Artos Documentation

Release 01.00.0000

Arpit Shah
contributors

Feb 18, 2019

CONTENTS

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 ARTOS feature highlight

- Built-in and pre-configured log framework
  - Organized logging for ease of use
  - Text or HTML formatted logs
  - Five log level support
  - Real-time log files in addition to general log file (for performance measurement)
  - Runtime log enable/disable
  - Tracking of generated log files
  - Separate log files per test suite during parallel testing
- Built-in report generation
  - Professional looking Extent report
  - Simple text or HTML formatted report
  - Separate test reports per test suite during parallel testing
- Test time measurement
  - Test-suite, test-case and test-units execution duration measurement with millisecond accuracy
- Test importance
  - Test importance is highlighted to focus on important test cases
- Easy debugging
  - GUI test selector for selective testing and avoids user error
  - FAIL stamp injection in the log file to pin point exact line of failure
  - Failure highlight at the end of test suite execution
  - Bug reference reporting against failed test case
  - Test time tracking with millisecond accuracy
  - BDD formatted test plan injection in the log file to avoid disconnect between test plan and test case
- TestCase development

  - Group based test case and test unit filtering
  - Exception checking
  - Known to fail test case support
  - Data Provider support
  - Parallel testing
  - 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..)
- Additional features
  - Stop on fail support
  - Properties based test framework configuration
  - Automated test script generation
- Listeners for future plug-in or application development.

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 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 final outcome of the test unit (namely: PASS, FAIL, SKIP or KTF)
Test Outcome	The final outcome of the test case (namely: PASS, FAIL, SKIP or KTF)

1.3 Abbreviations

KTF : Known To Fail

1.4 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 execution
@AfterTestSuite	Method	Annotation used to mark a method that is invoked after test suite execution
@BeforeTest	Method	Annotation used to mark a method that is invoked before each test case(s) execution from a test suite
@AfterTest	Method	Annotation used to mark a method that is invoked after each test case(s) execution from a test suite
@BeforeTestUnit	Method	Annotation used to mark a method that is invoked before test units execution
@AfterTestUnit	Method	Annotation used to mark a method that is invoked after test units execution
@DataProvider	Method	Annotation used to mark method(s) behaving as supplier of test data to the test case(s)
@ExpectedException	Method	Annotation used to specify list of exception type(s) 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 and test unit

1.5 GUI test selector

ARTOS provides built-in GUI test selector that is designed to help test developers run selective test cases during development and debugging. GUI test selector feature can be enabled or disabled by changing framework configuration. GUI test selector details will be covered in later sections.

1.6 Test logs

ARTOS log levels, log decorations, log format can be configured using framework configuration. FAIL stamp is injected in the log file when test status is updated to FAIL (by the test), so that user can pin point exact line (in the log file) where failure has occurred.

1.7 Test report

ARTOS auto generates text and/or HTML based test report. This report only contains PASS/FAIL information so it can be shared with external parties keeping business critical information contained in log files.

ARTOS additionally generates professional looking Extent report if enabled.

CHAPTER

TWO

SYSTEM SETUP

2.1 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.

2.2 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>
```

2.3 Eclipse SDK

2.3.1 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.

2.3.2 Configure test templates

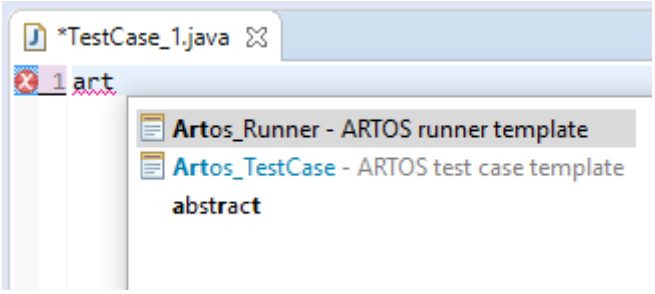
The use of a Java template increases consistency and test development speed. Templates can be modified to suite business requirements.

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

THREE

IMPLEMENT PROJECT

ARTOS test project consists of two components:

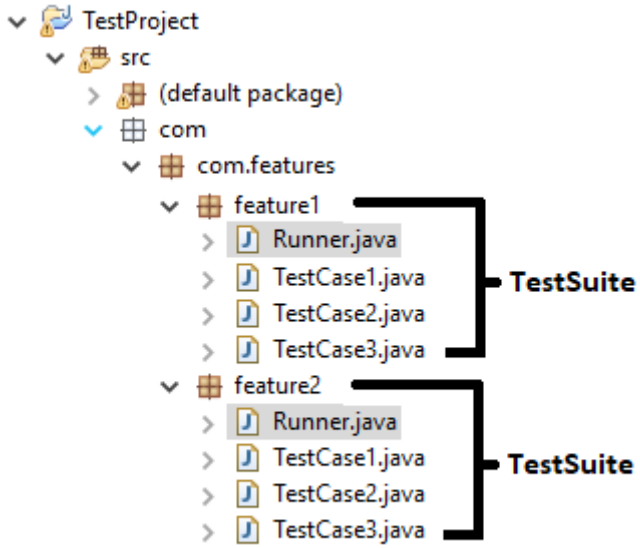
- Test Runner
- Test Case(s)

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

3.1 Recommended Project Structures

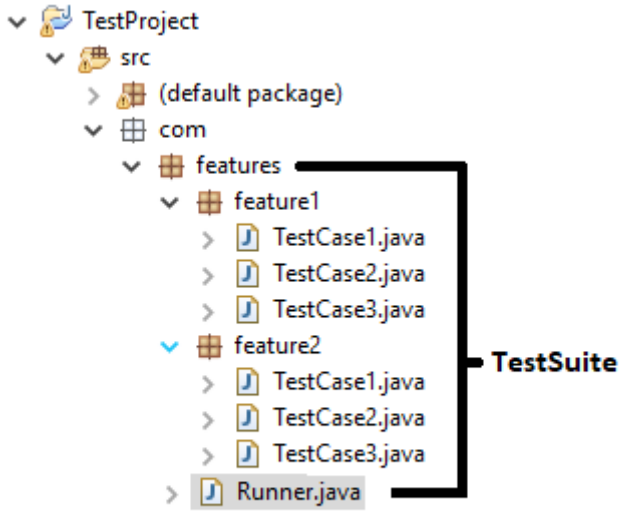
3.1.1 Feature Structure

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



3.1.2 Super Structure

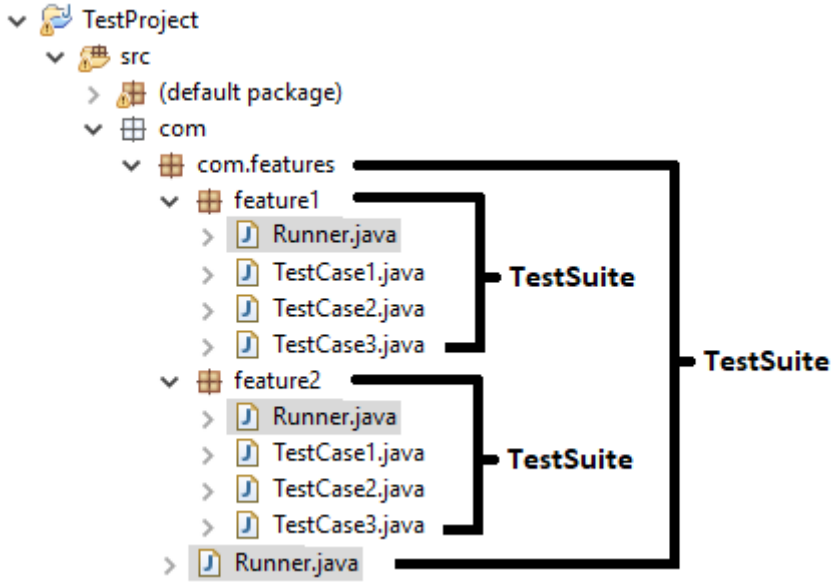
- Packages and sub-packages are organized based on features.
- Project contains a single Runner and all test cases are within Runner's scan scope thus entire project acts as single test suite.
- This can also be achieved by having Runner at project root location.



3.1.3 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

FOUR

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 4.1: 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

FIVE

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 5.1: 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

SIX

RUN TEST PROJECT

ARTOS can be run via

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

6.1 Command line

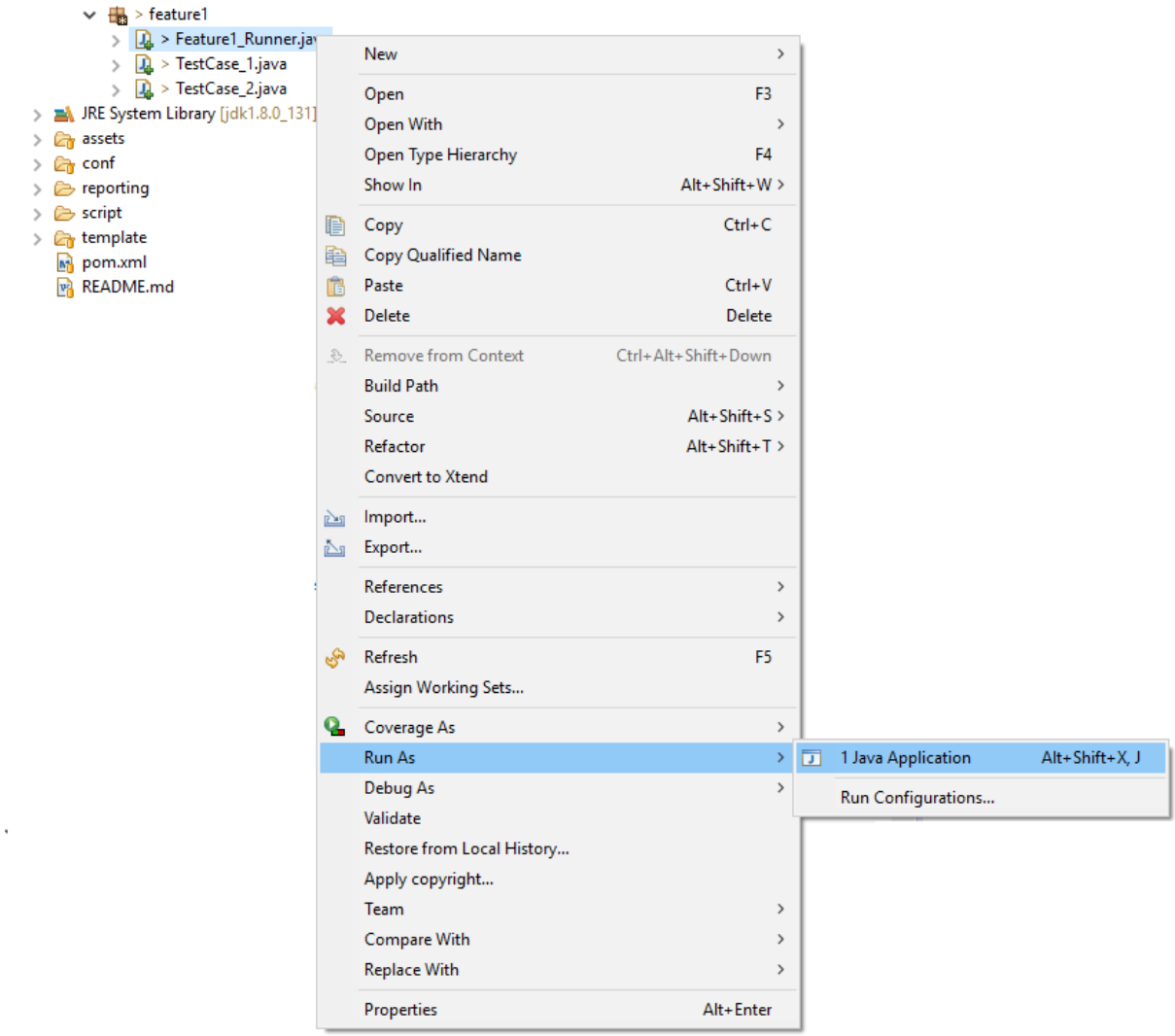
- Artos can be executed via command line by specifying minimum of
  - All library in the class path (Specify all dependencies)
  - Test runner class for given test suite (Starting point of test application)
  - Test script for given test suite (Contain all instructions to execute test suite correctly)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

6.2 Eclipse IDE

6.2.1 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

SEVEN

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

7.1 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>
```

7.2 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);
        }
}
```

7.3 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.

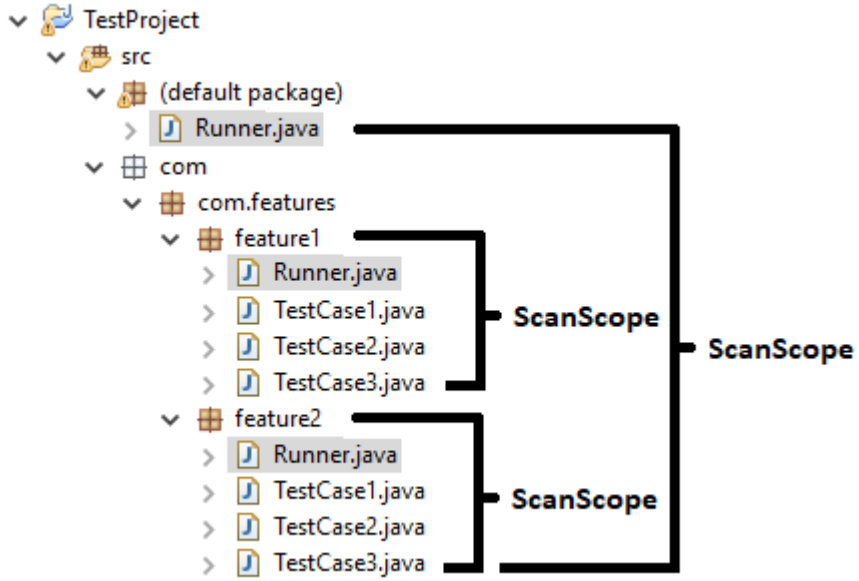
CHAPTER

EIGHT

TEST SUITE & TEST RUNNER

8.1 Scan Scope

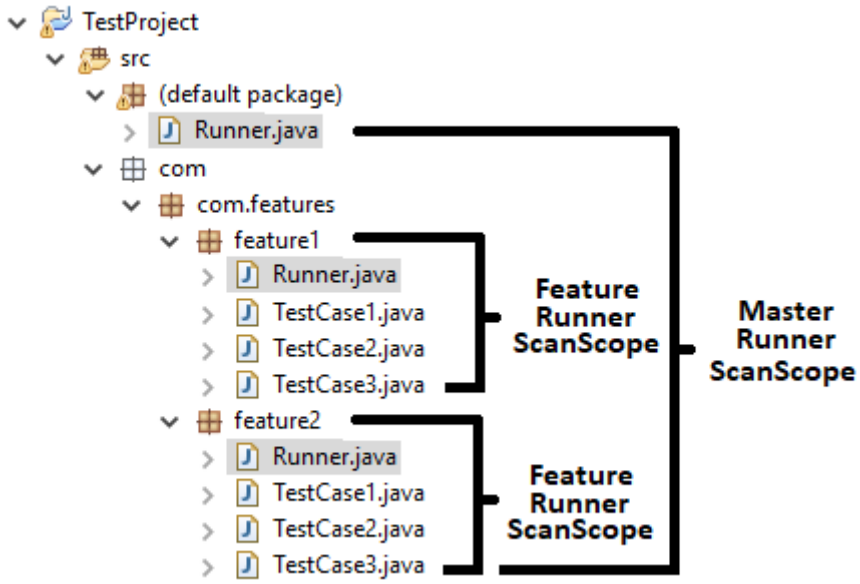
A section of Java project is scanned when test application is launched. A class that initiates scan is called a **Runner**. A package containing Runner class and it's child packages are scanned by the Runner in search of test cases, thus scanned section of the project is called a **Scan Scope** of the Runner.



8.2 The Runner

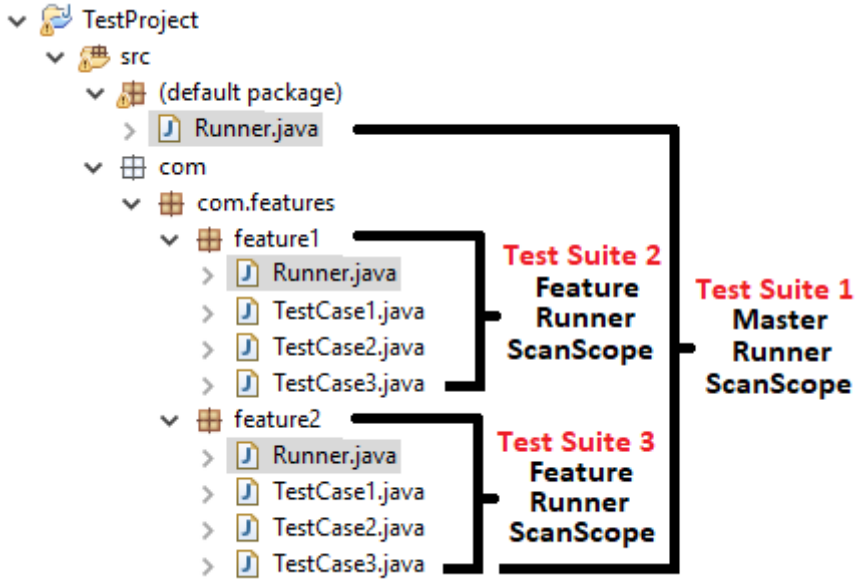
- A Runner is the entry point to a test application.
- A Runner at project root location[1] is called a **Master Runner** which has visibility of all test cases within a project.
- A Runner created within individual package is called a **Feature Runner** which has visibility inside its own package or 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



8.3 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 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

NINE

USE 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/unit executed without any errors
SKIP	1	Test case/unit execution is skipped
KTF	2	Test case/unit is known to fail
FAIL	3	Test case/unit failed

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

Recommended

Add short description during every status update.

Listing 9.1: : In this example test status is updated multiple time in single test unit. The most sever update out of all status updates will be considered as test unit outcome. In this example sever status update is **TestStatus.FAIL** so test unit outcome will be **FAIL**. Because there is only one test unit in the test case, the test case outcome is also **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         context.setTestStatus(TestStatus.PASS, "Test flow is as expected");
18
19         // TODO : logic goes here..
20         context.setTestStatus(TestStatus.PASS, "Test flow is as expected");
21
22         // TODO : logic goes here..
```

(continues on next page)

(continued from previous page)

```
        context.setTestStatus(TestStatus.FAIL, "Test flow is not as expected");

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

9.1 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 9.2: : In this example test outcome for each test unit is different. The most sever outcome among all test units will be considered as a test case outcome. In this example sever outcome is **TestStatus.FAIL** so test case outcome will be **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 = "ArpitS", 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..
        context.setTestStatus(TestStatus.PASS, "Test passes");
        // ------------------------------------------------------
    }

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

```

(continues on next page)

(continued from previous page)

```
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

TEN

TEST CONTEXT

A `TestContext` is the Java object which holds all required information about test suite and it is globally available. TestContext is unique per test suite. All test cases have access to TestContext object. A test case can get, set, update or store required object using TestContext. Some of the useful methods are listed below:

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

10.1 Context example

Listing 10.1: : 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

ELEVEN

FRAMEWORK CONFIGURATION

The `framework_configuration.xml` file is used to configure ARTOS framework, that is located inside `./conf` directory. ARTOS generates default configuration file in the same directory if file is not present.

Listing 11.1: : Default configuration file

```
<?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)

```
42   </features>
43 </configuration>
```

11.1 <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.

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

11.2 <logger>

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

Attribute

Name	Description
profile	profile name

Properties

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

11.3 <smtp_settings>

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

Attribute

Name	Description
profile	profile name

Properties

[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
```

```
// 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
```

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`

11.4 <features>

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

Attribute

Name	Description
profile	profile name

Properties

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

CHAPTER

TWELVE

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.

12.1 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

THIRTEEN

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 13.1: : Sample test case and test unit with ***Importance Indicator***

```
// This test case overall is of a CRITICAL importance
@TestImportance(Importance.CRITICAL)
@TestPlan(preparedBy = "Arpit", preparationDate = "18/02/2019", bdd = "GIVEN..WHEN..AND..THEN..")
@TestCase
public class Test_TestUnit_Importance implements TestExecutable {

        @TestImportance(Importance.CRITICAL)
        @Unit(sequence = 1)
        public void testUnit_1(TestContext context) {
                // ------------------------------------------------
                context.setTestStatus(TestStatus.FAIL, "This is a CRITICAL importance test unit");
                // ------------------------------------------------
        }

        @TestImportance(Importance.LOW)
        @Unit(sequence = 2)
        public void testUnit_2(TestContext context) {
                // ------------------------------------------------
                context.setTestStatus(TestStatus.FAIL, "This is a LOW importance test unit");
                // ------------------------------------------------
        }

        @TestImportance(Importance.MEDIUM)
        @Unit(sequence = 3)
        public void testUnit_3(TestContext context) {
                // ------------------------------------------------
                context.setTestStatus(TestStatus.FAIL, "This is a MEDIUM importance test unit");
                // ------------------------------------------------
        }

}
```

}

13.1 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]
********************************************************************************
```

13.2 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

FOURTEEN

LOGGING FRAMEWORK

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

14.1 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.

```
[INFO ][2019-02-06_18:50:11.025][main] -

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
      ___
     / _ \                          __ _ _
    | |_| |__ _   _   __ _  ____  \ \ \ \
    |  _  |  __|_| |_ / __ \/ ____|  \ \ \ \
    | | | | |   |_| |_| (__) )___  \  ) ) ) )
    |_| |_|_|    | |_ \__ _/| ___ /  / / / /
    ============\__/============== (_(_(_(
    == Artos ==  (RELEASE v0.0.6)

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

[INFO ][2019-02-06_18:50:11.025][main] -
************************************ 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 ********************************************
[TRACE][2019-02-06_18:51:05.648][pool-17-thread-1] - Res: 4869204920616D20436C69656E742030
[TRACE][2019-02-06_18:51:05.664][pool-3-thread-1] - Req: 4869204920616D20536572766572 2030
[TRACE][2019-02-06_18:51:05.664][pool-17-thread-1] - Res: 4869204920616D20436C69656E742031
[TRACE][2019-02-06_18:51:05.680][pool-3-thread-1] - Req: 4869204920616D20536572766572 2031
[TRACE][2019-02-06_18:51:05.680][pool-17-thread-1] - Res: 4869204920616D20436C69656E742032
[TRACE][2019-02-06_18:51:05.695][pool-3-thread-1] - Req: 4869204920616D20536572766572 2032
[TRACE][2019-02-06_18:51:05.695][pool-17-thread-1] - Res: 4869204920616D20436C69656E742033
[TRACE][2019-02-06_18:51:05.711][pool-3-thread-1] - Req: 4869204920616D20536572766572 2033
[TRACE][2019-02-06_18:51:05.711][pool-17-thread-1] - Res: 4869204920616D20436C69656E742034
[TRACE][2019-02-06_18:51:05.726][pool-3-thread-1] - Req: 4869204920616D20536572766572 2034
[TRACE][2019-02-06_18:51:05.726][pool-17-thread-1] - Res: 4869204920616D20436C69656E742035
[TRACE][2019-02-06_18:51:05.742][pool-3-thread-1] - Req: 4869204920616D20536572766572 2035
[TRACE][2019-02-06_18:51:05.742][pool-17-thread-1] - Res: 4869204920616D20436C69656E742036
[TRACE][2019-02-06_18:51:05.758][pool-3-thread-1] - Req: 4869204920616D20536572766572 2036
[TRACE][2019-02-06_18:51:05.758][pool-17-thread-1] - Res: 4869204920616D20436C69656E742037
[TRACE][2019-02-06_18:51:05.773][pool-3-thread-1] - Req: 4869204920616D20536572766572 2037
[TRACE][2019-02-06_18:51:05.773][pool-17-thread-1] - Res: 4869204920616D20436C69656E742038
[TRACE][2019-02-06_18:51:05.789][pool-3-thread-1] - Req: 4869204920616D20536572766572 2038
[TRACE][2019-02-06_18:51:05.789][pool-17-thread-1] - Res: 4869204920616D20436C69656E742039
[TRACE][2019-02-06_18:51:05.805][pool-3-thread-1] - Req: 4869204920616D20536572766572 2039
[TRACE][2019-02-06_18:51:05.805][pool-17-thread-1] - Res: 4869204920616D20436C69656E74203130
[TRACE][2019-02-06_18:51:05.820][pool-3-thread-1] - Req: 4869204920616D20536572766572 203130
[TRACE][2019-02-06_18:51:05.820][pool-17-thread-1] - Res: 4869204920616D20436C69656E74203131
[TRACE][2019-02-06_18:51:05.836][pool-3-thread-1] - Req: 4869204920616D20536572766572 203131
```

14.2 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.

14.3 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`

14.4 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.

14.5 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.

14.6 Log Rollover Policy

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

14.7 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.

14.8 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();`

14.9 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();`

14.10 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 ***********
*********************************
```

14.11 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");
```

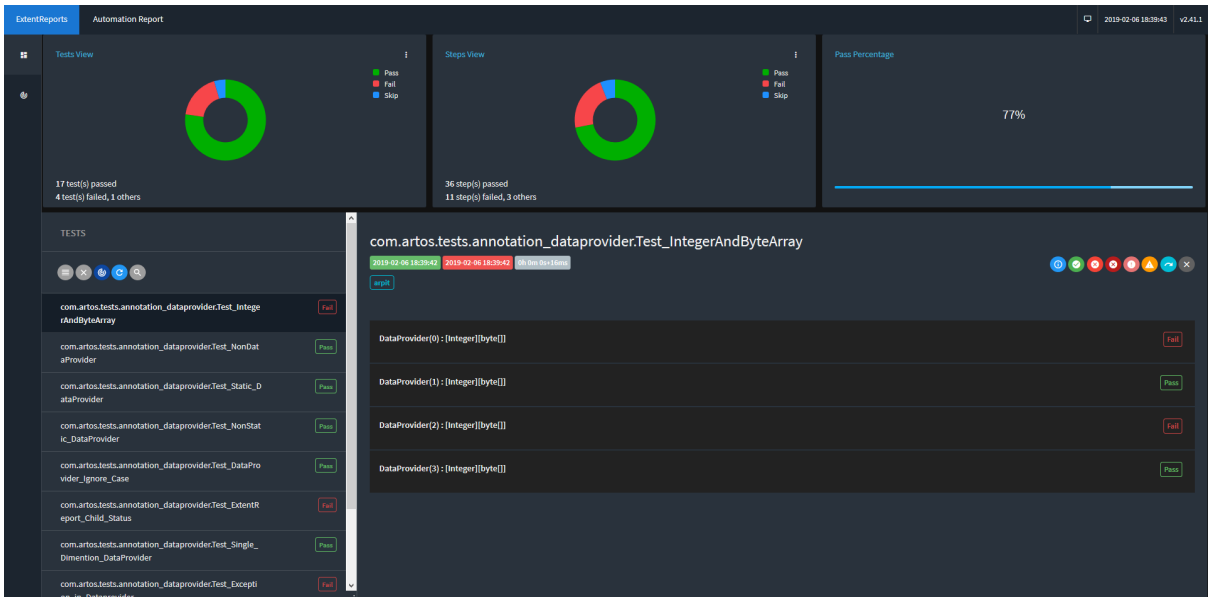
Warning: Parameterized logging is less efficient compare to string concatenation, if test application does not use multiple log levels then it is recommended to avoid parameterized logging. Parameterized logging overall improves performance in case where test application utilities multiple log levels 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

FIFTEEN

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.

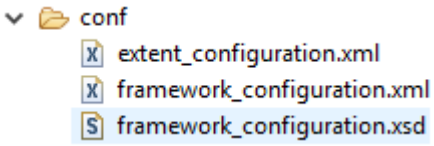


CHAPTER

SIXTEEN

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

SEVENTEEN

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

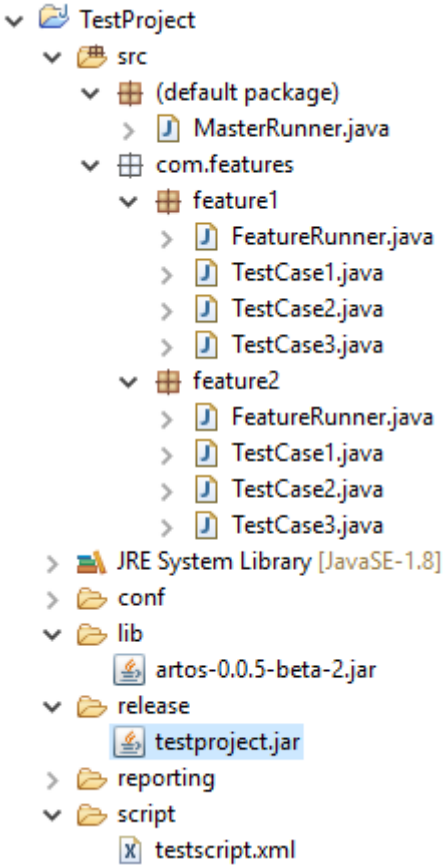
17.1 Example 1: Run from compiled classes

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

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

17.2 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"
```

17.3 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

EIGHTEEN

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 18.1: 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>
```

18.1 <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.

18.2 <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.

18.2.1 </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.

18.3 <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 18.2: 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>
```

18.4 <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 18.3: 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>
```

18.5 <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 18.4: All listed test cases will be added to execution list

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

Listing 18.5: 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>
```

18.6 <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 18.6: All test units will be added to execution list

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

Listing 18.7: Test units belongs to “Fast” OR “Slow” test units will be added to execution list

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

18.7 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

NINETEEN

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 19.1: 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>
    <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="*"/>
```

(continues on next page)

(continued from previous page)

```
37      </testunitgroups>
38    </suite>
39
40  </configuration>
```

CHAPTER

TWENTY

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

20.1 Simple server

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

Listing 20.1: : Simple server example

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

20.2 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 20.2: : Simple server with timeout example

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

20.3 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 20.3: : 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 20.4: : 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();
```

20.4 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 20.5: : Message parser example

```
public class MsgParser4ByteLength implements ConnectableMessageParser {
        Transform _transform = new Transform();
        byte[] leftOverBytes = null;
        List<byte[]> msgList = null;

        @Override
        public byte[] getLeftOverBytes() {
                return leftOverBytes;
        }

        @Override
        public List<byte[]> parse(byte[] data) {
                // reset variable before use
                msgList = new ArrayList<>();
                leftOverBytes = null;

                deserializeMsg(data);

                return msgList;
        }

        private void deserializeMsg(byte[] data) {
                // Check if at least length can be worked out
                if (!sufficientDataForLengthCalc(data)) {
                        leftOverBytes = data;
                        return;
                }

                // Check if message can be constructed
                if (!sufficientDataForMsg(data)) {
                        leftOverBytes = data;
                        return;
                }

                // Extract one complete message
                byte[] leftOvers = extractMsg(data);

        // process leftOver bytes to see if anymore messages can be extracted
                if (null != leftOvers) {
                        deserializeMsg(leftOvers);
                }
        }

        // 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));

```

(continues on next page)

(continued from previous page)

```
        // 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 20.6: : 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();
```

20.5 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 20.7: : RealTime Event Listener example

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

CHAPTER

TWENTYONE

@BEFORETESTUNIT @AFTERTESTUNIT

21.1 @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.

21.2 @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

TWENTYTWO

@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.

22.1 Annotation use case(s)

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

22.2 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

TWENTYTHREE

@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

23.1 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..")
```

23.2 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 logs");
                // ------------------------------------------------------
        }
}
```

- Log file snapshot for above test case.

```
**************************************************************************
Test Name       : com.tests.TestCase_1
```

(continues on next page)

(continued from previous page)

```
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

TWENTYFOUR

@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

24.1 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

24.2 Annotation use cases

```
1  // Single exception comparison
2  @ExpectedException(expectedExceptions = { NullPointerException.class })
3  // Single exception + exception message string comparison
4  @ExpectedException(expectedExceptions = { NullPointerException.class,␣
   ↪InvalidDataException.class }, contains = "exception example")
5  // Single exception + exception message matching with regular expression
6  @ExpectedException(expectedExceptions = { NullPointerException.class }, contains =
   ↪"[^0-9]*[12]?[0-9]{1,2}[^0-9]*")
7
8  // 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]*")
```

24.3 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
        // classes and description. 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 = 3)
        @ExpectedException(expectedExceptions = { Exception.class, NumberFormatException.class }, contains = "This is a test code")
        public void testUnit_3(TestContext context) throws Exception {
                // -------------------------------------------------
                // test logic goes here..
                throw new Exception("This is a test code");
                // -------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
        }

        // Code demonstrates how to specify multiple expected exception
        // classes and description using Regular expression. 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 = 4)
        @ExpectedException(expectedExceptions = { Exception.class,␣
↪NumberFormatException.class }, contains = ".*\\btest\\b.*")
        public void testUnit_4(TestContext context) throws Exception {
                // -------------------------------------------------
                // test logic goes here..
                throw new Exception("This is a test code");
                // -------------------------------------------------
        }

        // Code demonstrates how to specify exception but do not enforce
        // fail in absence of exception. If Exception is thrown then it
        // will be matched with expectedException class. If Exception
        // will not be thrown then test will continue execution and PASS
        // eventually
        @Unit(sequence = 5)
        @ExpectedException(enforce = false, expectedExceptions = { Exception.class,
↪ NumberFormatException.class })
        public void testUnit_5(TestContext context) throws Exception {
                // -------------------------------------------------
                // test logic goes here..
                context.getLogger().info("This test does not throw any exception");
                // -------------------------------------------------
        }

        // This will allow user to continue executing rest of the code
        // in case of exception. Guarding against wrong flow will help
        // user throw exception in case code did not do what was expected
        @Unit(sequence = 6)
        public void testUnit_6(TestContext context) throws Exception {
                // -------------------------------------------------
                try {
                        // Converting String into Integer should throw an error
                        Integer.parseInt("Test");

                        // Protects against code traveling in wrong direction
                        Guard.guardWrongFlow("Expected exception but did not occur
↪");

                } catch (NumberFormatException e) {
                        if (!e.getMessage().equals("For input string: \"Test\"")) {
                                throw e;
                        }
                }

                context.getLogger().info("Do something..");
                // logic goes here..
                // -------------------------------------------------
        }
}
```