JUnit 5 Release Notes

Stefan Bechtold, Sam Brannen, Johannes Link, Matthias Merdes, Marc Philipp, Christian Stein

Version 5.1.1

Table of Contents

This document contains the *change log* for all JUnit 5 releases since 5.0 GA.

Please refer to the User Guide for comprehensive reference documentation for programmers writing tests, extension authors, and engine authors as well as build tool and IDE vendors.

5.1.1

Date of Release: April 8, 2018

Scope: Bug fixes and minor improvements since the 5.1.0 release

For a complete list of all *closed* issues and pull requests for this release, consult the 5.1.1 milestone page in the JUnit repository on GitHub.

JUnit Platform

New Features and Improvements

- New `findAllClassesInModule()` method in `ReflectionSupport` which enables third-party `TestEngine` implementations to scan for classes in modules — for example, when processing a `ModuleSelector` during the discovery phase.

JUnit Jupiter

New Features and Improvements

- The `ParameterContext` API used in `ParameterResolver` implementations now includes the following convenience methods for looking up annotations on parameters. Extension authors are strongly encouraged to use these methods instead of those provided in the core `java.lang.reflect.Parameter` API due to a bug in `javac` on JDK versions prior to JDK 9 which causes lookups for annotations on parameters in inner class constructors to fail consistently — for example, when resolving a parameter for a `@Nested` test class constructor.
  - `boolean isAnnotated(Class<? extends Annotation> annotationType)`
  - `Optional<A> findAnnotation(Class<A> annotationType)`
  - `List<A> findRepeatableAnnotations(Class<A> annotationType)`

JUnit Vintage

No changes.

5.1.0

Date of Release: February 18, 2018

Scope:

- Discovering tests in Java 9 modules
- Improved Kotlin support
- Programmatic extension registration via `@RegisterExtension`
- Tag expression language for filtering tests to be executed
- Annotation-based conditional test execution with support for environment variables, system properties, operating systems, JRE versions, and dynamic scripts
- Various improvements for writing parameterized tests
- Refinements to the `ExtensionContext` API
- Support for re-running individual dynamic tests, parameterized tests, and test template invocations within an IDE

For a complete list of all *closed* issues and pull requests for this release, consult the 5.1 M1, 5.1 M2, 5.1 RC1, and 5.1 GA milestone pages in the JUnit repository on GitHub. This section describes all *changes* from version 5.0.3 to 5.1.0.

JUnit Platform

Bug Fixes

- Test methods selected by *fully qualified method name* via `DiscoverySelectors.selectMethod(String)`, via the `method` or `methods` element of the `selectors` configuration for the `junitPlatform` Gradle plugin, or via the `-select-method` or `-m` command-line options for the `ConsoleLauncher` can now contain special characters — for example, for JVM languages such as Kotlin and Groovy.

New Features and Improvements

Support for Java 9 Modules

- New `ModuleSelector` discovery selector for scanning Java 9 modules for test classes.
  - This is an alternative to the existing classpath scanning support.
- New console launcher option `--select-module <name>` or `-o <name>` for selecting Java 9 modules for test discovery.
  - This is an alternative to the existing classpath scanning support.
- New console launcher option `--scan-modules` for scanning all resolved Java 9 modules available on the boot layer configuration for test discovery.
  - This is an alternative to the existing classpath scanning support.
- When running on Java 9 or higher the `default` implementations of `getVersion()` and `getArtifactId()` in the `TestEngine` interface query the *Java Platform Module System* for this information.

Miscellaneous

- Tests can now be included or excluded based on their tags using a tag expression language when executed using the `Launcher`, `ConsoleLauncher`, Gradle plugin, or Maven Surefire provider.
- New `@SuiteDisplayName` annotation in the `junit-platform-suite-api` module for declaring a custom *display name* for a test suite.
    - Supported by the `JUnitPlatform` runner for JUnit 4 in the `junit-platform-runner` module.
- The summary table of a console launcher run now contains the initial ten stack trace lines to better describe the location of the failure.
- Class loading errors that occur during classpath scanning are now logged at `DEBUG` level (i.e., the `FINE` log level in `java.util.logging`) instead of as warnings.

JUnit Jupiter

Bug Fixes

- Test classes selected via one of the `selectClass()` variants in `DiscoverySelectors`, via the `aClass` or `classes` element of the `selectors` configuration for the `junitPlatform` Gradle plugin, or via the `-select-class` or `-c` command-line options for the `ConsoleLauncher` are no longer allowed to be `private`. This aligns with the behavior for test classes discovered via package, class path, and module path scanning.
- Null elements specified in the last column after the first row via `@CsvSource` for parameterized tests are now correctly converted to `null` instead of the empty `String`.

New Features and Improvements

Test Discovery

- The `JupiterTestEngine` supports the new JUnit Platform `ModuleSelector` for selecting Java 9 modules.
    - This is an alternative to the existing classpath scanning support.
- Selected dynamic tests and test template invocations can now be executed separately without running the complete test factory or test template. This allows to rerun single or selected parameterized, repeated or dynamic tests by selecting their unique IDs in subsequent discovery requests.

Programming Model

- Developers can now register extensions *programmatically* by annotating fields in test classes with `@RegisterExtension`.
    - See Programmatic Extension Registration in the User Guide for details.
- New predefined `@Enabled*` and `@Disabled*` annotations for declarative *conditional test execution*. See the following sections of the User Guide for details.
    - Operating Systems

  - Java Runtime Environment Versions
  - System Properties
  - Environment Variables
- New `@EnabledIf` and `@DisabledIf` annotations that can be used to control whether the annotated test class or test method is *enabled* or *disabled* by evaluating a script from a dynamic scripting language such as JavaScript or Groovy.
  - See Script-based Conditions in the User Guide for details.
- New `assertAll()` variants in `Assertions` that accept collections of executables.

Improved Kotlin Support

- New Kotlin friendly assertions added as *top-level functions* in the `org.junit.jupiter.api` package.
  - `assertAll()`: takes `Stream<() -> Unit>`, `Collection<() -> Unit>`, or `vararg () -> Unit`.
  - `assertThrows()`: uses Kotlin reified generics.
  - `fail()`: remove need to specify generic type explicitly.
    - When calling the `Assertions.fail` methods from Kotlin, the compiler required the generic return type of `fail` to be declared explicitly when calling it — for example, `fail<Nothing>("Some message")`. These new top-level functions remove this requirement by returning `Nothing`.

Parameterized Tests

- `@CsvFileSource` now supports a `numLinesToSkip` attribute which can be used to skip header lines in CSV files.
- `@ValueSource` now additionally supports literal values of type `short`, `byte`, `float`, `char`, and `java.lang.Class` for parameterized tests.
- The `value` attribute of `@MethodSource` is no longer mandatory. If no value (or an empty String) is supplied as a method name, a method with the same name as the current `@ParameterizedTest` method will be used as the factory method by convention.
  - See @MethodSource in the User Guide for details.
- New support for parameterized tests for implicit conversion from a `String` to an argument of any of the following common Java types. See the implicit conversion table in the User Guide for examples.
  - `java.io.File`
  - `java.math.BigDecimal`
  - `java.math.BigInteger`
  - `java.net.URI`
  - `java.net.URL`
  - `java.nio.charset.Charset`
  - `java.nio.file.Path`
  - `java.util.Currency`

    - `java.util.Locale`
    - `java.util.UUID`
- New fallback mechanism for parameterized tests for implicit conversion from a `String` to an argument of a given target type if the target type declares exactly one suitable *factory method* or a *factory constructor*.
    - See Fallback String-to-Object Conversion in the User Guide for details.

Extension Model

- Extensions for JUnit Jupiter can now access JUnit Platform configuration parameters at runtime via the new `getConfigurationParameter(String key)` method in the `ExtensionContext` API.
- Extensions for JUnit Jupiter can now access the `Lifecycle` of the current test instance via the new `getTestInstanceLifecycle()` method in the `ExtensionContext` API.
- New callback interface `CloseableResource` introduced in `ExtensionContext.Store`. A `Store` is bound to the lifecycle of its extension context. When the lifecycle of an extension context ends, the associated store is closed, and each stored value that is an instance of `ExtensionContext.Store.CloseableResource` is notified by an invocation of its `close()` method.
- New `getOrComputeIfAbsent(Class)` convenience method in `ExtensionContext.Store` that simplifies use cases where an extension wishes to store a single object of a given type (keyed by that type) in the `Store` and the object is created using the default constructor for that type.
    - For example, code such as `store.getOrComputeIfAbsent(X.class, key -> new X(), X.class)` can now replaced with `store.getOrComputeIfAbsent(X.class)`.

JUnit Vintage

Bug Fixes

- When using a tag filter to include/exclude a tag that represents a JUnit 4 category, e.g. `"com.example.Integration"`, the Vintage Engine no longer mistakenly executes all test methods of test classes that contain at least one included test method, e.g. one that is annotated with `@Category(com.example.Integration.class)`, regardless whether they belong to the same category.

New Features and Improvements

- The `VintageTestEngine` supports the new JUnit Platform `ModuleSelector` for selecting Java 9 modules.
    - This is an alternative to the existing classpath scanning support.
- Prior to this release, the Vintage test engine only returned a childless `TestDescriptor` for test classes annotated with `@Ignore`. However, build tools like Gradle need to show an accurate number of tests, i.e. they want to access and count the test methods of a test class regardless whether it’s ignored. The Jupiter engine already discovers skipped containers, e.g. test classes annotated with `@Disabled`, including their children and descendants. The Vintage engine now adopts this approach and returns a full subtree of `TestDescriptors` for classes annotated with `@Ignore`. During execution, it will only report the `TestDescriptor` of the test class as skipped which is consistent with how the Jupiter engine reports skipped containers.

5.0.3

Date of Release: January 15, 2018

Scope: Bug fixes and small improvements for `ConsoleLauncher`, Gradle plugin, and Maven Surefire provider

For a complete list of all *closed* issues and pull requests for this release, consult the 5.0.3 milestone page in the JUnit repository on GitHub.

Overall Improvements

- In 5.0.1, all artifacts were changed to have an *optional* instead of a mandatory dependency on the *@API Guardian* JAR in their published Maven POMs. However, although the Java compiler should ignore missing annotation types, a lot of users have reported that compiling tests without having the *@API Guardian* JAR on the classpath results in warnings emitted by `javac` that look like this:

  ```
  warning: unknown enum constant Status.STABLE
  reason: class file for org.apiguardian.api.API$Status not found
  ```

  To avoid confusion, the JUnit team has decided to make the dependency to the *@API Guardian* JAR *mandatory* again.

JUnit Platform

Bug Fixes

- Summary table is no longer printed via the `ConsoleLauncher` and Gradle plugin when details mode `NONE` is selected, unless there are errors.
- The XML report produced by the `ConsoleLauncher` and Gradle plugin is no longer invalid when the exception message of a failed test contains the XML CDATA end marker `]]>`.
- The `ConsoleLauncher`, the Gradle plugin, and the Maven Surefire provider now attempt to write a valid class name into the `classname` attribute of `<testcase/>` elements in the XML reports. In addition, the `name` attribute of dynamic tests and test template invocations (such as repeated and parameterized tests) is now suffixed with the index of the invocation so they are distinguishable by reporting tools.
- The Maven Surefire provider now includes the method parameter types when writing the `name` attribute of `<testcase/>` elements into XML reports. However, due to a limitation of Maven Surefire, instead of `methodName(Type)` they are written as `methodName{Type}`.
- Non-inherited *composed annotations* which are meta-annotated with a given `@Inherited` annotation are now considered to be implicitly *inherited* when searching for the given meta-annotation within a class hierarchy.

New Features and Improvements

- The JUnit Platform Maven Surefire provider now runs all specified tests in a single test run, i.e. all registered `TestExecutionListeners` will receive a single `TestPlan`. Previously, a separate `TestPlan` was discovered and executed for each test class.
- New `SUMMARY` details mode for the `ConsoleLauncher` and Gradle plugin which prints the table of success and failure counts at the end of test plan execution. This new mode is analogous to the previous behavior of the `NONE` details mode.
- The Maven Surefire provider now supports the `test` parameter that tells Surefire to only execute a subset of test classes or methods, e.g. by specifying `-Dtest=…` on the command line (see the Surefire documentation for details).

JUnit Jupiter

Bug Fixes

- The `@Tag` and `@Tags` annotations are now inherited within test class hierarchies.
- Due to a change in the JUnit Platform's `AnnotationUtils` class, non-inherited *composed annotations* which are meta-annotated with a given `@Inherited` annotation are now considered to be implicitly *inherited* when searching for the given meta-annotation within a class hierarchy.
  - For example, an `@Inherited` annotation such as `@TestInstance` will now be discovered on a custom *composed annotation* declared on a superclass even if the *composed annotation* is not declared as `@Inherited`.

JUnit Vintage

No changes.

5.0.2

Date of Release: November 12, 2017

Scope: Bug fixes and minor improvements since the 5.0.1 release

For a complete list of all *closed* issues and pull requests for this release, consult the 5.0.2 milestone page in the JUnit repository on GitHub.

JUnit Platform

Bug Fixes

- Failed tests are now reported correctly with Maven Surefire for test engines that do not use `MethodSource` (e.g. Spek).
- Tests that write to `System.out` or `System.err`, in particular via a logging framework, are now consistently reported correctly when executed with a non-zero `forkCount` with Maven Surefire.

New Features and Improvements

- The JUnit Platform Maven Surefire provider now supports the `redirectTestOutputToFile` Surefire feature.
- The JUnit Platform Maven Surefire provider now ignores empty strings supplied via `<includeTags/>`, `<groups/>`, `<excludeTags/>`, and `<excludedGroups/>`.

JUnit Jupiter

Bug Fixes

- Trailing spaces in a `@CsvSource` or `@CsvFileSource` input line no longer yield `null` values.
- `@EnableRuleMigrationSupport` previously failed to recognize `@Rule` methods that returned a subtype of one of the supported `TestRule` types. Moreover, it mistakenly instantiated some rules declared using methods multiple times. Now, once enabled, it will instantiate all declared rules (fields *and* methods) exactly once and call them in the same order used by JUnit 4.
- Previously, disabled test classes were eagerly instantiated when `Lifecycle.PER_CLASS` was used. Now, `ExecutionCondition` evaluation always takes place before test class instantiation.
- The `junit-jupiter-migrationsupport` module no longer incorrectly attempts to register the `JupiterTestEngine` via the `ServiceLoader` mechanism, thereby allowing it to be used as a module on the Java 9 module-path.

New Features and Improvements

- Failure messages for `assertTrue()` and `assertFalse()` in the `Assertions` class now include details about the expected and actual boolean values.
  - For example, the generated failure message for a call to `assertTrue(false)` is now "expected: <true> but was: <false>" instead of an empty string.
- If a parameterized test does not consume all arguments supplied to it via argument sources, the unconsumed arguments are no longer included in the display name.

JUnit Vintage

No changes.

5.0.1

Date of Release: October 3, 2017

Scope: Bug fixes for the 5.0.0 release

For a complete list of all *closed* issues and pull requests for this release, consult the 5.0.1 milestone page in the JUnit repository on GitHub.

Overall Improvements

- All artifacts now have an `optional` instead of a mandatory dependency on the *@API Guardian* JAR in their published Maven POMs.

JUnit Platform

No code changes.

JUnit Jupiter

Bug Fixes

- `ExpectedExceptionSupport` from the `junit-jupiter-migrationsupport` module no longer swallows exceptions if the test class does not declare a JUnit 4 `ExpectedException` rule.
  - Consequently, `@EnableRuleMigrationSupport` and `ExpectedExceptionSupport` may now be used without declaring an `ExpectedException` rule.

JUnit Vintage

Bug Fixes

- `PackageNameFilters` are now applied to tests selected via a `ClassSelector`, `MethodSelector` or `UniqueIdSelector`.

5.0.0

Date of Release: September 10, 2017

Scope: First General Availability Release

For complete details consult the 5.0.0 Release Notes online.