Artos Documentation

Release 01.00.0000

Arpit Shah
contributors

Feb 02, 2019

INTRODUCTION:

CHAPTER

ONE

ARTOS (ART OF SYSTEM TESTING)

ARTOS is developed by a team of experienced test engineers for test developers/engineers. It was designed and developed with the aim to provide a test framework which 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 other platform that runs Java. It can be used for functional, system, end to end and/or unit testing. Most test frameworks only provide the test runner while the rest is left on engineers to develop, whereas ARTOS comes with many inbuilt and well tested utilities saving time for users to focus on what they do best!

1.1 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 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	A state of a test case at the line of execution (PASS, FAIL, SKIP or KTF)
Unit Outcome	The final outcome of the test unit (PASS, FAIL, SKIP or KTF)
Test Outcome	The final outcome of a test case (PASS, FAIL, SKIP or KTF)

1.2 Annotations

ARTOS makes use of Java annotations for most of its features. A list of supported annotations is provided below. Annotation in detail will be covered in later sections.

Annotation	Applies To	Usage
@TestCase	Class	Annotation used to mark a class as a test case
@TestPlan	Class	Annotation used to document a test plan and other test case related information.
@Unit	Method	Annotation used to mark a method inside a test case as a test unit.
@BeforeTestSuite	Method	Annotation used to mark a method that is invoked before test suite
@AfterTestSuite	Method	Annotation used to mark a method that is invoked after test suite
@BeforeTest	Method	Annotation used to mark a method that is invoked before each test cases of a test suite
@AfterTest	Method	Annotation used to mark a method that is invoked after each test cases of a test suite
@BeforeTestUnit	Method	Annotation used to mark a method that is invoked before test units.
@AfterTestUnit	Method	Annotation used to mark a method that is invoked after test units.
@DataProvider	Method	Annotated method(s) are marked as a supplier of data for the test case(s).
@ExpectedException	Method	Annotation used to specify list of exceptions and/or exception message. Attribute values are used to derive test outcome
@Group	Class & Method	Annotation used to specify list of groups that a test case or a test unit belongs to.
@KnownToFail	Class & Method	Annotation used to enforce known to fail check for annotated test case.

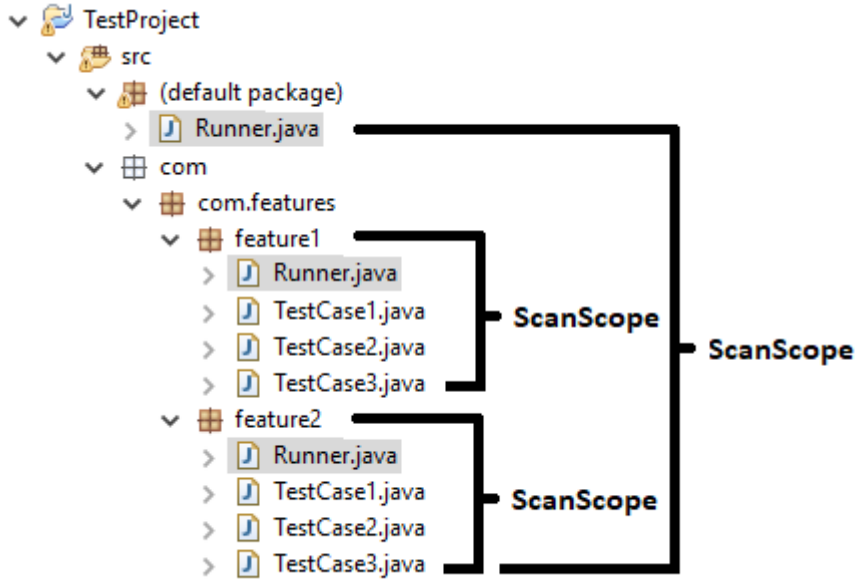
CHAPTER

TWO

TEST SUITE & TEST RUNNER

2.1 Scan Scope

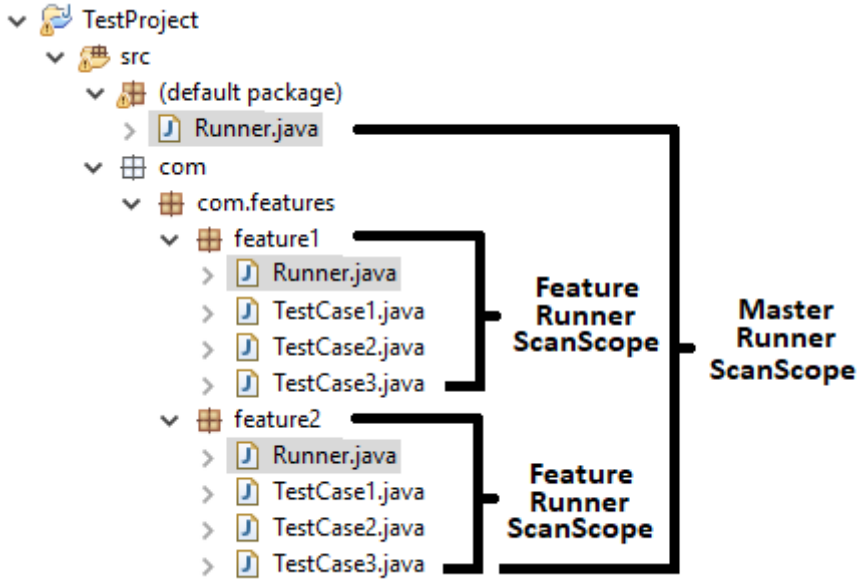
ARTOS runner scans a section of Java project as an initial step of test suite execution. A class which initiates this scan is called the **Runner**. The Runner only scans withing same package or its child packages. A section of the Java project that will be scanned during the search is called a **Scan Scope** of the Runner.



2.2 The Runner

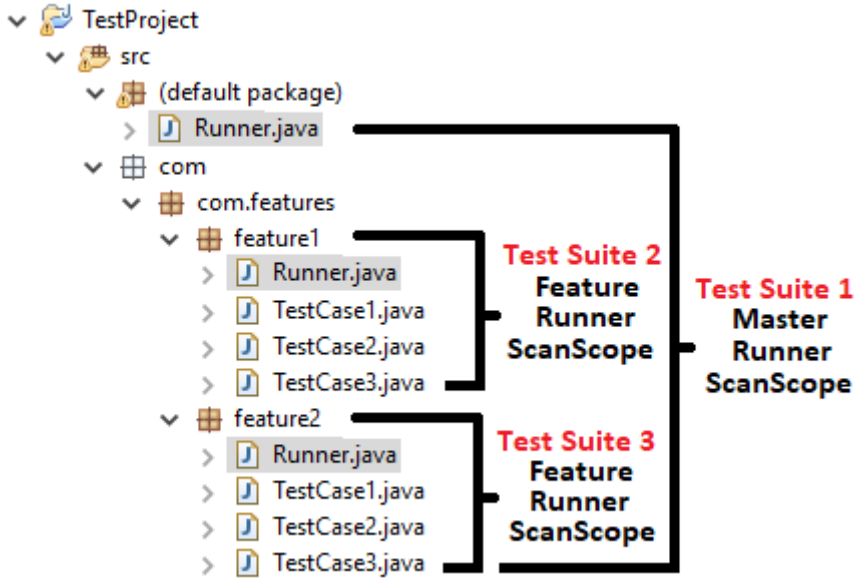
- A Runner is the entry point to a test suite.
- A Runner performs a scan within its scan scope in search of test cases (Classes annotated with `@TestCase` annotation).
- A Runner at project root location[1] is called a **Master Runner** which has visibility of all test cases within a project.
- A Runner created within individual package is called a **Feature Runner** which has visibility inside its own package or its sub-packages.
- A test project can have more than one Runner.
- Non-Maven project root location => `src`.
- Maven project root location => `src/main/java`.
- **Eclipse IDE** root location is also known as "default package".

[1] Project Root location



2.3 Test Suite

- A Runner and test cases within Runner's scan scope combined consists a **Test Suite**.
- A project contains as many test suites as 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.



CHAPTER

THREE

SYSTEM SETUP

3.1 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.

3.2 Add Artos Jar as a dependency

- Non-Maven Projects
  - Download latest Artos jar from location - Artos_Maven_Repository.
  - Add jar to project build path.
- Maven Projects
  - Copy latest jar dependency xml block from 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>
```

3.3 Eclipse SDK

3.3.1 Install ANSI plug-in for Linux OS

- Go to Eclipse SDK => Help => Eclipse Marketplace.
- Find “ANSI escape in console” plug-in.
- Install plug-in.
- Restart Eclipse SDK.

3.3.2 Configure test templates

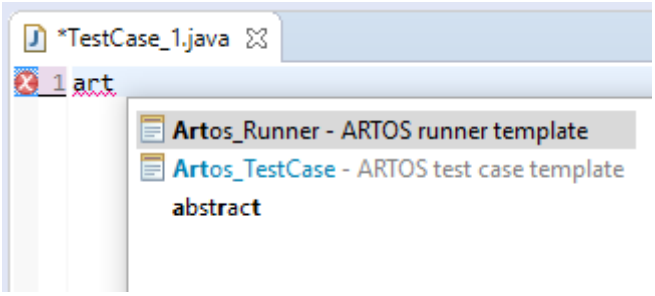
The use of a Java template increases consistency and test development speed. Templates can be modified to suite business requirements.

Import default templates:

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

Use template:

- Create new Java class.
- Select and delete all the content of the class.
- Type **art** and press **CTRL+SPACE**.
- Template suggestion list will appear so user can select appropriate template.



CHAPTER

FOUR

IMPLEMENT PROJECT

ARTOS test project consists of two components:

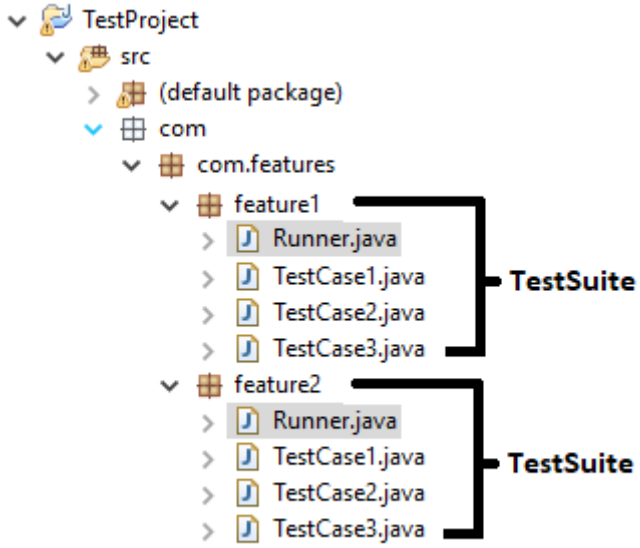
- Test Runner
- Test Case(s)

Project can be configured many different ways as per business requirement.

4.1 Recommended Project Structures

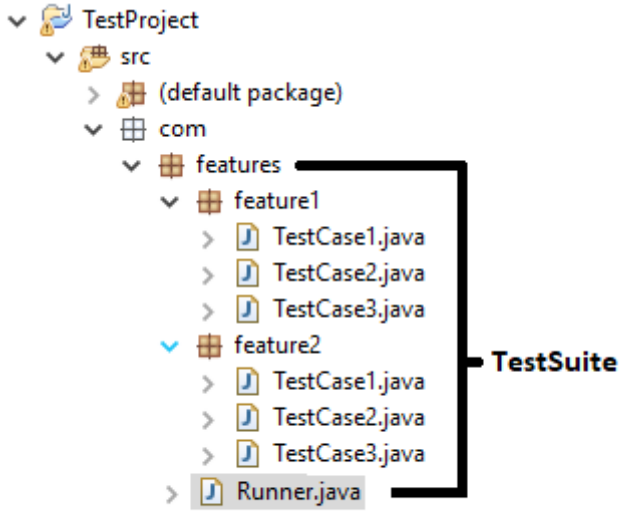
4.1.1 Feature Structure

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



4.1.2 Super Structure

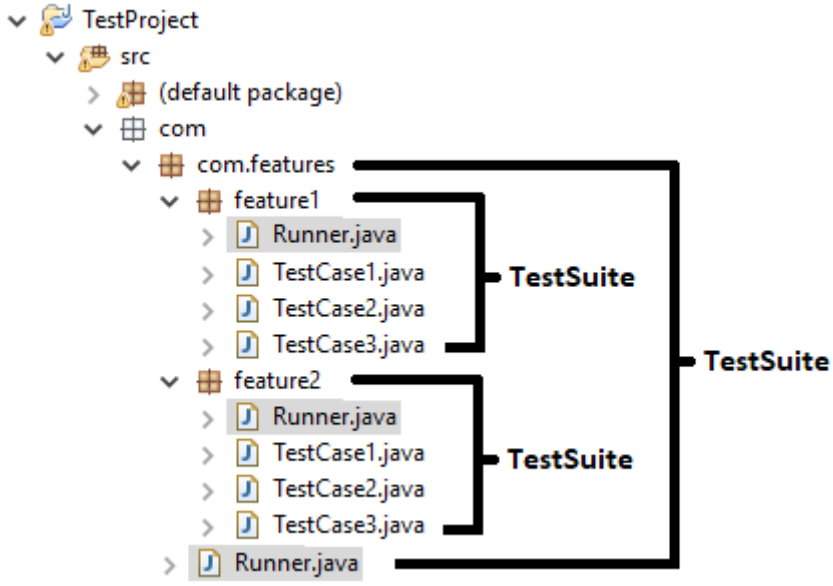
- Packages and sub-packages are organized based on features.
- Project contains a single Runner and all test cases are within Runner's scan scope thus entire project acts as single test suite.
- This can also be achieved by having Runner at project root location.



4.1.3 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 in parent/root position and Runner class within each feature packages.
- The test suite executes limited or all test cases depending on the used Runner.



CHAPTER

FIVE

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 (In this example : com.tests.ArtosMain.java).
- Implement `main` method and Runner code as shown in the example below.

Listing 5.1: 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

SIX

IMPLEMENT TESTCASE

Test case is the Java class which meets the following requirements:

- Class is `public`
- Class is annotated with `@TestCase` annotation.
- Class implements `TestExecutable` interface.
- Class contains at least one test unit.

Recommended

Add test plan for each test cases using `@TestPlan` annotation.

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).`

Important:

- Test units must be independent of each other.
- All test units are executed using new class instance so variables/objects can not be shared between two test units unless stored in context.
- Use method `context.setGlobalObject(key, obj);` or `context.setGlobalString(key, str);` to share objects between test cases.

Steps

- Create new Java class inside created package structure (In this example : TestCase_1.java)
- Copy paste below code in newly created Java file.

Listing 6.1: Example: Test case code 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")
```

(continues on next page)

(continued from previous page)

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

        @Unit()
        public void unit_test1(TestContext context) throws Exception {
                // ------------------------------------------------------------------
↪------------------------------
                // Print on console
                System.out.println("Hello World 1");
                // Print inside a log file
                context.getLogger().debug("Hello World 1");
                // ------------------------------------------------------------------
↪------------------------------
        }

        @Unit()
        public void unit_test2(TestContext context) throws Exception {
                // ------------------------------------------------------------------
↪------------------------------
                // Print on console
                System.out.println("Hello World 2");
                // Print inside a log file
                context.getLogger().debug(doSomething());
                // ------------------------------------------------------------------
↪------------------------------
        }

        // This method is not a test unit
        public String doSomething() {
                return "Hello World 2";
        }
}
```

CHAPTER

SEVEN

RUN TEST PROJECT

ARTOS can be run via

- Command line
- IDE (Example : Eclipse, IntelliJ etc..)

7.1 Command line

- Artos can be executed via command line by specifying minimum of
  - All library in the class path (Specify all dependencies)
  - Test runner class for given test suite (Starting point of test application)
  - Test script for given test suite (Contain all instructions to execute test suite correctly)
  - Profile name (helps in selection of correct framework configuration)

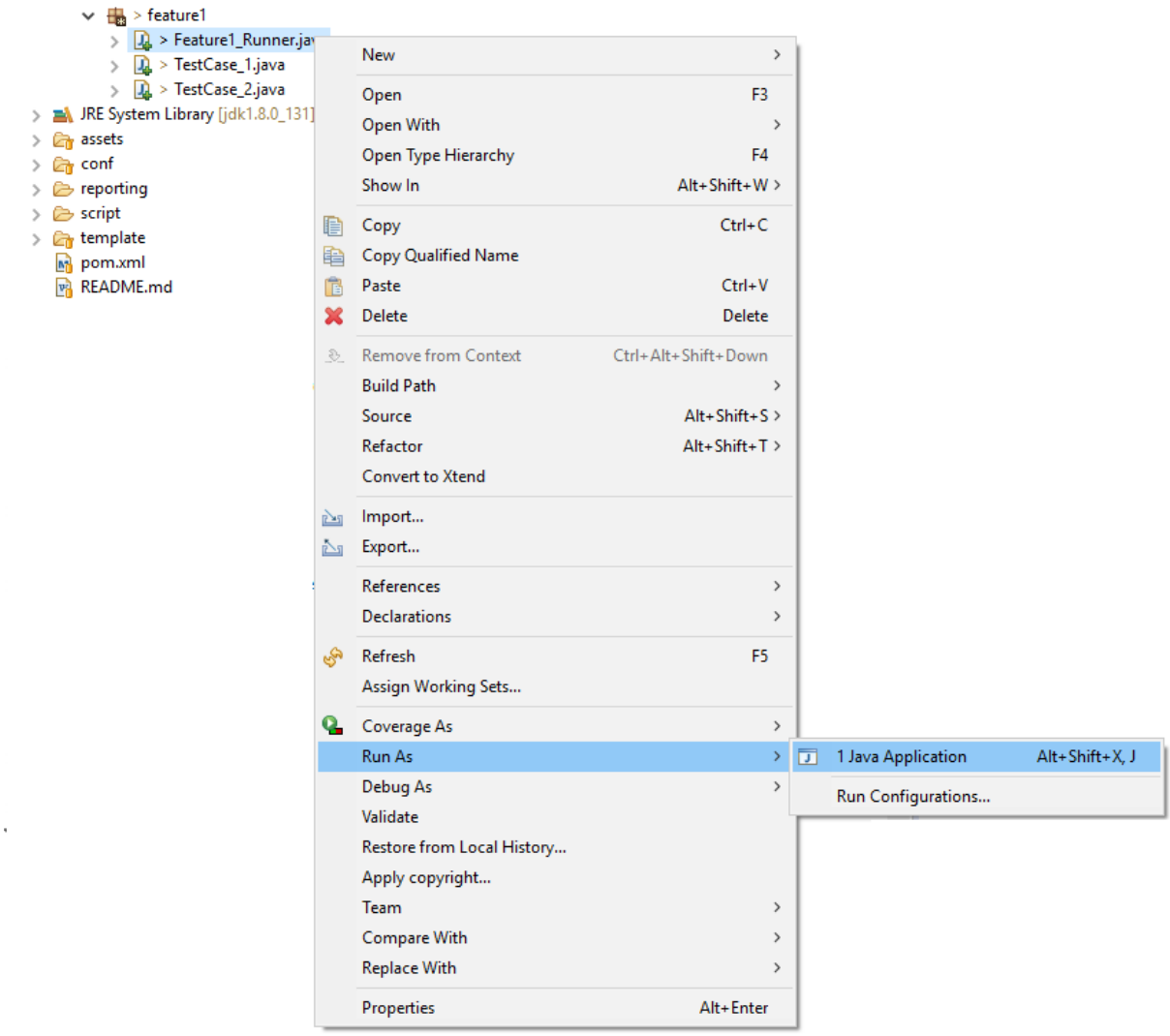
```
// Current example is written with following assumption:
// * Test project only has artos-0.0.1.jar and testproject.jar as a dependency.
// * artos-0.0.1.jar is located at .\lib\artos-0.0.1.jar.
// * Test script is located at .\script\testscript.xml.
// * Class with main method name is TestRunner.java (Test runner).
// * "dev" profile is used from framework_configuration.xml.

java -cp .\lib\artos-0.0.1.jar .\lib\testproject.jar TestRunner --testscript=".
↪\script\testscript.xml" --profile="dev"
```

7.2 Eclipse IDE

7.2.1 Using Runner Class

- Right click on the test runner class.
- Select options Run as => Java Application.


feature1
Feature1_Runner.ja
TestCase_1.java
TestCase_2.java
JRE System Library [jdk1.8.0_131]
assets
conf
reporting
script
template
pom.xml
README.md
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
Convert to Xtend
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Validate
Restore from Local History...
Apply copyright...
Team
Compare With
Replace With
Properties Alt+Enter
1 Java Application Alt+Shift+X, J
Run Configurations...

CHAPTER

EIGHT

HELLO WORLD

Artos is ready to execute test cases in three simple steps

- Add Artos Jar as a dependency
- Create a Runner
- Create a TestCase

8.1 Add Artos Jar as a dependency

- Non-Maven Projects
  - Download latest Artos jar from location - Artos_Maven_Repository.
  - Add jar to project build path.
- Maven Projects
  - Copy latest jar dependency xml block from 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>
```

8.2 Create a Runner

- Create required package structure (In this example : com.tests).
- Create new Java class inside created package structure (In this example : ArtosMain.java).
- Copy paste below code in newly created Java file.

```
package com.tests;

import com.artos.framework.infra.Runner;

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

8.3 Create a TestCase

- Create new Java class inside created package structure (In this example : TestCase_1.java)
- Copy paste below code in newly created Java file.

```
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
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("Hello World 2");
30                 // ------------------------------------------------------------------
   ↪-----------------------------
31         }
32 }
```

- Invoke `main()` method by running project as Java application.
- You have successfully executed your first test case using ARTOS.
- Notice logs generated in `./reporting` directory.
- Notice configuration files generated in `./conf` directory.

CHAPTER

NINE

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
-ts <arg>	–testscript <arg>	Test script file path
-v	–version	ARTOS' version

- Test script and profile can be specified via command line using following example:

```
>>> Short convention
java -cp .\lib\artos-0.0.1.jar .\lib\testproject.jar TestRunner -ts=".
↪\script\testscript.xml" --p="dev"
>>> Long convention
java -cp .\lib\artos-0.0.1.jar .\lib\testproject.jar TestRunner --testscript=".
↪\script\testscript.xml" --profile="dev"
```

CHAPTER

TEN

TEST STATUS

Test status allows user to update test status during test unit execution. Test status can be updated as frequently as required. Each status update will be visible in log file. Highest severity status update is recorded as test outcome.

Status	Severity	Usage
PASS	0	Test case executed without any errors
SKIP	1	Test case execution is skipped
KTF	2	Test case is known to fail
FAIL	3	Test case failed

Test status can be updated using method `TestContext().setTestStatus(TestStatus.FAIL, "Test did bad thing..");`

Recommended

Add short description during every status update.

Listing 10.1: Example code with multiple status updates. Outcome will be **FAIL**

```
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 : logic goes here..
17         TestContext().setTestStatus(TestStatus.PASS, "Test flow is as expected");
18
19         // TODO : logic goes here..
20         TestContext().setTestStatus(TestStatus.PASS, "Test flow is as expected");
21
22         // TODO : logic goes here..
23         TestContext().setTestStatus(TestStatus.FAIL, "Test flow is not as expected
   ↪");
```

(continues on next page)

(continued from previous page)

```
24
25          // TODO : logic goes here..
26          TestContext().setTestStatus(TestStatus.PASS, "Test flow is as expected");
27          // ------------------------------------------------------------------------
    ↪--------------------
28      }
29  }
```

10.1 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 10.2: Example code with multiple unit outcomes. TestCase outcome will be **FAIL**

```
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   // TestCase outcome is FAIL
10  @TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "GIVEN..WHEN..
    ↪AND..THEN..")
11  @TestCase()
12  public class TestCase_1 implements TestExecutable {
13
14      // TestUnit outcome  is FAIL
15      @Unit()
16      public void unit_test1(TestContext context) throws Exception {
17          // ------------------------------------------------------------------------
    ↪--------------------
18          // TODO : logic goes here..
19          TestContext().setTestStatus(TestStatus.FAIL, "Test fails");
20          // ------------------------------------------------------------------------
    ↪--------------------
21      }
22
23      // TestUnit outcome  is PASS
24      @Unit()
25      public void unit_test1(TestContext context) throws Exception {
26          // ------------------------------------------------------------------------
    ↪--------------------
27          // TODO : logic goes here..
28          TestContext().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          TestContext().setTestStatus(TestStatus.KTF, "Test is known to fail");
```

(continues on next page)

(continued from previous page)

```
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          TestContext().setTestStatus(TestStatus.SKIP, "Test is skipped");
47          // ------------------------------------------------------------------------
    ↪--------------------
48      }
49  }
```

CHAPTER

ELEVEN

TESTCONTEXT

- Test context is a container that holds live information about test suite, test cases and related information.
- Test context provides many useful methods to make test case development easy.
- Each test suite maintains its own `TestContext` object instance that allows parallel test suite execution without any conflict.
- All annotated methods are injected with relevant `TestContext` object so test suite related information is available to each test case/unit all the time.

11.1 Logger

User can get relevant logger object and log to appropriate log files using one of the following methods.

```
>>> Logs to log file with set log level
context.getLogger().info();
context.getLogger().debug();
context.getLogger().warn();
context.getLogger().error();
context.getLogger().fatal();
context.getLogger().trace();
```

11.2 DataProvider Parameters

DataProvider method returns 2D array. ARTOS repeats test case execution as many time as the length of 2D array. ARTOS updates relevant TestContext variables with next values each execution. Test units can access those values using following methods.

```
>>> Gets DataProvider parameter value
context.getParameterisedObject1();
context.getParameterisedObject2();
```

11.3 Global object storage

ARTOS TestContext can store key value pairs. Any method with an access of TestContext can retrieve, update or remove set value pairs. User can use one of the following methods to achieve that.

```
>>> Sets key value pair
context.setGlobalObject(String key, Object obj);
context.setGlobalString(String key, String obj);
```

```
>>> Gets key value
context.setGlobalObject(String key);
context.getGlobalString(String key);
```

11.4 Live TestCase Count

User can query number of test cases by their outcome using following methods.

```
>>> Gets TestCase count at that point of time
context.getCurrentFailCount();
context.getCurrentKTFCount();
context.getCurrentPassCount();
context.getCurrentSkipCount();
```

CHAPTER

TWELVE

PARALLEL SUITE EXECUTION & TESTSCRIPT

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

- Test script can be auto generated using ARTOS inbuilt feature or can be manually constructed.
- User specifies test script file path via command line parameter.

Example: `--testscript="./script/testscript.xml"`.

Listing 12.1: 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>
```

12.1 <suite> and parallel test suite execution

- `<suite></suite>` represents one test suite.
- One test script file can have multiple `<suite></suite>` elements.
- Multiple test suites will be run in parallel if specified in single test script. see below Sample script.

</suite> can have following attributes:

loopcount

- Loop count value represents number of time test suite will be executed in the sequence.
- Loop count value is an unsigned integer value represented as String.
- If loop count value is missing or invalid then default “1” will be applied.

name

- Test suite name is a string which will be used to construct log and report file name.

- String should not be more than 10 characters long
- String can only contain one of the following characters. [A-Z][a-z][0-9][-]

Listing 12.2: 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>
    <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>
```

12.2 <tests>

- `<tests></tests>` contains list of test cases and their execution sequence.
- If any test cases are specified then only those test cases will be executed in given sequence.
- If test case is explicitly marked with attribute `TestCase(skip=true)` then marked test case will be skipped even though it is listed in a test script.
- If test case is listed in the script but not within a scan scope of the Runner then Runner will ignore that test case and continue execution with rest of the test cases.
- If `<tests></tests>` is empty then all test cases within Runner's scan scope will be executed with use of sequence number specified by `TestCase(sequence=1)` attribute.
- Test case name is case sensitive.

Test cases can be listed as shown below:

Listing 12.3: 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>
```

12.3 <parameters>

- `<parameters></parameters>` contains list of parameters which required to be available throughout test suite execution.
- All parameters will be set to `TestContext` object using method `context.setGlobalObject(key, obj);` prior to test suite execution.
- Parameters can be queried during test case execution using method `context.getGlobalObject(key);`.
- Parameter(s) can be removed or value can be updated by test case.
- Separate `TestContext` object is assigned to each test suite thus parameters values with same name will not be overwritten between test suites.
- Parameter name and value are case sensitive

Listing 12.4: Sample <tests> 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>
```

12.4 <testcasegroups>

- `<testcasegroups></testcasegroups>` contains list of group which is used to select test cases.
- Test cases belongs to one or more listed group are added to execution list.
- Group filter is applied to test cases listed under `<tests>` tag.
- In case of empty `<tests>` tag, group filter is applied to all test cases which are within scan scope of the Runner.
- Test case explicitly marked with attribute `TestCase(skip=true)` will be filtered out regardless of the group.
- If `<testcasegroups>` tag is missing then group filter will not be applied and all listed test cases will be added to execution list.
- Group name is case in-sensitive.

Listing 12.5: All listed test cases will be added to execution list

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

Listing 12.6: 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>
```

12.5 <testunitgroups>

- `<testunitgroups></testunitgroups>` contains list of group which is used to filter test units for execution.
- Group filter is only applied to test cases filtered using `<testcasegroups>` group list.
- Test unit explicitly marked with attribute `Unit(skip=true)` will be filtered out regardless of the group.
- If `<testunitgroups>` tag is missing then group filter will not be applied and all test units will be added to execution list.
- Group name is case in-sensitive.

Listing 12.7: All test units will be added to execution list

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

Listing 12.8: Test units belongs to “Fast” OR “Slow” test units will be added to execution list

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

12.6 Auto Generate test script

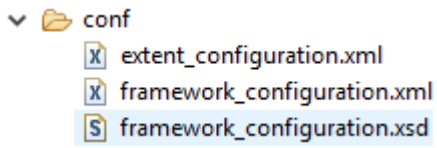
- ARTOS can auto generate test script.
- To enable test script generation user can set `<property name="generateTestScript">true</property>` inside `conf/framework_configuration.xml` file.
- Once enabled, test script will be auto generated inside `./script` directory.

CHAPTER

THIRTEEN

GENERATE DEFAULT CONFIGURATIONS

The ARTOS' first launch generates required configurations and directories in `conf` directory under project root. If configuration file is already present then it will not be overwritten but all missing files are re-created with default values.



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

FOURTEEN

FRAMEWORK CONFIGURATION

The `framework_configuration.xml` file is used to configure ARTOS framework. This file should be located inside `conf` directory at the project root. ARTOS framework will create default file in the same directory if file is not present.

Default settings:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="framework_configuration.xsd">
  <organization_info profile="dev">
    <property name="Name">&lt;Organisation&gt; PTY LTD</property>
    <property name="Address">XX, Test Street, Test address</property>
    <property name="Country">NewZealand</property>
    <property name="Contact_Number">+64 1234567</property>
    <property name="Email">artos.framework@gmail.com</property>
    <property name="Website">www.theartos.com</property>
  </organization_info>
  <logger profile="dev">
    <!--LogLevel Options : info:debug:trace:fatal:warn:all-->
    <property name="logLevel">debug</property>
    <property name="logRootDir">.\reporting\</property>
    <property name="enableLogDecoration">false</property>
    <property name="enableTextLog">true</property>
    <property name="enableHTMLLog">false</property>
    <property name="enableExtentReport">true</property>
  </logger>
  <smtp_settings profile="dev">
    <property name="ServerAddress">smtp.gmail.com</property>
    <property name="SSLPort">587</property>
    <property name="SMTPAuth">true</property>
    <property name="SendersName">John Murray</property>
    <property name="SendersEmail">test@gmail.com</property>
    <property name="emailAuthSettingsFilePath">.\conf\user_auth_settings.xml</property>
    <property name="ReceiversEmail">test@gmail.com</property>
    <property name="ReceiversName">Mac Murray</property>
    <property name="EmailSubject">Artos Email Client</property>
    <property name="EmailMessage">This is a test Email from Artos</property>
  </smtp_settings>
  <features profile="dev">
    <property name="enableGUITestSelector">true</property>
    <property name="enableGUITestSelectorSeqNumber">false</property>
    <property name="enableBanner">true</property>
    <property name="enableOrganisationInfo">true</property>
    <property name="enableEmailClient">false</property>
    <property name="enableArtosDebug">false</property>
    <property name="generateEclipseTemplate">false</property>
    <property name="generateTestScript">true</property>
    <property name="stopOnFail">false</property>
```

(continues on next page)

(continued from previous page)

```
  </features>
</configuration>
```

14.1 <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 `<property name="enableOrganisationInfo">true</property>` under the `<feature>` tag.

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

14.2 <logger>

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

Property Name	Content	Description
logLevel[1]	String	Set log level
logRootDir[2]	String	Set log root directory relative to project
enableLogDecoration[3]	Boolean	Enable or disable log decoration
enableTextLog[4]	Boolean	Enable or disable text log and report
enableHTMLLog[4]	Boolean	Enable or disable HTML log and report
enableExtentReport[4]	Boolean	Enable or disable the Extent report

[1] One of the following log level can be selected:

- info
- debug
- trace
- fatal
- warn
- all

[2] Log file path construction: “logRootDir + test suite packageName + log file”.

```
>>> Example : /reporting/com.artos.featuretest/com.artos.tests_0_1546845327744-all.log
```

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

```
* [%-5level] = Log level upto 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
```

[4] When enabled: Log files and reports are generated with following specification.

```
>>> File naming convention:
Runner package name + Thread number + TestSuite name (Optional) + Time stamp + Type
```

14.3 <smtp_settings>

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

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`

14.4 <features>

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

Property Name	Content	Description
enableGUITestSelector	Boolean	Enable GUI test selector
enableGUITestSelectorSeqNum	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 debug log
generateEclipseTemplate	Boolean	Enable generation of Eclipse template
generateTestScript	Boolean	Enable test script generation
stopOnFail	Boolean	Enable test execution stop on fail

14.5 Profile based configuration:

ARTOS supports multiple configuration based on profile names.

If user requires different configurations between:

- Development and production environments OR

```
// Text log file example
* com.artos.feature1_0_xyz_1546845327744-all.log
* com.artos.feature1_0_xyz_1546845327744-realtime.log
* com.artos.feature1_0_xyz_1546845327744-summary.log

// HTML log file example
* com.artos.feature1_0_xyz_1546845327744-all.html
* com.artos.feature1_0_xyz_1546845327744-realtime.html
* com.artos.feature1_0_xyz_1546845327744-summary.html

// Extent report file example
* com.artos.feature1_0_xyz_1546847059200-all-extent.html
```

- Linux and Windows environments OR
- Local machine and build server environments

Then user can create multiple configurations within same file with unique profiles names. During test application launch user can pass profile name either via command line or via `Runner` class object.

- User can specify profile name via command line using parameter `--profile="prod"`.
- Profile name is case in-sensitive.
- If configurations blocks are missing from the file then default value will be used.
- If profile name is not specified then first configuration block from the top of the `framework_configuration.xml` file will be used.
- If any configuration line is missing from configuration block then previous configuration block line will be used. If configuration line is missing from all blocks then default values will be used.

14.6 Example

Given example contains two profiles, "dev" and "prod" respectively. "dev" profile has GUI test selector enabled where as "prod" profile has GUI test selector disabled.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="framework_configuration.xsd">

  <!-- Development configuration -->
  <organization_info profile="dev">
    <property name="Name">Organisation PTY LTD</property>
    <property name="Address">XX, Test Street, Test address</property>
    <property name="Country">NewZealand</property>
    <property name="Contact_Number">+64 1234567</property>
    <property name="Email">artos.framework@gmail.com</property>
    <property name="Website">www.theartos.com</property>
  </organization_info>
  <logger profile="dev">
    <!--LogLevel Options : info:debug:trace:fatal:warn:all-->
    <property name="logLevel">debug</property>
    <property name="logRootDir">.\reporting\</property>
    <property name="logSubDir">SN-123</property>
    <property name="enableLogDecoration">false</property>
    <property name="enableTextLog">true</property>
    <property name="enableHTMLLog">false</property>
    <property name="enableExtentReport">true</property>
  </logger>
  <smtp_settings profile="dev">
    <property name="ServerAddress">smtp.gmail.com</property>
    <property name="SSLPort">587</property>
    <property name="SMTPAuth">true</property>
    <property name="SendersName">John Murray</property>
    <property name="SendersEmail">test@gmail.com</property>
    <property name="emailAuthSettingsFilePath">.\conf\user_auth_settings.xml</property>
    <property name="ReceiversEmail">test@gmail.com</property>
    <property name="ReceiversName">Mac Murray</property>
    <property name="EmailSubject">Artos Email Client</property>
    <property name="EmailMessage">This is a test Email from Artos</property>
  </smtp_settings>
  <features profile="dev">
    <property name="enableGUITestSelector">true</property>
    <property name="enableGUITestSelectorSeqNumber">true</property>
    <property name="enableBanner">true</property>
```

(continues on next page)

(continued from previous page)

```
    <property name="enableOrganisationInfo">true</property>
    <property name="enableEmailClient">false</property>
    <property name="enableArtosDebug">false</property>
    <property name="generateEclipseTemplate">false</property>
    <property name="generateTestScript">true</property>
    <property name="stopOnFail">false</property>
  </features>

  <!-- Production configuration -->
  <organization_info profile="prod">
    <property name="Name">Organisation PTY LTD</property>
    <property name="Address">XX, Test Street, Test address</property>
    <property name="Country">NewZealand</property>
    <property name="Contact_Number">+64 1234567</property>
    <property name="Email">artos.framework@gmail.com</property>
    <property name="Website">www.theartos.com</property>
  </organization_info>
  <logger profile="prod">
    <!--LogLevel Options : info:debug:trace:fatal:warn:all-->
    <property name="logLevel">debug</property>
    <property name="logRootDir">.\reporting\</property>
    <property name="logSubDir">SN-123</property>
    <property name="enableLogDecoration">false</property>
    <property name="enableTextLog">true</property>
    <property name="enableHTMLLog">false</property>
    <property name="enableExtentReport">true</property>
  </logger>
  <smtp_settings profile="prod">
    <property name="ServerAddress">smtp.gmail.com</property>
    <property name="SSLPort">587</property>
    <property name="SMTPAuth">true</property>
    <property name="SendersName">John Murray</property>
    <property name="SendersEmail">test@gmail.com</property>
    <property name="emailAuthSettingsFilePath">.\conf\user_auth_settings.xml</
↪property>
    <property name="ReceiversEmail">test@gmail.com</property>
    <property name="ReceiversName">Mac Murray</property>
    <property name="EmailSubject">Artos Email Client</property>
    <property name="EmailMessage">This is a test Email from Artos</property>
  </smtp_settings>
  <features profile="prod">
    <property name="enableGUITestSelector">false</property>
    <property name="enableGUITestSelectorSeqNumber">false</property>
    <property name="enableBanner">true</property>
    <property name="enableOrganisationInfo">true</property>
    <property name="enableEmailClient">false</property>
    <property name="enableArtosDebug">false</property>
    <property name="generateEclipseTemplate">false</property>
    <property name="generateTestScript">true</property>
    <property name="stopOnFail">false</property>
  </features>

</configuration>
```

CHAPTER

FIFTEEN

LOGGING FRAMEWORK

- Purpose:
  - Logs and reports are heart of any test framework.
  - Artos log framework:
    - Pre-configured and ready to be used.
    - Capable of producing text and/or HTML log files.
    - Generates real-time logs which can be used for performance measurement.
    - Generates summary report which can be used as customer facing reports. (Separate to Extent Report)
    - Provides features which can be very useful in test development.
- Facilities:
  - Two log layouts are supported:
    - Text layout (Default enabled).
    - HTML layout (Default disabled).
  - Two log files are generated per test suite execution:
    - General log file: All logs (Same as console log).
    - Real Time log file: Send/receive message logs from connectable interfaces with time-stamp.
  - Report is generated per test suite:
    - Summary report: Summarized pass/fail summary with test name, execution time and bug reference number.
  - Log patterns (decoration):
    - Decoration enabled pattern:

      ```
      "[%-5level][%d{yyyy-MM-dd_HH:mm:ss.SSS}][%t][%F][%M][%c{1}] -
      %msg%n%throwable"
      ```

    - Decoration disabled pattern:

      ```
      "%msg%n%throwable"
      ```

      Refer: Log4j_Pattern
  - Log level support:
    - 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.

  - Note: Log level can be configured using `conf/framework_configuration.xml` file.
  - Log can be enabled/disabled dynamically (Only applicable to general log)
    - Disable log dynamically: `TestContext().getLogger().disableGeneralLog();`
    - Enable log dynamically: `TestContext().getLogger().enableGeneralLog();`
  - Parallel test suite execution will generate separate log files per test suite.
    - Log files will be labeled using thread number.
  - Log files are organized under test suite name for ease of use.
    - Log files are created under the following hierarchy. `RootDir => SubDir => TestSuiteName => Log file`.

```
>>> Example: ``./reporting/SN-123/com.test.testsuite1/..``
```

    - Root directory and sub directory location can be configured using `conf/framework_configuration.xml` file.
  - Log framework abstraction
    - Log framework is abstracted so that log framework can be changed in future without breaking existing test scripts.
  - Log file tracking
    - All log files are tracked during runtime. If user requires to retrieve current log files (inclusive of text/HTML) they can utilize this functionality.
    - This functionality will also be used to find current log files and attach to email client.
  - **FAIL** stamp injection
    - **FAIL** Stamp is injected to log stream straight after test status is updated to FAIL. This allows user to know at which exact line the test case failed during execution.
  - Log rollover policy
    - Current log rollover policy is triggered based on a file size of 20MB.
    - 20MB was chosen to meet emailing requirement. Trigger policy can be exposed to user in future.
  - Parameterized logging for efficiency
    - Parameterized logging is less efficient compare to string concatenation but it is efficient in case of log level is reduced to **INFO** or **ERROR**, because system do not have to spend time concatenating string.
- Usage:
  - Enable/disable text/HTML log files:

    * Can be configured using `conf/framework_configuration.xml` file.

- Change log level and log directory:
    * Can be configured using `conf/framework_configuration.xml` file.
- logging simple string with level info or debug:

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

- logging parameterized string with level info or debug:

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

    * Disable logging during execution time:

```
TestContext().getLogger().disableGeneralLog();
```

    * Enable logging during execution time:

```
TestContext().getLogger().enableGeneralLog();
```

CHAPTER

SIXTEEN

OFF-LINE EXTENT REPORT

- Purpose:

Professional looking Extent report which can be distributed among customer or external parties.

- Facilities:
  - If Extent configuration file is not present, then framework will generate default configuration file at location `conf/extent_configuration.xml`.
  - Artos has inbuilt support for off-line Extent reporting.
  - If enabled, Artos will produce Extent report for every test suite execution.
  - Extent report includes test name, test writer's name and test case duration.
  - Every test status update and description/reason will be reported via Extent report.
  - Final test result with bug reference number will be reported via Extent report.
- Usage:
  - Extent reporting can be enabled/disabled via `conf/framework_configuration.xml` file.

CHAPTER

SEVENTEEN

REAL TIME LOGGING

- Purpose:
  - Performance testing requires real-time logs so time between messages can be measured accurately. Artos has inbuilt interface which can be used to provide real time logging.
- Facilities:
  - Artos has inbuilt hooks for real time logging. All inbuilt connectors support real-time logging.
  - User can write new connectors by implementing `Connectable` interface and use listener `RealTimeLogEventListener` to capture real-time log events.
  - Real time logs are printed with time stamp, user is not allowed to disable time-stamp.
  - If test suites are executed in parallel then separate real time log file will be produced per test suite.
  - Real time log file will roll over at 20MB of file size.
  - Real time logs cannot be disabled. If *RealTimeLogEventListener* is not provided then log file will remain empty.
- Usage:
  - Collect send receive events in real time with time stamp.

CHAPTER

EIGHTEEN

STOP ON FAIL

- **Purpose:**
  - Depending on the goal, some user may want to continue next test execution after a test case failure, in other case user may choose to stop after very first test case failure. Artos support both behaviors.
- Facilities:
  - By default, Artos is setup to stop on first failure. it can be configured to continue executing rest of the test cases.
- Usage:
  - Stop on Fail feature can be enabled/disabled via `conf/framework_configuration.xml` file.

CHAPTER

NINETEEN

@BEFORETESTUNIT @AFTERTESTUNIT

19.1 @BeforeTestUnit

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

Location	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.

19.2 @AfterTestUnit

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

Location	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

TWENTY

@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
label()	Test label	Optional	Empty String
dataprovider()	Data provider method Name	Optional	Empty String
testtimeout()	Test timeout in milliseconds	Optional	0

- **skip()**
  - Temporarily 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.
- **label()**
  - Reserved for future use.
- **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.

20.1 Annotation use case(s)

```
@TestCase(skip = false, sequence = 1, label = { "Regression" }, dataprovider =
↪"username", testtimeout = 5000)
```

20.2 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, label = { "Regression" }, dataprovider =
↪"username", testtimeout = 5000)
public class TestCase_1 implements TestExecutable {

}
```

CHAPTER

TWENTYONE

@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

21.1 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..")
```

21.2 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 {

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

- Log file snapshot for above test case.

```
1  ************************************************************************
2  Test Name       : com.tests.TestCase_1
3  Written BY      : ArpitS
4  Date            : 1/1/2018
5  BDD Test Plan   :
6  GIVEN test project is set correctly
7  AND logger is used to log HELLO WORLD string
8  THEN hello world should be printed correctly
9  ************************************************************************
10 Observe formatted test plan in the log file
11
12 [PASS] : unit_test()
13
14
15 Test Result : PASS
```

CHAPTER

TWENTYTWO

@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

22.1 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

22.2 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
```

(continues on next page)

(continued from previous page)

```
@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]*")
```

22.3 Example functional test case

```
package com.tests;

import com.artos.annotation.ExpectedException;
import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.exception.InvalidDataException;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;

@ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class }, contains = "exception example")
@TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "")
@TestCase(skip = false, sequence = 0)
public class TestCase_2 implements TestExecutable {

        @Override
        public void execute(TestContext context) throws Exception {
                // ------------------------------------------------------------------
↪-----------------------------
                throw new NullPointerException("exception example");
                // ------------------------------------------------------------------
↪-----------------------------
        }
}
```

22.4 Example unit test case

```
package com.tests;

import com.artos.annotation.ExpectedException;
import com.artos.annotation.TestCase;
import com.artos.annotation.TestPlan;
import com.artos.exception.InvalidDataException;
import com.artos.framework.infra.TestContext;
import com.artos.interfaces.TestExecutable;


@TestPlan(preparedBy = "ArpitS", preparationDate = "1/1/2018", bdd = "")
@TestCase(skip = false, sequence = 0)
public class TestCase_2 implements TestExecutable {

        @Unit
        @ExpectedException(expectedExceptions = { NullPointerException.class,␣
↪InvalidDataException.class }, contains = "exception example")
        public void unit_test(TestContext context) throws Exception {
                // ------------------------------------------------------------------
↪-----------------------------
```

(continues on next page)

(continued from previous page)

```
19                  throw new NullPointerException("exception example");
20                  // ------------------------------------------------------------------
    ↪----------------------------
21          }
22  }
```