Artos Documentation

Release 01.00.0004

Arpit Shah
contributors

Apr 19, 2020

Contents

CHAPTER 1

Introduction

1.1 ARTOS (Art of System Testing)

Welcome to Artos Docs! Here, you can find Artos documentation, guidelines, as well as tips and tricks to help you start a successful test automation journey.

Artos is designed and developed by a team of experienced test engineers as a **free to use** and **open-source** project to help test community throughout the world. It is aimed at providing a test framework that is easy to use, reliable and works out of the box. Artos is written in Java which makes it suitable for Windows, Linux, MAC or any platform that runs Java. It can be used for functional, system, end to end and/or unit testing. Artos includes simple but powerful runner as well as many inbuilt and well-tested utilities that will save time for users to let them focus on what they do best!

1.2 Framework glossary

Keyword	Description
Test suite	A collection of test cases that are designed specifically to test the system under test
Test runner	A class which is the entry point to a test application. It is responsible for running and tracking test cases from the start to end
Test case	A class which contains set of instructions that will be performed on the system under test
Test unit	A method within a test case that represents the smallest and independent executable unit
Test context	A container object that stores and tracks test suite, test case and test unit related information
Test script	A set of instructions to guide the test runner on how to execute test cases. The test script is represented by xml script
Scan scope	A section of the Java project which will be scanned during the search of test cases
Test status	The state of a test case at the time of execution (namely: PASS, FAIL, SKIP or KTF)
Unit outcome	The outcome of the test unit (namely: PASS, FAIL, SKIP or KTF)
Test outcome	The outcome of the test case (namely: PASS, FAIL, SKIP or KTF)
Fail stamp	The text stamp added to a log stream at the line of failure
GUI test selector	Prompt that displays scoped test cases and allow the user to run selective test cases
Failure highlight	Textually representation of failed test cases and its unit for user

1.3 Abbreviations

Abbreviation	Description
KTF	Known To Fail
GUI	Graphic user interface

1.4 Annotations

ARTOS uses Java annotations for most of the feature sets. A list of supported annotations is provided below. Annotation in detail will be covered in later sections.

Annotation	Applies To	Usage
@TestCase	Class	Denotes that class is a test case
@TestPlan	Class	Declares information required for test plan
@Unit	Method	Denotes that method is a test unit
@BeforeTestSuite	Method	Denotes that the annotated method should be executed once before test suite execution
@AfterTestSuite	Method	Denotes that the annotated method should be executed once after test suite execution
@BeforeTest	Method	Denotes that the annotated method should be executed before each test case execution
@AfterTest	Method	Denotes that the annotated method should be executed after each test case execution
@BeforeTestUnit	Method	Denotes that the annotated method should be executed before each test unit execution
@AfterTestUnit	Method	Denotes that the annotated method should be executed after each test unit execution
@AfterFailedUnit	Method	Denotes that the annotated method should be executed after each failed test unit execution
@DataProvider	Method	Denotes that the annotated method is the supplier of a test data and declares a unique name for the method
@ExpectedException	Method	Declares rules of managing exception of the annotated method
@Group	Class/Method	Declares group name(s) that the annotated test case/unit belongs to
@KnownToFail	Method	Declares that the annotated test unit is known to fail
@TestImportance	Class/method	Declares importance of the annotated test case/unit
@TestDependency	Class	Declares test case dependency on other test case(s)
@UnitDependency	Method	Declares unit dependency on other unit(s) implemented within single test case
@StepDefinition	method	Declares step definition that binds test unit to feature file

1.5 Example test case and unit

```
1  import com.artos.annotation.*;
2  import com.artos.framework.Enums.Importance;
3  import com.artos.framework.Enums.TestStatus;
4  import com.artos.framework.infra.TestContext;
5  import com.artos.interfaces.TestExecutable;
6
7  @TestImportance(Importance.CRITICAL)
8  @Group(group = "Regression")
9  @TestPlan(preparedBy = "ArtosTeam", bdd = "GIVEN..WHEN..AND..THEN..")
10 @TestCase(sequence = 1)
11 public class Sample_1 implements TestExecutable {
12
13    @Group(group = "BADPATH")
14    @TestImportance(Importance.LOW)
15    @Unit(sequence = 1)
16    public void testUnit_1(TestContext context) {
17       // ------------------------------------------------
18       context.setTestStatus(TestStatus.FAIL, "Bad path");
19       // ------------------------------------------------
20    }
21
22    @Group(group = "GOODPATH")
23    @TestImportance(Importance.HIGH)
24    @Unit(sequence = 2)
25    public void testUnit_2(TestContext context) {
```

(continues on next page)

(continued from previous page)

```
26          // ------------------------------------------------
27          context.setTestStatus(TestStatus.PASS, "Good path");
28          // ------------------------------------------------
29      }
30
31  }
```

CHAPTER 2

Artos feature highlight

- Built-in and pre-configured log framework
  - Enable/Disable text and/or HTML formatted logs.
  - Enable/Disable logging at runtime.
  - Enable/Disable time-stamp and thread information.
  - Multiple log levels.

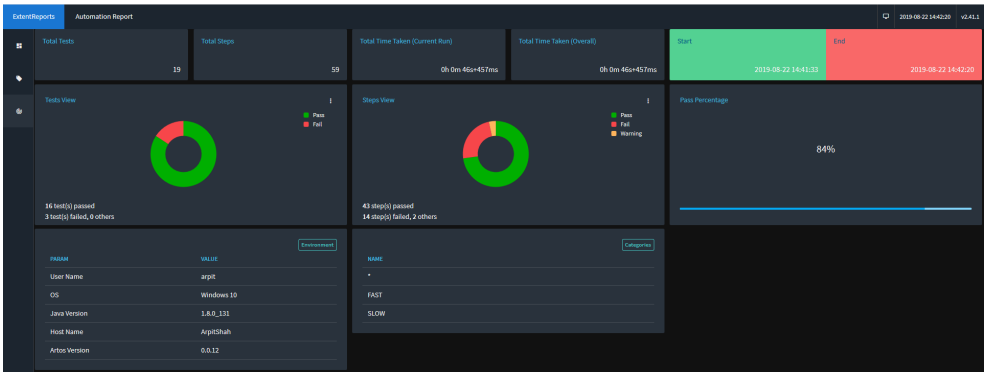
	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
OFF							
FATAL	x						
ERROR	x	x					
WARN	x	x	x				
INFO	x	x	x	x			
DEBUG	x	x	x	x	x		
TRACE	x	x	x	x	x	x	
ALL	x	x	x	x	x	x	x

  - Real-time log files in addition to a general log file (for performance measurement).

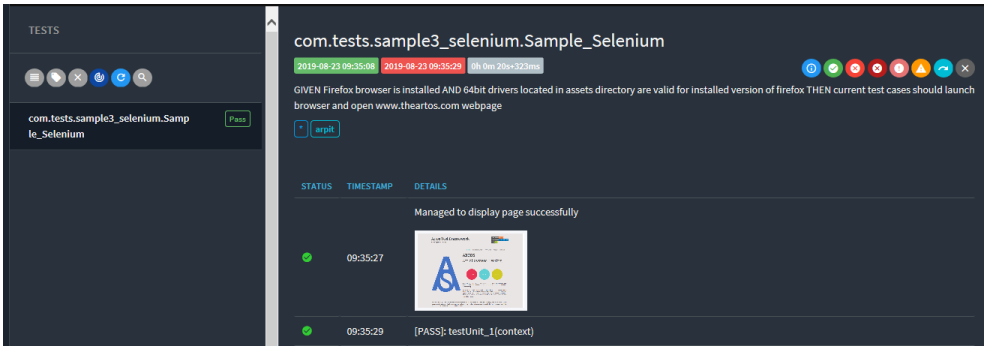
```
24 [TRACE][2019-08-22_14:42:19.855][pool-3-thread-1] - Req: 4869
25 [TRACE][2019-08-22_14:42:19.856][pool-6-thread-1] - Res: 4869
26 [TRACE][2019-08-22_14:42:19.857][pool-5-thread-1] - Res: 48656C6C6F2066726F6D206120536572766572
27 [TRACE][2019-08-22_14:42:19.857][Thread-4] - Req: 48656C6C6F2066726F6D206120536572766572
28 [TRACE][2019-08-22_14:42:19.858][pool-6-thread-1] - Res: 48656C6C6F2066726F6D206120436C696E6574
29 [TRACE][2019-08-22_14:42:19.858][pool-3-thread-1] - Req: 48656C6C6F2066726F6D206120436C696E6574
30 [TRACE][2019-08-22_14:42:19.858][pool-5-thread-1] - Res: 48656C6C6F2066726F6D206120536572766572
31 [TRACE][2019-08-22_14:42:19.858][Thread-4] - Req: 48656C6C6F2066726F6D206120536572766572
32 [TRACE][2019-08-22_14:42:19.859][pool-3-thread-1] - Req: 48656C6C6F2066726F6D206120436C696E6574
33 [TRACE][2019-08-22_14:42:19.859][pool-6-thread-1] - Res: 48656C6C6F2066726F6D206120436C696E6574
34 [TRACE][2019-08-22_14:42:19.859][Thread-4] - Req: 48656C6C6F2066726F6D206120536572766572
35 [TRACE][2019-08-22_14:42:19.859][pool-5-thread-1] - Res: 48656C6C6F2066726F6D206120536572766572
36 [TRACE][2019-08-22_14:42:19.860][pool-6-thread-1] - Res: 48656C6C6F2066726F6D206120436C696E6574
```

  - Separate log files per test suite during parallel testing.
  - Inbuilt log file roll over.
  - Log file tracking so user can parse, process or email log files at run time.

- Built-in report generation
  - Professional looking Extent report.



  - Attach snapshot to the Extent report.

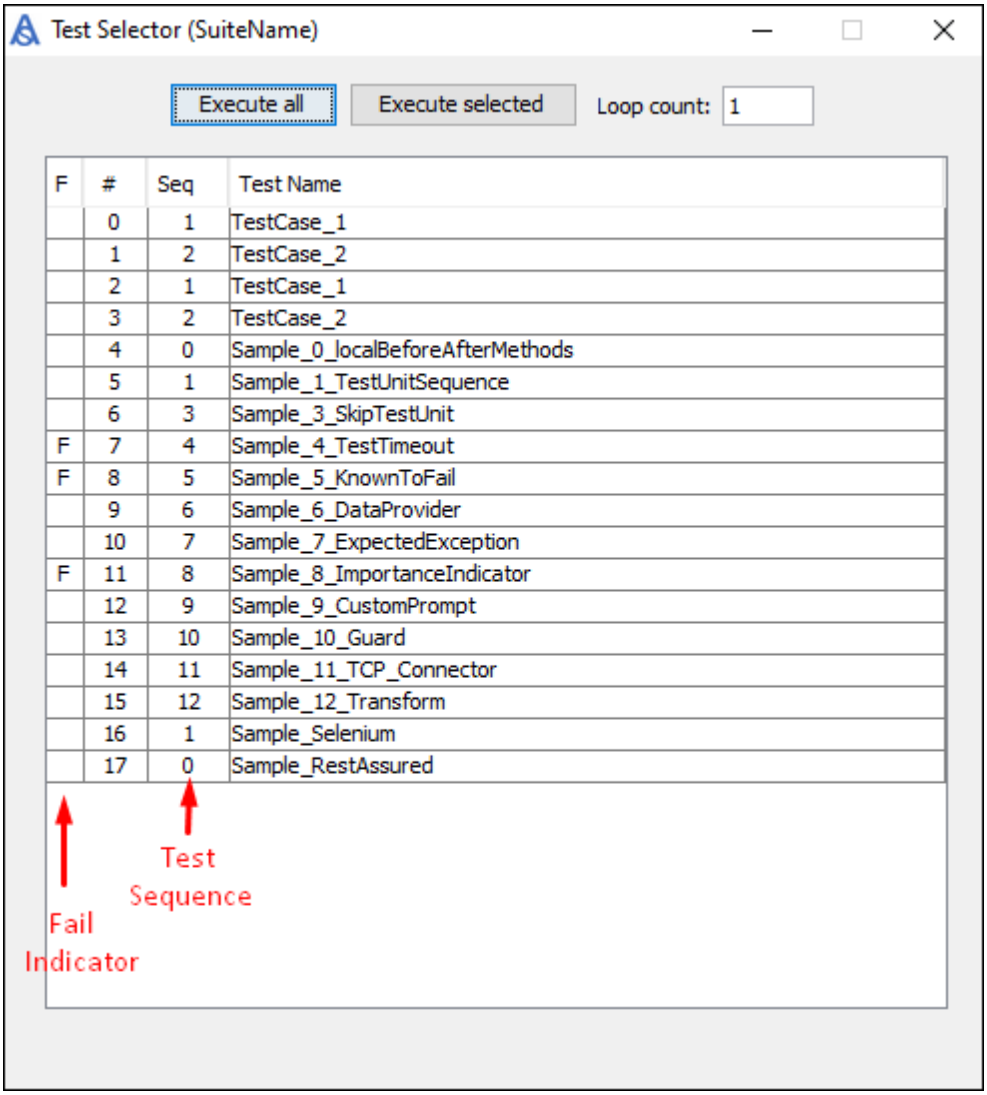


  - Text and/or HTML formatted summary report.
  - JUnit xml report.
  - Separate test reports per test suite during parallel testing.
- Test duration
  - Test-suite, test-case and test-units execution duration measurement with millisecond accuracy
- Importance indicator
  - Assign importance to test case/units which is visible in summary report.

```
**********************************************************************************
                    FAILED TEST CASES (3)

**********************************************************************************
1   com.tests.sample1_general.Sample_4_TestTimeout
        |-- testUnit_1(context) [LOW]
2   com.tests.sample1_general.Sample_5_KnownToFail
        |-- testUnit_2(context)
3   com.tests.sample1_general.Sample_8_ImportanceIndicator [CRITICAL]
        |-- testUnit_1(context) [LOW]
        |-- testUnit_2(context) [HIGH]
        |-- testUnit_3(context) [MEDIUM]
        |-- testUnit_4(context) [CRITICAL]
        |-- testUnit_5(context) [FATAL]
**********************************************************************************
```

- GUI test selector
  - Execute selective test cases or all without modifying code/script and avoids typing error.
  - Run same test multiple time using loop count.

  - Visible test sequence number.
  - Highlights previously failed test for quick re-run.
  - Enable/Disable GUI based on requirement.



- Failure indicator
  - Print failed test case tree at the end of console log.
  - Fail stamp injection in the log file to pinpoint the exact line of failure.
  - Highlight failed test from previous run in GUI test selector.
  - Bug/JIRA reference will be printed in summary for quick debugging.
- Known to fail
  - Segregate known to fail test cases from new failures
  - Warn user and trigger force failure in-case known to fail test case passes.
- Easy debugging
  - Add bug/JIRA reference to test case.
  - Add test description and BDD/Gherkin test plan within test case to avoid disconnect between test plan and test case.
  - Test case/unit writer/reviewers names can be added to test meta-data.

  - BDD test plan injection in the log file to avoid switch between test script and log files during debugging.
  - Real time log and general log are recorded in their own log files for easy debugging.
  - Automated fail test script generation for quick re-run.
  - Prints system info at the start of test execution for easy debugging.
  - Warns user with user friendly information printed on the console.
- TestCase development
  - Auto-generated test templates to speed up development
  - Group based test case and test unit filtering
  - Exception checking
  - Known to fail test case support
  - Data Provider support
  - Sequentialize test cases to maintain dependency and repeatability
  - Disable/Skip test cases
  - Global parameter support
  - Built-in utilities for test development (Data Transformation, CountDownTimer, Live display, Guardian, etc..)
  - Built-in connectors (TCP, UDP, etc..)
  - Global and local before-after method support.
- Stop on fail
  - Enable **Stop on Fail** to stop test execution after first failure.
  - Drop execution of remaining test case/unit if critical test case fails by enabling drop flag for given test case/unit.
- Parallel testing
  - Run test suites in parallel using test script.
  - Separate logs per test suite execution.
  - Separate extent reports per test suite execution.
  - Separate global variables and context per suite to avoid interference.
  - Execute same/different test cases per test suite.
  - Test multiple product/hardware/software at the same time using parallel testing.
- Cucumber BDD script (Behavior driven development)
  - Accepts Cucumber generated feature file for BDD testing
  - Auto generated BDD test templates to speed up development
  - BDD skeleton generation in case of missing methods
- Deployment
  - Support multiple framework configuration profiles for different environment.
  - Execute test cases with pre-defined test script.
  - Disable GUI test selector for production environment.
  - Automated test script generation.
  - Automated test script generation for failed test cases.

  - Log file tracking in-case log files are required to be emailed.
  - Dynamically enable/disable log to avoid bulk printing.
  - Execute as unit tests or functional test via JAR.
- Listeners
  - Listeners are supported for future plug-in or application development.

CHAPTER 3

Contribution Guidelines

Artos team is committed to make test development and deployment easy and welcome you to participate in the journey by establishing yourself as a contributor. We encourage you to help improve Artos and docs by providing feedback, suggestions, or issues regarding feature or a product. Below are several ways for you to contribute to Artos' documentation, including editing, requesting change, or raising issues.

Basic Requests for docs change due to bugs, e.g. missed item, incorrect item, or confusing content Typos or grammatical errors

Advanced Reorganizing or rewriting docs content New docs about Artos features

CHAPTER 4

System Setup

4.1 System requirements

- Platform
  - Windows, Linux, MAC or any platform which can run **Java 8** or above.
- JDK
  - Artos can be integrated with any Java project compiled with **JDK 8U45** or higher.

4.2 Add Artos Jar as a dependency

- Non-Maven Projects
  - Download latest Artos jar from the location - Artos_Maven_Repository.
  - Add jar to project build path.
- Maven Projects
  - Copy latest jar dependency XML block from the location - Artos_Maven_Repository.
  - Add dependency to project pom.xml file

```
<!-- Example dependency block -->
<dependency>
    <groupId>com.theartos</groupId>
    <artifactId>artos</artifactId>
    <version>x.x.xx</version>
</dependency>
```

4.3 Eclipse IDE

4.3.1 Install ANSI plug-in for Linux OS

- Go to Eclipse SDK => Help => Eclipse Marketplace.

- Find “ANSI escape in console” plug-in.
- Install the plug-in.
- Restart Eclipse SDK.

4.3.2 Configure test templates:

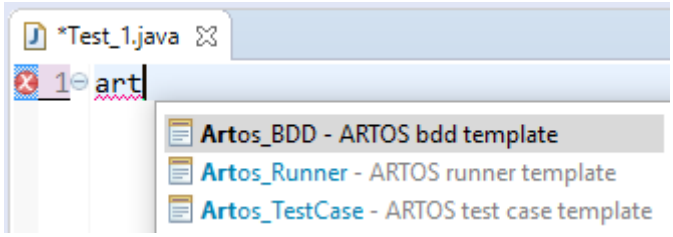
The use of a Java template avoids typing error and increase test development speed. Once imported, templates can be edited to suit business needs.

Import default templates:

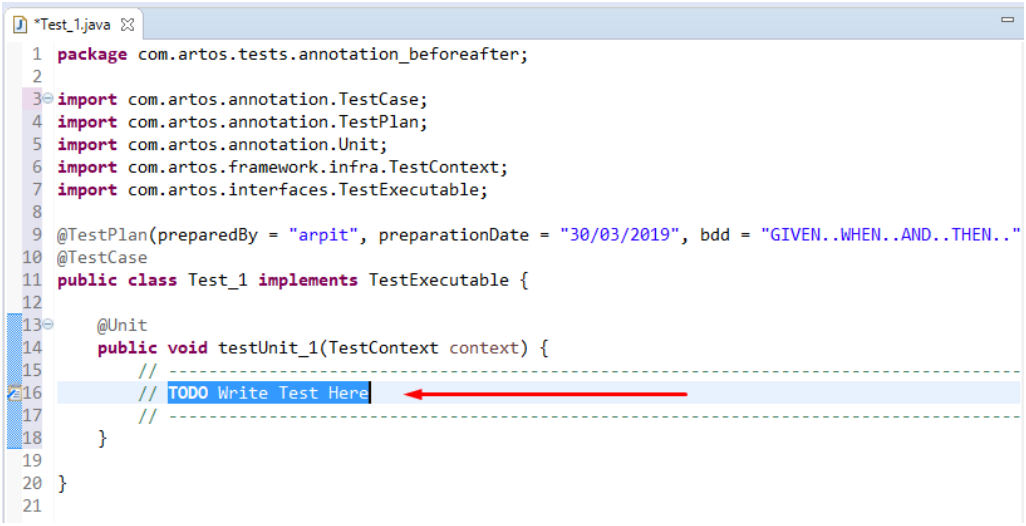
- Download **template.xml** file from location : Artos_Eclipse_Template .
- In Eclipse IDE, browse to Window => Preferences => Java => Editor => Templates.
- Click on **Import** button.
- Import downloaded **template.xml** file.
- Following templates will be added
  - Artos_Runner
  - Artos_TestCase
  - Artos_BDD

4.3.3 Use template:

- Create a new Java class.
- Select and delete all the content of the class.
- Type **art** and press **CTRL + SPACE**.
- Template suggestion (IntelliSense) list will appear as shown below.



- Select desired template.
- Skeleton code will be added to the class.
- Write BDD test plan inside `bdd` attribute under `@TestPlan` annotation.
- Start writing test logic between two green lines.

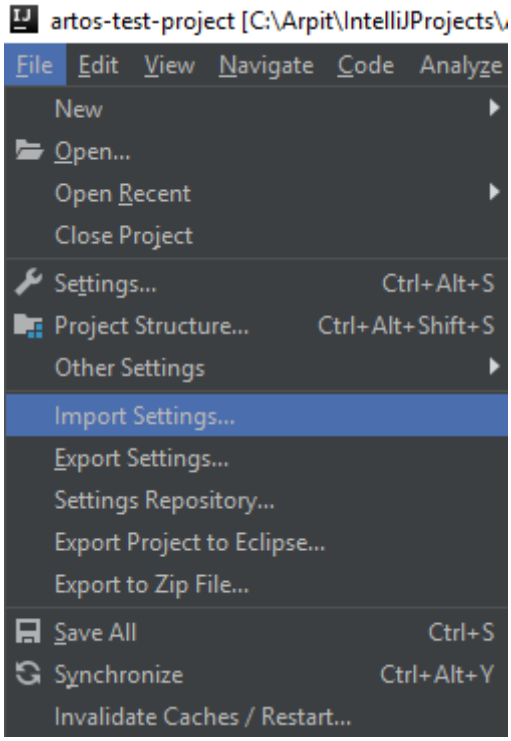


4.4 IntelliJ IDE

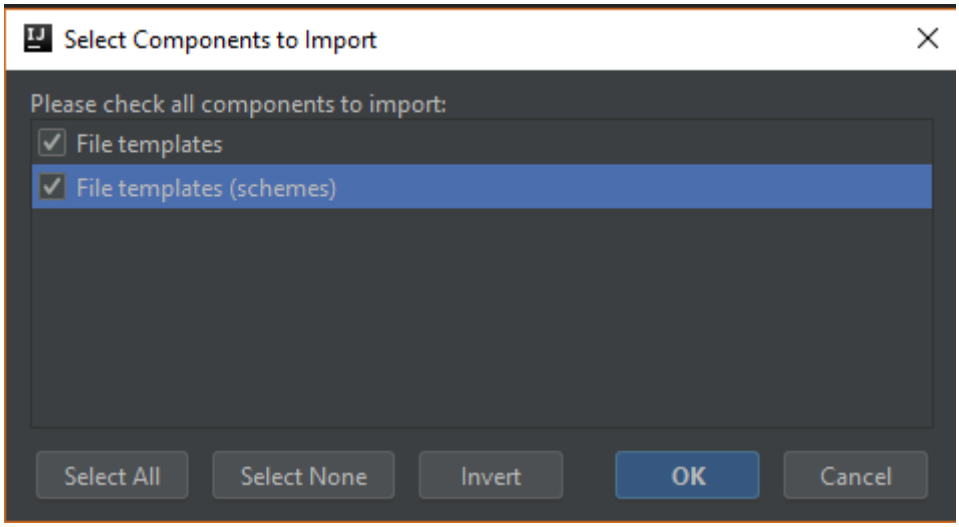
The use of a Java template avoids typing error and increase test development speed. Once imported, templates can be edited to suit business needs.

4.4.1 Configure test templates:

- Download **IntelliJ_template.zip** file from location : Artos_IntelliJ_Template .
- Browse to File => Import Settings.
- Browse and import downloaded **IntelliJ_template.zip** file.



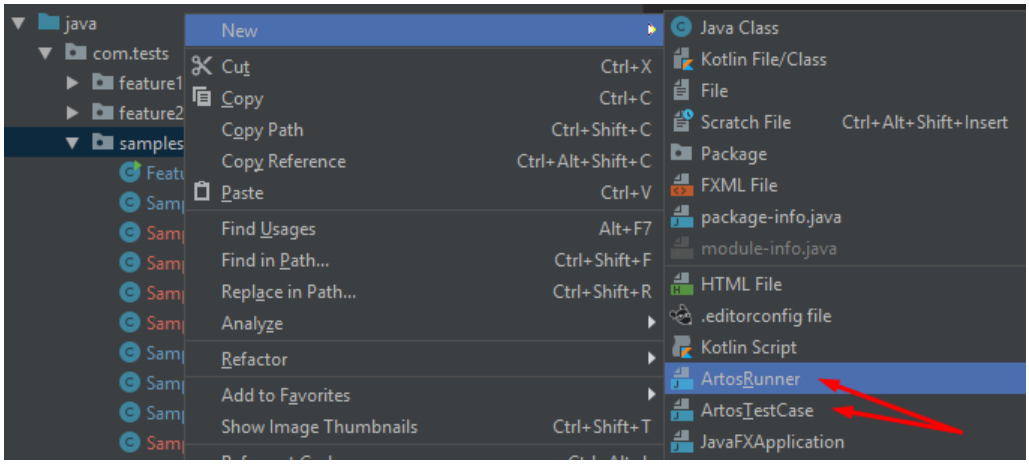
- Select both of the checkboxes.



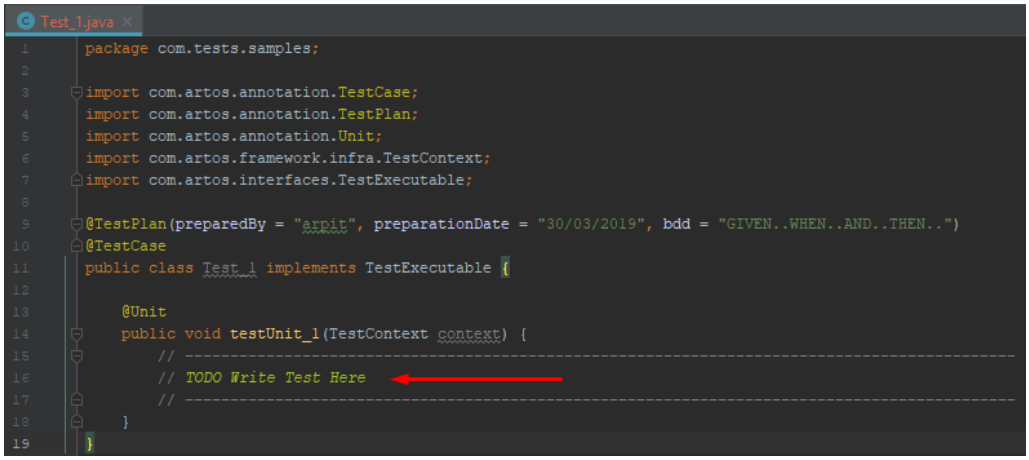
- Following templates will be added
  - ArtosRunner
  - ArtosTestCase

4.4.2 Use template:

- Right-click on the package.
- Select new and choose the desired template.



- Provide a class name.
- Skeleton code will be added to the class.
- Write BDD test plan inside `bdd` attribute under `@TestPlan` annotation.
- Start writing test logic between two green lines.


Test_1.java
package com.tests.samples;
import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.annotation.Unit;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;
@TestPlan(preparedBy = "arpit", preparationDate = "30/03/2019", bdd = "GIVEN..WHEN..AND..THEN..")
@TestCase
public class Test_1 implements TestExecutable {
@Unit
public void testUnit_1(TestContext context) {
// TODO Write Test Here
}
}

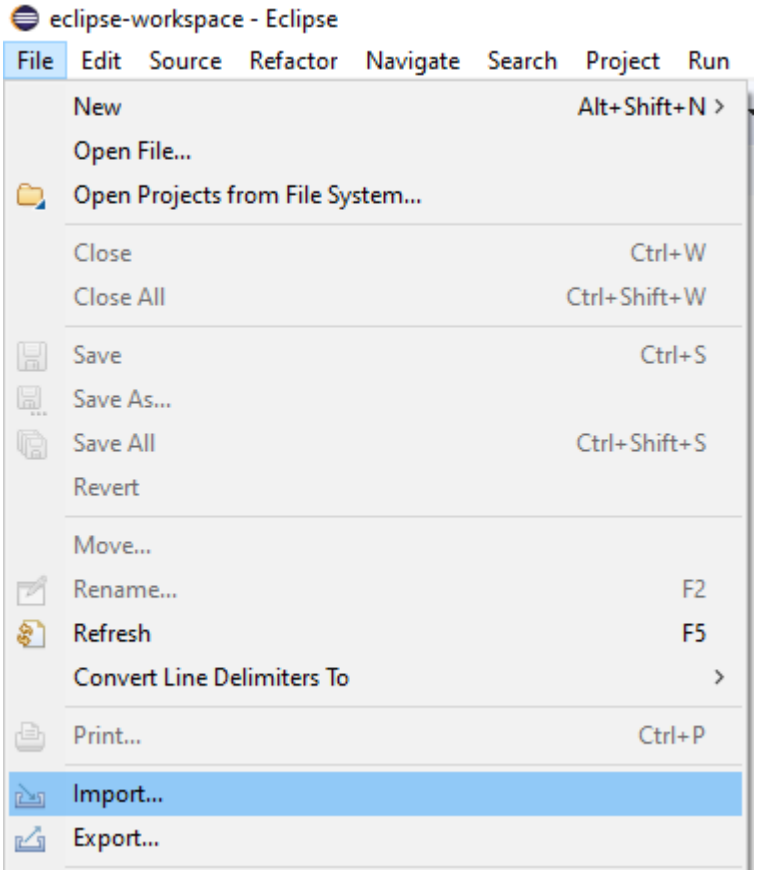
CHAPTER 5

Get Started with Sample Project

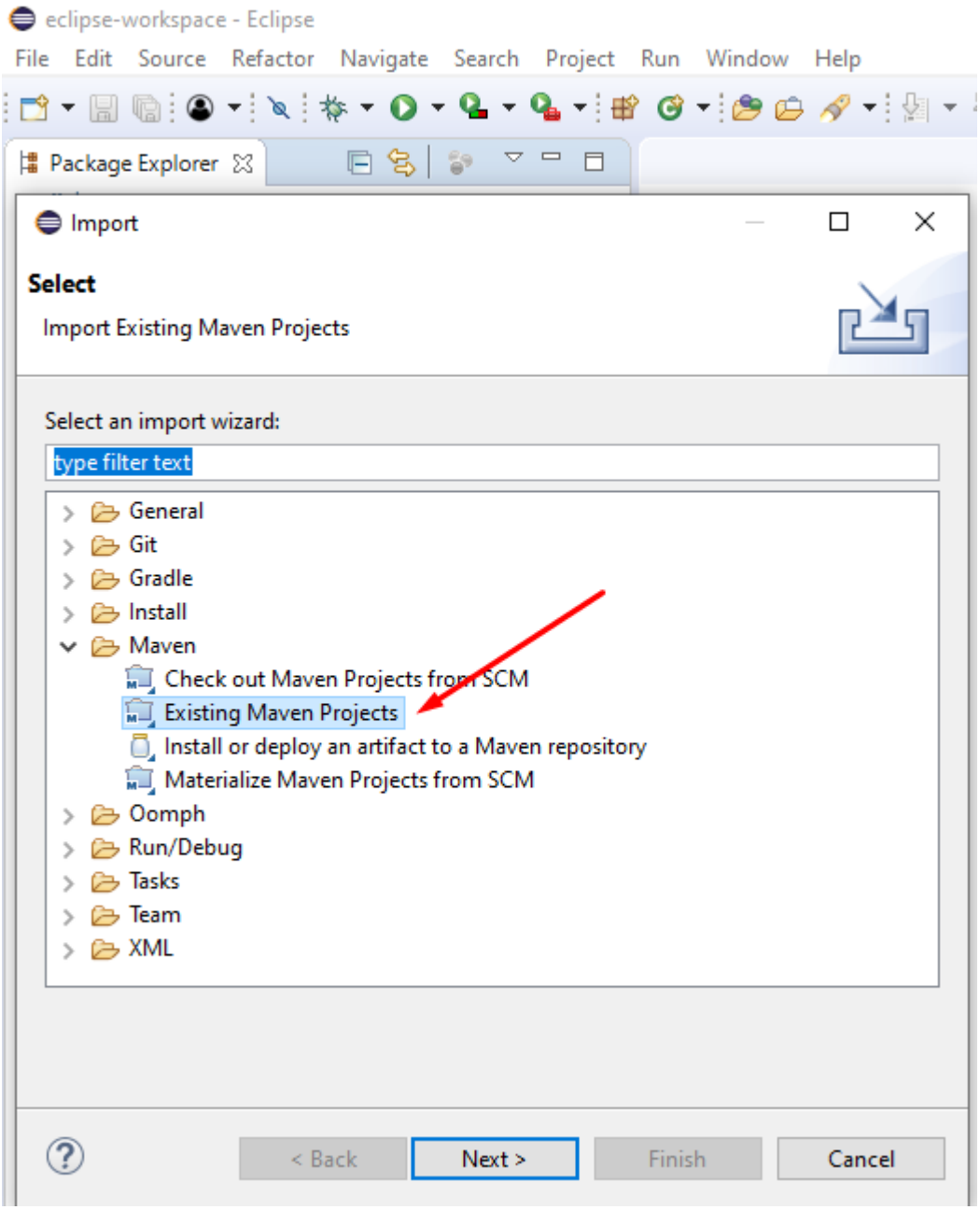
Note

Below guide is written using Eclipse IDE.

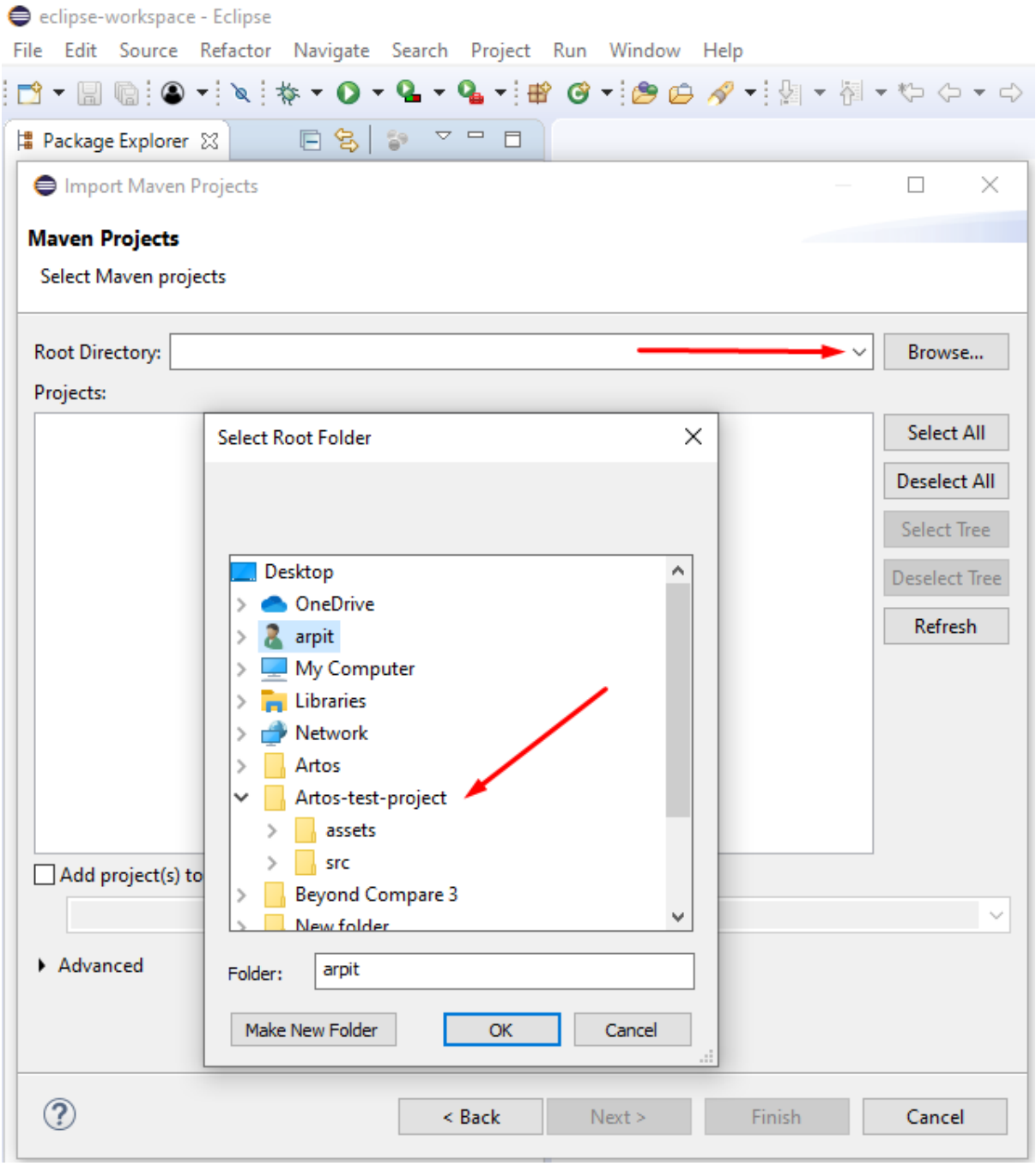
- Download sample project from Artos_Sample_Project.
- Unzip the project to a local directory.
- Open Eclipse IDE.
- Click on **File => Import..**



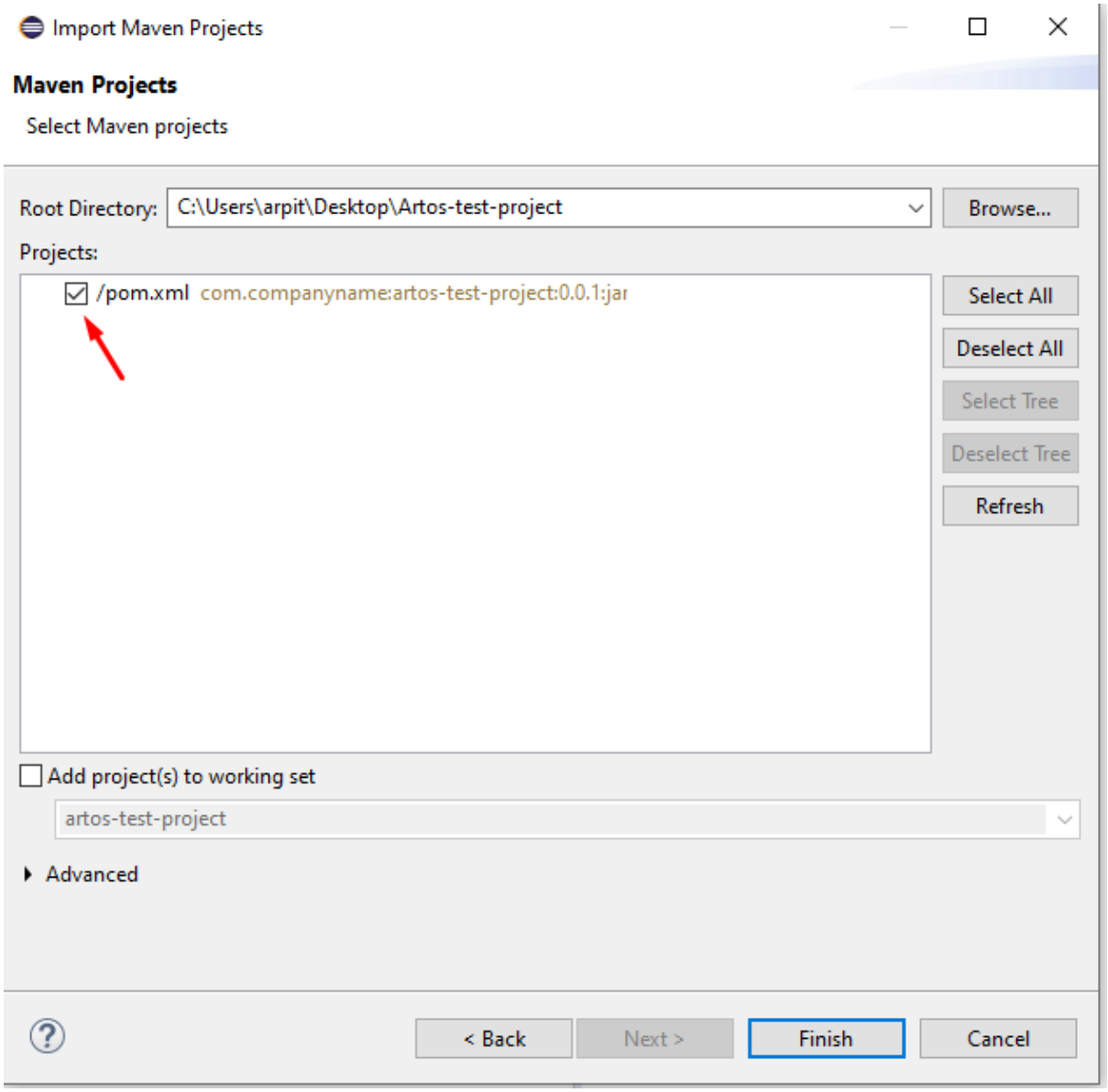
- Select **Existing Maven Projects** and click on **Next** button.



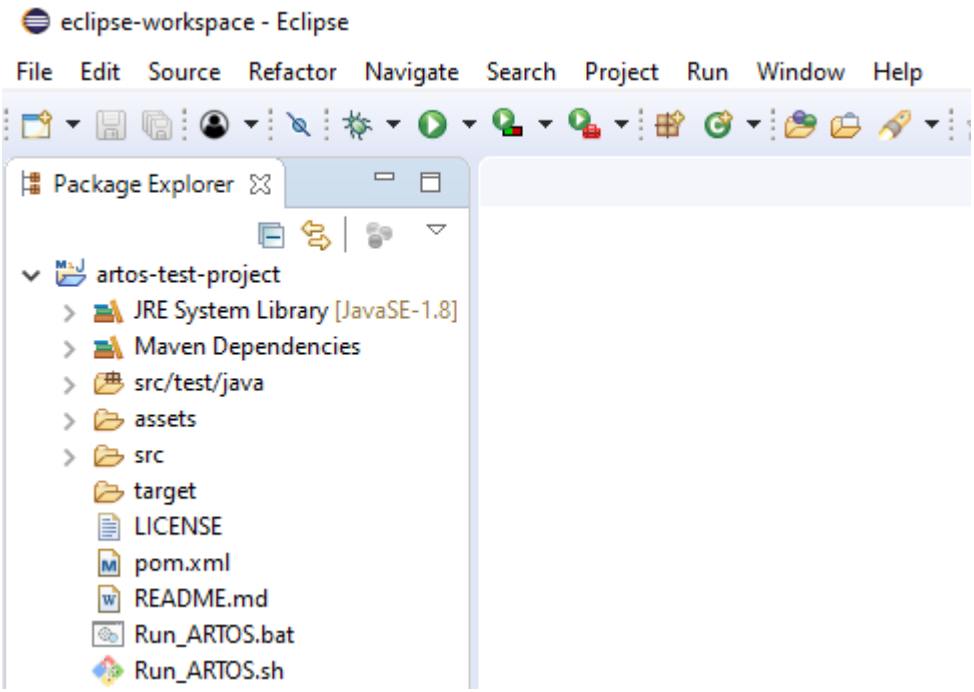
- Browse to Artos-test-project directory, select the directory and click **OK** button.



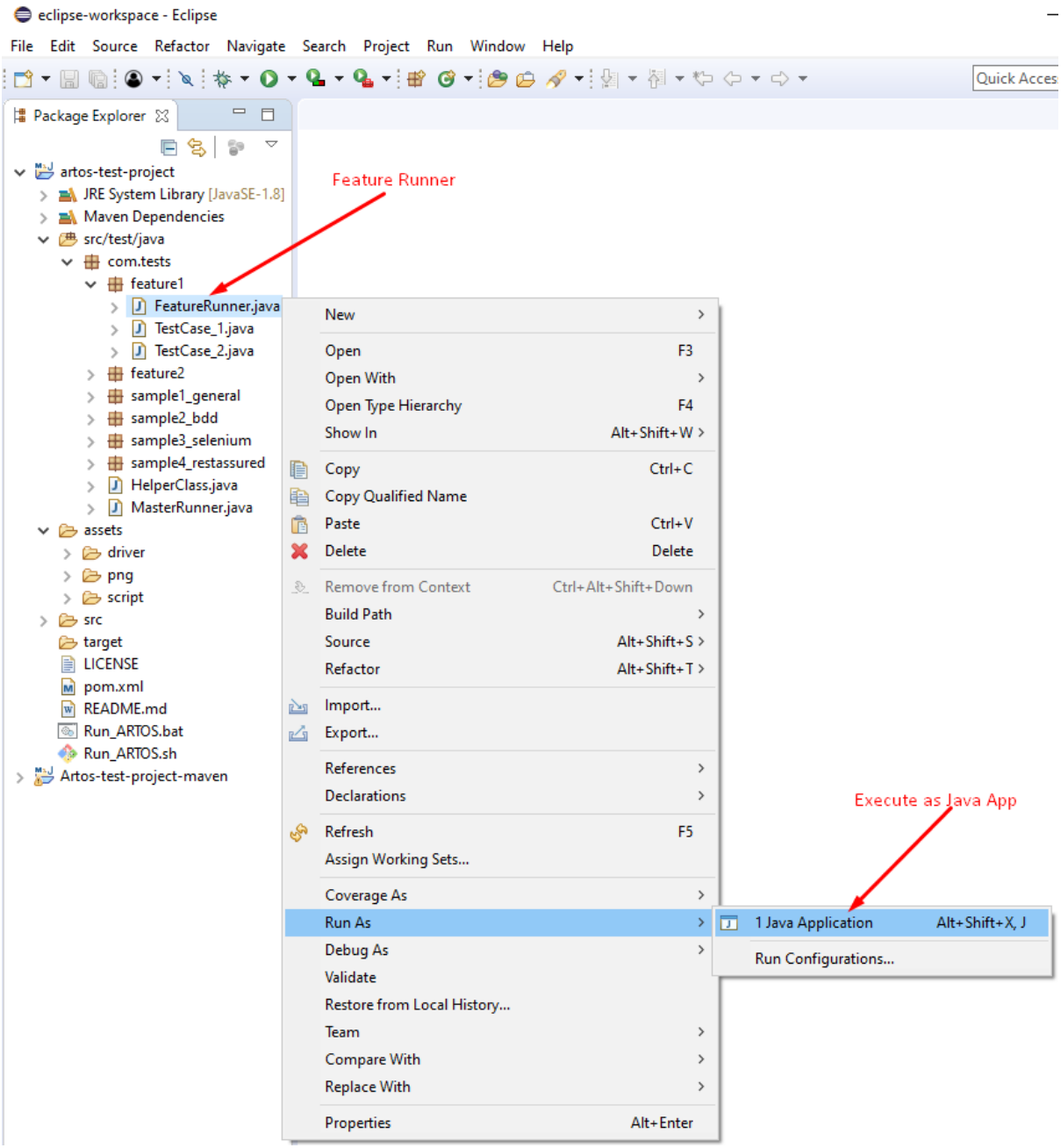
- Ensure pom.xml tick box is checked as shown below. Click on **Finish** button.



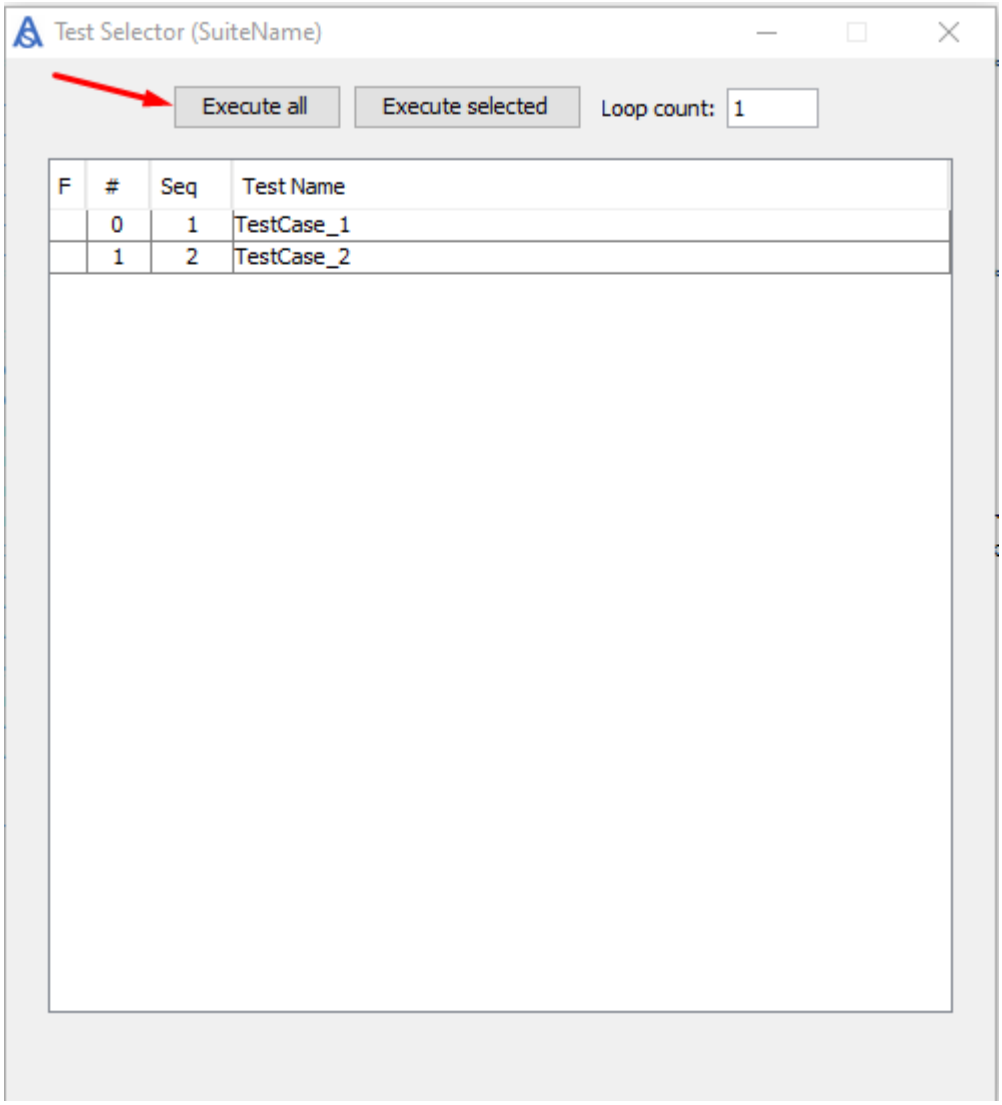
- Project will appear as shown below



- Select Feature Runner as shown below and execute as the Java application.



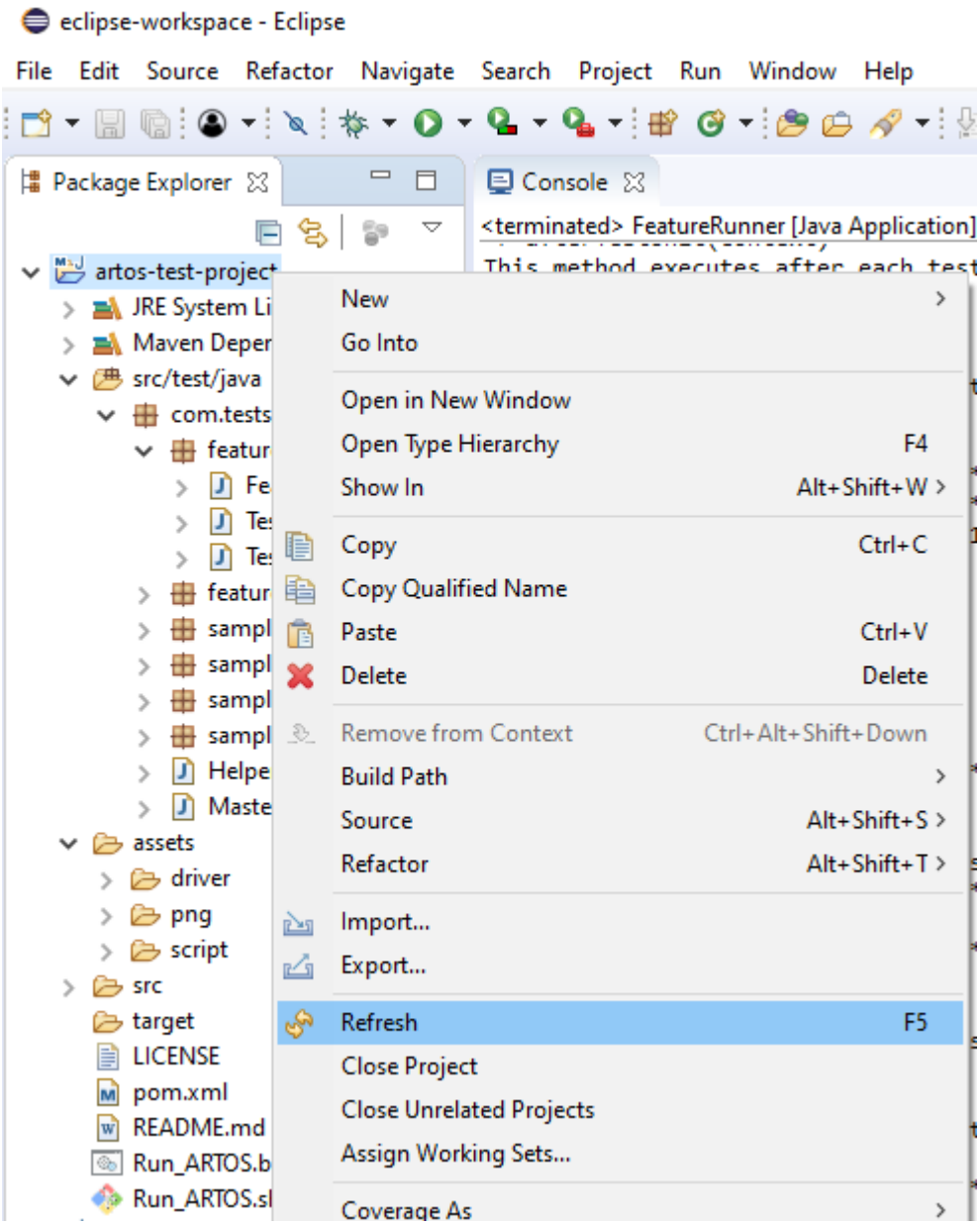
- GUI Selector will appear on the screen. Click on **Execute All**.



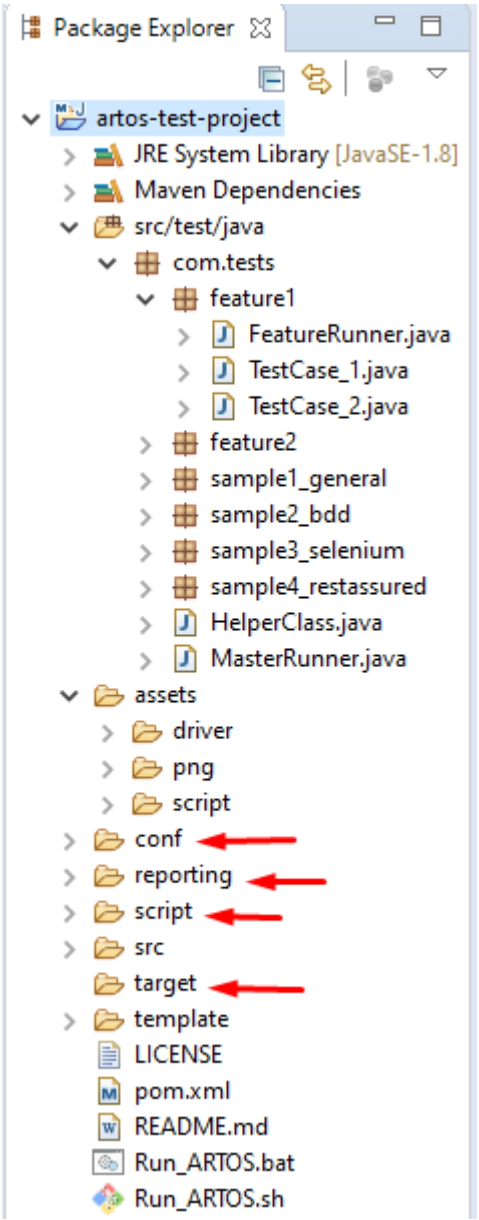
- Logs and summary will be printed on a console.

Congratulation. You have successfully run your very first Artos test suite.

- Select Artos-Test-Project, use mouse right-click and select refresh.



- New directories will appear. Those directories contain logs, reports, Artos' configurations, auto-generated test script, and Artos templates to speed up test case writing.



Recommended

- Explore Artos' directories to gain more understanding about Artos.
- Execute and explore other Runners and example codes that are designed to help user get started with Artos.
- Ask us questions on Stack Overflow with [ARTOS] tag or email Artos team for any support queries.

CHAPTER 6

Implement Project

ARTOS test project consists of two components:

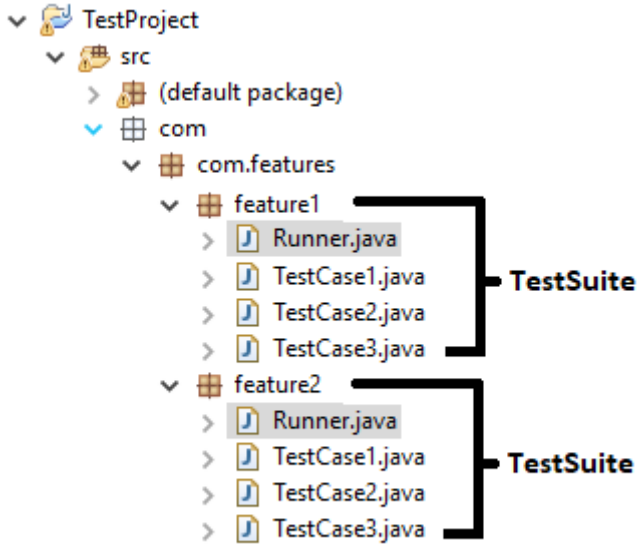
- Test Runner
- Test Case(s)

The project can be configured in many different ways as per the business requirement.

6.1 Recommended Project Structures

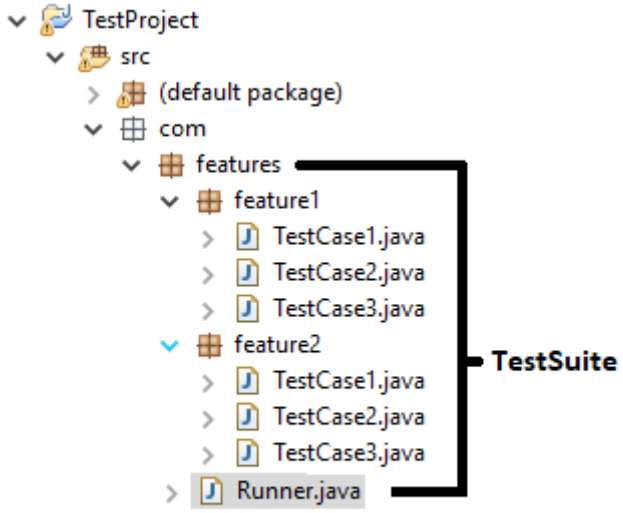
6.1.1 Feature Structure

- Packages and sub-packages are organized based on features.
- Each package has its own Runner class thus each package acts as a test suite.



6.1.2 Super Structure

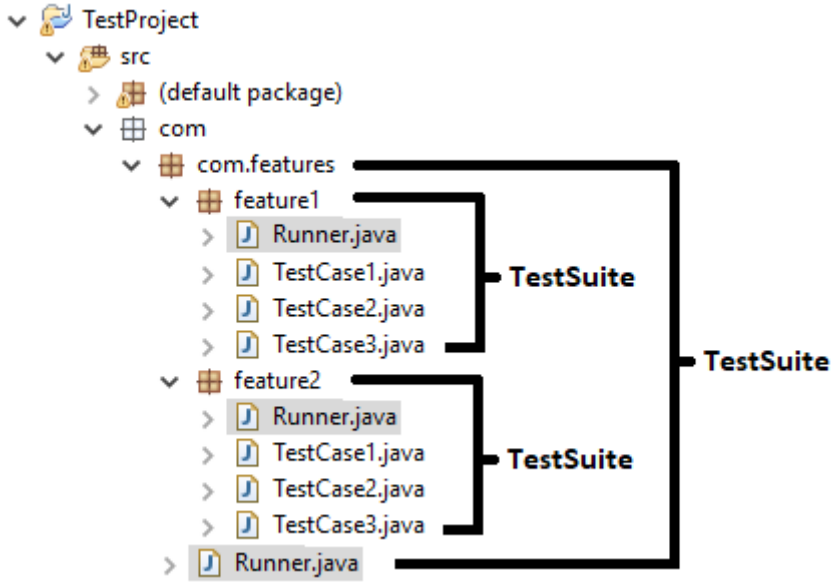
- Packages and sub-packages are organized based on features.
- Project contains a single Runner. All test cases are within Runner's scan scope thus entire project acts as a single test suite.
- This can also be achieved by having Runner at project root location.



6.1.3 Tree Structure (Feature Tree)

This structure is a mixture of the above structures.

- Packages and sub-packages are organized based on features or a test group.
- Project contains Runner class at parent/root position as well as inside each feature packages.
- The test suite executes limited or all test cases depending on the executed Runner.



CHAPTER 7

Implement Runner

A runner is a Java class which meets the following requirements:

- Class is `public` and implements `main()` method.
- The `main()` method invokes Artos' runner object as shown in below example.

Steps

- Create a Java class under required package structure
- Implement `main()` method and Runner code as shown in the example below.

Listing 7.1: Example: Test Runner code

```
1  package com.tests;
2
3  import com.artos.framework.infra.Runner;
4
5  public class ArtosMain {
6          public static void main(String[] args) throws Exception {
7                  Runner runner = new Runner(ArtosMain.class);
8                  runner.run(args);
9          }
10 }
```

CHAPTER 8

Implement TestCase and TestUnits

The test case is the Java class which meets the following requirements:

- Class is `public` and annotated with `@TestCase` annotation.
- Class implements `TestExecutable` interface.

The test unit is a Java method which meets the following requirements:

- Method is `public` and belongs to a test case.
- Method is annotated with `@Unit` annotation.
- Method signature is `public void methodName(TestContext context).`

Recommended

A test plan can be added to a test case or a test unit using `@TestPlan` annotation.

Steps

- Create new Java class inside created package structure.
- Use the template to generate test case skeleton.
- Add details and test units as required.

Listing 8.1: Example: Test case with multiple test units

```
1  package com.tests;
2
3  import com.artos.annotation.TestCase;
4  import com.artos.annotation.TestPlan;
5  import com.artos.annotation.Unit;
6  import com.artos.framework.infra.TestContext;
7  import com.artos.interfaces.TestExecutable;
8
9  @TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "given␣
   ↪project has no errors then hello world will be printed")
10 @TestCase()
11 public class TestCase_1 implements TestExecutable {
12
```

(continues on next page)

(continued from previous page)

```
13      @Unit()
14      public void unit_test1(TestContext context) throws Exception {
15          // -------------------------------------------------------
16          // Print on console
17          System.out.println("Hello World 1");
18          // Print inside a log file
19          context.getLogger().debug("Hello World 1");
20          // -------------------------------------------------------
21      }
22
23      @Unit()
24      public void unit_test2(TestContext context) throws Exception {
25          // -------------------------------------------------------
26          // Print on console
27          System.out.println("Hello World 2");
28          // Print inside a log file
29          context.getLogger().debug(doSomething());
30          // -------------------------------------------------------
31      }
32
33      // This method is not a test unit
34      public String doSomething() {
35          return "Hello World 2";
36      }
37  }
```

CHAPTER 9

Test Sequence

Artos is designed to support unit, functional and end-to-end automated/manual testing thus test sequence is not a mandatory requirement. If test cases are independent of each other and test execution order is not important then the test sequence assignment can be ignored.

9.1 Sequence benefits

- Brings consistency to test execution so the bug can easily be reproduced.
- Brings consistency into log parsing if that is a requirement.
- If test cases or test units are dependent on each other then sequence can be used to describe test dependency or execution order.

9.2 Ordering process

- Within Runner's scan scope => packages are sorted by name.
- Within each package => test cases are sorted by given sequence number.
- Within each test case => test units are sorted by given sequence number.

Listing 9.1: Example: Test case and unit sequence

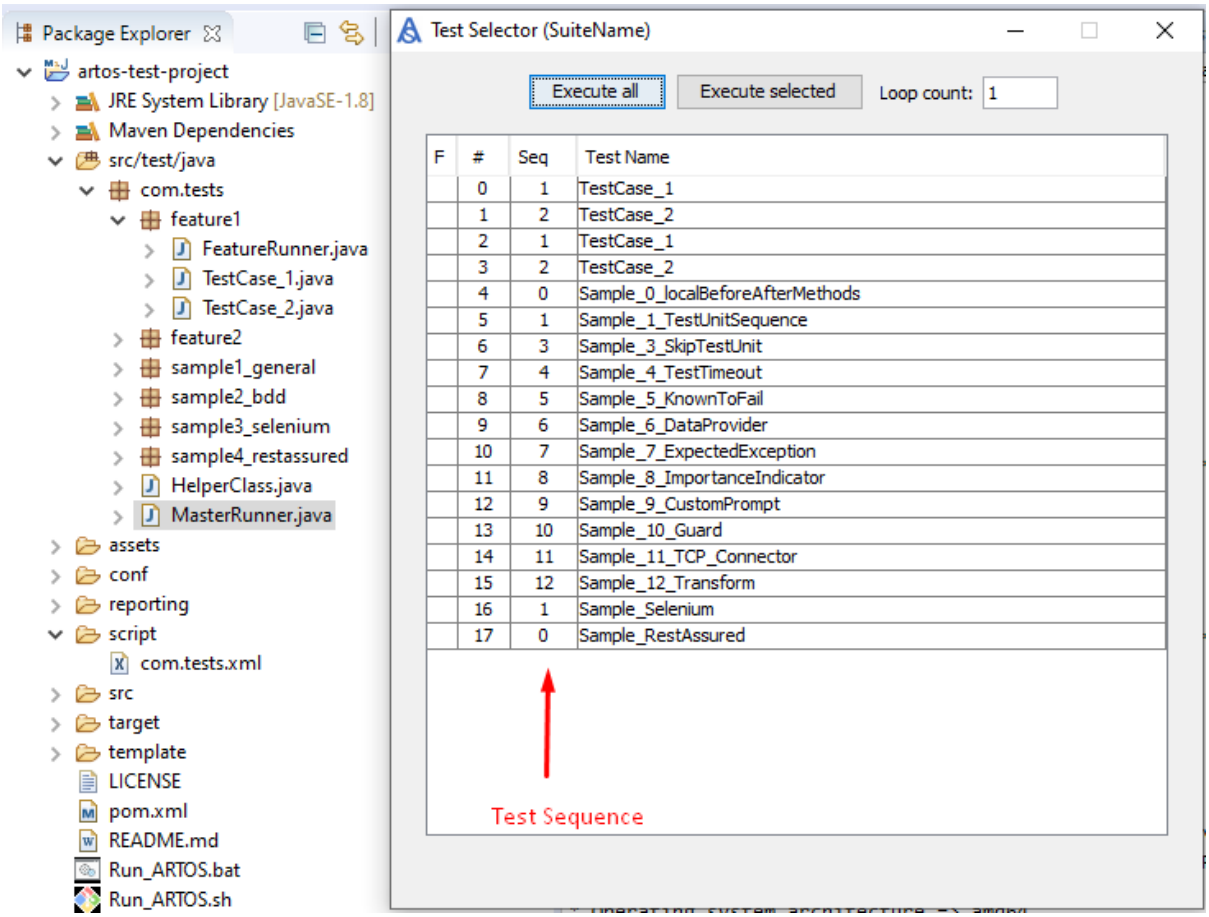
```
1  package com.tests.feature1;
2
3  import com.artos.annotation.TestCase;
4  import com.artos.annotation.TestPlan;
5  import com.artos.annotation.Unit;
6  import com.artos.framework.infra.TestContext;
7  import com.artos.interfaces.TestExecutable;
8
9  @TestPlan(preparedBy = "User1", preparationDate = "19/02/2019", bdd = "GIVEN..WHEN.
   ↪.AND..THEN..")
10 @TestCase(sequence = 1)
11 public class TestCase_1 implements TestExecutable {
12
```

(continues on next page)

(continued from previous page)

```
13      @Unit(sequence = 1)
14      public void testUnit_1(TestContext context) {
15          // ----------------------------------------------------------------------
    ↪----------------------
16          context.getLogger().info("First Unit");
17          // ----------------------------------------------------------------------
    ↪----------------------
18      }
19
20      @Unit(sequence = 2)
21      public void testUnit_2(TestContext context) {
22          // ----------------------------------------------------------------------
    ↪----------------------
23          context.getLogger().info("Second Unit");
24          // ----------------------------------------------------------------------
    ↪----------------------
25      }
26
27      @Unit(sequence = 3)
28      public void testUnit_3(TestContext context) {
29          // ----------------------------------------------------------------------
    ↪----------------------
30          context.getLogger().info("Third Unit");
31          // ----------------------------------------------------------------------
    ↪----------------------
32      }
33
34  }
```

Assigned test case sequence can be reviewed via GUI test selector



9.3 Ignoring sequence assignment

- If sequence numbers are not assigned to some or all test cases/units then those test cases/units execution order can not be guaranteed, all though those test cases/units will retain their positions respective to other test cases/units sequence numbers within execution order.

9.4 Test sequence override via test script

- A test script can override test sequence by specifying fully qualified test class paths within a test script.
- Test units sequence can not be overridden from the test script.

Listing 9.2: Example: Test script overrides test sequence if test cases are specified

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration version="1">
  <suite enable="true" loopcount="1" name="UniqueName">
    <tests>
      <test name="com.tests.feature1.TestCase_1"/>
      <test name="com.tests.feature1.TestCase_2"/>
      <test name="com.tests.feature2.TestCase_1"/>
      <test name="com.tests.feature2.TestCase_2"/>
    </tests>
    <featurefiles>
    </featurefiles>
    <parameters>
      <parameter name="PARAMETER_0">parameterValue_0</parameter>
      <parameter name="PARAMETER_1">parameterValue_1</parameter>
      <parameter name="PARAMETER_2">parameterValue_2</parameter>
    </parameters>
    <testcasegroups>
      <group name="*"/>
    </testcasegroups>
    <testunitgroups>
      <group name="*"/>
    </testunitgroups>
  </suite>
</configuration>
```

Listing 9.3: Example: Test script sequence override is disabled if test cases are not specified

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration version="1">
  <suite enable="true" loopcount="1" name="UniqueName">
    <tests>
    </tests>
    <featurefiles>
    </featurefiles>
    <parameters>
      <parameter name="PARAMETER_0">parameterValue_0</parameter>
      <parameter name="PARAMETER_1">parameterValue_1</parameter>
      <parameter name="PARAMETER_2">parameterValue_2</parameter>
    </parameters>
    <testcasegroups>
      <group name="*"/>
    </testcasegroups>
    <testunitgroups>
```

(continues on next page)

(continued from previous page)

```
      <group name="*"/>
    </testunitgroups>
  </suite>
</configuration>
```

9.5 Test sequence override via Runner

- A Runner can override test sequence by specifying test sequence within the Runner.
- Test unit sequence can not be overridden from the Runner.

Listing 9.4: Example: Runner overrides test case sequence if test list is passed as shown below.

```
package com.tests;

import java.util.ArrayList;
import com.artos.framework.infra.Runner;
import com.artos.interfaces.TestExecutable;

public class MasterRunner {

    public static ArrayList<TestExecutable> getTestList() throws Exception {
        ArrayList<TestExecutable> tests = new ArrayList<TestExecutable>();

        // ------------------------------------------------------------------------
↪---------------------
        // TODO User May Add Test Case Manually as show in sample below
        tests.add(new com.tests.feature1.TestCase_1());
        tests.add(new com.tests.feature1.TestCase_2());
        tests.add(new com.tests.feature2.TestCase_1());
        tests.add(new com.tests.feature2.TestCase_2());
        // ------------------------------------------------------------------------
↪---------------------

        return tests;
    }

    public static void main(String[] args) throws Exception {
        Runner runner = new Runner(MasterRunner.class);
        runner.setTestList(getTestList());
        runner.run(args);
    }
}
```

Listing 9.5: Example: Runner test case sequence overrides is disabled if empty test list or null is passed.

```
package com.tests;

import com.artos.framework.infra.Runner;
import com.artos.interfaces.TestExecutable;

public class MasterRunner {

    public static void main(String[] args) throws Exception {
        Runner runner = new Runner(MasterRunner.class);
        runner.setTestList(null);
```

(continues on next page)

(continued from previous page)

```
11          runner.run(args);
12      }
13  }
```

9.6 Override priority

- If test script is used and override sequence is specified then Runner override sequence is ignored.
- If test script is not used then Runner override sequence is used.
- If test script is not used and Runner override is disabled then sequence specified within `@TestCase` annotation is used to execute test cases.

Important

- Test cases will be ignored if they are not specified in the override script.
- Console warnings will be printed in case override sequence specifies test cases that are outside Runner's scan scope. In given situation test execution will be continued by ignoring out of scope test cases.

CHAPTER 10

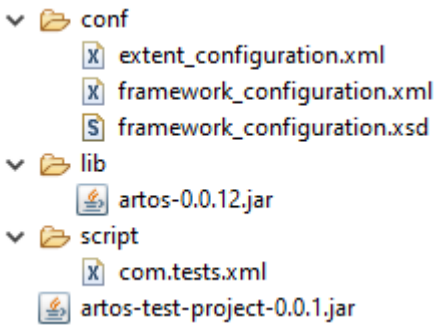
Launch test suite with test-script

Artos test project can be executed in multiple different ways

- Launch project jar using command-line
- Launch Maven project via command-line
- Launch Runner via IDE (Eclipse, IntelliJ etc..)

10.1 Launch project jar (Windows command-line)

- Step 1: Note down Runners fully qualified class-path (Example: `com.tests.feature1.FeatureRunner`)
- Step 2: Build Artos test project jar (Example: `artos-test-project-0.0.1.jar`)
- Step 3: Create a directory called `lib` at the location where `artos-test-project-0.0.1.jar` is situated.
- Step 4: Copy artos jar downloaded from maven repository inside `lib` directory.
- Step 5: Create a directory called `conf` at the location where `artos-test-project-0.0.1.jar` is situated.
- Step 6: Copy artos configuration to `conf` directory.
- Step 7: Create a directory called `script` at the location where `artos-test-project-0.0.1.jar` is situated.
- Step 8: Copy XML based test script inside the `script` directory.



- Step 9: Open command line prompt or PowerShell.
- Step 10: change directory location to the same directory where artos-test-project-0.0.1.jar is present using `cd` command.
- Step 11: Execute below command line argument to execute the Artos test project.

Listing 10.1: Example: Command line to launch test project jar

```
java -cp "artos-test-project-0.0.1.jar;.\lib\artos-0.0.12.jar" com.tests.feature1.
↪FeatureRunner --testscript="com.tests.xml" --profile="dev"
```

Important

- By default `dev` profiled configuration is used from framework_configuration.xml file.
- If a user has separate configurations for `dev` and `production` environment OR `Window` and `Linux` environments then they should pass appropriate profile name via command-line arguments.

10.2 Launch Maven project as unit test framework

Note

Artos sample project can be downloaded from Artos website.

Below example explains how to launch Artos Maven test project in both the environments (Windows or Linux). Artos sample project POM file is pre-configured to launch Artos test project at compile time like a unit test framework:

- `exec-maven-plugin` is used to execute `Run_ARTOS.bat` for Windows and `Run_ARTOS.sh` for Linux.
- `exec-maven-plugin` is configured to execute script at unit level by setting `<phase>test</phase>`.
- `<profiles></profiles>` feature is used to select correct extension of the files based on platform.

Run_ARTOS.bat and **Run_ARTOS.sh** script content is specified below.

Listing 10.2: Run_ARTOS.bat content

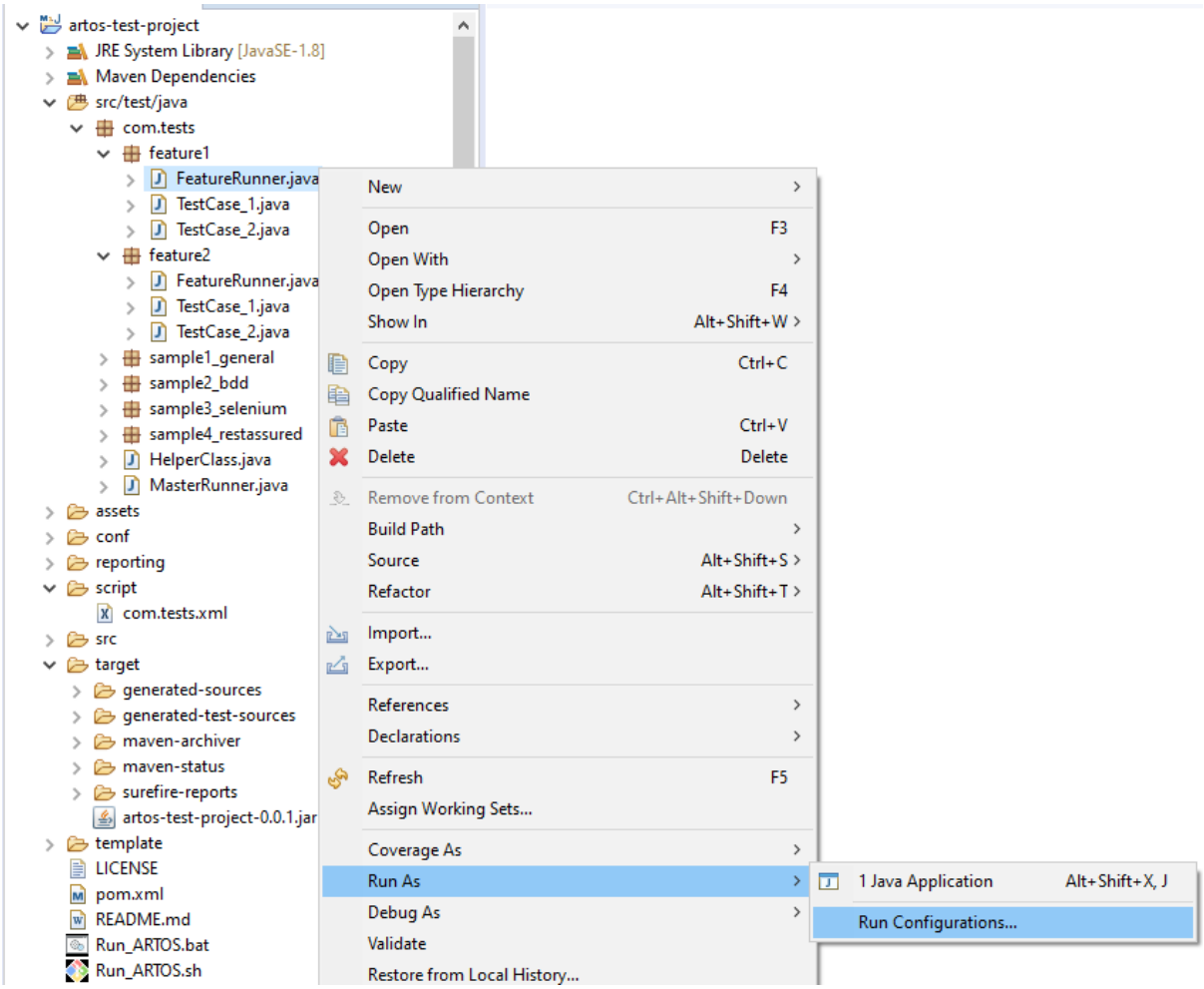
```
@echo off
set mypath=%cd%
@echo %mypath%
mvn exec:java -D"exec.mainClass"="com.tests.FeatureRunner" -Dexec.args="-
↪v -p=dev -t=testscript.xml"
pause
```

Listing 10.3: Run_ARTOS.sh content

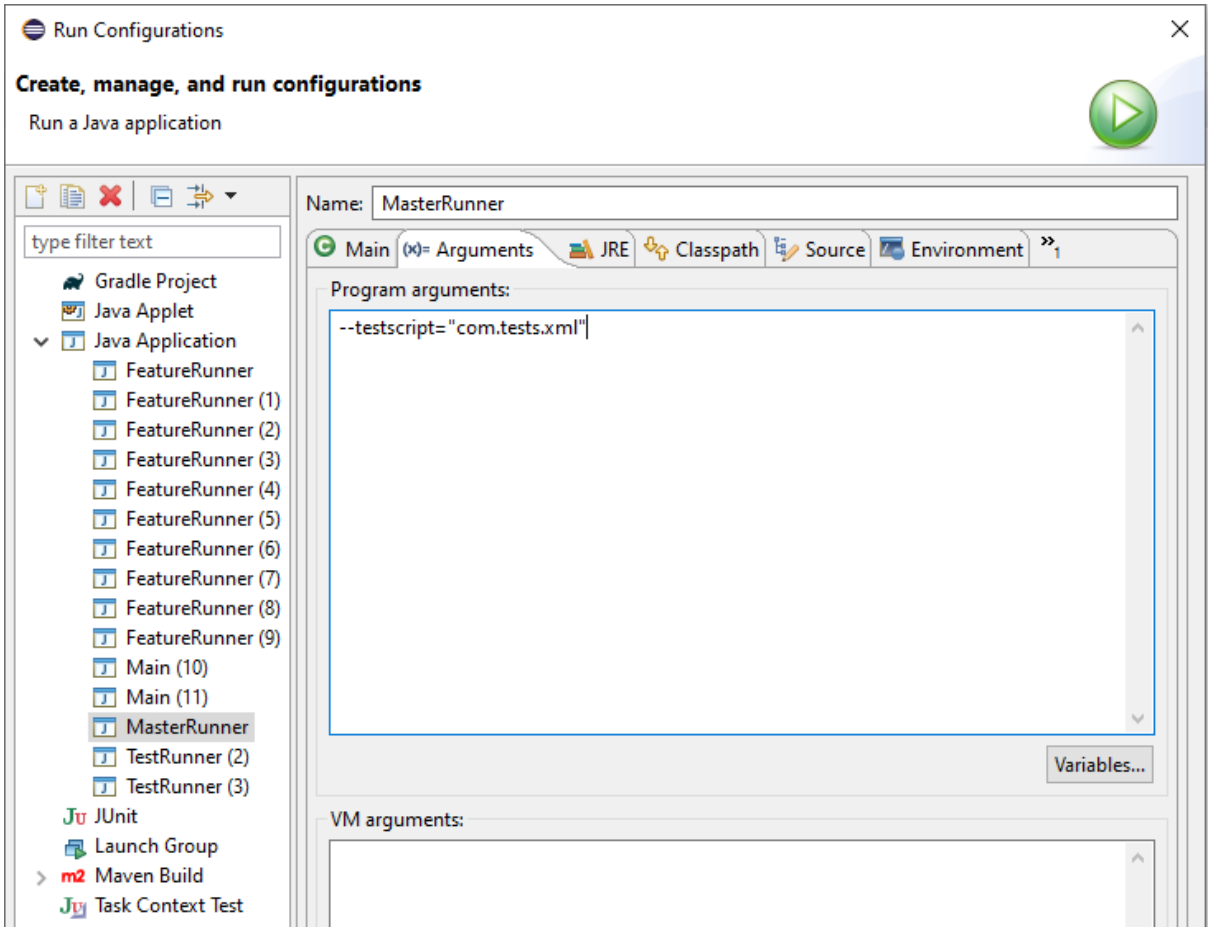
```
#!/bin/sh
#echo off
mvn exec:java -Dexec.mainClass="com.tests.FeatureRunner" -Dexec.args="-v -
↪p=dev -t=testscript.xml"
```

10.3 Runner launch using test script (Eclipse IDE)

- Right-click on Runner. Go to **Run As => Run Configurations...**



- Provide test script argument as shown below and click on run button


Run Configurations
Create, manage, and run configurations
Run a Java application
type filter text
Gradle Project
Java Applet
Java Application
FeatureRunner
FeatureRunner (1)
FeatureRunner (2)
FeatureRunner (3)
FeatureRunner (4)
FeatureRunner (5)
FeatureRunner (6)
FeatureRunner (7)
FeatureRunner (8)
FeatureRunner (9)
Main (10)
Main (11)
MasterRunner
TestRunner (2)
TestRunner (3)
JUnit
Launch Group
Maven Build
Task Context Test
Name: MasterRunner
Main
Arguments
JRE
Classpath
Source
Environment
Program arguments:
--testscript="com.tests.xml"
Variables...
VM arguments:

CHAPTER 11

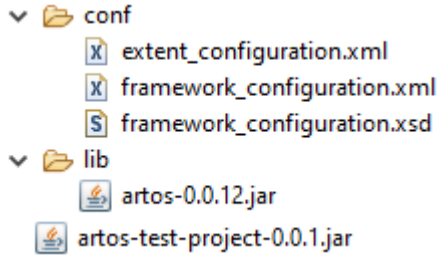
Launch test suite without test-script

Artos test project can be executed in multiple different ways

- Launch project jar using command-line
- Launch Maven project via command-line
- Launch Runner via IDE (Eclipse, IntelliJ etc..)

11.1 Launch project jar (Windows command-line)

- Step 1: Note down Runners fully qualified class-path (Example: `com.tests.feature1.FeatureRunner`)
- Step 2: Build Artos test project jar (Example: `artos-test-project-0.0.1.jar`)
- Step 3: Create a directory called `lib` at the location where `artos-test-project-0.0.1.jar` is situated.
- Step 4: Copy artos jar downloaded from maven repository inside `lib` directory.
- Step 5: Create a directory called `conf` at the location where `artos-test-project-0.0.1.jar` is situated.
- Step 6: Copy artos configuration to `conf` directory.
- Step 7: Create a directory called `script` at the location where `artos-test-project-0.0.1.jar` is situated.



- Step 9: Open command line prompt or PowerShell.

- Step 10: change directory location to the same directory where artos-test-project-0.0.1.jar is present using `cd` command.
- Step 11: Execute below command line argument to execute the Artos test project.

Listing 11.1: Example: Command line to launch test project jar

```
java -cp "artos-test-project-0.0.1.jar;.\lib\artos-0.0.12.jar" com.tests.feature1.
↪FeatureRunner --profile="dev"
```

Important

- By default `dev` profiled configuration is used from framework_configuration.xml file.
- If a user has separate configurations for `dev` and `production` environment OR `Window` and `Linux` environments then they should pass appropriate profile name via command-line arguments.

11.2 Launch Maven project as unit test framework

Note

Artos sample project can be downloaded from Artos website.

Below example explains how to launch Artos Maven test project in both the environments (Windows or Linux). Artos sample project POM file is pre-configured to launch Artos test project at compile time like a unit test framework:

- `exec-maven-plugin` is used to execute `Run_ARTOS.bat` for Windows and `Run_ARTOS.sh` for Linux.
- `exec-maven-plugin` is configured to execute script at unit level by setting `<phase>test</phase>`.
- `<profiles></profiles>` feature is used to select correct extension of the files based on platform.

Run_ARTOS.bat and **Run_ARTOS.sh** script content is specified below.

Listing 11.2: Run_ARTOS.bat content

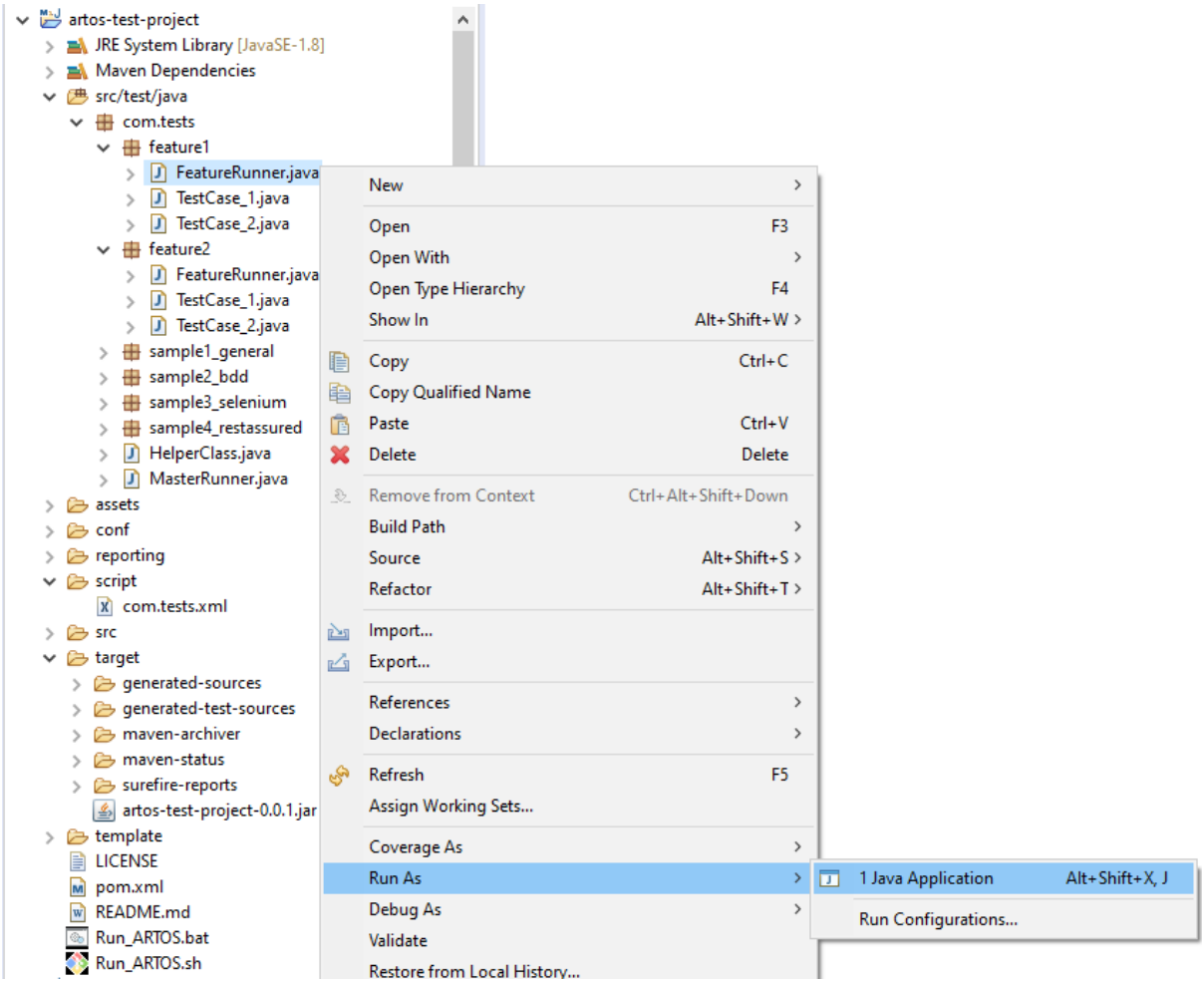
```
@echo off
set mypath=%cd%
@echo %mypath%
mvn exec:java -D"exec.mainClass"="com.tests.FeatureRunner" -Dexec.args="-
↪v -p=dev"
pause
```

Listing 11.3: Run_ARTOS.sh content

```
#!/bin/sh
#echo off
mvn exec:java -Dexec.mainClass="com.tests.FeatureRunner" -Dexec.args="-v -
↪p=dev"
```

11.3 Runner launch using test script (Eclipse IDE)

- Right-click on Runner. Go to **Run As => Run Configurations...**


artos-test-project
JRE System Library [JavaSE-1.8]
Maven Dependencies
src/test/java
com.tests
feature1
FeatureRunner.java
TestCase_1.java
TestCase_2.java
feature2
FeatureRunner.java
TestCase_1.java
TestCase_2.java
sample1_general
sample2_bdd
sample3_selenium
sample4_restassured
HelperClass.java
MasterRunner.java
assets
conf
reporting
script
com.tests.xml
src
target
generated-sources
generated-test-sources
maven-archiver
maven-status
surefire-reports
artos-test-project-0.0.1.jar
template
LICENSE
pom.xml
README.md
Run_ARTOS.bat
Run_ARTOS.sh
New
Open F3
Open With
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Validate
Restore from Local History...
1 Java Application Alt+Shift+X, J
Run Configurations...

CHAPTER 12

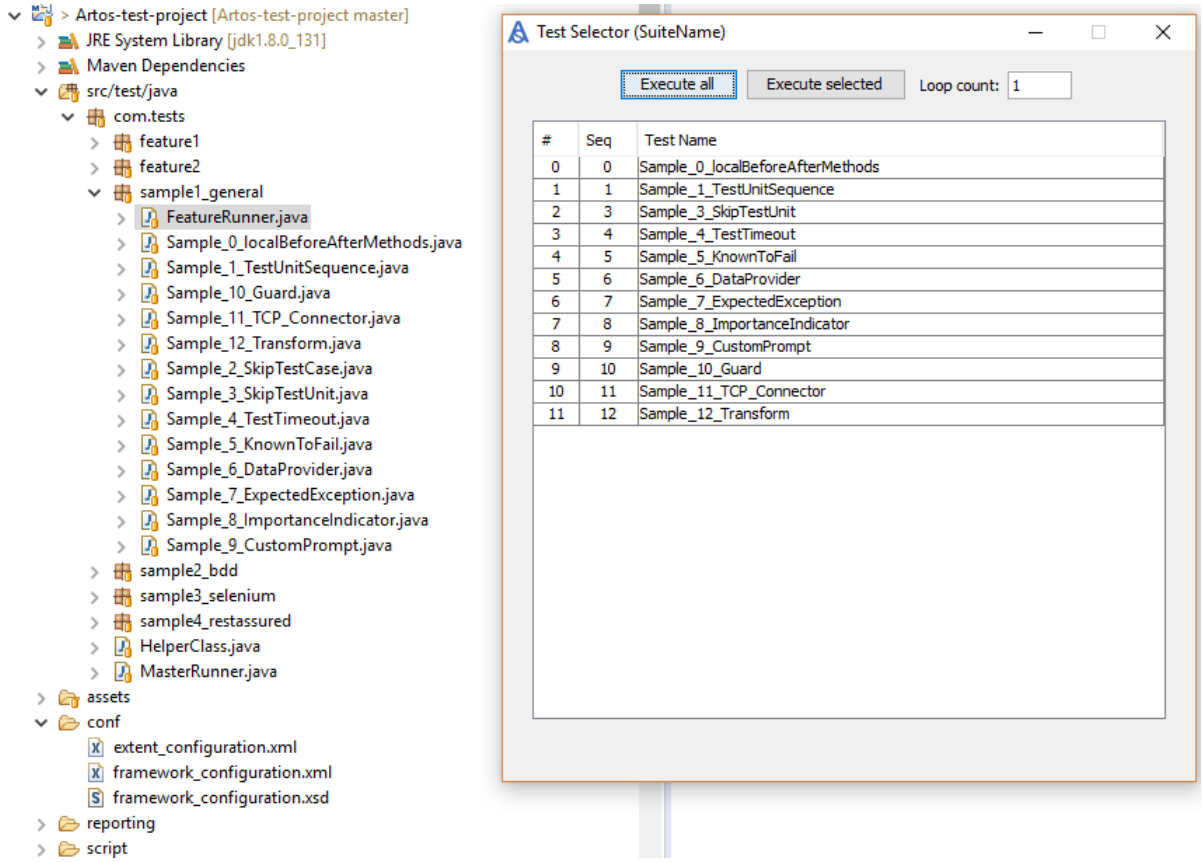
Execute Test Cases

Upon a project launch Artos performs scan to find test cases within a runner's scan scope. Test cases are processed in the following order.

- Test cases with **skip=true** attribute are dropped from the list.
- Test cases are dropped from the list, in case if they are not listed in user provided test list (list is provided via test script or test runner)
- Test cases are dropped if they do not belong to desired test case groups. (Groups are provided via test script or test runner)
- Remaining test cases are sorted in the sequence of provided test list (list is provided via test script or test runner). If test list is not provided then test cases are sorted using sequence declared using **sequence** attribute of @TestCase.

All sorted test cases are displayed to user using GUI test selector. Using GUI test selector user may do following

- User may choose to execute selected or all test cases
- User may change loop count
- User may cancel the test suite execution.



Once execution is started GUI selector will be hidden.

Important:

- If only one test case is present then GUI test selector will not appear and test execution will be initiates automatically.
- User can disable GUI test selector by changing **framework_configuration.xml** settings to **<property name="enableGUITestSelector">false</property>**.

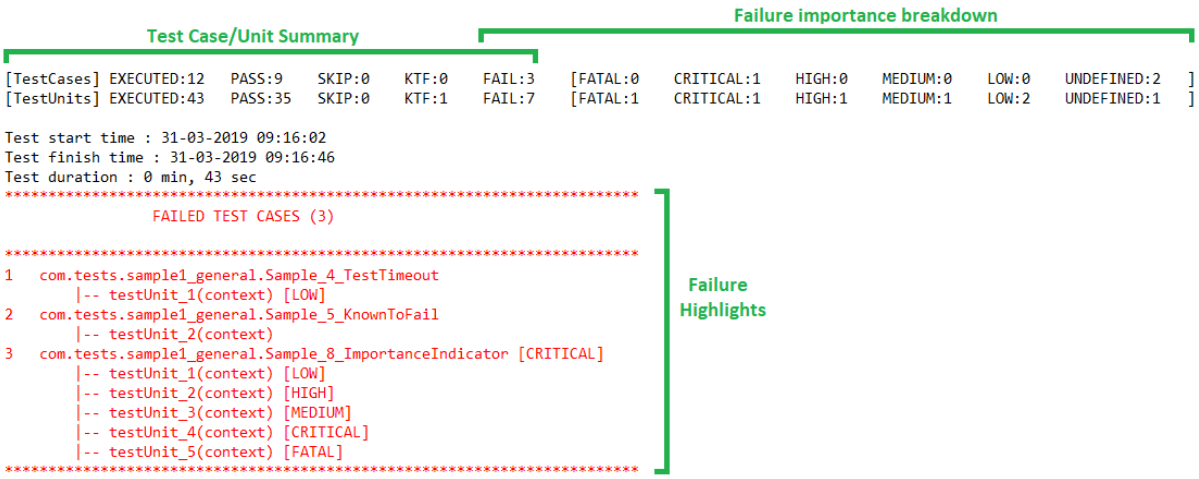
CHAPTER 13

Test Result

Test result can be seen in:

- Console log
- Test suite logs
- Test summary report

13.1 Console log example



CHAPTER 14

System Setup

14.1 System requirements

- Platform
  - Windows, Linux, MAC or any platform which can run **Java 8** or above.
- JDK
  - Artos can be integrated with any Java project compiled with **JDK 8U45** or higher.

14.2 Add Artos Jar as a dependency

- Non-Maven Projects
  - Download latest Artos jar from the location - Artos_Maven_Repository.
  - Add jar to project build path.
- Maven Projects
  - Copy latest jar dependency XML block from the location - Artos_Maven_Repository.
  - Add dependency to project pom.xml file

```
<!-- Example dependency block -->
<dependency>
    <groupId>com.theartos</groupId>
    <artifactId>artos</artifactId>
    <version>x.x.xx</version>
</dependency>
```

14.3 Eclipse IDE

14.3.1 Install ANSI plug-in for Linux OS

- Go to Eclipse SDK => Help => Eclipse Marketplace.

- Find “ANSI escape in console” plug-in.
- Install the plug-in.
- Restart Eclipse SDK.

14.3.2 Configure test templates:

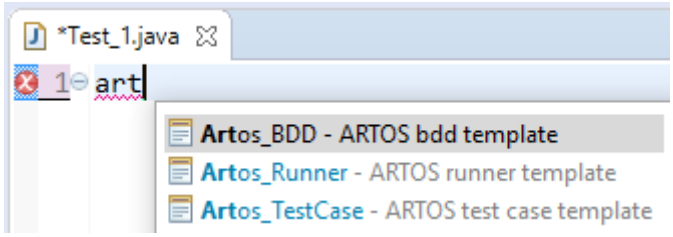
The use of a Java template avoids typing error and increase test development speed. Once imported, templates can be edited to suit business needs.

Import default templates:

- Download **template.xml** file from location : Artos_Eclipse_Template .
- In Eclipse IDE, browse to Window => Preferences => Java => Editor => Templates.
- Click on **Import** button.
- Import downloaded **template.xml** file.
- Following templates will be added
  - Artos_Runner
  - Artos_TestCase
  - Artos_BDD

14.3.3 Use template:

- Create a new Java class.
- Select and delete all the content of the class.
- Type **art** and press **CTRL + SPACE**.
- Template suggestion (IntelliSense) list will appear as shown below.



- Select desired template.
- Skeleton code will be added to the class.
- Start writing test logic between two green lines.

Feature1.java

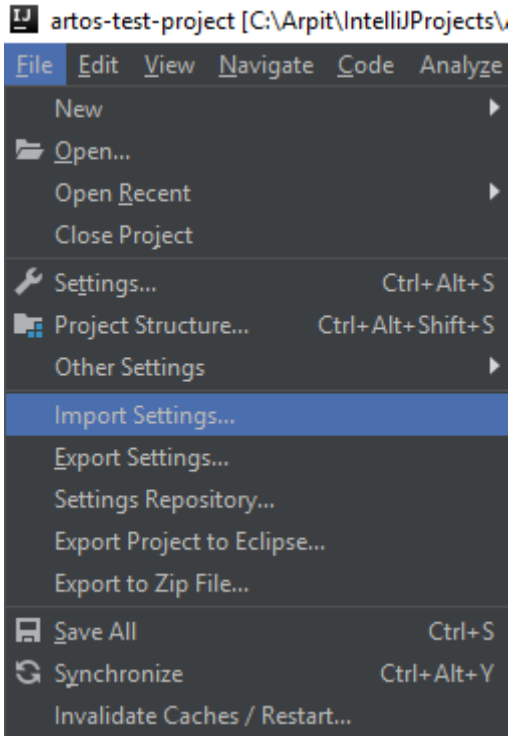
```
1  package com.theartos.stepdefinition;
2
3  import com.artos.annotation.StepDefinition;
6
7  public class Feature1 implements TestExecutable {
8
9      @StepDefinition("User login using username \"username\" password \"password\"")
10     public void _user_login_using_username_password(TestContext context) {
11         // ---------------------------------------------------------------------------
12         // TODO Write Test Here
13         context.getLogger().info("UserName: " + context.getStepParameter("Param0"));
14         context.getLogger().info("Password: " + context.getStepParameter("Param1"));
15         // ---------------------------------------------------------------------------
16     }
17
18     @StepDefinition("User browse to page \"<pagename>\"")
19     public void _user_browse_to_page(TestContext context) {
20         // ---------------------------------------------------------------------------
21         context.getLogger().info("Page Name: " + context.getStepParameter("pagename"));
22         // ---------------------------------------------------------------------------
23     }
24
25     @StepDefinition("User click on button \"<buttonname>\"")
26     public void _user_click_on_button(TestContext context) {
27         // ---------------------------------------------------------------------------
28         context.getLogger().info("Button Text: " + context.getStepParameter("buttonname"));
29         // ---------------------------------------------------------------------------
30     }
31
32 }
33
```

14.4 IntelliJ IDE

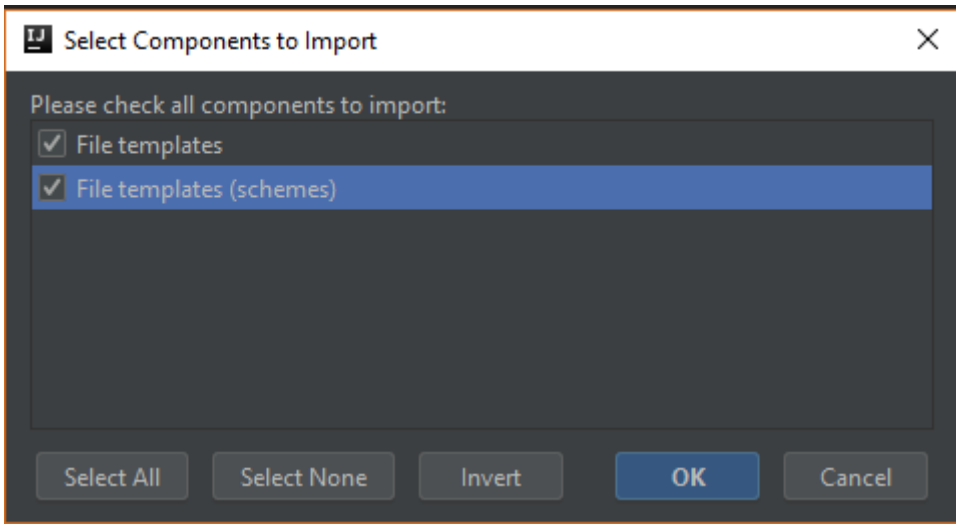
The use of a Java template avoids typing error and increase test development speed. Once imported, templates can be edited to suit business needs.

14.4.1 Configure test templates:

- Download **IntelliJ_template.zip** file from location : Artos_IntelliJ_Template .
- Browse to File => Import Settings.
- Browse and import downloaded **IntelliJ_template.zip** file.



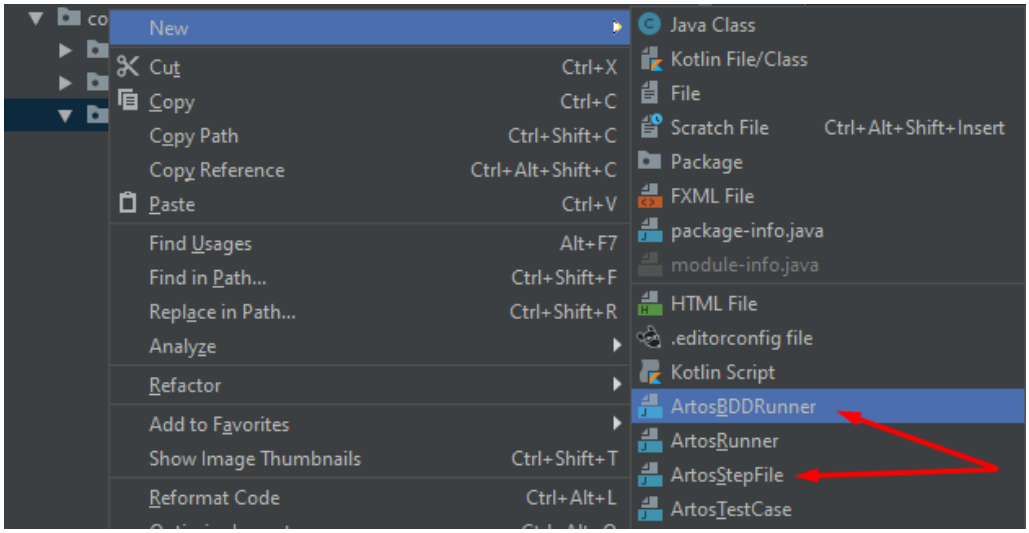
- Select both of the checkboxes.



- Following templates will be added
  - ArtosRunner
  - ArtosTestCase

14.4.2 Use template:

- Right-click on the package.
- Select new and choose the desired template.



- Provide a class name.
- Skeleton code will be added to the class.
- Start writing test logic between two green lines.

```
package com.tests.samples;

import com.artos.annotation.StepDefinition;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

public class StepFile implements TestExecutable {

    @StepDefinition("User login using username \"username\" password \"password\"")
    public void user_login_using_username_password(TestContext context) {
        // ------------------------------------------------------------------------------------------
        // TODO Write Test Here
        context.getLogger().info( msg: "UserName: " + context.getStepParameter( key: "Param0"));
        context.getLogger().info( msg: "Password: " + context.getStepParameter( key: "Param1"));
        // ------------------------------------------------------------------------------------------
    }

    @StepDefinition("User browse to page \"<pagename>\"")
    public void user_browse_to_page(TestContext context) {
        // ------------------------------------------------------------------------------------------
        context.getLogger().info( msg: "Page Name: " + context.getStepParameter( key: "pagename"));
        // ------------------------------------------------------------------------------------------
    }

    @StepDefinition("User click on button \"<buttonname>\"")
    public void user_click_on_button(TestContext context) {
        // ------------------------------------------------------------------------------------------
        context.getLogger().info( msg: "Button Text: " + context.getStepParameter( key: "buttonname"));
        // ------------------------------------------------------------------------------------------
    }
}
```

CHAPTER 15

Implement Project

ARTOS BDD test project consists of three components:

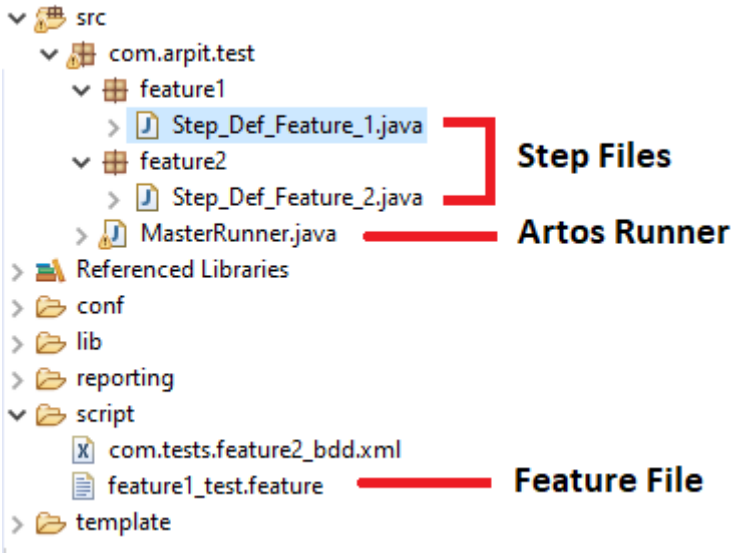
- Test Runner
- Class(s) containing test steps
- Feature File(s)

The project can be configured in many different ways as per the business requirement.

15.1 Recommended Project Structures

15.1.1 Super Structure

- Packages and sub-packages are organized based on features.
- Project contains a single Runner that has visibility of all classes that contains step definition.



CHAPTER 16

Implement Feature File

Feature file is used to describe test scenario(s) in human readable language using Gherkin syntax. Artos accepts feature file(s) as an input and binds scenario steps to relevant method within the test project.

- Feature file is a text file with an extension `.feature`.
- Script file should be placed inside `script` directory of the project.

16.1 Sample Feature File

Listing 16.1: Example: Feature File Sample

```
Feature: Television Test

  Background: User powers on Television
    Given television is connected to power socket
    WHEN television power is turned on
    And television audio volume is to "50"
    Then television displays "Television is ready for the test"
    AND television "greed" LED is on

  @smoke
  Scenario Outline: Television Remote Functions works
    Given remote button "<FunctionButtons>" is pressed
    Then television displays "<ExpectedAction>"

    Examples:
      | FunctionButtons | ExpectedAction                          |
      | ON              | television is already ON      |
      | MUTE            | television audio is muted     |
      | OFF             | NA                                                  |
      | ON              | television is turned ON       |

  @smoke @manual
  Scenario Outline: Volume check
    Given television power is turned on
    And television is configured
      | brightness | pixel        | input | headphone | externalSpeaker |
```

(continues on next page)

(continued from previous page)

```
      |        100 | <resolution> | HDMI  | off       | true            |
    When remote button "<FunctionButtons>" is pressed
    Then User validates "<ExpectedAction>"

    Examples:
      | FunctionButtons     | ExpectedAction           | resolution |
      | VolumeIncrease      | Is volume higher by one? | HD         |
      | VolumeDecrease      | Is volume lower by one?  | FULL_HD    |
```

16.2 Feature File Components

16.2.1 Feature

- Declared using `Feature:` followed by description of the the feature
- Only one `Feature:` can be available in one feature file

Listing 16.2: Example: Feature declaration

```
Feature: Television Test
```

16.2.2 Scenario

- Scenario is declared using `Scenario Outline:` followed by description of the scenario.
- One feature file can have one or more scenarios.
- Scenario is defined using **Gherkin Syntax** which utilizes keywords “Given”, “When”, “And”, “Then”
- Scenario lines declared with Gherkin keywords are called **Steps**.
- Scenario can be attached to one or more groups by adding group name above `Scenario Outline:` as shown in below example.
- Same scenario can be executed multiple time with various data by declaring example table as shown below.
  - Declare wild card within step definition by using `"<wildcard>"` syntax.
  - Provide a table of data which maps to the wild card.
  - First row of the each column maps the wild card.
  - During execution wild card text is replaced by text from the table.
  - Scenario will be executed repeatedly until all rows of the table is exhausted.

Listing 16.3: Example: Scenario Example

```
@smoke @regression
Scenario Outline: Television Remote Functions works
  Given television is on
  And remote battery is fully charged
  When remote button "<FunctionButtons>" is pressed
  Then television displays "<ExpectedAction>"

Examples:
    | FunctionButtons | ExpectedAction            |
    | ON              | television is already ON  |
    | MUTE            | television audio is muted |
    | OFF             | NA                        |
    | ON              | television is turned ON   |
```

16.2.3 Background

- Declared using `Background:` followed by description of the background scenario
- feature file can have only one `Background:` scenario.
- Scenario declared under `Background:` is executed prior to each scenarios declared under `Scenario Outline:`
- `Background:` is declared as very first scenario in feature file.

Listing 16.4: Example: Background example

```
Background: User powers on Television
    Given television is connected to power socket
    WHEN television power is turned on
    And television audio volume is to "50"
    Then television displays "Television is ready for the test"
    AND television "greed" LED is on
```

CHAPTER 17

Implement Runner

A runner is a Java class which meets the following requirements:

- Class is `public` and implements `main()` method.
- The `main()` method invokes Artos' runner object as shown in below example.

Steps

- Create a Java class under required package structure
- Implement `main()` method and Runner code as shown in the example below.

Listing 17.1: Example: Test Runner code

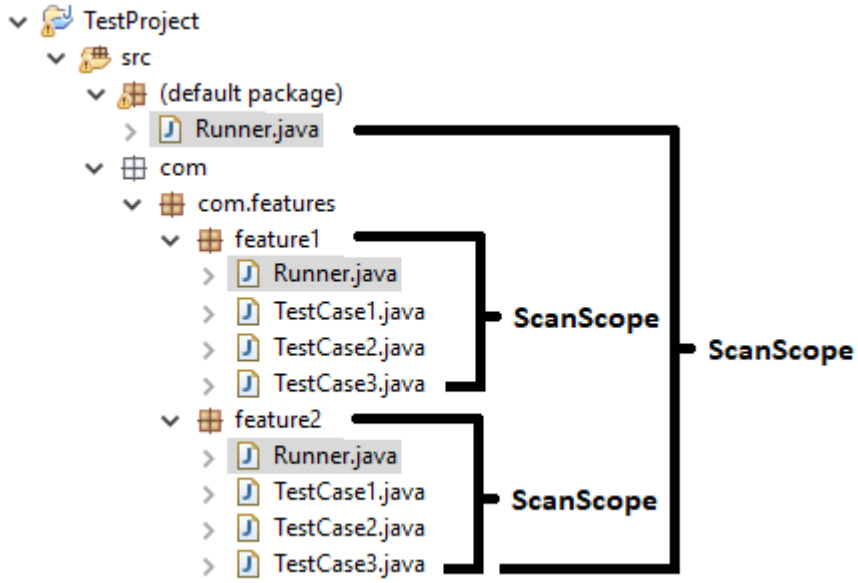
```
1  package com.tests;
2
3  import com.artos.framework.infra.Runner;
4
5  public class ArtosMain {
6          public static void main(String[] args) throws Exception {
7                  Runner runner = new Runner(ArtosMain.class);
8                  List<String> featureFileNameList = new ArrayList<String>();
9                  featureFileNameList.add("feature1_test.feature");
10                 runner.setFeatureFileList(featureFileNameList);
11                 runner.run(args);
12         }
13 }
```

CHAPTER 18

Test Suite & Test Runner

18.1 Scan Scope

A class containing `main()` method that initializes runner object is called a **Runner**. Upon test execution, a Java package containing Runner and it's child packages are scanned by the Runner in search of test cases, thus scanned section of the project is called a **Scan Scope** of the Runner.



18.2 The Runner

- A Runner is the entry point to a test application.
- A Runner at project root location[1] is called a **Master Runner** which has visibility of all test cases within a project.

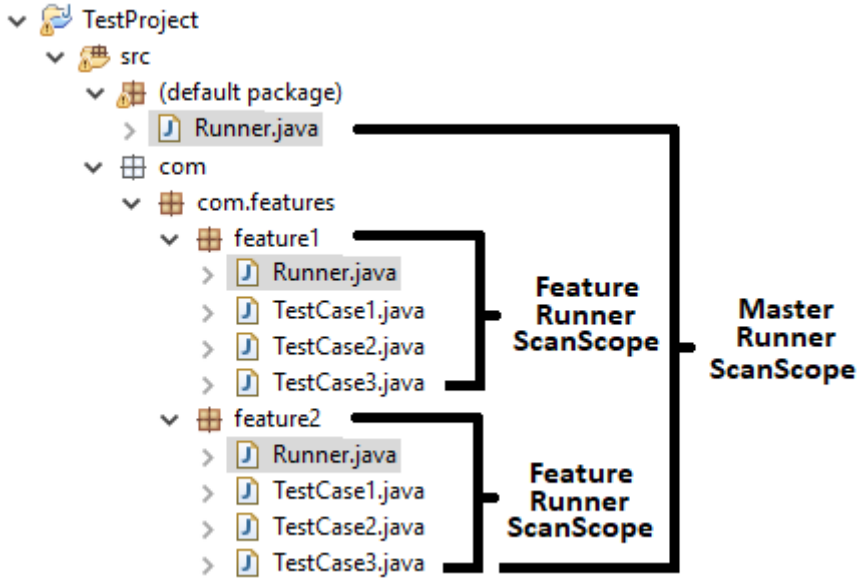
[1] Project Root location

- A Runner created within the individual package is called a **Feature Runner** which has visibility inside its package or its child-packages.
- A test project can have more than one Runner.

Important

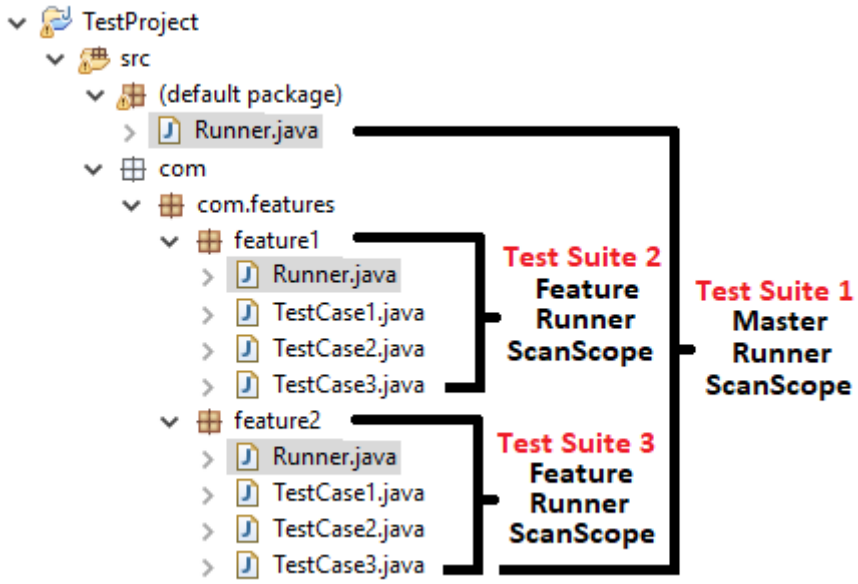
Project root can be different depending on the project configuration. Below guideline is written with default project setup in mind.

- Non-Maven project root location => `src`.
- Maven project root location => `src/main/java`.
- **Eclipse IDE** root location is also known as "default package".



18.3 Test Suite

- A Runner and test cases within Runner's scan scope combined constructs a **Test Suite**.
- A project contains as many test suites as a number of test Runners.
- A test suite can not execute test cases outside its Runner's scan scope.
- Test suites may share one or more test cases.


TestProject
src
(default package)
Runner.java
com
com.features
feature1
Runner.java
TestCase1.java
TestCase2.java
TestCase3.java
feature2
Runner.java
TestCase1.java
TestCase2.java
TestCase3.java
Test Suite 2
Feature
Runner
ScanScope
Test Suite 3
Feature
Runner
ScanScope
Test Suite 1
Master
Runner
ScanScope

CHAPTER 19

Test Status and Outcome

Test status is used to provide a periodic update about status of the test at runtime. Test status can be updated/changed multiple time during test case/unit execution. Each status updates will be visible in log file and Extent report. At the end of unit execution, status update with highest severity ranking will be considered as a test unit result/outcome. If test case has multiple units then highest severity ranking unit result will be considered as a test case result/outcome.

Status	Severity	Usage
PASS	0	Test case/unit executed without any errors
SKIP	1	Test case/unit is skipped during run time
KTF	2	Test case/unit is known to fail
FAIL	3	Test case/unit failed

Listing 19.1: : Test status can be updated using following method.

```
// Status update with simple description
context.setTestStatus(TestStatus.FAIL, "Test did bad thing..");

// Status update with snapshot image
context.setTestStatus(TestStatus.FAIL, new File("./reporting/test.png"), "Test␣
↪failed with error shown in png");
```

Listing 19.2: : In this example test status is updated multiple time in single test unit. The most sever update is **TestStatus.FAIL**, thus test unit outcome will be **FAIL**. Most sever outcome of all test units (in this case only one) is considered as the test case outcome, thus test case outcome is also **FAIL**.

```
package com.tests;

import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.annotation.Unit;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

@TestPlan(preparedBy = "UserName", preparationDate = "1/1/2018", bdd = "GIVEN..
↪WHEN..AND..THEN..")
```

(continues on next page)

(continued from previous page)

```
@TestCase()
public class TestCase_1 implements TestExecutable {

    @Unit()
    public void unit_test1(TestContext context) throws Exception {
        // ------------------------------------------------------
        // TODO : logic goes here..
        context.setTestStatus(TestStatus.PASS, "Test flow is as expected");

        // TODO : logic goes here..
        context.setTestStatus(TestStatus.PASS, "Test flow is as expected");

        // TODO : logic goes here..
        context.setTestStatus(TestStatus.FAIL, "Test flow is not as expected");

        // TODO : logic goes here..
        context.setTestStatus(TestStatus.PASS, "Test flow is as expected");
        // ------------------------------------------------------
    }
}
```

19.1 TestUnit vs TestCase Status

- Test unit outcome is most sever test status update during test unit execution.
- Test case outcome is most sever test outcome among all the test units execution.

Listing 19.3: : In this example test outcome for each test unit is different. The most sever outcome among all test units is **TestStatus.FAIL** so test case outcome is **FAIL**.

```
package com.tests;

import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.annotation.Unit;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

// TestCase outcome is FAIL
@TestPlan(preparedBy = "UserName", preparationDate = "1/1/2018", bdd = "GIVEN..
↪WHEN..AND..THEN..")
@TestCase()
public class TestCase_1 implements TestExecutable {

    // TestUnit outcome  is FAIL
    @Unit()
    public void unit_test1(TestContext context) throws Exception {
        // ------------------------------------------------------
        // TODO : logic goes here..
        context.setTestStatus(TestStatus.FAIL, "Test fails");
        // ------------------------------------------------------
    }

    // TestUnit outcome  is PASS
    @Unit()
    public void unit_test1(TestContext context) throws Exception {
        // ------------------------------------------------------
        // TODO : logic goes here..
```

(continues on next page)

(continued from previous page)

```
28          context.setTestStatus(TestStatus.PASS, "Test passes");
29          // ------------------------------------------------------
30      }
31
32      // TestUnit outcome  is KTF
33      @Unit()
34      public void unit_test1(TestContext context) throws Exception {
35          // ------------------------------------------------------
36          // TODO : logic goes here..
37          context.setTestStatus(TestStatus.KTF, "Test is known to fail");
38          // ------------------------------------------------------
39      }
40
41      // TestUnit outcome  is SKIP
42      @Unit()
43      public void unit_test1(TestContext context) throws Exception {
44          // ------------------------------------------------------
45          // TODO : logic goes here..
46          context.setTestStatus(TestStatus.SKIP, "Test is skipped");
47          // ------------------------------------------------------
48      }
49  }
```

CHAPTER 20

Test Context

A `TestContext` is globally available Java container object and used to hold information relating to test case(s) through out test suite life cycle. During parallel testing each thread

Command	Usage
context.setTestStatus(testStatus, description);	Update test status with description
context.getLogger();	Get logger object
context.getLogger().info();	To log information level string
context.getLogger().debug();	To log debug level string
context.getLogger().warn();	To log warning level string
context.getLogger().error();	To log error level string
context.getLogger().fatal();	To log fatal level string
context.getLogger().trace();	To log trace level string
context.getLogger().disableGeneralLog();	Temporary disable logging
context.getLogger().enableGeneralLog();	Enable logging
context.getLogger().getCurrentGeneralLogFiles();	Get list of test suite relevant log files
context.getLogger().getCurrentRealTimeLogFiles();	Get list of test suite relevant real time log files
context.getLogger().getCurrentSummaryLogFiles();	Get list of test suite relevant summary report
context.getParameterisedObject1();	Get 2D DataProvider object 1
context.getParameterisedObject2();	Get 2D DataProvider object 2
context.getDataProviderMap();	Get Map containing all available DataProviders
context.printMethodName();	Prints executing method name
context.setGlobalObject(String key, Object obj);	Store any object with key
context.setGlobalString(String key, String obj);	Store string object with key
context.setGlobalObject(String key);	Get any object using key
context.getGlobalString(String key);	Get string object using key
context.getCurrentFailCount();	Get total fail count at point of time
context.getCurrentKTFCount();	Get total fail count at point of time
context.getCurrentPassCount();	Get total fail count at point of time
context.getCurrentSkipCount();	Get total fail count at point of time
context.registerListener(listener);	Register new listener to provide test live update
context.deRegisterListener(listener);	Remove registered listener
context.isKnownToFail();	Returns test case known to fail flag
context.getCurrentTestStatus();	Returns current test status
context.getCurrentTestUnitStatus();	Returns current test unit status

20.1 Context example

Listing 20.1: : Example highlights that each test unit is passed with TestContext argument, so each test unit has access to relevant context object in run time

```
1  package com.tests;
2
3  import com.artos.annotation.TestCase;
4  import com.artos.annotation.TestPlan;
5  import com.artos.annotation.Unit;
6  import com.artos.framework.infra.TestContext;
7  import com.artos.interfaces.TestExecutable;
8
9  @TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "GIVEN..WHEN..
   ↪AND..THEN..")
10 @TestCase()
11 public class TestCase_1 implements TestExecutable {
12
13         @Unit()
14         public void unit_test1(TestContext context) throws Exception {
15                 // ------------------------------------------------------
16                 // TODO write logic here
17                 // ------------------------------------------------------
18         }
19
20         @Unit()
21         public void unit_test2(TestContext context) throws Exception {
22                 // ------------------------------------------------------
23                 // TODO write logic here
24                 // ------------------------------------------------------
25         }
26 }
```

CHAPTER 21

Framework Configuration

The `framework_configuration.xml` is used to configure the Artos behavior. File should be located/generated at `./conf` directory of the project. ARTOS generates default configuration file if the file is not present.

Listing 21.1: Default configuration file

```
1        <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2        <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="framework_configuration.xsd">
3         <organization_info profile="dev">
4        <property name="Name">&lt;Organisation&gt; PTY LTD</property>
5        <property name="Address">XX, Test Street, Test address</property>
6        <property name="Country">NewZealand</property>
7        <property name="Contact_Number">+64 1234567</property>
8        <property name="Email">artos.framework@gmail.com</property>
9        <property name="Website">www.theartos.com</property>
10    </organization_info>
11    <logger profile="dev">
12       <!--LogLevel Options : info:debug:trace:fatal:warn:all -->
13       <property name="logLevel">debug</property>
14       <property name="logRootDir">.\reporting\</property>
15       <property name="enableLogDecoration">true</property>
16       <property name="enableTextLog">true</property>
17       <property name="enableHTMLLog">false</property>
18       <property name="enableRealTimeLog">false</property>
19       <property name="enableExtentReport">true</property>
20       <property name="enableJUnitReport">true</property>
21       <property name="enableLogCleanup">true</property>
22    </logger>
23    <smtp_settings profile="dev">
24       <property name="ServerAddress">smtp.gmail.com</property>
25       <property name="SSLPort">587</property>
26       <property name="SMTPAuth">true</property>
27       <property name="SendersName">John Murray</property>
28       <property name="SendersEmail">test@gmail.com</property>
29       <property name="emailAuthSettingsFilePath">.\conf\user_auth_settings.xml</
↪property>
30       <property name="ReceiversEmail">test@gmail.com</property>
31       <property name="ReceiversName">Mac Murray</property>
```

(continues on next page)

(continued from previous page)

```
        <property name="EmailSubject">Artos Email Client</property>
        <property name="EmailMessage">This is a test Email from Artos</property>
    </smtp_settings>
    <dashboard profile="dev">
        <!--Default : 127.0.0.1 -->
        <property name="dashBoardRemoteIP">127.0.0.1</property>
        <property name="dashBoardRemotePort">11111</property>
    </dashboard>
    <features profile="dev">
        <property name="enableGUITestSelector">true</property>
        <property name="enableGUITestSelectorSeqNumber">true</property>
        <property name="enableBanner">true</property>
        <property name="enableOrganisationInfo">true</property>
        <property name="enableEmailClient">false</property>
        <property name="enableArtosDebug">false</property>
        <property name="generateEclipseTemplate">true</property>
        <property name="generateIntelliJTemplate">false</property>
        <property name="generateTestScript">true</property>
        <property name="stopOnFail">false</property>
        <property name="enableDashBoard">false</property>
    </features>
        </configuration>
```

Each XML block lets user control specific functionalities of the framework. Default configuration's are applied if user has missing block or invalid profile is specified. `profile="dev"` indicates development profile. Development profile is used by default unless specific profile is set by user at launch time. User can hardcode profile name in their runner or specify profile name via command line parameter at launch time. Profile lets user apply different conigurations between development or production environments. There are not any limitations on how many profile user can create and they all can be listed in same file.

21.1 <organization_info>

The `<organization_info>` tag provides a way to specify organization information which will be printed at the start of each log files and on a console. This does not apply to rolled over log files. Printing of organization information can be enabled or disabled by setting property under the `<feature>` tag as shown below.

```
    <property name="enableOrganisationInfo">true</property>
```

Attribute

Name	Description
profile	profile name

Properties

Property Name	Content	Description
Name	String	Organization name
Address	String	Organization address
Country	String	Country name
Contact_Number	String	Organization contact number
Email	String	Organization email address
Website	String	Organization Website

21.2 <logger>

The `<logger>` tag provides a way to configure the log framework and the report behavior.

Attribute

Name	Description
profile	profile name

Properties

Table 21.1: :header: Property Name, Content, Description :widths: 30, 30, 40 :stub-columns: 0

logLevel	String	Set log level
logRootDir	String	Set log root directory relative to project
enableLogDecoration	Boolean	Enable or disable log decoration
enableTextLog	Boolean	Enable or disable text log and report
enableHTMLLog	Boolean	Enable or disable HTML log and report
enableRealTimeLog	Boolean	Enable or disable read time logs
enableExtentReport	Boolean	Enable or disable the Extent report
enableJUnitReport	Boolean	Enable or disable JUnit report
enableLogCleanup	Boolean	Enable or disable cleaning up previous logs prior to execution

One of the following log level can be selected:

.	.	.	.	.	.
info	debug	trace	fatal	warn	all

Log file path construction: `logRootDir + test suite packageName + log file.`

```
    ./reporting/com.artos.featuretest/com.artos.tests_0_190713-17.53.20.183-all.log
```

Enabling log decoration will add following information in front of each log line.

```
* [%-5level] = Log level up to 5 char max
* [%d{yyyy-MM-dd_HH:mm:ss.SSS}] = Date and time
* [%t] = Thread number
* [%F] = File where logs are coming from
* [%M] = Method which generated log
* [%c{-1}] = ClassName which issued logCommand
```

Log files and reports are generated with following specification:

```
>>> File naming convention:
Runner package name + Thread number + TestSuite name (if specified) + date and␣
↪time + Log Type + file extension
```

```
// Text log file example
* com.artos.feature1_0_xyz_190713-17.53.20.183-all.log
* com.artos.feature1_0_xyz_190713-17.53.20.183-realtime.log
* com.artos.feature1_0_xyz_190713-17.53.20.183-summary.log

// HTML log file example
* com.artos.feature1_0_xyz_190713-17.53.20.183-all.html
* com.artos.feature1_0_xyz_190713-17.53.20.183-realtime.html
* com.artos.feature1_0_xyz_190713-17.53.20.183-summary.html
```

(continues on next page)

(continued from previous page)

```
// Extent report file example
* com.artos.feature1_0_xyz_190713-17.53.20.183-all-extent.html
```

21.3 <smtp_settings>

The <smtp_settings> tag provides a way to configure SMTP settings for the email.

Attribute

Name	Description
profile	profile name

Properties

Property Name	Content	Description	Example
ServerAddress	String	SMTP server address	smtp.gmail.com
SSLPort	Integer	SSL Port number	587
SMTPAuth	Boolean	Enable SMTP Authentication	true
SendersName	String	Email sender's name	John Murray
SendersEmail	String	Sender's email address	test@gmail.com
emailAuthSettingsFilePath	String	Email credential file path	.\conf\user_auth_settings.xml
ReceiversEmail	String	Receiver's email address	test@gmail.com
ReceiversName	String	Receiver's Name	Mac Murray
EmailSubject	String	Email subject line	Test results
EmailMessage	String	Email body	This is a test Email from Artos

21.4 <dashboard>

Attribute

Name	Description
profile	profile name

Properties

Property Name	Content	Description
dashBoardRemoteIP	Boolean	Remote IP for Dashboard events
dashBoardRemotePort	Boolean	Remote Port for UDP events

21.5 <features>

The `<features>` tag provides a way to enable/disable the Artos's feature.

Attribute

Name	Description
profile	profile name

Properties

Table 21.2: :header: Property Name, Content, Description :widths: 40, 15, 45 :stub-columns: 0

enableGUITestSelector	Boolean	Enable GUI test selector
enableGUITestSelectorSeqNumber	Boolean	Enable test seq on GUI test selector
enableBanner	Boolean	Enable ARTOS banner
enableOrganisationInfo	Boolean	Enable organization information printing
enableEmailClient	Boolean	Enable email client
enableArtosDebug	Boolean	Enable Artos's debug log
generateEclipseTemplate	Boolean	Enable generation of Eclipse template
generateIntelliJTemplate	Boolean	Enable generation of IntelliJ template
generateTestScript	Boolean	Enable test script generation
stopOnFail	Boolean	Enable test execution stop on fail feature
enableDashBoard	Boolean	Enable or disable dashboard listener

CHAPTER 22

Importance Indicator

In fast paced environment, debugging failed test-cases that are of high importance is crucial than spending time debugging test-cases that are of low importance. ARTOS **Importance Indicator** feature lets user specify importance of the test case using `@TestImportance` annotation. Specified importance level is reflected in the test reports and on the console during **Failure Highlights**. This allows test developer/lead/manager to judge seriousness of the failure quickly and they can take informed decision by just glancing over the failure report.

Importance Indicator can be defined at a test level or test unit level or both

Listing 22.1: : Sample test case and test unit with ***Importance Indicator***

```
1  // This test case overall is of a CRITICAL importance
2  @TestImportance(Importance.CRITICAL)
3  @TestPlan(preparedBy = "Arpit", preparationDate = "18/02/2019", bdd = "GIVEN..WHEN.
   ↪.AND..THEN..")
4  @TestCase
5  public class Test_TestUnit_Importance implements TestExecutable {
6
7          @TestImportance(Importance.CRITICAL)
8          @Unit(sequence = 1)
9          public void testUnit_1(TestContext context) {
10                 // -------------------------------------------------
11                 context.setTestStatus(TestStatus.FAIL, "This is a CRITICAL␣
   ↪importance test unit");
12                 // -------------------------------------------------
13         }
14
15         @TestImportance(Importance.LOW)
16         @Unit(sequence = 2)
17         public void testUnit_2(TestContext context) {
18                 // -------------------------------------------------
19                 context.setTestStatus(TestStatus.FAIL, "This is a LOW importance␣
   ↪test unit");
20                 // -------------------------------------------------
21         }
22
23         @TestImportance(Importance.MEDIUM)
24         @Unit(sequence = 3)
25         public void testUnit_3(TestContext context) {
```

(continues on next page)

(continued from previous page)

```
26                  // ------------------------------------------------
27                  context.setTestStatus(TestStatus.FAIL, "This is a MEDIUM␣
↪importance test unit");
28                  // ------------------------------------------------
29          }
30
31  }
```

}

22.1 Importance Indicator in console failure highlights

```
********************************************************************************
                  FAILED TEST CASES (1)

********************************************************************************
1   com.artos.tests.annotation_testimportance.Test_TestUnit_Importance [CRITICAL]
        |-- testUnit_1(context) [CRITICAL]
        |-- testUnit_2(context) [LOW]
        |-- testUnit_3(context) [MEDIUM]
********************************************************************************
```

22.2 Importance Indicator in summary report

```
********************************** Header Start ******************************************
Organisation_Name : <Organisation> PTY LTD
Organisation_Country : NewZealand
Organisation_Address : XX, Test Street, Test address
Organisation_Phone : +64 1234567
Organisation_Email : artos.framework@gmail.com
Organisation_Website : www.theartos.com
********************************** Header End ********************************************
FAIL = com.artos.tests.annotation_testimportance.Test_TestUnit_Importance.............................. P:0    F:1    S:0    K:0    [CRITICAL  ] duration:000:00:00.06
  |--FAIL = testUnit_1(context)                                                                          :      :      :      :      [CRITICAL  ] duration:000:00:00.01
  |--FAIL = testUnit_2(context)                                                                          :      :      :      :      [LOW       ] duration:000:00:00.01
  |--FAIL = testUnit_3(context)                                                                          :      :      :      :      [MEDIUM    ] duration:000:00:00.01

********************************************************************************
PASS:0 FAIL:1 SKIP:0 KTF:0 EXECUTED:1

Test start time : 18-02-2019 08:55:14
Test finish time : 18-02-2019 08:55:14
Test duration : 0 min, 0 sec
```

CHAPTER 23

Failure-Highlights

Testers/Developers are generally interested in failed test-cases. Monitoring logs on a console is generally a first step towards debugging and following that tester/developer starts to look for log files/reports. ARTOS generates **Failure-Highlights** on a console to help user speed up debugging. Failure-Highlights help them quickly judge the area/feature of mot failures. Failure-Highlights also includes `Importance Indicator` if specified which helps user priorities test debugging.

If mature IDE is used then Failure-Highlights will appear red in color to draw an attention of user.

Note: Failure information is already present in log files/reports so Failure-Highlights will not be recorded in any of the files. It will go away as soon as console is cleared.

23.1 Failure-Highlights Example

Failure Highlight without Importance Indicator

```
*************************************************************************
                FAILED TEST CASES (3)

*************************************************************************
1   com.artos.tests.annotation_dataprovider.Test_IntegerAndByteArray
        |-- execute(context) : DataProvider[0]
        |-- execute(context) : DataProvider[2]
2   com.artos.tests.annotation_dataprovider.Test_ExtentReport_Child_Status
        |-- execute(context) : DataProvider[1]
3   com.artos.tests.annotation_dataprovider.Test_Exception_in_Dataprovider
        |-- execute(context) : DataProvider[0]
*************************************************************************
```

Failure Highlight + Importance Indicator

```
************************************************************************
                FAILED TEST CASES (3)

************************************************************************
1   com.artos.tests.annotation_dataprovider.Test_IntegerAndByteArray [CRITICAL]
        |-- execute(context) : DataProvider[0] [CRITICAL]
        |-- execute(context) : DataProvider[2] [CRITICAL]
2   com.artos.tests.annotation_dataprovider.Test_ExtentReport_Child_Status [MEDIUM]
        |-- execute(context) : DataProvider[1] [MEDIUM]
3   com.artos.tests.annotation_dataprovider.Test_Exception_in_Dataprovider [CRITICAL]
        |-- execute(context) : DataProvider[0] [CRITICAL]
************************************************************************
```

CHAPTER 24

Logging Framework

Logs and reports are heart of any test framework. ARTOS includes Log4j based pre-configured and ready to use log framework.

24.1 Log files

ARTOS by default generates two types of log files per test suite execution.

- **General logs** : Test application logs will be recorded in a general log file.
  - By default general logs do not include time-stamp or other log decoration, which makes it easy to read.
  - Time-stamp and log decoration can be enabled using `framework_configuration.xml` file.
- **Real-time logs** : In addition to general log file, ARTOS generates real time log file. Real time logs are produced by ARTOS' built in connectors or any class which implements `Connectable` interface. Real time logs includes time stamp and log decoration which may be useful in debugging. All data sent and received from built in connectors are logged in real time log file which provides following benefits:
  - Real time logs can be used to measure system performance by measuring time between the sent/receive events. Log parsing is easy with fixed format of the log file.
  - Real time logs are always recorded with time stamp, thread name, calling method name and other required information so test developers may choose to omit those information from general log and keep general logs noise free and human readable.Separate log file is generated to record sent and received events/data from classes that implements `Connectable` interface.

24.2 Test report

ARTOS by default generates live text based report per test suite execution. Test report includes following information. Test report does not contain any business critical information so it can be shared to external parties.

- Test case fully qualified name and PASS/FAIL/SKIP/KTF summary
- PASS/FAIL/SKIP/KTF count
- Bug reference for failed test cases
- Test time duration with Millisecond accuracy
-

```
PASS = com.tests.samples.Sample_ExpectedException..................................................... P:1    F:0    S:0    K:0    [          ] duration:000:00:00.21
  |--PASS = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:00.08
  |--PASS = testUnit_2(context)                                                                       :      :      :      :      [          ] duration:000:00:00.02
  |--PASS = testUnit_3(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_4(context)                                                                       :      :      :      :      [          ] duration:000:00:00.02
  |--PASS = testUnit_5(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_6(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01
PASS = com.tests.samples.Sample_DataProvider.......................................................... P:2    F:0    S:0    K:0    [          ] duration:000:00:00.20
  |--PASS = testUnit_1(context) : data[0]                                                             :      :      :      :      [          ] duration:000:00:00.01 JIRA-124
  |--PASS = testUnit_1(context) : data[1]                                                             :      :      :      :      [          ] duration:000:00:00.01 JIRA-124
  |--PASS = testUnit_1(context) : data[2]                                                             :      :      :      :      [          ] duration:000:00:00.01 JIRA-124
  |--PASS = testUnit_1(context) : data[3]                                                             :      :      :      :      [          ] duration:000:00:00.01 JIRA-124
  |--PASS = testUnit_2(context) : data[0]                                                             :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_2(context) : data[1]                                                             :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_2(context) : data[2]                                                             :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_2(context) : data[3]                                                             :      :      :      :      [          ] duration:000:00:00.01
KTF  = com.tests.samples.Sample_KnownToFail.......................................................... P:2    F:0    S:0    K:1    [          ] duration:000:00:00.02
  |--KTF  = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01 JIRA-123
  |--PASS = testUnit_2(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01 JIRA-123
PASS = com.tests.samples.Sample_localBeforeAfterMethods.............................................. P:3    F:0    S:0    K:1    [          ] duration:000:00:00.02
  |--PASS = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:00.01
  |--PASS = testUnit_2(context)                                                                       :      :      :      :      [          ] duration:000:00:00.00
PASS = com.tests.samples.Sample_CustomPrompt.......................................................... P:4    F:0    S:0    K:1    [          ] duration:000:00:34.426
  |--PASS = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:05.62
  |--PASS = testUnit_2(context)                                                                       :      :      :      :      [          ] duration:000:00:05.101
  |--PASS = testUnit_3(context)                                                                       :      :      :      :      [          ] duration:000:00:05.118
  |--PASS = testUnit_4(context)                                                                       :      :      :      :      [          ] duration:000:00:05.98
  |--PASS = testUnit_5(context)                                                                       :      :      :      :      [          ] duration:000:00:05.102
  |--PASS = testUnit_6(context)                                                                       :      :      :      :      [          ] duration:000:00:03.652
  |--PASS = testUnit_7(context)                                                                       :      :      :      :      [          ] duration:000:00:05.180
  |--PASS = testUnit_8(context)                                                                       :      :      :      :      [          ] duration:000:00:00.107
PASS = com.tests.samples.Sample_Transform............................................................. P:5    F:0    S:0    K:1    [          ] duration:000:00:00.07
  |--PASS = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:00.07
PASS = com.tests.samples.Sample_Guard................................................................. P:6    F:0    S:0    K:1    [          ] duration:000:00:00.10
  |--PASS = testUnit_1(context)                                                                       :      :      :      :      [          ] duration:000:00:00.09

************************************************************************
PASS:6 FAIL:0 SKIP:0 KTF:1 EXECUTED:7

Test start time : 18-02-2019 11:04:34
Test finish time : 18-02-2019 11:05:09
Test duration : 0 min, 34 sec
```

Note: Test application and device under test may communicate using well defined protocols like Serial, RS485, TCP, UDP, TLS, USB, Protocol buffers etc.., Test application connector may queue incoming/outgoing events/data and application processes them one by one. Logging sent and received data live with time-stamp in separate log file (Realtime log file) keeps all the other noise away. Those logs can be later processed easily using Python script or similar. For all other debugging General logs can be used.

24.3 Log File Path and Naming Convention

- Log files path := `./reporting/subdirectory/`. Runner's package name is used as a sub-directory name to keep log files organized.
- General log filename := `package name + "_" + suitename (optional) + "_" + threadnumber + "_" + timestamp + "-all.log"`
- RealTime log filename := `package name + "_" + suitename (optional) + "_" + threadnumber + "_" + timestamp + "-realtime.log"`
- Summary report filename := `package name + "_" + suitename (optional) + "_" + threadnumber + "_" + timestamp + "-summary.log"`

Example:

General log file : `./reporting/com.test.feature1/com.test.feature1_suite1_0_1549353269885-all.log`
Real time log file : `./reporting/com.test.feature1/com.test.feature1_suite1_0_1549353269885-realtime.log`
Summary report file : `./reporting/com.test.feature1/com.test.feature1_suite1_0_1549353269885-summary.log`

24.4 Log Format

- ARTOS supports text and HTML formatted logs.
- Text formatted log and report are enabled by default.
- Text and/or HTML logs can be enabled/disabled using `framework_configuration.xml` file.

24.5 Log Pattern

- General logs are not decorated by default to maintain simplicity.
- Log decoration can be enabled/disabled using `framework_configuration.xml` file.
  - Decoration disabled log pattern: `"%msg%n%throwable"`
  - Decoration enabled log patter: `"[%-5level][%d{yyyy-MM-dd_HH:mm:ss.SSS}][%t][%F][%M][%c{1}] - %msg%n%throwable"`
  - Refer: Log4j_Pattern for more information.

24.6 Log Rollover Policy

- Log rollover policy is triggered based on a file size of 20MB.

24.7 Log Level

Log level can be configured using `conf/framework_configuration.xml` file.

- Following log levels are supported:

Level	Description
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates severe error events that will presumably lead the application to abort.
INFO	Designates informational messages that highlight the progress of the application at coarse level.
OFF	The highest possible rank and is intended to turn off logging.
TRACE	Designates finer-grained informational events than the DEBUG.
WARN	Designates potentially harmful situations.

24.8 Runtime Log Enable/Disable

General log can be enabled/disabled at run time using following methods:

- Disable log: `context.getLogger().disableGeneralLog();`
- Enable log: `context.getLogger().enableGeneralLog();`

24.9 Log File Tracking

All log files relevant to test suite are tracked and can be queried at runtime using following methods:

- General log file list: `context.getLogger().getCurrentGeneralLogFiles();`
- Real-Time log file list: `context.getLogger().getCurrentRealTimeLogFiles();`
- Summary report file list: `context.getLogger().getCurrentSummaryLogFiles();`

24.10 FAIL Stamp Injection

FAIL Stamp is injected to log stream after test status is updated to FAIL. This allows user to know at which exact line the test unit failed during execution.

```
>>> Sample Stamp
**********************************
********** FAIL HERE ***********
**********************************
```

24.11 Parameterized logging

ARTOS supports parameterized logging.

- Logging using string concatenation:

```
context.getLogger().info("This is a test String" + "This is
a test String 1");     context.getLogger().debug("This is a test
String" + "This is a test String 2");
```

- Logging using parameterized string:

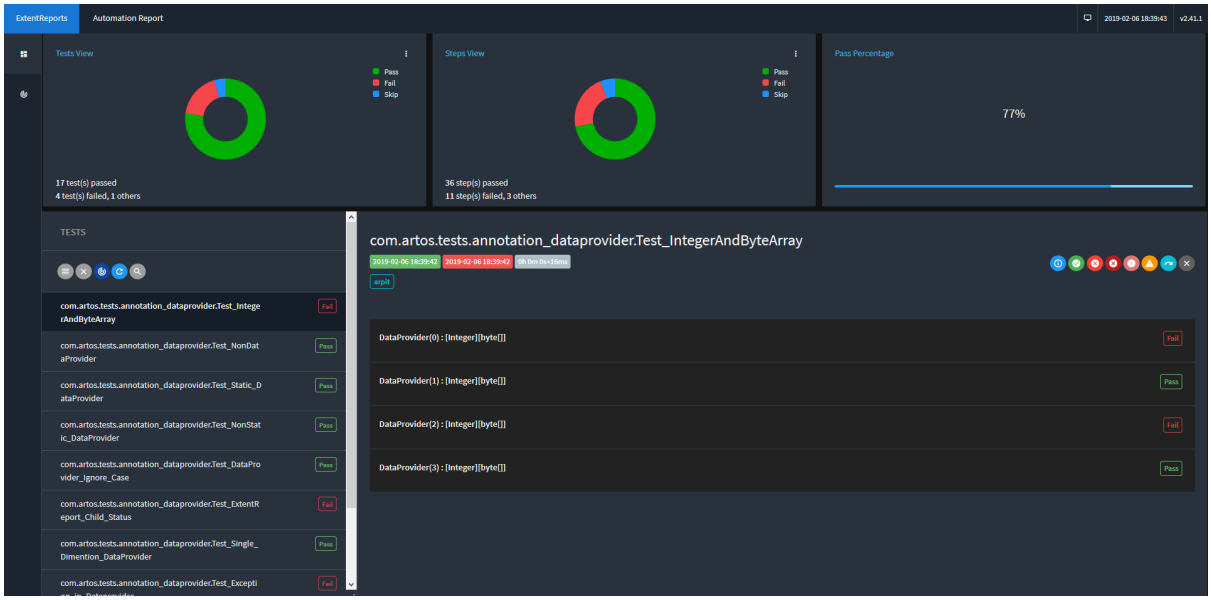
```
context.getLogger().info("This is a test String {} {}", "one",
"two"); context.getLogger().debug("This is a test String {} {}",
"one", "two");
```

Warning: Parameterized logging is less efficient compare to string concatenation, if test application does not use multiple log levels then it is recommended to avoid parameterized logging. Parameterized logging overall improves performance in case where test application utilities multiple log levels and user switches between log levels because system does not waste time in concatenating strings for logs which are disabled using log level configuration.

CHAPTER 25

Extent Report

- ARTOS by default generates professional looking Extent test report.
- Separate extent report is generate per test suite execution.
- Extent reporting can be enabled/disabled via `conf/framework_configuration.xml` file.



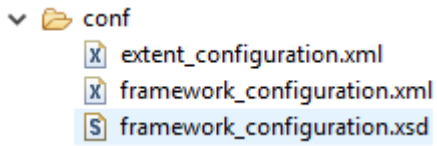
Important:

- Extent report requires Internet access to get CSS information to display theme correctly. If Internet access is blocked or not available then extent report may look broken.

CHAPTER 26

Generate Default Configurations

The ARTOS' generates required configuration files and directories upon launch if not present. Configuration files are generated in `conf` directory. If configuration files are already present then it will not be overwritten.



Configuration Name	Description
conf/extent_configuration.xml	configuration for extend reports
conf/framework_configuration.xml	configuration for ARTOS framework
conf/framework_configuration.xsd	XML schema definition for framework_configuration.xml

CHAPTER 27

Use Command line Parameters

ARTOS support short and long convention of command line parameters. Supported commands are listed below:

Short	Long	Description
-c	–contributors	Prints ARTOS contributors name
-h	–help	Command line help
-p <arg>	–profile <arg>	Framework configuration profile name
-t <arg>	–testscript <arg>	Test script file path
-v	–version	ARTOS' version

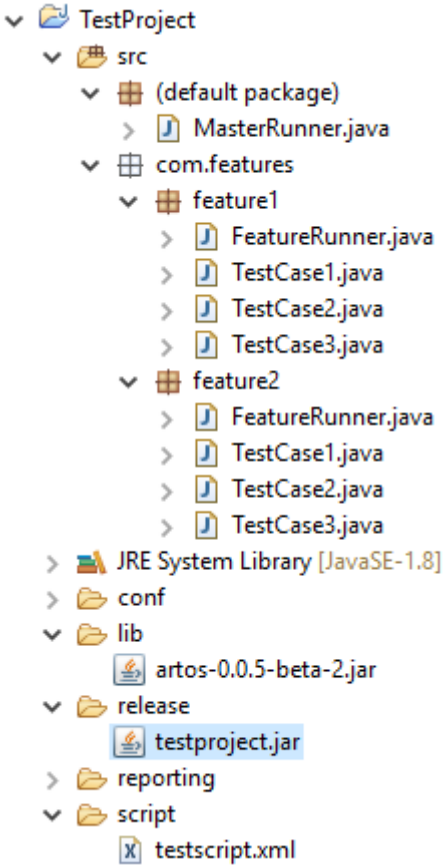
27.1 Example 1: Run from compiled classes

```
// long convention
java -cp ".\lib\*;.\bin\" MasterRunner --testscript="testscript.xml" --profile="dev
↪"

// short convention
java -cp ".\lib\*;.\bin\" MasterRunner -t="testscript.xml" -p="dev"
```

27.2 Example 2: Run from Jar

```
// long convention
java -jar .\lib\testproject.jar --testscript="testscript.xml" --profile="dev"

// short convention
java -jar .\lib\testproject.jar -t="testscript.xml" -p="dev"
```

27.3 Above examples are created using below project structure:



- Project compiled classes are located inside `bin` directory.
- Project dependency Jars are located inside `lib` directory.
- Project test script is located inside `script` directory.
- Project exported as JAR inside directory `release`.
- Project jar name = `testproject.jar`
- Project Master Runner class name = `MasterRunner`.
- Project Test Script name = `testscript.xml`.
- Project framework_config.xml profile = `dev`.

```
>>> Project Jar manifest was created using following information:
Manifest-version: 1.0
Created-By: 1.0 (ARTOS Team)
Main-Class: MasterRunner
Class-Path: ../lib/artos-0.0.5-beta-2.jar
```

CHAPTER 28

TestScript

Test script is XML file used to instruct test Runner on how to execute test suite.

- Test script overrides configuration specified in the Runner class.
- Test script can only be specified using command line argument.
- Test script must be present inside script directory of the project.

Listing 28.1: Sample test script

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration version="1">
  <suite loopcount="1" name="SuiteName">
    <tests>
      <test name="com.featureXYZ.TestCase_1"/>
      <test name="com.featureXYZ.TestCase_2"/>
    </tests>
    <parameters>
      <parameter name="PARAMETER_0">parameterValue_0</parameter>
      <parameter name="PARAMETER_1">parameterValue_1</parameter>
      <parameter name="PARAMETER_2">parameterValue_2</parameter>
    </parameters>
    <testcasegroups>
      <group name="*"/>
    </testcasegroups>
    <testunitgroups>
      <group name="*"/>
    </testunitgroups>
  </suite>
</configuration>
```

28.1 <configuration version="1">

- `<configuration></configuration>` is a root tag which is responsible to hold multiple `<suite>` and its child tags.
- `version` attributes is used to identify test script version.

28.2 <suite>

- `<suite></suite>` represents one test suite.
- Test script may have multiple `<suite></suite>` elements.
- All specified test suites runs in parallel upon test script execution.

28.2.1 </suite> attributes:

loopcount

- Loop count specifies number of time test suite execution will be repeated.
- Loop count value "1" will be used in case of missing or invalid argument is provided.

name

- String value which is used in construction of log file name.
- Value longer than 10 characters will be trimmed to 10 characters.
- Allowed character sets are [A-Z][a-z][0-9][-]
- User should choose unique name per test suite so log files can be identified using identifier.

28.3 <tests>

- `<tests></tests>` contains list of test cases. test cases can be specified using their fully qualified path name. Test case names are case sensitive.
- Test cases will be executed in same sequence as specified in the script.
- Test cases are listed in the script but marked with attribute `TestCase(skip=true)` will be omitted from execution list.
- Test cases are listed in the script but outside Runner's scan scope will be omitted from execution list.
- If `<tests></tests>` is empty then all test cases within Runner's scan scope are executed following sequence specified using `TestCase(sequence=1)` attribute.
- Remaining test cases will be further filtered using group filter which will be explained under `<testcasegroups>` tag description.

Listing 28.2: Sample <tests> element

```
<tests>
        <test name="com.test.feature1.TestCase_1"/>
        <test name="com.test.feature1.TestCase_2"/>

        <test name="com.test.feature2.TestCase_1"/>
        <test name="com.test.feature2.TestCase_2"/>
</tests>
```

28.4 <parameters>

Provides a way to specify test suite specific information which is accessible at run time. (for example: product serial number, ip address, file paths etc..)

- `<parameters></parameters>` contains list of parameters and their string value. Parameter name and value are case sensitive.

- All listed parameters value can be requested at run time using method `context.getGlobalObject(key);`.
- All listed parameters value can be updated at run time using method `context.setGlobalObject(key, obj);`.
- Each test suite parameters are maintained separately so they can be updated or removed without conflict.

Listing 28.3: Sample <parameters> element

```
<parameters>
    <parameter name="SerialNumber">ABC_0567</parameter>
    <parameter name="DownloadPath">/usr/temp/download</parameter>
    <parameter name="Product_IP">192.168.1.101</parameter>
</parameters>
```

28.5 <testcasegroups>

- `<testcasegroups></testcasegroups>` contains list of group names or regular expression. Group names are case in-sensitive.
- Test cases short listed following steps described in `<tests>` are further filtered using group names listed in `<testcasegroups>` tag. Test cases do not belong to any of the listed group are omitted from execution list.
- Filter will not be applied in case of missing `<testcasegroups>` tag.

Listing 28.4: All listed test cases will be added to execution list

```
<testcasegroups>
    <group name="*"/>
</testcasegroups>
```

Listing 28.5: Test case belongs to “Automated” OR “Semi-Automated” test cases will be added to execution list

```
<testcasegroups>
    <group name="Automated"/>
    <group name="Semi-Automated"/>
</testcasegroups>
```

28.6 <testunitgroups>

- `<testunitgroups></testunitgroups>` contains list of group names or regular expression. Group names are case in-sensitive.
- Unit group filter is only applied to test cases that are short listed after applying `<testcasegroups>` group filter.
- Filter will not be applied in case of missing `<testunitgroups>` tag.

Listing 28.6: All test units will be added to execution list

```
<testunitgroups>
    <group name="*"/>
</testunitgroups>
```

Listing 28.7: Test units belongs to “Fast” OR “Slow” test units will be added to execution list

```
<testunitgroups>
    <group name="Fast"/>
    <group name="Slow"/>
</testunitgroups>
```

28.7 Auto Generate test script

Test script is generated manually or auto generated using ARTOS inbuilt feature.

- To enable auto generation feature
  - Change `generateTestScript` property within `conf/framework_configuration.xml` file to **true**.

```
>>> <property name="generateTestScript">true</property>
```

- Once enabled
  - Run ARTOS using Runner class via IDE
  - Test script will be auto generated inside `script` directory.

CHAPTER 29

Parallel Suite Execution

- All test suites specified in test script will run in parallel upon test application launch. Test suites can have same or different test cases. User can specify different parameters per test suite which will be available during run time.
- Parallel suite execution feature can be used in following scenarios:
  - Test multiple product at the same time.
  - Test one product by splitting test cases into multiple test suites.

Sample script is given below which targets two different products based on specified IP address.

Listing 29.1: Sample test script

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration version="1">

  <suite loopcount="1" name="TestSuite1">
    <tests>
      <test name="com.featureXYZ.TestCase_1"/>
      <test name="com.featureXYZ.TestCase_2"/>
    </tests>
    <parameters>
      <parameter name="SerialNumber">ABC_0123</parameter>
      <parameter name="DownloadPath">/usr/temp/download</parameter>
      <parameter name="Product_IP">192.168.1.100</parameter>
    </parameters>
    <testcasegroups>
      <group name="*"/>
    </testcasegroups>
    <testunitgroups>
      <group name="*"/>
    </testunitgroups>
  </suite>

  <suite loopcount="1" name="TestSuite2">
    <tests>
      <test name="com.featureXYZ.TestCase_1"/>
      <test name="com.featureXYZ.TestCase_2"/>
    </tests>
```

(continues on next page)

(continued from previous page)

```
    <parameters>
      <parameter name="SerialNumber">ABC_0567</parameter>
      <parameter name="DownloadPath">/usr/temp/download</parameter>
      <parameter name="Product_IP">192.168.1.101</parameter>
    </parameters>
    <testcasegroups>
      <group name="*"/>
    </testcasegroups>
    <testunitgroups>
      <group name="*"/>
    </testunitgroups>
  </suite>

</configuration>
```

CHAPTER 30

TCP Server (Single Client)

TCPServer class is designed to talk to a single client. If client connection is established, server will launch separate thread that is responsible for listening to incoming messages from client. All received messages are added to a queue that can be polled by driving application. TCPServer implements `Connectable` interface so same connector object can be used for `HeartBeat` class.

TCPServer have following facilities:

- Message Filter
- Message Parser
- Real-Time log

30.1 Simple server

Below code will start a server and will listen for incoming client.

Listing 30.1: : Simple server example

```
// launch server
int port = 1200;
TCPServer server = new TCPServer(port);
server.connect();
```

30.2 Simple server with timeout

Below code will start a server and will listen for client connection until timeout is reached (5000 milliseconds in this case).

Listing 30.2: : Simple server with timeout example

```
// connect server with soTimeout
int port = 1200;
int soTimeout = 5000;
TCPServer server = new TCPServer(port);
server.connect(soTimeout);
```

30.3 Server with message filter

User may require filtering some messages out during server/client communication (Heartbeat, status messages etc..). Message filter interface allows user to define how to filter messages. ARTOS real time log file will log filtered and non-filtered messages so user can go through all received events/data. Message filter is applied after message parse object de-serialises incoming messages, so user should assume non-fused messages while writing message filter code. User can apply more than one filter object in same TCPServer object.

Note: Implementation inside meetCriteria() method may impact performance of receiver thread so user should keep implementation simple and light.

Below code creates filter object using `ConnectableFilter` interface. User must provide implementation of the method `meetCriteria(byte[] data)` so receiver thread can filter messages which meets those criteria(s). In current example any received byte array matches `"00 00 00 04 01 02 03 04"` will be filtered out and will not be added to message queue.

Listing 30.3: : Filter object creation example

```
Transform _transform = new Transform();

// Create filter object
ConnectableFilter filter = new ConnectableFilter() {
        @Override
        public boolean meetCriteria(byte[] data) {
                if (Arrays.equals(data, _transform.strHexToByteArray("00 00 00 04␣
↪01 02 03 04"))) {
                        return true;
                }
                return false;
        }
};
```

Below code will launch server which is listening on port 1200 with supplied filter list. Messages which meets criteria specified in supplied filter(s) will be dropped from the message queue.

Listing 30.4: : TCP Server with filter example

```
// add filter to filterList
List<ConnectableFilter> filterList = new ArrayList<>();
filterList.add(filter);

// launch server with filter
int port = 1200;
TCPServer server = new TCPServer(port, null, filterList);
server.connect();
// receive msg with 2 seconds timeout
byte[] msg = server.getNextMsg(2000, TimeUnit.MILLISECONDS);
// server disconnect
server.disconnect();
```

30.4 Server with message parser (fused message parser)

TCP does not have concept of fixed size packets like UDP. If two or more byte-arrays are sent at the same time, TCP protocol can concatenate(fuse) them (TCP guarantees to maintain order) and send it to make transfer efficient. Due to this behavior, at receiver end user may have to implement logic which can de-serialise message according to their specification.

TCPServer allows user to supply de-serialising logic so prior to populating messages to queue, messages can be separated from fused byte arrays. If filter object is supplied then filtering will be processed after message de-serialising. ARTOS will record messages to realtime log file prior to de-serialisation so performance measurement does not have any impact on time stamp.

Note: Implementation of de-serialisation method may impact performance of receiver thread so user should keep implementation simple and light.

Below Example de-serialises concatenated messages with following specification:

```
>>> First four bytes (Big Endian) as payload length excluding length bytes + data
Example Message: "00 00 00 04 11 22 33 44"
Length: "00 00 00 04"
Data: "11 22 33 44"
```

User can construct similar class which implements `ConnectableMessageParser` to de-serialise concatenated messages. Below example de-serialise concatenated messages and construct list of messages according to specification. If any bytes are left over then those bytes are handed back to receiver thread.

Listing 30.5: : Message parser example

```
1  public class MsgParser4ByteLength implements ConnectableMessageParser {
2          Transform _transform = new Transform();
3          byte[] leftOverBytes = null;
4          List<byte[]> msgList = null;
5
6          @Override
7          public byte[] getLeftOverBytes() {
8                  return leftOverBytes;
9          }
10
11         @Override
12         public List<byte[]> parse(byte[] data) {
13                 // reset variable before use
14                 msgList = new ArrayList<>();
15                 leftOverBytes = null;
16
17                 deserializeMsg(data);
18
19                 return msgList;
20         }
21
22         private void deserializeMsg(byte[] data) {
23                 // Check if at least length can be worked out
24                 if (!sufficientDataForLengthCalc(data)) {
25                         leftOverBytes = data;
26                         return;
27                 }
28
29                 // Check if message can be constructed
30                 if (!sufficientDataForMsg(data)) {
31                         leftOverBytes = data;
32                         return;
33                 }
34
35                 // Extract one complete message
36                 byte[] leftOvers = extractMsg(data);
37
38         // process leftOver bytes to see if anymore messages can be extracted
39                 if (null != leftOvers) {
40                         deserializeMsg(leftOvers);
```

(continues on next page)

(continued from previous page)

```
                }
        }

        // Extract complete message inclusive of 4 bytes of length
        private byte[] extractMsg(byte[] data) {
                int length = _transform.bytesToInteger(Arrays.copyOfRange(data, 0, 4), ByteOrder.BIG_ENDIAN);

        // if complete message is found then add to message list.
                msgList.add(Arrays.copyOfRange(data, 0, 4 + length));

        // Return leftover bytes after extracting one complete message
                if (data.length > 4 + length) {
                        return Arrays.copyOfRange(data, 4 + length, data.length);
                }
                return null;
        }

        // Returns true if atleast 4 bytes are present to calculate length of the data
        private boolean sufficientDataForLengthCalc(byte[] data) {
                if (data.length < 4) {
                        return false;
                }
                return true;
        }

        // Returns true if enough bytes are present to construct one complete message
        private boolean sufficientDataForMsg(byte[] data) {
                int length = _transform.bytesToInteger(Arrays.copyOfRange(data, 0, 4), ByteOrder.BIG_ENDIAN);
                if (data.length < 4 + length) {
                        return false;
                }
                return true;
        }

}
```

Below example will launch server with message parser designed to de-serialise concatenated messages for provided specification.

Listing 30.6: : TCPServer with message parser example

```
// create msg parser object
MsgParser4ByteLength msgParser = new MsgParser4ByteLength();

// launch server with message parser
int port = 1200;
TCPServer server = new TCPServer(port, msgParser, null);
server.connect();
// receive msg with 2 seconds timeout
byte[] msg = server.getNextMsg(2000, TimeUnit.MILLISECONDS);
// server disconnect
server.disconnect();
```

30.5 Server real-time log

- This interface allows user to listen server send/receive events and can log sent/received byte arrays real-time.
- User can create their own listener by implementing `RealTimeLoggable` interface and can process events differently.
- User is allowed register more than one listener at a time.

Note: Implementation of event listener may impact performance of sender and receiver thread so user should keep implementation simple and light.

Below code explains how to enable real time log using inbuilt listener. Once enabled, user will see all send receive log bytes are added to real-time log file with time stamp.

Listing 30.7: : RealTime Event Listener example

```
TCPServer server = new TCPServer(1300);
RealTimeLogEventListener realTimeListener = new RealTimeLogEventListener(context);
server.setRealTimeListener(realTimeListener);
server.connect();
```

CHAPTER 31

Visual Regression

A visual regression can be applied to front-end, user interface(UI) or embedded products with display rendering capability. A visual regression tool performs front-end, user interface(UI) or embedded product regression testing by capturing the screen-shots of web pages/UI/Frame Buffer and compare them with the gold sample images (either historical screen-shots or reference images that can be trusted). If a new image is not 100% match then visual regression tool can alert you and/or provide percentile of the match and/or return a result/diff image highlighting the areas where changes have been detected. User can use that information for record purpose so appropriate action can be triggered.

Many organization already use tools like Selenium, Cypress, WebdriverIO for an end to end testing, so image capturing capabilities are already inbuilt. In the context of embedded product testing, the product shall have an API that lets user capture a frame-buffer at run time. Once you have an image then only step required is to have a utility that compares images and provide diff between two images and percentage of mismatch.

Understanding the above requirements, Artos has taken a generic approach to cater to all kinds of testing. Artos provides inbuilt utilities which let you do the following.

- Compare PNG or JPG images pixel by pixel and store result image to desired location.
- Request match percentage after image compare.
- Request result/diff image after image compare.
- Record images/snapshots to the Extent report.

Listing 31.1: : Sample code to demonstrate visual regression functionality

```
1  package com.tests.selenium;
2
3  import java.io.File;
4  import java.util.concurrent.TimeUnit;
5
6  import org.openqa.selenium.OutputType;
7  import org.openqa.selenium.TakesScreenshot;
8  import org.openqa.selenium.WebDriver;
9  import org.openqa.selenium.firefox.FirefoxDriver;
10
11 import com.artos.annotation.*;
12 import com.artos.framework.Enums.TestStatus;
13 import com.artos.framework.infra.TestContext;
```

(continues on next page)

(continued from previous page)

```
14 import com.artos.interfaces.TestExecutable;
15 import com.artos.utils.Guard;
16 import com.artos.utils.ImageCompare;
17 import com.artos.utils.UtilsFile;
18
19 @TestPlan(preparedBy = "arpit", preparationDate = "25/08/2019", bdd = "GIVEN␣
   ↪webpage is not modified since last regression then visual regression should pass
   ↪")
20 @TestCase
21 public class Sample_Selenium implements TestExecutable {
22
23     @Unit
24     public void testUnit_1(TestContext context) throws Exception {
25         // ------------------------------------------------------
26         WebDriver fireFoxDriver = (WebDriver) context.getGlobalObject("FIREFOX_
   ↪DRIVER");
27         fireFoxDriver.navigate().to("https://www.theartos.com");
28         Thread.sleep(8000);
29
30         // Take snapshot using Selenium
31         String relativePathToImage = "./reporting/testImage.png";
32         File destFile = takeSnapShot(fireFoxDriver, relativePathToImage);
33
34         // Store snapshot to the report file
35         context.setTestStatus(TestStatus.PASS, destFile, "Managed to display page␣
   ↪successfully");
36
37         File goldSample = new File("./goldsamples/test.png");
38         ImageCompare ic = new ImageCompare();
39         ic.compare(goldSample, destFile, new File("./reporting"), "Result_Image");
40         File resultFile = ic.getResultImage();
41
42         // Store result image to report
43         context.setTestStatus(TestStatus.PASS, resultFile, "Result Image is stored␣
   ↪for user reference");
44         // if image match is not 100% then throw an exception
45         Guard.guardEquals(100, ic.getPercentageMatch());
46         // ------------------------------------------------------
47     }
48
49     @BeforeTestUnit
50     public void beforeTest(TestContext context) {
51         System.setProperty("webdriver.gecko.driver", "./assets/driver/geckodriver_
   ↪64bit.exe");
52         WebDriver firefoxDriver = new FirefoxDriver();
53         firefoxDriver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
54         context.setGlobalObject("FIREFOX_DRIVER", firefoxDriver);
55     }
56
57     @AfterTestUnit
58     public void tearDown(TestContext context) {
59         WebDriver fireFoxDriver = (WebDriver) context.getGlobalObject("FIREFOX_
   ↪DRIVER");
60         fireFoxDriver.quit();
61
62     }
63
64     public static File takeSnapShot(WebDriver webdriver, String fileWithPath)␣
   ↪throws Exception {
65
66         // Convert web driver object to TakeScreenshot
```

(continues on next page)

(continued from previous page)

```
67          TakesScreenshot scrShot = ((TakesScreenshot) webdriver);
68
69          // Call getScreenshotAs method to create image file
70          File srcFile = scrShot.getScreenshotAs(OutputType.FILE);
71
72          // Move image file to new destination
73          File destFile = new File(fileWithPath);
74
75          // Copy file at destination
76          UtilsFile.copyFile(srcFile, destFile, true);
77
78          return destFile;
79      }
80
81  }
```

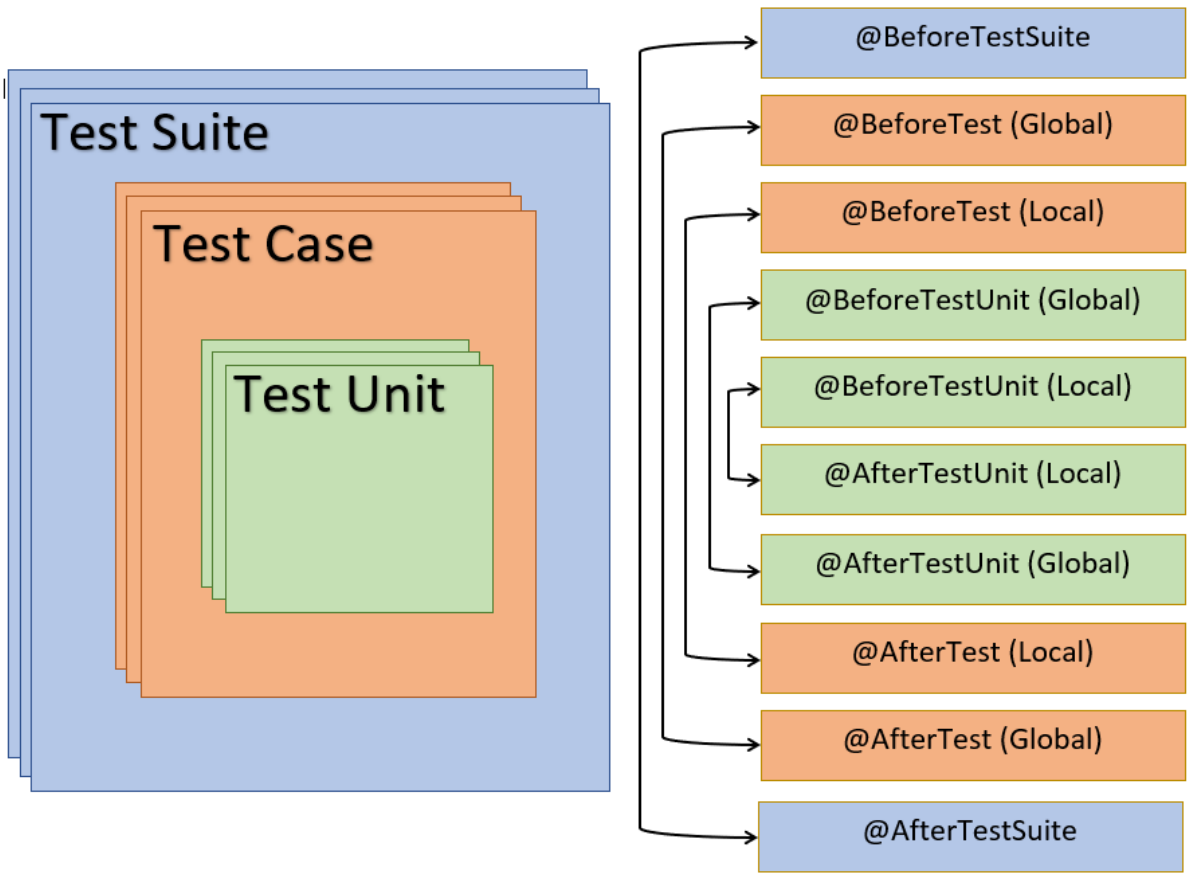
31.1 Original Image capture

31.2 Modified Image capture

31.3 Diff => Original and Modified Images

CHAPTER 32

@BeforeTestUnit @AfterTestUnit



32.1 @BeforeTestUnit

Method marked with annotation `@BeforeTestUnit` is executed in different order depending on where it is implemented. All possible combinations are listed below:

Implementation	Execution sequence
Inside a Runner	Invoked before each test units **within a test suite**.
Inside a Test-Case	Invoked before each test units **within a test case**.
Inside a Runner and a Test-Case	Method implemented within the Runner class is invoked before each test units **within a test suite** and the method implemented in the test case will be invoked before each test units **within a test case**. Method implemented in the Runner class will execute before method implemented in the test case.

32.2 @AfterTestUnit

Method marked with annotation `@AfterTestUnit` is executed in different order depending on where it is implemented. All possible combinations are listed below:

Implementation	Execution sequence
Inside a Runner	Invoked after each test units **within a test suite**.
Inside a Test-Case	Invoked after each test units **within a test case**.
Inside a Runner and a Test-Case	Method implemented within the Runner class is invoked after each test units **within a test suite** and the method implemented in the test case will be invoked after each test units **within a test case**. Method implemented in the Runner class will execute after method implemented in the test case.

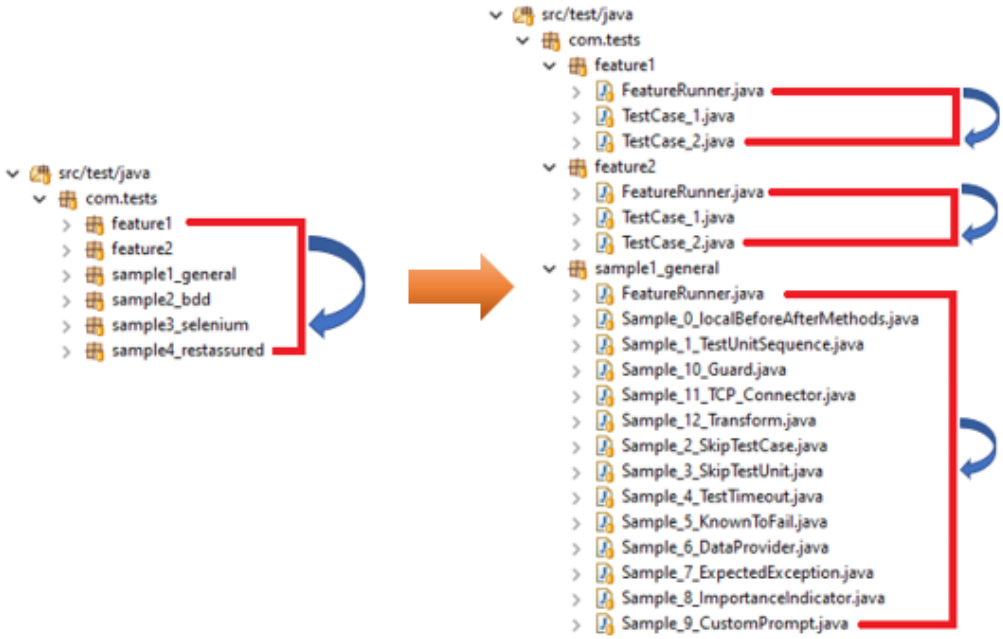
CHAPTER 33

@TestCase

Annotation @TestCase is used to mark java class as a test case.

Attribute	Description	Mandatory/Optional	Default Value
skip()	Skip or Keep	Optional	false
sequence()	Test sequence number	Optional	0
bugref()	Bug rReference	Optional	Empty String
dropRemainingTestsOnFailure()	Trigger to drop remaining test cases from the execution list	Optional	false

- **skip()**
  - Removes test case from execution list, skipped test case will not appear in GUI test selector.
  - Skip attribute will be applied regardless of test execution method (test list, test script or test scanning).
- **sequence()**
  - Provides sequence number to a test case.
  - Test case(s) are assigned sequence number '0' if no sequence number is specified by the user.
  - Sequence number is ignored in case of test sequence is provided by the user (via test script or test list).
  - In absence of user provided test sequence (empty test list in the test-script or empty/null test list), test case execution sequence will be decided by first sorting packages by name in ascending order and secondly bubble sorting test cases using sequence number within their respective packages.
  - Test cases are sorted using bubble sort mechanism so any test case(s) (within same package) with same sequence number will be arranged as per their scan order, thus between them order of execution cannot be guaranteed.



- **bugref()**
  - User can define bug reference (up to 20 bytes long)
- **dropRemainingTestsUponFailure()**
  - Enables dropping of remaining test cases from execution list if annotated test case fails.

33.1 Annotation use case(s)

```
@TestCase(skip = false, sequence = 1, bugref = "JIRA-1234, JIRA-234",␣
↪dropRemainingTestsUponFailure = true)
```

33.2 Example test case

```
import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

@TestCase(skip = false, sequence = 1, bugref = "JIRA-1234, JIRA-234",␣
↪dropRemainingTestsUponFailure = true)
public class TestCase_1 implements TestExecutable {

}
```

CHAPTER 34

@Unit

Annotation @Unit is used to mark Java methods as a test case.

Attribute	Description	Mandatory/Optional	Default Value
skip()	Skip or Keep	Optional	false
sequence()	Test sequence number	Optional	0
dataprovider()	Data provider method Name	Optional	Empty String
testtimeout()	Test timeout in milliseconds	Optional	0
bugref()	Bug reference	Optional	Empty String
dropRemainingUnitsOnFailure()	Trigger to drop remaining test cases from the execution list	Optional	false

- **skip()**
  - Removes test case from execution list, skipped test case will not appear in GUI test selector.
  - Skip attribute will be applied regardless of test execution method (test list, test script or test scanning).
- **sequence()**
  - Provides sequence number to a test case.
  - Test case(s) are assigned sequence number '0' if no sequence number is specified by the user.
  - Sequence number is ignored in case of test sequence is provided by the user (via test script or test list).
  - In absence of user provided test sequence (empty test list in the test-script or empty/null test list), test case execution sequence will be decided by first sorting packages by name in ascending order and secondly bubble sorting test cases using sequence number within their respective packages.
  - Test cases are sorted using bubble sort mechanism so any test case(s) (within same package) with same sequence number will be arranged as per their scan order, thus between them order of execution cannot be guaranteed.
- **dataprovider()**
  - Used to specify data provider method name which provides 2D data array in return.
  - Test case is repeatedly executed until all data from the array is applied.
  - Data provider method name is case in-sensitive.

  - If mentioned method is not found or not visible or not valid then test execution will stop prior to test suite launch.

- **testtimeout()**
  - Used to set test execution timeout.
  - Test case will be marked failed if test execution takes longer than specified time.
  - 0 timeout means infinite timeout.
  - timeout is in milliseconds.
- **bugref()**
  - User can define bug reference (up to 20 bytes long)
- **dropRemainingUnitsUponFailure()**
  - Enables dropping of remaining test units from execution list if annotated unit fails.

34.1 Annotation use case(s)

```
@Unit(skip = false, sequence = 1, bugref = "JIRA-1234, JIRA-234", dataprovider =
↪"username", testtimeout = 5000, dropRemainingUnitsUponFailure = true)
```

34.2 Example test unit

```
import com.artos.annotation.TestCase;
import com.artos.annotation.Unit;
import com.artos.annotation.TestPlan;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

@TestCase(sequence = 1)
public class TestCase_1 implements TestExecutable {

        @Unit(skip = false, sequence = 1, bugref = "JIRA-1234, JIRA-234",␣
↪dataprovider = "username", testtimeout = 5000, dropRemainingUnitsUponFailure =␣
↪true)
        public void testUnit_1(TestContext context) {

        }

}
```

CHAPTER 35

@TestPlan

Annotation @TestPlan is used to describe short BDD (Behavior Driven Development) or Simple text styled test plan. If attribute "bdd" is specified by user then bdd text is formatted and then printed in the log file during test execution. This annotation encourages user to maintain test plan within a test case.

Attribute	Description	Mandatory/Optional	Default Value
description()	Short description	Optional	Empty String
preparedBy()	Test developer/engineer name	Optional	Empty String
preparationDate()	Test preparation date	Optional	Empty String
reviewedBy()	Reviewer name	Optional	Empty String
reviewDate()	Review date	Optional	Empty String
bdd()	BDD style test plan	Optional	Empty String

35.1 Annotation use cases

```
@TestPlan(description = "Automated Test Case", preparedBy = "ArpitS",␣
↪preparationDate = "1/1/2018", reviewedBy = "ArpitS", reviewDate = "1/2/2018",␣
↪bdd = "GIVEN..AND..WHEN..THEN..")
```

35.2 Example test case

```
import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.annotation.Unit;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

@TestPlan(preparedBy = "arpit", preparationDate = "1/1/2018", bdd = "given test␣
↪project is set correctly and logger is used to log HELLO WORLD string then hello␣
↪world should be printed correctly")
@TestCase(skip = false, sequence = 0)
public class TestCase_1 implements TestExecutable {

```

(continues on next page)

(continued from previous page)

```
        @Unit
        public void unit_test(TestContext context) throws Exception {
                // ------------------------------------------------------
                context.getLogger().debug("Observe formatted test plan in logs");
                // ------------------------------------------------------
        }
}
```

- Log file snapshot for above test case.

```
*************************************************************************
Test Name        : com.tests.TestCase_1
Written BY       : ArpitS
Date             : 1/1/2018
BDD Test Plan    :
GIVEN test project is set correctly
AND logger is used to log HELLO WORLD string
THEN hello world should be printed correctly
*************************************************************************
Observe formatted test plan in the log file

[PASS] : unit_test()


Test Result : PASS
```

CHAPTER 36

@ExpectedException

Annotation @ExpectedException is used to manage an exception during test where user must specify at lease one exception. User can optionally provide exception message/description either as a string or regular expression to deal with complex scenarios.

Note: String specified in "contains" attribute must be 100% match with exception message/description inclusive of non-printable characters. For partial or dynamic string matching, use regular expression.

Attribute	Description	Mandatory/Optional	Default Value
expectedExceptions()	One or more exception classes	Mandatory	NA
contains()	String or regular expression	Optional	Empty String
enforce()	Enforce exception checking	Optional	true

36.1 Test combinations and expected outcome

expectedExceptions()	contains()	enforce()	Test Exception	Outcome
specified	default	true	exception match	PASS
specified	specified	true	exception + description match	PASS
specified	default	true	exception miss-match	FAIL
specified	specified	true	exception/description miss-match	FAIL
specified	default	true	no exception	FAIL
specified	specified	true	no exception	FAIL
specified	default	false	exception match	PASS
specified	specified	false	exception + description match	PASS
specified	default	false	exception miss-match	FAIL
specified	specified	false	exception + description miss-match	FAIL
specified	default	false	no exception	PASS
specified	specified	false	no exception	PASS

36.2 Annotation use cases

```
// Single exception comparison
@ExpectedException(expectedExceptions = { NullPointerException.class })
// Single exception + exception message string comparison
@ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class }, contains = "exception example")
// Single exception + exception message matching with regular expression
@ExpectedException(expectedExceptions = { NullPointerException.class }, contains =
↪"[^0-9]*[12]?[0-9]{1,2}[^0-9]*")

// Multiple exception comparison
@ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class })
// Multiple exception + exception message string comparison
@ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class }, contains = "exception example")
// Multiple exception + exception message matching with regular expression
@ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class }, contains = "[^0-9]*[12]?[0-9]{1,2}[^0-9]*")
```

36.3 Example usage

```
@TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "GIVEN..WHEN..
↪AND..THEN..")
@TestCase(sequence = 1)
public class Sample_ExpectedException implements TestExecutable {


        // Code demonstrates how to specify single expected exception class
        // Test Unit execution will be terminated as soon as exception is
        // thrown and next test unit will be run. If Exception is not as
        // expected or exception did not occur then test unit will be
        // marked as a FAIL
        @Unit(sequence = 1)
        @ExpectedException(expectedExceptions = { NumberFormatException.class })
        public void testUnit_1(TestContext context) {
                // -------------------------------------------------
                // Converting String into Integer should throw an error
                Integer.parseInt("Test");
                // -------------------------------------------------
        }

        // Code demonstrates how to specify multiple expected exception
        // classes. Test Unit execution will be terminated as soon as
        // exception is thrown and next test unit will be run.
        // If Exception is not as expected or exception did not occur
        // then test unit will be marked as a FAIL
        @Unit(sequence = 2)
        @ExpectedException(expectedExceptions = { Exception.class,␣
↪NumberFormatException.class })
        public void testUnit_2(TestContext context) {
                // -------------------------------------------------
                // Converting String into Integer should throw an error
                Integer.parseInt("Test");
                // -------------------------------------------------
        }

        // Code demonstrates how to specify multiple expected exception
```

(continues on next page)

(continued from previous page)

```
35          // classes and description. Test Unit execution will be terminated
36          // as soon as exception is thrown and next test unit will be run.
37          // If Exception is not as expected or exception did not occur then
38          // test unit will be marked as a FAIL
39          @Unit(sequence = 3)
40          @ExpectedException(expectedExceptions = { Exception.class,␣
    ↪NumberFormatException.class }, contains = "This is a test code")
41          public void testUnit_3(TestContext context) throws Exception {
42                  // -------------------------------------------------
43                  // test logic goes here..
44                  throw new Exception("This is a test code");
45                  // -------------------------------------------------
46          }
47
48          // Code demonstrates how to specify multiple expected exception
49          // classes and description using Regular expression. Test Unit
50          // execution will be terminated as soon as exception is thrown
51          // and next test unit will be run. If Exception is not as
52          // expected or exception did not occur then test unit will be
53          // marked as a FAIL
54          @Unit(sequence = 4)
55          @ExpectedException(expectedExceptions = { Exception.class,␣
    ↪NumberFormatException.class }, contains = ".*\\btest\\b.*")
56          public void testUnit_4(TestContext context) throws Exception {
57                  // -------------------------------------------------
58                  // test logic goes here..
59                  throw new Exception("This is a test code");
60                  // -------------------------------------------------
61          }
62
63          // Code demonstrates how to specify exception but do not enforce
64          // fail in absence of exception. If Exception is thrown then it
65          // will be matched with expectedException class. If Exception
66          // will not be thrown then test will continue execution and PASS
67          // eventually
68          @Unit(sequence = 5)
69          @ExpectedException(enforce = false, expectedExceptions = { Exception.class,
    ↪ NumberFormatException.class })
70          public void testUnit_5(TestContext context) throws Exception {
71                  // -------------------------------------------------
72                  // test logic goes here..
73                  context.getLogger().info("This test does not throw any exception");
74                  // -------------------------------------------------
75          }
76
77          // This will allow user to continue executing rest of the code
78          // in case of exception. Guarding against wrong flow will help
79          // user throw exception in case code did not do what was expected
80          @Unit(sequence = 6)
81          public void testUnit_6(TestContext context) throws Exception {
82                  // -------------------------------------------------
83                  try {
84                          // Converting String into Integer should throw an error
85                          Integer.parseInt("Test");
86
87                          // Protects against code traveling in wrong direction
88                          Guard.guardWrongFlow("Expected exception but did not occur
    ↪");
89
90                  } catch (NumberFormatException e) {
91                          if (!e.getMessage().equals("For input string: \"Test\"")) {
```

(continues on next page)

(continued from previous page)

```
92                                  throw e;
93                              }
94                  }
95
96                  context.getLogger().info("Do something..");
97                  // logic goes here..
98                  // ------------------------------------------------
99          }
100 }
```

CHAPTER 37

@TestDependency

Annotation @TestDependency is used to declare test case dependency on other test classes. One or more test cases can be declared as a dependency. If dependency is declared and any of the following requirements are **not met** then test case execution will be skipped.

Requirements:

- Declared dependency test case must be within a scan scope.
- Declared dependency test case execution must be completed before declaring test case execution.
- Declared dependency test case outcome must be **PASS**.

Warning will be printed when test case execution is skipped due to dependency requirements not met.

```
========================================================================
================ DEPENDENCY REQUIREMENTS ARE NOT MET ====================
====================== TEST CASE WILL BE SKIPPED ========================
========================================================================
```

Attribute	Description	Mandatory/Optional	Default Value
dependency()	Array of Test Cases	Mandatory	N/A

37.1 Annotation use case(s)

```
@TestDependency(dependency = { TestCase_1.class, TestCase_2.class })
```

37.2 Example test case

```
@TestDependency(dependency = { TestCase_1.class, TestCase_2.class })
@TestCase(skip = false, sequence = 1, bugref = "JIRA-1234, JIRA-234",␣
↪dropRemainingTestsUponFailure = true)
public class TestCase_1 implements TestExecutable {

}
```

CHAPTER 38

@UnitDependency

Annotation @UnitDependency is used to declare unit dependency on other units withing same test case. One or more unit name can be declared as a dependency. If dependency is declared and any of the following requirements are **not met** then unit execution will be skipped.

Requirements:

- Declared dependency unit(s) must be within a same test case.
- Declared dependency unit(s) execution must be completed before declaring unit execution.
- Declared dependency unit(s) outcome must be **PASS**.

Warning will be printed when unit execution is skipped due to dependency requirements not met.

```
================================================================================
================ DEPENDENCY REQUIREMENTS ARE NOT MET ===================
====================== TEST UNIT WILL BE SKIPPED ========================
================================================================================
```

Attribute	Description	Mandatory/Optional	Default Value
dependency()	Array of method name	Mandatory	N/A

38.1 Annotation use case(s)

```
@UnitDependency(dependency = { "TestUnit_1", "TestCase_2" })
```

38.2 Example test case

```
@TestPlan(preparedBy = "user", preparationDate = "19/02/2019", bdd = "GIVEN..WHEN..
↪AND..THEN..")
@TestCase(sequence = 1)
public class TestCase_1 implements TestExecutable {

@Unit(sequence = 1)
```

(continues on next page)

(continued from previous page)

```
6  public void testUnit_1(TestContext context) {
7          // ------------------------------------------------------------------------
   ↪--------------------
8          context.getLogger().info("do something");
9          // ------------------------------------------------------------------------
   ↪--------------------
10 }
11
12 @Unit(sequence = 2)
13 public void testUnit_2(TestContext context) {
14         // ------------------------------------------------------------------------
   ↪--------------------
15         context.getLogger().info("do something");
16         // ------------------------------------------------------------------------
   ↪--------------------
17 }
18
19 @UnitDependency(dependency = { "testUnit_1", "testUnit_2" })
20 @Unit(sequence = 3)
21 public void testUnit_3(TestContext context) {
22         // ------------------------------------------------------------------------
   ↪--------------------
23         context.getLogger().info("do something");
24         // ------------------------------------------------------------------------
   ↪--------------------
25 }
```

}

CHAPTER 39

@DataProvider

DataProvider is used when same code(logic) is repeatedly executed with fix set of data. Data could be primitive or it can be complex object.

Annotation @DataProvider is used to denote a Java method as a DataProvider. Method annotated with @DataProvider can be within a test case class or can be in separate class as long as it is within Runner's scan scope. Warning will be printed if two data provider have duplicate name/descriptions or data provider is missing.

Attribute	Description	Mandatory/Optional	Default Value
name()	name/description	Mandatory	NA

- **name()**
  - name/description is used to bind data provider to a test unit.
  - name/description is case insensitive.

39.1 Annotation use case(s)

```
@DataProvider(name = "UsernameAndPassword")
```

39.2 DataProvider Samples

Data provider could be simple or complex 2D object.

```
public class Collection_Of_DataProvider {

        @DataProvider(name = "UsernameAndPassword")
        public Object[][] dataproviderMethod_1(TestContext context) {

                Object[][] data = new Object[][] {
                                // {User-name, Password}
                                {"Joe", "1234"},
                                {"Sam", "2345"},
                                {"Mark", "1453"},
```

(continues on next page)

(continued from previous page)

```
                                {"Maya", "1458"},

                };
                return data;
        }

        @DataProvider(name = "UsernameAndStudentDetails")
        public Object[][] dataproviderMethod_1(TestContext context) {

                Object[][] data = new Object[][] {
                                // {Name, id, phone, marks, grade}
                                { "Joe", new String[] { "1", "0210123456", "20", "F" } },
                                { "Sam", new String[] { "2", "0210123457", "40", "D" } },
                                { "Mark", new String[] { "3", "0210123458", "70", "C" } },
                                { "Maya", new String[] { "4", "0210123459", "90", "A" } },
                };
                return data;
        }

        @DataProvider(name = "UsernameAndStudentDetailsPerClass")
        public Object[][] dataproviderMethod_2(TestContext context) {

                Classroom classroom = (Classroom) context.getGlobalObject("CLASSROOM_ENUM");
                switch (classroom) {
                case CLASS_A:
                        return new Object[][] {
                                // {Name, id, phone, marks, grade}
                                {"Joe", new String[] { "1", "0210123456", "20", "F" }},
                                {"Sam", new String[] { "2", "0210123457", "40", "D" }},
                        };
                case CLASS_B:
                        return new Object[][] {
                                // {Name, id, phone, marks, grade}
                                {"Mark", new String[] { "3", "0210123458", "70", "C" }},
                                {"Maya", new String[] { "4", "0210123459", "90", "A" }},
                        };
                default:
                        return null;
                }

        }

}

class CLASSROOM {
        public enum Classroom {
                CLASS_A,
                CLASS_B
        }
}
```

39.3 Bind DataProvider to unit

```
@TestCase(sequence = 1)
public class Test_Student_Grade implements TestExecutable {

        @Unit(sequence = 1, dataprovider = "UsernameAndPassword")
        public void testUnit_1(TestContext context) throws Exception {
                // ------------------------------------------------------------------
                String username = (String) context.getParameterisedObject1();
                String password = (String) context.getParameterisedObject2();

                context.getLogger().debug(username + " : " + password);
                // ------------------------------------------------------------------
        }

        @Unit(sequence = 2, dataprovider = "UsernameAndStudentDetails")
        public void testUnit_2(TestContext context) throws Exception {
                // ------------------------------------------------------------------
                String username = (String) context.getParameterisedObject1();
                String[] userDetails = (String[]) context.getParameterisedObject2();

                context.getLogger().debug(username + " : " + userDetails[0] + " ," + userDetails[1]);
                // ------------------------------------------------------------------
        }

        @Unit(sequence = 3)
        public void testUnit_3(TestContext context) throws Exception {
                // ------------------------------------------------------------------
                // Set global variable so it can be used in unit 4
                context.setGlobalObject("CLASSROOM_ENUM", Classroom.CLASS_A);
                // ------------------------------------------------------------------
        }

        @Unit(sequence = 4, dataprovider = "UsernameAndStudentDetailsPerClass")
        public void testUnit_4(TestContext context) throws Exception {
                // ------------------------------------------------------------------
                String username = (String) context.getParameterisedObject1();
                String[] userDetails = (String[]) context.getParameterisedObject2();

                context.getLogger().debug(username + " : " + userDetails[0] + " ," + userDetails[1]);
                // ------------------------------------------------------------------
        }
}
```

CHAPTER 40

Report Portal Integration with Artos

Note

Based on this example you can create your own integration with a Artos test framework. We can provide a support for integration with a new framework or supporting exsisting one. If you need more details, please email on artos.framework@gmail.com

Recommended

- This example is recommended to be used with Artos build 0.0.14 or above

40.1 Preparation

- Go to reportportal-client
- Download latest jar or copy Maven dependency signature for Maven project.
- Add jar to test project build path. If maven test project then add Maven dependency to POM file.

```
<!-- https://mvnrepository.com/artifact/com.theartos/reportportal-client -->
<dependency>
        <groupId>com.theartos</groupId>
        <artifactId>reportportal-client</artifactId>
        <version>0.0.1</version>
</dependency>
```

- Re-build the Maven project to ensure all required dependency are downloaded to local repository.
- Create **conf** directory to project root (./conf) if not already present.
- Create **reportportal_configuration.xml** file inside conf directory (./conf/reportportal_configuration.xml).
- Add following details to **reportportal_configuration.xml** file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
  <reportportal_info>
        <property name="TestSuite_Name">TEST_SUITE_1</property>
        <property name="Project_Name">RPT_PRJ_1</property>
        <property name="Release_Name">01.02.0002</property>
        <property name="Base_URL">http://192.168.1.26:8080</property>
        <property name="UUID">0acb493c-37d2-4afc-a029-3c7aad01fb78</property>
  </reportportal_info>
</configuration>
```

- Update xml values to match your local reportportal configuration.
- Save and close the file.

40.2 Create Listener for Artos

- Create **ReportPortalListener.java** class inside test project.
- Copy and paste below code to your local project
- Fix package location as per your project

```
/
*******************************************************************************
 * Copyright (C) 2018-2020 Arpit Shah and Artos Contributors
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 *
*******************************************************************************/
package listener;

import java.io.File;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

```

(continues on next page)

(continued from previous page)

```
32      import com.artos.framework.Enums.TestStatus;
33      import com.artos.framework.infra.BDDScenario;
34      import com.artos.framework.infra.BDDStep;
35      import com.artos.framework.infra.TestObjectWrapper;
36      import com.artos.framework.infra.TestUnitObjectWrapper;
37      import com.artos.interfaces.TestProgress;
38      import com.google.common.base.Throwables;
39      import com.theartos.ReportPortalLauncher;
40      import com.theartos.ReportPortalLauncher.LogStatus;
41      import com.theartos.ReportPortalLauncher.Status;
42
43      public class ReportPortalListener implements TestProgress {
44
45              ReportPortalLauncher rpl;
46              boolean activeTest = false;
47              boolean activeChildUnit = false;
48              boolean debug = false;
49
50              // ************************************************************
51              // Common functionality between BDD or Non-BDD testcase/testunits
52              // ************************************************************
53
54              @Override
55              public void testSuiteExecutionStarted(String description) {
56
57                      if (debug) {
58                              System.err.println("testSuiteExecutionStarted(String␣
    ↪description)");
59                      }
60
61                      rpl = new ReportPortalLauncher();
62                      rpl.StartLaunch();
63
64                      String TestSuiteName = "TestSuite"; // Default value
65
66                      // Find Test Suite Name
67                      if (null != description) {
68                              String[] desc = description.split("\\.");
69                              int descLength = desc.length;
70                              if (descLength > 2) {
71                                      TestSuiteName = desc[descLength - 2];
72                              } else {
73                                      TestSuiteName = description;
74                              }
75                      }
76
77                      rpl.StartSuite("TestSuite: " + TestSuiteName, description);
78                      try {
79                              Thread.sleep(500);
80                      } catch (InterruptedException e) {
81                              e.printStackTrace();
82                      }
83              }
84
85              @Override
86              public void testSuiteExecutionFinished(String description) {
87
88                      if (debug) {
89                              System.err.println("testSuiteExecutionFinished(String␣
    ↪description)");
90                      }
```

(continues on next page)

(continued from previous page)

```
91
92                      rpl.endSuite();
93                      rpl.endLaunch();
94
95                      // Give some time for API to finish its communication
96                      try {
97                              Thread.sleep(3000);
98                      } catch (InterruptedException e) {
99                              e.printStackTrace();
100                     }
101             }
102
103             @Override
104             public void testException(Throwable e) {
105                     rpl.log(LogStatus.Error, Throwables.getStackTraceAsString(e));
106             }
107
108             @Override
109             public void unitException(Throwable e) {
110                     rpl.log(LogStatus.Error, Throwables.getStackTraceAsString(e));
111             }
112
113             @Override
114             public void testCaseSummaryPrinting(String FQCN, String description) {
115                     if (debug) {
116                             System.err.println("testCaseSummaryPrinting(String␣
    ↪FQCN, String description)");
117                     }
118             }
119
120             @Override
121             public void testUnitSummaryPrinting(String FQCN, String description) {
122                     if (debug) {
123                             System.err.println("testUnitSummaryPrinting(String␣
    ↪FQCN, String description)");
124                     }
125             }
126
127             @Override
128             public void testSuiteSummaryPrinting(String description) {
129                     if (debug) {
130                             System.err.println("testSuiteSummaryPrinting(String␣
    ↪description)");
131                     }
132             }
133
134             @Override
135             public void testExecutionLoopCount(int count) {
136                     // TODO Auto-generated method stub
137
138             }
139
140             @Override
141             public void testCaseExecutionSkipped(TestObjectWrapper t) {
142                     // TODO Auto-generated method stub
143
144             }
145
146             @Override
147             public void testCaseExecutionFinished(TestObjectWrapper t) {
148
```

(continues on next page)

(continued from previous page)

```
149             }
150
151             @Override
152             public void testSuiteFailureHighlight(String description) {
153                     // TODO Auto-generated method stub
154
155             }
156
157             @Override
158             public void testSuiteException(Throwable e) {
159                     // TODO Auto-generated method stub
160
161             }
162
163             // ******************************************************
164             // Non-BDD Test Cases
165             // ******************************************************
166             @Override
167             public void testCaseExecutionStarted(TestObjectWrapper t) {
168
169                     if (debug) {
170                             System.err.println(
↪"testCaseExecutionStarted(TestObjectWrapper t)");
171                     }
172
173                     Set<String> tags = new HashSet<String>();
174                     for (String s : t.getGroupList()) {
175                             tags.add(s);
176                     }
177                     // Start next test
178                     rpl.StartTest(t.getTestClassObject().getName(),
179                                     "".equals(t.getTestPlanDescription().trim()) ?
↪ t.getTestPlanBDD() : t.getTestPlanDescription(), tags);
180                     activeTest = true;
181             }
182
183             @Override
184             public void testResult(TestObjectWrapper t, TestStatus testStatus,␣
↪File snapshot, String description) {
185
186                     if (debug) {
187                             System.err.println(
188                                             "testResult(TestObjectWrapper t,␣
↪TestStatus testStatus, File snapshot, String description)");
189                     }
190
191                     if (testStatus == TestStatus.PASS || testStatus == TestStatus.
↪KTF) {
192                             rpl.endTest(Status.PASSED);
193                     } else if (testStatus == TestStatus.SKIP) {
194                             rpl.endTest(Status.SKIPPED);
195                     } else if (testStatus == TestStatus.FAIL) {
196                             rpl.endTest(Status.FAILED);
197                     }
198             }
199
200             @Override
201             public void childTestUnitExecutionStarted(TestObjectWrapper t,␣
↪TestUnitObjectWrapper unit, String paramInfo) {
202
203                     if (debug) {
```

(continues on next page)

(continued from previous page)

```
204                             System.err.println(
205
↪"childTestUnitExecutionStarted(TestObjectWrapper t, TestUnitObjectWrapper unit,␣
↪String paramInfo)");
206                     }
207
208                     Set<String> tags = new HashSet<String>();
209                     for (String s : unit.getGroupList()) {
210                             tags.add(s);
211                     }
212                     rpl.StartStep(unit.getTestUnitMethod().getName() + " " +␣
↪paramInfo,
213                                     "".equals(unit.getTestPlanDescription()) ?␣
↪unit.getTestPlanBDD() : unit.getTestPlanDescription(), tags);
214                     activeChildUnit = true;
215             }
216
217             @Override
218             public void testUnitExecutionStarted(TestUnitObjectWrapper unit) {
219
220                     if (debug) {
221                             System.err.println(
↪"testUnitExecutionStarted(TestUnitObjectWrapper unit)");
222                     }
223
224                     if (!activeChildUnit) {
225                             Set<String> tags = new HashSet<String>();
226                             for (String s : unit.getGroupList()) {
227                                     tags.add(s);
228                             }
229                             rpl.StartStep(unit.getTestUnitMethod().getName(),
230                                             "".equals(unit.
↪getTestPlanDescription().trim()) ? unit.getTestPlanBDD()
231                                                             : unit.
↪getTestPlanDescription(),
232                                             tags);
233                     }
234             }
235
236             @Override
237             public void testUnitResult(TestUnitObjectWrapper unit, TestStatus␣
↪testStatus, File snapshot, String description) {
238
239                     if (debug) {
240                             System.err.println(
241                                             "testUnitResult(TestUnitObjectWrapper␣
↪unit, TestStatus testStatus, File snapshot, String description)");
242                     }
243
244                     if (testStatus == TestStatus.PASS || testStatus == TestStatus.
↪KTF) {
245                             rpl.endStep(Status.PASSED);
246                     } else if (testStatus == TestStatus.SKIP) {
247                             rpl.endStep(Status.SKIPPED);
248                     } else if (testStatus == TestStatus.FAIL) {
249                             rpl.endStep(Status.FAILED);
250                     }
251                     activeChildUnit = false;
252             }
253
254             @Override
```

(continues on next page)

(continued from previous page)

```
255             public void testCaseStatusUpdate(TestStatus testStatus, File snapshot,
↪ String msg) {
256
257                     if (debug) {
258                             System.err.println("testCaseStatusUpdate(TestStatus␣
↪testStatus, File snapshot, String msg)");
259                     }
260
261                     if (snapshot == null) {
262                             rpl.log(LogStatus.Info, msg);
263                     } else {
264                             rpl.log(LogStatus.Info, msg, snapshot);
265                     }
266             }
267
268             @Override
269             public void childTestUnitExecutionFinished(TestUnitObjectWrapper␣
↪unit) {
270                     if (debug) {
271                             System.err.println(
↪"childTestUnitExecutionFinished(TestUnitObjectWrapper unit)");
272                     }
273             }
274
275             @Override
276             public void testUnitExecutionFinished(TestUnitObjectWrapper unit) {
277                     if (debug) {
278                             System.err.println(
↪"testUnitExecutionFinished(TestUnitObjectWrapper unit)");
279                     }
280             }
281
282             @Override
283             public void childTestCaseExecutionStarted(TestObjectWrapper t, String␣
↪paramInfo) {
284                     if (debug) {
285                             System.err.println(
↪"childTestCaseExecutionStarted(TestObjectWrapper t, String paramInfo)");
286                     }
287             }
288
289             @Override
290             public void childTestCaseExecutionFinished(TestObjectWrapper t) {
291                     if (debug) {
292                             System.err.println(
↪"childTestCaseExecutionFinished(TestObjectWrapper t)");
293                     }
294
295             }
296
297             @Override
298             public void printTestPlan(TestObjectWrapper t) {
299                     // TODO Auto-generated method stub
300
301             }
302
303             @Override
304             public void printTestUnitPlan(TestUnitObjectWrapper unit) {
305                     // TODO Auto-generated method stub
306
307             }
```

(continues on next page)

(continued from previous page)

```
308
309             // ********************************************************
310             // BDD Test Cases
311             // ********************************************************
312
313             @Override
314             public void testCaseExecutionStarted(BDDScenario scenario) {
315
316                     if (debug) {
317                             System.err.println(
↪"testCaseExecutionStarted(BDDScenario scenario)");
318                     }
319
320                     Set<String> tags = new HashSet<String>();
321                     for (String s : scenario.getGroupList()) {
322                             tags.add(s);
323                     }
324
325                     StringBuilder sb = new StringBuilder();
326                     LinkedHashMap<String, List<String>> globalDatatable =␣
↪scenario.getGlobalDataTable();
327                     List<String> keyList = new ArrayList<String>();
328                     int numberOfDataRaw = 0;
329                     for (Map.Entry<String, List<String>> entry : globalDatatable.
↪entrySet()) {
330                             keyList.add(entry.getKey());
331                             numberOfDataRaw = entry.getValue().size();
332                     }
333
334                     for (int j = 0; j < keyList.size(); j++) {
335                             sb.append("|");
336                             sb.append(keyList.get(j));
337                     }
338                     sb.append("|\n");
339
340                     for (int i = 0; i < numberOfDataRaw; i++) {
341                             for (int j = 0; j < keyList.size(); j++) {
342                                     sb.append("|");
343                                     sb.append(globalDatatable.get(keyList.get(j)).
↪get(i));
344                             }
345                             sb.append("|\n");
346                     }
347
348                     // Start next test
349                     rpl.StartTest("Scenario: " + scenario.
↪getScenarioDescription(), sb.toString().equals("|") ? "" : sb.toString(),
350                                     tags);
351                     activeTest = true;
352             }
353
354             @Override
355             public void testResult(BDDScenario scenario, TestStatus testStatus,␣
↪File snapshot, String description) {
356
357                     if (debug) {
358                             System.err.println(
359                                             "testResult(BDDScenario scenario,␣
↪TestStatus testStatus, File snapshot, String description)");
360                     }
361
```

(continues on next page)

(continued from previous page)

```
362                     if (!activeChildUnit) {
363                             if (testStatus == TestStatus.PASS || testStatus ==␣
↪TestStatus.KTF) {
364                                     rpl.endTest(Status.PASSED);
365                             } else if (testStatus == TestStatus.SKIP) {
366                                     rpl.endTest(Status.SKIPPED);
367                             } else if (testStatus == TestStatus.FAIL) {
368                                     rpl.endTest(Status.FAILED);
369                             }
370
371                             activeTest = false;
372                     } else {
373
374                             // First close child test
375                             if (testStatus == TestStatus.PASS || testStatus ==␣
↪TestStatus.KTF) {
376                                     rpl.endStep(Status.PASSED);
377                             } else if (testStatus == TestStatus.SKIP) {
378                                     rpl.endStep(Status.SKIPPED);
379                             } else if (testStatus == TestStatus.FAIL) {
380                                     rpl.endStep(Status.FAILED);
381                             }
382
383                             activeChildUnit = false;
384                     }
385             }
386
387             @Override
388             public void childTestUnitExecutionStarted(BDDScenario scenario,␣
↪BDDStep step, String paramInfo) {
389
390                     if (debug) {
391                             System.err.println(
↪"childTestUnitExecutionStarted(BDDScenario scenario, BDDStep step, String␣
↪paramInfo)");
392                     }
393             }
394
395             @Override
396             public void testUnitExecutionStarted(BDDStep step) {
397
398                     if (debug) {
399                             System.err.println("testUnitExecutionStarted(BDDStep␣
↪step)");
400                     }
401
402                     if (!activeChildUnit) {
403                             rpl.StartStep("Step: " + step.getStepDescription(),␣
↪step.getStepDescription(), null);
404                     } else {
405                             rpl.StartChildStep("Parameterized Step: " + step.
↪getStepDescription(), step.getStepDescription(), null);
406                     }
407             }
408
409             @Override
410             public void childTestCaseExecutionStarted(BDDScenario scenario,␣
↪String paramInfo) {
411                     if (debug) {
412                             System.err.println(
↪"childTestCaseExecutionStarted(BDDScenario scenario, String paramInfo)");
```

(continues on next page)

(continued from previous page)

```
413                     }
414
415                     Set<String> tags = new HashSet<String>();
416                     for (String s : scenario.getGroupList()) {
417                             tags.add(s);
418                     }
419
420                     StringBuilder sb = new StringBuilder();
421                     LinkedHashMap<String, List<String>> globalDatatable =␣
↪scenario.getGlobalDataTable();
422                     List<String> keyList = new ArrayList<String>();
423
424                     for (Map.Entry<String, List<String>> entry : globalDatatable.
↪entrySet()) {
425                             keyList.add(entry.getKey());
426                     }
427
428                     for (int j = 0; j < keyList.size(); j++) {
429                             sb.append("|");
430                             sb.append(keyList.get(j));
431                     }
432                     sb.append("|\n");
433
434                     int indexNumber = Integer.parseInt(paramInfo.
↪substring(paramInfo.indexOf("(") + 1, paramInfo.indexOf(")")));
435                     for (int j = 0; j < keyList.size(); j++) {
436                             sb.append("|");
437                             sb.append(globalDatatable.get(keyList.get(j)).
↪get(indexNumber));
438                     }
439                     sb.append("|\n");
440
441                     rpl.StartStep("Step: " + scenario.getScenarioDescription() +
↪" " + paramInfo, sb.toString(), tags);
442                     activeChildUnit = true;
443             }
444
445             @Override
446             public void childTestCaseExecutionFinished(BDDScenario scenario) {
447                     if (debug) {
448                             System.err.println(
↪"childTestCaseExecutionFinished(BDDScenario scenario)");
449                     }
450
451             }
452
453             @Override
454             public void childTestUnitExecutionFinished(BDDStep step) {
455                     if (debug) {
456                             System.err.println(
↪"childTestUnitExecutionFinished(BDDStep step)");
457                     }
458
459             }
460
461             @Override
462             public void testCaseExecutionFinished(BDDScenario scenario) {
463                     if (debug) {
464                             System.err.println(
↪"testCaseExecutionFinished(BDDScenario scenario)");
465                     }
```

(continues on next page)

(continued from previous page)

```
466
467                     if (activeTest) {
468                             // Then close the testcase
469                             rpl.endTest(Status.PASSED);
470                             activeTest = false;
471                     }
472
473             }
474
475             @Override
476             public void testUnitResult(BDDStep step, TestStatus testStatus, File␣
    ↪snapshot, String description) {
477                     if (debug) {
478                             System.err.println("testUnitExecutionFinished(BDDStep␣
    ↪step)");
479                     }
480
481                     if (!activeChildUnit) {
482                             // First close child test
483                             if (testStatus == TestStatus.PASS || testStatus ==␣
    ↪TestStatus.KTF) {
484                                     rpl.endStep(Status.PASSED);
485                             } else if (testStatus == TestStatus.SKIP) {
486                                     rpl.endStep(Status.SKIPPED);
487                             } else if (testStatus == TestStatus.FAIL) {
488                                     rpl.endStep(Status.FAILED);
489                             }
490                     } else {
491                             // First close child test
492                             if (testStatus == TestStatus.PASS || testStatus ==␣
    ↪TestStatus.KTF) {
493                                     rpl.endChildStep(Status.PASSED);
494                             } else if (testStatus == TestStatus.SKIP) {
495                                     rpl.endChildStep(Status.SKIPPED);
496                             } else if (testStatus == TestStatus.FAIL) {
497                                     rpl.endChildStep(Status.FAILED);
498                             }
499                     }
500             }
501
502             @Override
503             public void testUnitExecutionFinished(BDDStep step) {
504                     // TODO Auto-generated method stub
505             }
506
507             @Override
508             public void printTestPlan(BDDScenario sc) {
509                     // TODO Auto-generated method stub
510
511             }
512
513             @Override
514             public void printTestUnitPlan(BDDStep step) {
515                     // TODO Auto-generated method stub
516
517             }
518
519             // *******************************************************
520             // Before After
521             // *******************************************************
522
```

(continues on next page)

(continued from previous page)

```
523             @Override
524             public void beforeTestSuiteMethodExecutionStarted(String methodName,␣
↪String description) {
525                     // TODO Auto-generated method stub
526
527             }
528
529             @Override
530             public void beforeTestSuiteMethodExecutionFinished(String␣
↪description) {
531                     // TODO Auto-generated method stub
532
533             }
534
535             @Override
536             public void afterTestSuiteMethodExecutionStarted(String methodName,␣
↪String description) {
537                     // TODO Auto-generated method stub
538
539             }
540
541             @Override
542             public void afterTestSuiteMethodExecutionFinished(String description)
↪{
543                     // TODO Auto-generated method stub
544
545             }
546
547             @Override
548             public void globalBeforeTestUnitMethodExecutionStarted(String␣
↪methodName, TestUnitObjectWrapper unit) {
549                     // TODO Auto-generated method stub
550
551             }
552
553             @Override
554             public void globalBeforeTestUnitMethodExecutionStarted(String␣
↪methodName, BDDStep step) {
555                     // TODO Auto-generated method stub
556
557             }
558
559             @Override
560             public void␣
↪globalBeforeTestUnitMethodExecutionFinished(TestUnitObjectWrapper unit) {
561                     // TODO Auto-generated method stub
562
563             }
564
565             @Override
566             public void globalBeforeTestUnitMethodExecutionFinished(BDDStep step)
↪{
567                     // TODO Auto-generated method stub
568
569             }
570
571             @Override
572             public void globalAfterTestUnitMethodExecutionStarted(String␣
↪methodName, TestUnitObjectWrapper unit) {
573                     // TODO Auto-generated method stub
574
```

(continues on next page)

(continued from previous page)

```
575             }
576
577             @Override
578             public void globalAfterTestUnitMethodExecutionStarted(String␣
↪methodName, BDDStep step) {
579                     // TODO Auto-generated method stub
580
581             }
582
583             @Override
584             public void globalAfterFailedUnitMethodExecutionStarted(String␣
↪methodName, TestUnitObjectWrapper unit) {
585                     // TODO Auto-generated method stub
586
587             }
588
589             @Override
590             public void globalAfterFailedUnitMethodExecutionStarted(String␣
↪methodName, BDDStep step) {
591                     // TODO Auto-generated method stub
592
593             }
594
595             @Override
596             public void␣
↪globalAfterTestUnitMethodExecutionFinished(TestUnitObjectWrapper unit) {
597                     // TODO Auto-generated method stub
598
599             }
600
601             @Override
602             public void globalAfterTestUnitMethodExecutionFinished(BDDStep step) {
603                     // TODO Auto-generated method stub
604
605             }
606
607             @Override
608             public void␣
↪globalAfterFailedUnitMethodExecutionFinished(TestUnitObjectWrapper unit) {
609                     // TODO Auto-generated method stub
610
611             }
612
613             @Override
614             public void globalAfterFailedUnitMethodExecutionFinished(BDDStep␣
↪step) {
615                     // TODO Auto-generated method stub
616
617             }
618
619             @Override
620             public void␣
↪localBeforeTestUnitMethodExecutionStarted(TestObjectWrapper t,␣
↪TestUnitObjectWrapper unit) {
621                     // TODO Auto-generated method stub
622
623             }
624
625             @Override
626             public void␣
↪localBeforeTestUnitMethodExecutionFinished(TestUnitObjectWrapper unit) {
```

(continues on next page)

(continued from previous page)

```
627                     // TODO Auto-generated method stub
628
629             }
630
631             @Override
632             public void␣
↪localAfterTestUnitMethodExecutionStarted(TestObjectWrapper t,␣
↪TestUnitObjectWrapper unit) {
633                     // TODO Auto-generated method stub
634
635             }
636
637             @Override
638             public void␣
↪localAfterTestUnitMethodExecutionFinished(TestUnitObjectWrapper unit) {
639                     // TODO Auto-generated method stub
640
641             }
642
643             @Override
644             public void globalBeforeTestCaseMethodExecutionStarted(String␣
↪methodName, TestObjectWrapper t) {
645                     // TODO Auto-generated method stub
646
647             }
648
649             @Override
650             public void globalBeforeTestCaseMethodExecutionStarted(String␣
↪methodName, BDDScenario scenario) {
651                     // TODO Auto-generated method stub
652
653             }
654
655             @Override
656             public void␣
↪globalBeforeTestCaseMethodExecutionFinished(TestObjectWrapper t) {
657                     // TODO Auto-generated method stub
658
659             }
660
661             @Override
662             public void globalBeforeTestCaseMethodExecutionFinished(BDDScenario␣
↪scenario) {
663                     // TODO Auto-generated method stub
664
665             }
666
667             @Override
668             public void globalAfterTestCaseMethodExecutionStarted(String␣
↪methodName, TestObjectWrapper t) {
669                     // TODO Auto-generated method stub
670
671             }
672
673             @Override
674             public void globalAfterTestCaseMethodExecutionStarted(String␣
↪methodName, BDDScenario scenario) {
675                     // TODO Auto-generated method stub
676
677             }
678
```

(continues on next page)

(continued from previous page)

```
679             @Override
680             public void␣
↪globalAfterTestCaseMethodExecutionFinished(TestObjectWrapper t) {
681                     // TODO Auto-generated method stub
682
683             }
684
685             @Override
686             public void globalAfterTestCaseMethodExecutionFinished(BDDScenario␣
↪scenario) {
687                     // TODO Auto-generated method stub
688
689             }
690
691             @Override
692             public void localBeforeTestCaseMethodExecutionStarted(String␣
↪methodName, TestObjectWrapper t) {
693                     // TODO Auto-generated method stub
694
695             }
696
697             @Override
698             public void␣
↪localBeforeTestCaseMethodExecutionFinished(TestObjectWrapper t) {
699                     // TODO Auto-generated method stub
700
701             }
702
703             @Override
704             public void localAfterTestCaseMethodExecutionStarted(String␣
↪methodName, TestObjectWrapper t) {
705                     // TODO Auto-generated method stub
706
707             }
708
709             @Override
710             public void␣
↪localAfterTestCaseMethodExecutionFinished(TestObjectWrapper t) {
711                     // TODO Auto-generated method stub
712
713             }
714
715     }
```

40.3 Register listener to the Artos runner

- Change Artos runner launch code as shown below to register listener.

```
1     public static void main(String[] args) throws Exception {
2             List<Class<?>> listeners = new ArrayList<Class<?>>();
3             listeners.add(ReportPortalListener.class);
4             Runner runner = new Runner(FeatureRunner.class, listeners);
5             runner.setTestList(getTestList());
6             runner.run(args);
7     }
```

- Execute runner.
- You should see new Launch is created in Report Portal instance.

CHAPTER 41

Articles

- Test_Result_Reporting
- Visual_Regression

CHAPTER 42

Blogs

- Artos_Introduction_By_Vinay_Bedre
- Artos_For_Testers_By_Testers_By_Swapna_Soni
- Artos_OpenSource_By_Swapna_Soni