FAQs and Troubleshooting

In this section:

General

How can I get quick access to information about usage on the command line?

Use the `-help` command line switch:

```
jtestcli.exe -help
```

How can I work with Jtest via proxy?

Typically, if you connect through a proxy server, you need to configure the connection by passing protocol-specific system properties to the JVM – using the `-D` command line option.

To work with Jtest, ensure that the system properties for the HTTPS protocol are configured. At a minimum, you must configure `https.proxySet=true`, `https.proxyHost=[hostname]`, and `https.proxyPort=[port number]`. If your proxy server requires authentication, you can configure your credentials with the `https.proxyUser` and `https.proxyPassword` properties.

Your command line may resemble the following:

```
java -Dhttps.proxySet=true -Dhttps.proxyHost=myserver.example.com -Dhttps.proxyPort=8080 -Dhttps.
proxyUser=user1 -Dhttps.proxyPassword=MyPassword
```

In addition, you can configure the `https.nonProxyHosts` property to specify hosts where connection via proxy is not required.

If you use Jtest on desktop with Eclipse or IntelliJ IDEA, the proxy settings are automatically detected and do not need to be configured in the command line.

Installation

How can I prevent my machine ID from floating?

Changes in the network environment may affect the interface that is used to compute your machine ID and result in machine ID instability. You can use the PARASOFT_SUPPORT_NET_INTERFACES environment variable to specify a stable interface and prevent the machine ID from floating.

1. Set up the **PARASOFT_SUPPORT_NET_INTERFACES** environment variable.
2. Set the variable value to a stable Ethernet network interface. Do not use virtual, temporary or loopback interfaces.
   - On Windows: Set the value to the MAC address of your network card. You can use the `ipconfig -all` command to obtain the address. For example:

   ```
   SET PARASOFT_SUPPORT_NET_INTERFACES=00-10-D9-27-AC-85
   ```

   - On Linux: Set the value to one of the network interfaces from the "inet" or "inet6" family. You can use the `ifconfig` command to obtain the list of available interfaces. For example:

   ```
   export PARASOFT_SUPPORT_NET_INTERFACES=eth1
   ```

If the problem persists, you can obtain diagnostic information by setting up the environment variable **PARASOFT_DEBUG_NET_INTERFACES** and setting its value to true. This will print to the standard output the checking procedure that can be shared with technical support, as well as the interface that is used to compute your machine ID. The interface will be marked with the [SELECTED] prefix.

Testing and Analysis

What if the static analysis performance decreased after Jtest was updated to a newer version?

The new version of Jtest may require more memory to run static analysis. You can increase memory allocation by configuring the -Xmx option in the [INSTALL_DIR]/etc/jtestcli.jvm configuration file.

What if Jtest reports compilation problems when building the project with Maven?

If Jtest reports compilation problems related to importing dependencies in the Maven test scope, try running the `jtest:jtest` goal with the `mvn test` co mmand:

```
mvn test jtest:jtest
```

Why does running the `jtest:coverage` goal (Maven) or `jtest-coverage` task (Gradle) fail the build?

The Jtest `coverage` goal/task was dropped with release 10.2.2 (plugin version 1.2.4) and is no longer available (see Migration From 10.x to 10.2.2 or later) . Executing this goal fails the build with the following messages:

`[ERROR] Could not find goal 'coverage' in plugin com.parasoft.jtest:jtest-maven-plugin`. (Maven)

Task 'jtest-coverage' not found (Gradle)

What if my Gradle build fails when configured to perform test impact analysis with Jtest?

A Gradle build may fail when you perform test impact analysis with Jtest on multi-module projects if a submodule does not contain tests that are identified as affected tests during test impact analysis. To prevent a Gradle build from failing when no tests are found, configure the following option: `setFailOnNoMatchingTests(false)`

What if Jtest cannot collect coverage information for tests run with Gradle?

To enable collecting coverage, Jtest automatically configures the JVM arguments for the Jtest coverage agent. Overriding these arguments prevents Jtest from collecting coverage data. If your Gradle build script modifies JVM arguments, ensure they are added (**+=**) to other JVM arguments to prevent overriding:

✗ jvmArgs **=** ['--module-path', classpath.asPath, '--add-modules', 'ALL-MODULE-PATH', '--add-reads', "$moduleName=junit", '--patch-module', "$moduleName=" + files(sourceSets.test.java.outputDir). asPath]	✓ jvmArgs **+=** ['--module-path', classpath.asPath, '--add-modules', 'ALL-MODULE-PATH', '--add-reads', "$moduleName=junit", '--patch-module', "$moduleName=" + files(sourceSets.test.java.outputDir). asPath]

What if Jtest cannot resolve required dependencies while running analysis or executing tests with Maven?

To ensure that all of the required dependencies are resolved, you may need to enable resolving transitive dependencies by adding the following option to your command line:

```
-Djtest.resolve.transitive=true
```

Why is test impact analysis with Maven re-running one or more tests that are NOT affected by code changes?

- Test impact analysis re-runs entire test suites when at least one test included in the test suit is affected by code changes. As a result, if the test suite contains tests that are not affected, they are re-run as well.
- In rare cases, test impact analysis may be unable to recognize the include or exclude pattern. As a workaround, you can filter your tests manually; see Manually Filtering Tests to Re-run for details.

How can I test source files where a byte order mark (BOM) appears at the start of the text stream?

To ensure that files that include BOM are properly processed and tested, you need to configure Jtest to remove BOM when the file is parsed. Add one of the following settings to the `jtest.properties` configuration file:

- `jtest.parser.bom=remove` - removes BOM and leaves the original file encoding unchanged.
- `jtest.parser.bom=recognize` - removes BOM and overrides the original fine encoding with the encoding defined by BOM

What if Jtest reports flaky tests as failed when running tests with Maven or Gradle?

By default, Jtest does not differentiate between tests that always fail when run on the same code and tests that can either pass or fail (flaky tests). You can configure Jtest to recognize flaky tests by enabling the deprecated XML processing mechanism to be used for test execution. Add the following option in the `jtestcli.properties` configuration file:

```
jtest.unittest.xml.results.processing.enabled=true
```

Note that enabling the deprecated XML processing mechanism may slow down test execution.

What if tests fail when running PowerMock with mockito-inline?

When running PowerMock tests with mockito-inline on the classpath before powermock, you might encounter test failures with the following exception:

```
NotAMockException: Argument should be a mock, but is: class java.lang.Class
```

This is caused by an issue with how PowerMock interacts with the mockito-inline mock maker (see PowerMock issue for details).

There are several possible solutions depending on your requirements:

- Remove mockito-inline or put powermock before mockito-inline in classpath. You can use this option if you do not need static and constructor mocking with Mockito.
- Update PowerMock statements to use method calls which correctly support mockito-inline. Some PowerMock statements have no clean update path and will require a workaround. See the examples:
  - Updating PowerMock tests (static mocking) - Example
  - Updating PowerMock tests (verify) - Example
- Switch to Mockito static and constructor mocking using mockito-inline. This requires a rewrite of all Powermock calls to use analogous Mockito API. See the examples:
  - Converting PowerMock tests to Mockito (static mocking) - Example
  - Converting PowerMock tests to Mockito (constructor mocking) - Example

There are some limitations concerning the classes and methods which can be mocked. For details, see Mockito static and constructor mocking limitations.

Updating PowerMock tests (static mocking) - Example

The following PowerMock calls which cause an issue can be easily modified:

- thenReturn()
- doReturn()
- doNothing()

For example:

```
PowerMockito.when(MyClass.staticCall(any())).thenReturn("value");
PowerMockito.when(MyClass.staticCall(any())).thenReturn("value1", "value2");
PowerMockito.doReturn("value").when(MyClass.class);
MyClass.staticCall(any());
PowerMockito.doNothing().when(MyClass.class, "staticCall", any());
```

Updated code:

```
PowerMockito.when(MyClass.staticCall(any())).thenAnswer(invocation -> "value");
PowerMockito.when(MyClass.staticCall(any())).thenAnswer(invocation -> "value1").thenAnswer(invocation ->
"value2");
PowerMockito.doAnswer(invocation -> "value").when(MyClass.class);
MyClass.staticCall(any());
PowerMockito.doAnswer(invocation -> null).when(MyClass.class, "staticCall", any());
```

Updating PowerMock tests (verify) - Example

When calling verifyStatic() or verifyPrivate() you might encounter the following test exception:

```
org.mockito.exceptions.misusing.NotAMockException:
Argument passed to verify() is of type Class and is not a mock!
Make sure you place the parenthesis correctly!
See the examples of correct verifications:
    verify(mock).someMethod();
    verify(mock, times(10)).someMethod();
    verify(mock, atLeastOnce()).someMethod();
```

The code should be refactored to manually verify the required call.

For example:

```
PowerMockito.when(MyClass.staticCall(any())).thenAnswer(invocation -> "value");
...
PowerMockito.verifyStatic(MyClass.class, times(1));
MyClass.staticCall(any());
```

Updated code:

```
int counter[] = new int[]{0};
PowerMockito.when(MyClass.staticCall(any())).thenAnswer(invocation -> {
    counter[0]++;
    return "value";
});
...
assertEquals(1, counter[0]);
```

For verifyPrivate() the only solution is to manually verify the required call since Mockito does not support mocking private methods.

For example:

```
PowerMockito.mockStatic(MyClass.class);
PowerMockito.when(MyClass.class, "privateStatic", any()).thenAnswer(invocation -> inv.callRealMethod());
...
PowerMockito.verifyPrivate(MyClass.class, times(1)).invoke("privateStatic");
```

Updated code:

```
PowerMockito.mockStatic(MyClass.class);
int counter[] = new int[]{0};
PowerMockito.when(MyClass.class, "privateStatic", any()).thenAnswer(invocation -> {
    counter[0]++;
    return inv.callRealMethod();
});
...
assertEquals(1, counter[0]);
```

Converting PowerMock tests to Mockito (static mocking) - Example

PowerMock code:

```
PowerMockito.mockStatic(MyClass.class);
PowerMockito.when(MyClass.staticCall(any())).thenReturn("value");
// test code
PowerMockito.verifyStatic(MyClass.class, times(1));
MyClass.staticCall(any());
```

Code converted to Mockito:

```
try (MockedStatic<FunctionsToMock> mocked = mockStatic(FunctionsToMock.class)) {
    mocked.when(() -> FunctionsToMock.staticCall(any())).thenReturn("value");
    // test code
    mocked.verify(() -> FunctionsToMock.staticCall(any()), times(1));
}
```

Converting PowerMock tests to Mockito (constructor mocking) - Example

PowerMock code:

```
FunctionsToMock mock = Mockito.mock(FunctionsToMock.class);
PowerMockito.whenNew(FunctionsToMock.class).withAnyArguments().thenReturn(mock);
when(mock.calculate()).thenReturn(1);
// test code
```

Code converted to Mockito:

```
try (MockedConstruction<FunctionsToMock> mocked = mockConstruction(FunctionsToMock.class, (mock, context) -> {
    when(mock.calculate()).thenReturn(1);
})) {
    // test code
}
```

Mockito static and constructor mocking limitations

The following cases are not supported by Mockito:

- Private methods are not mockable by Mockito
- The following classes are prevented from being static and constructor mocked to not interfere with Mockito internals:
    - java.lang.ClassLoader
    - java.lang.Thread
    - java.lang.Object
    - java.lang.System

    - java.util.Arrays
    - java.util.Locale
    - java.util.concurrent.ConcurrentHashMap
    - Abstract classes

  Mockito also does not recommend static mocking any class of the JDK.

- Mocking these classes will cause a stack overflow exception or other infinite loop:
    - java.io.File

What if a file cannot be tested because its name is inconsistent between the source control and the testing input?

To correctly process and analyze a file, Jtest must be able to match the file name you provided in the testing scope (i.e. the file name in your project) with the file name in the source control system. For this reason, if you are using a case-sensitive source control system, such a Git, you need to ensure that the file name capitalization is identical.

Reports

What if some characters fail to be properly displayed in Jtest reports?

Reports generated by Parasoft products require a sans-serif font to be available in your environment. If your report fails to correctly display some characters, such as national characters, ensure that a sans-serif font is installed on your system.

Why are the issue tracking tags not available in the report after I run Jtest with Maven?

Your report may not include the issue tracking tags you have configured (see Associating Tests with Development Artifacts) if Jtest is unable to resolve one or more required dependencies. To prevent this, see What if Jtest cannot resolve required dependencies while running analysis or executing tests with Maven?.

Working in the IDE

Why does test execution fail when the path to UTA files exceeds 250 characters?

On Windows, a file path longer than 250 characters may prevent accessing the file. If your workspace is deeply nested in the folder structure, the path to some UTA files may exceed 250 characters. If the files that are key to test execution cannot be accessed, test execution fails.

To ensure that the path to the UTA files does not exceed the limit, you may consider moving your workspace folder higher in the folder hierarchy.

What if the Jtest run configuration does not work in IntelliJ IDEA?

In rare cases, the Jtest run configuration may not work due to a known issue in some IntelliJ versions (see https://youtrack.jetbrains.com/issue/IDEA-223124 for details). If this happens, you need restart your IDE. The Jtest run configuration will be working after the restart.