or set up a complex staging environment. This facilitates collaboration between QA and Development: QA can provide developers traced test sessions with code-level test results, and these tests help developers identify, understand, and resolve the problematic code.

Prerequisites

- You must have Parasoft Jtest (with a Tracer license) available on a system that your team can access. This cross-product capability requires Jtest and SOAtest to be installed installed via p2 updatesite archives (see Eclipse p2 Update Site Installation for details).
- You must have the appropriate platform Jtest Tracer libraries (*.dll for Windows, lib*.so for Linux, and libpmt.sl for HP-UX) for the machine that is running the application you want to trace. If the application you want to trace is running on a different platform than the machine where you have Jtest installed, then the Jtest installation for that platform will need to be downloaded so that the libraries (*.dll, lib*.so, or libpmt.sl) can be extracted and copied to that machine.

Configuring Jtest Tracer Client

Before you can "trace" an application to create test cases, you must perform the following one-time configuration steps:

1. Ensure that the Tracer libraries are available on the machine that is running the application you want to trace.
    - See the Jtest documentation for details.
2. Set that system's path to reference the Tracer libraries.
    - See the Jtest documentation for details.
3. Start the server with the appropriate JVM arguments for tracing.
    - See the Jtest documentation for details.
4. Specify which test(s) you want to trace by adding Jtest Tracer Client tools to the test suite. *You need to add two Jtest Tracer Client tools: one as a set-up tool and one as a tear-down tool.*
    - See the following section for details.

Specifying which Test(s) to Trace

You specify which test(s) to trace by adding two Jtest Tracer Client test tools to your test suite:

- A Jtest Tracer Client - Start Trace as a set-up test before the first test action that you want to trace.
- A Jtest Tracer Client - Stop Trace tool as a tear-down test after the last test action that you want to trace.

To configure this:

1. Right-click the Test Case Explorer node for the test suite or test where you want to start tracing and select **Add New> Test** from the shortcut menu.
2. In the Add Test wizard, select **Set-Up** from the left pane, **Jtest Tracer Client** from the right pane, and click the **Finish** button. A **Set-Up: Jtest Tracer Client** node displays within the selected test suite.
3. Configure the set-up test as follows:
    a. Go to the **Jtest Tracer Client** test configuration panel (if it is not visible, double-click the Jtest Tracer Client node to open it).
    b. Specify the host and port that you will connect to. 6543 is the default port.

        - For example, if the service endpoint is http://www.foo.bar.com:1234, you would enter `www.foo.bar.com` for host and `1234` for port.
        - http endpoints are supported. tcp ones are not.
    c. Ensure that the **Start trace** button is selected.
    d. Specify where you want to save the result file (the file that Jtest will use to generate test cases).
        - If you want to save it to a local system (the machine that is running SOAtest), select **Local file** and browse to (or enter) the local file path.
        - If you want to save it to a remote system (e.g., the system where the traced application is running), select **Remote file** and then enter a path to the desired location.
    e. Specify which classes/packages you want to monitor in the Monitored Classes/Packages area.
        - If you want to monitor all classes and packages that were specified in the JVM argument, leave **All** (the default) selected.
        - If you want to monitor only a subset of the classes or packages that were specified in the JVM argument, choose **Custom** and then specify the classes/packages in the table that opens.This is useful for focusing on specific use cases that only pertain to a certain subset of classes. All classes and packages must be fully qualified and packages must end with a `".*"` character sequence.
    f. Click **Save**.
4. Right-click the Test Case Explorer node for the test suite or test where you want to stop tracing and select **Add New> Test** from the shortcut menu.
5. In the Add Test wizard, select **Tear-Down** from the left pane, **Jtest Tracer Client** from the right pane, and click the **Finish** button. A **Tear-Down: Jtest Tracer Client** node displays within the selected test suite.
6. Configure the tear-down test exactly as you configured the set-up test, but with one exception: Select the **Stop trace** button instead of the **Start trace** button.

Generating Test Cases

To generate test cases, you need to:

1. Run the SOAtest test suite that includes the Jtest Tracer Client as a set-up tool and tear-down tool (as described above). This will produce the Tracer output file. When the test execution is complete, a Tracer output file will be saved on either the local or remote machine (depending on whether the configuration was set to use a local file or remote file).
2. Use Parasoft Jtest to generate JUnit test cases from the tracer output file.
    - See the Jtest documentation for details.

Troubleshooting

Object states are reproduced by using Jtest's object repository feature in cases where the object was constructed prior to the start request. Due to memory issues, there is a limit to how large such objects can be. If the object exceeds the set limit, the restored object may not contain the necessary information to recreate the method invocation accurately. You can see which objects have not been created correctly by looking at the standard output of the application /service or by seeing if the generated test case looks incorrect. However, if the invocations do not interact with the unrecorded portions of the object, then your test case will be recreated perfectly. You may want to consider removing that large class from the monitor argument so we do not sniff it. If you believe an adequate amount of memory is available to capture larger objects, contact Jtest Support for instructions on how to increase the recorded object size limit. You may also increase the amount of memory that your application or web service uses. We suggest you always allocate the most memory possible for your application.

Java compilation also has size limitations and you may run into them when generating tests with Jtest. When recreating tests, we make sure that the calling sequence is true to the one that was actually enacted. Therefore, there are times where in order to recreate the calling sequence perfectly, we are adding in a large number of method invocations. The compiler may not be able to handle such large methods.

We already have measures to parse out method calls that do not mutate any data. If you see this problem, you can modify your monitor argument to not sniff this class or you can use Jtest Tracer Tool's start and stop action commands to create a more focused sub section of calls.