Document Number and Revision: 911-2411 Rev 11

Oracle Manufacturing Data Transfer Specification

Overview

This document is intended as a reference on how test result data should be structured and delivered from the manufacturing test process, into the Oracle Test Data Management System known as TDMS.

Audience

This document is targeted at test and manufacturing systems software engineers who are required to collate test results for TDMS delivery. A fundamental knowledge of XML, networking and software design is mandatory before attempting to construct the result data.

Table of Contents

1. ACRONYMS

The following details the acronyms which you come come across within the body of this document.

1. **TDMS Central: Central server**
   This is the master TDMS server residing within the Oracle Data Center. This server is responsible for pulling data from remote manufacturing sides running satellite TDMS instances or sites which are autonomous, running their own test systems but providing results which are TDMS compatible. The central server will tunnel to each of the satellite TDMS servers and pull test result data, via an internet connection, or will access the test results within a cloud store.

2. **TDMS: Test Data management System**
   The Test Data Management system is software developed by Oracle which allows for collection, collation and presentation of test result data generated as part of the manufacturing process when building Oracle products.
   NOTE: The test results stored within TDMS must be stored as a legal requirement for a period of 10 years.

3. **PO: Purchase Order**
   A Purchase Order which initiates the whole process of building a product. The Purchase order relates to a sales order that is raised for the customer. Within the purchase order are details of the work orders, where a work order is the order used on the shop floor to manufacturer a system and associate a serial number.

4. **WIP: Work in Progress**
   When a serial number is assigned to a product, based on the Purchase order, this is referred to as Work In Progress.

5. **ATS: Automated Test System**
   An automated system used to test products under test.

6. **SFTP: Secure File transfer protocol**
   Protocol used to transfer files over network in a secure manner.

7. **XML: Extensible Markup Language**
   Method of encoding data into a text format using elements and attributes
   .

8. **JSON: JavaScript Object Notation**
   JavaScript syntax for encoding structured data into text format.

9. **XSD: XML Schema Definition**
   The definition file for generating XML documents. The XSD defines the syntax and grammar for the XML files.

10. **XSLT:** Extensible Stylesheet Language Transformations)
    Definition language used to parse and process XML documents into other XML documents or other similar formats such as HTML, plain text
    .

11. **Nexus (*aka ATS+*):** Oracle Test Automation Software
    Nexus is Oracle's software used to automate the testing process and is used within all our manufacturing facilities, and some of our external partnered manufacturing sites.

12. **VTS:** Oracle Hardware Validation Test
    Oracle Hardware Test validation suite.
13. **FRU:** Field Replaceable Unit
    An item such as memory or a drive can be referred to as a FRU part as this can be replaced in the field, if a failure should occur.
14. **SITEID:** the unique site identifier.
    Each site within TDMS is given a unique site identifier which is an Integer type. When identifying a data source, this identifier should be used.

2. INTRODUCTION/OVERVIEW

When testing a product in manufacturing, data will be produced during the normal operation of test, and ultimately processed and loaded into the central TDMS database. This test data, if to be loaded into the database, should be formatted as XML. This XML should adhere to an XSD specification document, supplied by Oracle as a set to accompany this document (currently, the preferred method for XSD file distribution is via the supplier Secure Site portal with corresponding notification of change and expected action via email). These XML results will then be packaged into files which are then transferred to Oracle, using one of the methods outlined within this document.

3. KEY CONCEPTS OF A TEST

When manufacturing a product at a site (identified to TDMS via a SITEID), it is assumed that the product will flow through a set of operations (known as a *LOGICALOPERATION*), and each operation will have an outcome (known as STATUS, which can have a value of P-pass   F-fail or A-Aborted). The complete audit of the test being carried out, known as the *TESTSUITE*, details the full set of steps (TESTSETS) executed to test the product. A TESTSET must have a unique name within a TESTSUITE but there can be N TESTSET(s) within the TESTSUITE.

The complete test result itself is identified by a unique global identifier (GUTI, globally unique test identifier) and a start time (*TESTSTARTTIME*). These 2 elements coupled with the SITEID, constitute a compound primary key for a test result.

(**Note** , *Where possible, please attempt to make the GUTI a unique identifier, per test run . Though TDMS can accommodate multiple test start times for the same GUTI, it is advised that each test result should have a different GUTI*).

It is usual that if a unit should stay within a test location for multiple operations, the GUTI can stay the same and the TESTSTARTTIME time and OPERATION change. I.e. if a unit were to carry out multiple operations at the same physical location.

The product being tested will have a unique *SERIALNUMBER* and have a PARTNUMBER assigned and will also belong to a particular type of product FAMILY.

A LOGICAL OPERATION (LOGOP) will have a unique name within the manufacturing process (i.e. there is only one SFT operation, one IST operation, one HIPOT etc.), and will be executed at a particular unique shop floor location at the site (known as the TESTLOCATION).

The LOGOP could be automated, executed by a TESTSEQUENCER, or carried out manually, whereby the operation is carried out by a physical test operator who will have an employee identifier (OPERATORID).

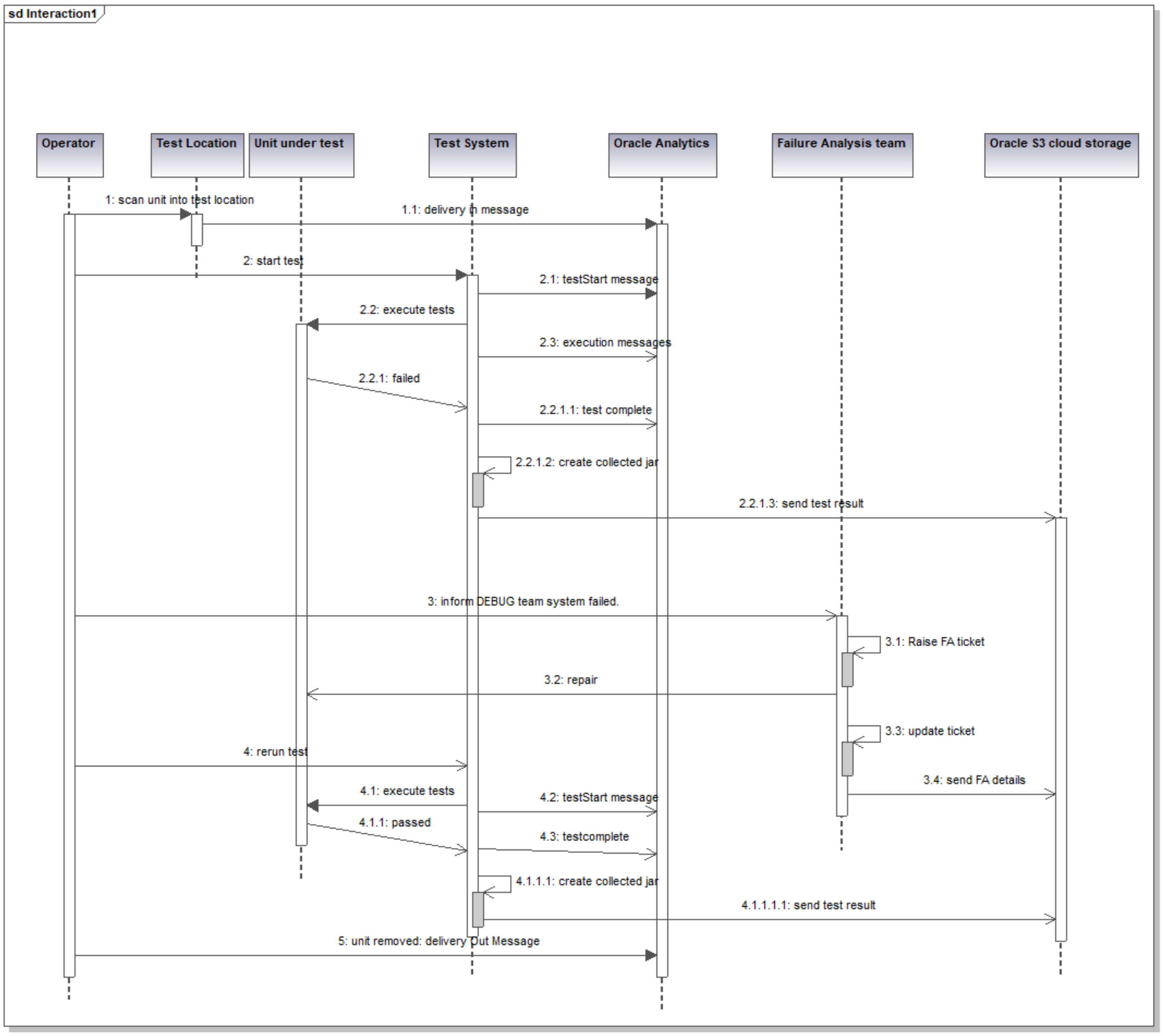
The LOGOP will have a duration, calculated by the complete time – start time (TESTSUITECOMPLETE – TESTSTARTTIME) and a final STATUS (of P F or A) . A test result may not have an empty result.

1 WORKFLOW OF A TEST

When a test is carried out on a Unit, it is assumed that at the end of the test, no matter the outcome, a result will be created and made available to Oracle.

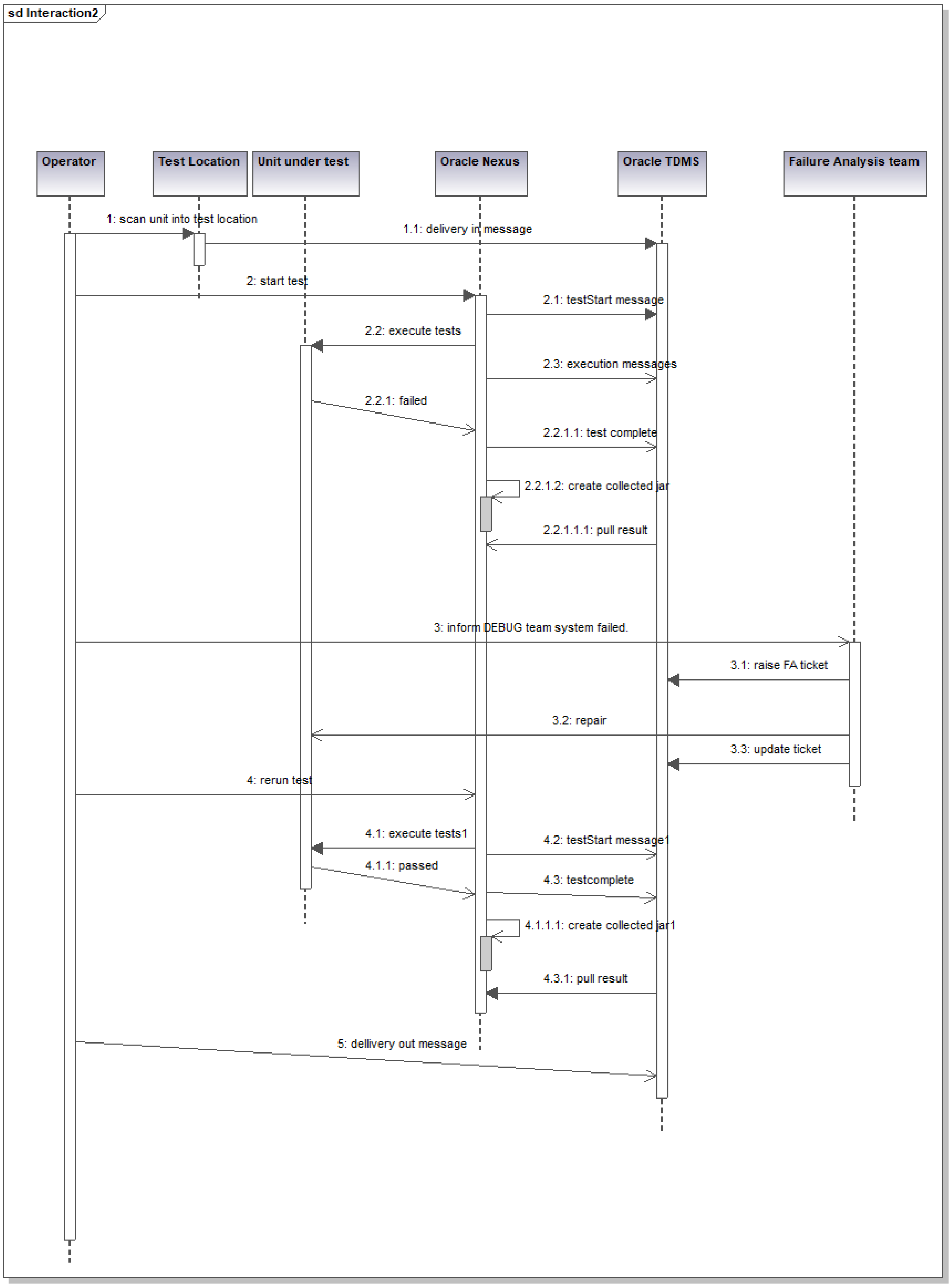
1.1 Workflow when Site is Independent of Oracle

If you are operating within a site which has no TDMS instance and no local Oracle owned test process, then the process steps and information flow to TDMS should be similar to the following.



1.2 Workflow for an integrated site.

If you are working within a site which has a locally installed TDMS and an Oracle Nexus test process, then the information flow would be like the following.


sd Interaction2
Operator
Test Location
Unit under test
Oracle Nexus
Oracle TDMS
Failure Analysis team
1: scan unit into test location
1.1: delivery in message
2: start test
2.1: testStart message
2.2: execute tests
2.3: execution messages
2.2.1: failed
2.2.1.1: test complete
2.2.1.2: create collected jar
2.2.1.1.1: pull result
3: inform DEBUG team system failed.
3.1: raise FA ticket
3.2: repair
3.3: update ticket
4: rerun test
4.1: execute tests1
4.2: testStart message1
4.1.1: passed
4.3: testcomplete
4.1.1.1: create collected jar1
4.3.1: pull result
5: dellivery out message

2 DATA ITEM DEFINITIONS

Within the data being sent to Oracle are many data items, however there are a few which merit focus to ensure clarity of what is expected when populating with test details.

2.1 STATUS

When a test completes, the test result could be a PASS a FAIL or the test itself could have been ABORTED. However, it is important that although the test may have been aborted, that this be captured, along with a reason for the result. The data would be placed into the TEST/PROCESS/TESTSUITE/RESULT element, with the result having their own element type

1- <PASS />

2- <FAIL />

3- <ABORTED />

2.2 GUTI

Each test result will have a unique identifier which will be called the GUTI (Globally Unique Test Identifier) The definition of which is as follows:

- Maximum size of 36 characters
- Zero padded up to 36 characters if value is less than 36 characters in length
- Alphanumeric character set, both upper and lower case. (only a-z, A-Z, 0-9, -,)

You should assign a GUTI as a unit enters a test location and reset the GUTI when the unit leaves the location. Each test run will provide as stated, compound key of GUTI+TESTSTARTTIME as a unique reference to the result data for the operation(s) being executed.

Though it is possible for a unit under test to have the same GUTI but different test start times such that there are multiple test runs being performed under the same GUTI , it is advised that a GUTI be generated for each test run thus making the GUTI unique to the test being ran

Note: Each test run should produce a separate test result for TDMS submission.

2.3 TESTSUITE

The TESTSUITE will contain the TESTSETS which make up the complete operation being performed. The TESTSUITE will contain N number of TESTSETS but each TESTSET within the TESTSUITE must be unique.

The TESTSUITE will have an overall result of the test being executed and a meaningful message to indicate a reason behind the result.

```
e.g.
RESULT=F ( Failed )
MESSAGE=”Dimm failure due to Errorcode 1” ,
RESULT=P ( Passed )
MESSAGE=”Passed All Testsets”
```

2.4 TESTSET

As previously stated, a TESTSUITE contains 1-N TESTSET elements, where a TESTSET is a unique item (identified by its name). There can never be duplicate test set names as TDMS cannot process a test run with a non-unique testset. Names such as PRETEST, MAINTEST, MEMTEST, CPUSRESSTEST_1, POSTTEARDOWN are expected by TDMS where the name does not contain spaces and adheres to one of: uppercase, lowercase, or camel case.

TESTSET names should not contain spaces. If this is required, then the “underscore _ “character should be adopted.

Note: If you should require to execute the same testset more than once during an operation, it is acceptable to append a numeric counter to the name of the test set, such as “powerCycleTestSet_1” .

Within the TESTSET and also TESTEVENT you must be sure that your ordering of the XML elements is correct and is in line with the details of the XSD. If you are unsure as to what the ordering should be it is advised that you take the XSD supplied and generate a sample XML file (there are many tools available online which provide this feature).

2.5 TESTEVENT

A TESTSET can have 0-N TESTEVENT element, with each element adhering to the same naming constraint to that of the TESTSET. The TESTEVENT is intended as a more fine-grained method of audit tracking a test. An example use case being a test set such as TESTNETWORK, and the TESTEVENT(s) carrying out various types of network tests such as loopback, UDP testing, TCP testing etc.

3 MANDATORY DATA ITEMS

1. **Serial number**
   XPATH: TEST/PRODUCT/UNIT/SERIALNUMBER, TEST/PRODUCT/UNIT/@name
   Each product being tested must have a *serial number* by which the product can be identified. This number is unique.

2. **Part number:**
   XPATH: TEST/PRODUCT/UNIT/PARTNUMBER
   Each product should belong to a particular part number. A family of products may have 1-N part numbers.

3. **Test Location**
   XPATH: TEST/PRODUCT/UNIT/TESTLOCATION
   The location of where the operation was performed. This test location should be unique within the facility that is producing the product.

4. **Family of product that is being tested**
   XPATH: TEST/PRODUCT/UNIT/FAMILY
   The family of the product. Whereby a family is a common set of products.

5. **Logical Operation**
   XPATH: TEST/PRODUCT/UNIT/OPERATION
   The operation that is being performed at the location. However, it is possible that there may be multiple operations carried out at the same location, without the unit under test physically moving.

6. **GUTI:**
   XPATH: TEST/@GUTI
   Globally unique test identifier. This is a 36-character max length, unique identifier which identifies the test being performed.

7. **Test Suite:**
   XPATH: TEST/PROCESS/TESTSUITE
   When a test result is being generated, a single test suite will indicate the time of the test, the duration , and the overall result. The test suite is the collection of TESTEVENT(s) which were executed upon the system being tested.

8. **Test Set**
   XPATH: TEST/PROCESS/TESTSUITE/TESTSET
   There should be at a minimum, 1 test set within a test suite. The test set is a record of the test which was carried out within the test suite.

9. **Messages:**
   XPATH: TEST/PROCESS/TESTSUITE/RESULT
   When a test fails, you should make every attempt to populate a meaningful message into the **TESTSUITE** and test **TESTEVENT** to indicate what exactly went wrong during the test.

10. **Target:**
    XPATH: TESTPRODUCT/ADDITIONALDETAILS/TARGET
    When we are testing the product, the area within the product which the test is targeting should be specified. For example, if this is a system wide test, then the target value should be set to "**SYSTEM**".

If we're testing *motherboards,* then this value may be set to BOARD etc. The target value should be dictated by the product engineer.

11. **Buildid**
    XPATH: TEST/PROCESS/TESTSUITE/DEFINITIONS/BUILDID
    The build identifier is used to specify an identifier for the system such that it is easy to identify P1, P2 and full production builds. This is useful to filter test results. The value is usually designated by the product engineers within Oracle. However, sample values for this identifier would be RR, P1, P2, REPAIR.

12. **isProductionRun**
    XPATH: TEST/PRODUCT/RELEASE/@isProductionRun
    This attribute is used to indicate if the test result is an actual production result or can be ignored as it was part of a pilot build, or a test process qualification.

13. **SEQUENCER**
    XPATH: TEST/SEQUENCER
    To provide tracking of the engine used to execute the tests, it is requested that the details of the test sequencer used to test the product be provided as part of the test result. This provides the ability to trace defects within the process and to trace process failures and software updates.

14. **RELEASE**
    XPATH:
    To provide tracking the version details in respect to the test operations. This differs from the SEQUENCER in that the SEQUENCER would run the test software, and the test software carries the actual tasks on the system under test.

4 DATE TIME FORMATTING

All date/times within the TDMS XML data must adhere to the ISO-8601 format.

e.g.

```
YYYY-MM-DDTHH:mm:ss.sssZ
```

It is expected that the times will be UTC, with the time zone recorded within the time Zone attribute of the test result. However, if the "Z" option is removed and replaced with a time zone offset, this is also valid.

Please be aware that all times produces during test run should be millisecond resolution.

Additionally when specifying ISO compliant dates, it is assumed that the last character of the string is always a Z. (Uppercase Z) .

The format for ISO8601 would be as follows:

`YYY-MM-DDThh:mm:ss[.mmm]TZD` *(e.g. 2012-03-29T10:05:45.987-06:00)*

Taking a small example string: `2022-11-09T01:25:01.871Z` or `2022-1109T01:25:01.871+05:00`

This would provide us with a valid timestamp when converted to a java.sql.Timestamp.

The first example indicates that this time is UTC, however the second indicates that we are offset from UTC by 5 hours. (UTC with timezone).

Note: TDMS only accepts ISO 8601 compliant time values to millisecond precision.

5 TEST RESULT DELIVERY INTO TDMS

Once a test completes the test result will require packaging into a format which TDMS can process. The format expected by TDMS is XML due to its ability to enforce strict grammar checking.

The XML test result representation is bundled into a JAR formatted (see section 9).

It is also possible to add additional data and attach to the overall result. This data inclusion is covered within section.6

Requirement for inclusion of this data should be addressed on a product-by-product basis and would be specified by Oracle's Product and Test Engineers. Please refer to Oracle engineers for guidance when structuring your data to ensure that you understand the expectation level for data collection.

5.1 Structure of the data

As stated, all data from a test result, must be formatted as XML.

The XML from your test run should be placed into a file which adheres to the TESTRESULT.xsd which you should have been provided.

This topmost xml test result file should be placed into a file, named after the serial number of the product under test e.g.

```
> 1111M8888.xml
```

Within this XML, there will be following sections.

1. **TESTSUITE**
   When you run a test, it is referred to as a TEST SUITE. Where it will have a start and complete time, and ultimately an overall outcome of PASS FAIL or ABORTED.
2. **TESTSET**
   Within the TESTSUITE, there will be 1-N TESTSET(s), where a test set has its own outcome of PASS FAIL ABORT.
3. **TESTEVENT**
   If you require a finer level of granularity, you can define each sub level step within a TESTSET such that a TESTSET can have 0-N TESTEVENT(s).
4. **PRODUCT**
   This section details specific pieces of information related to the product under test such as its family, MAC address etc.
5. **PROCESS**
   This section details information related to the actual test process being carried out. Such as the revision of the software being used, the area within the PRODUCT that this operation is targeting such as the SYSTEM, MOTHERBOARD, NETWORKCARD etc. An operation my target an area or areas within the system under test.

If we refer to the PROCESS element of the XSD, you can see the positional mapping of the above elements.

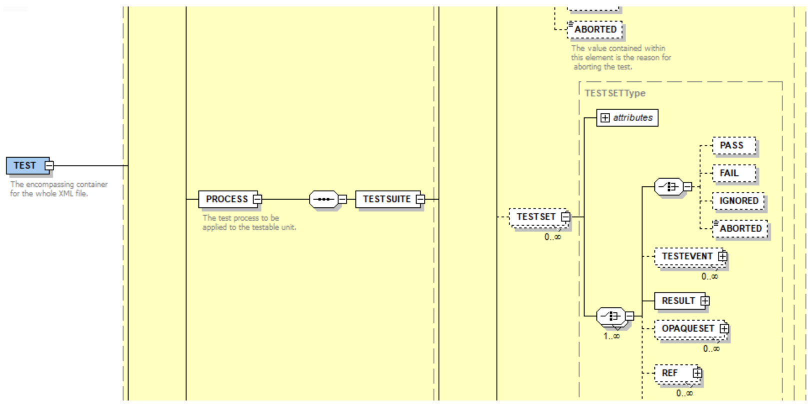
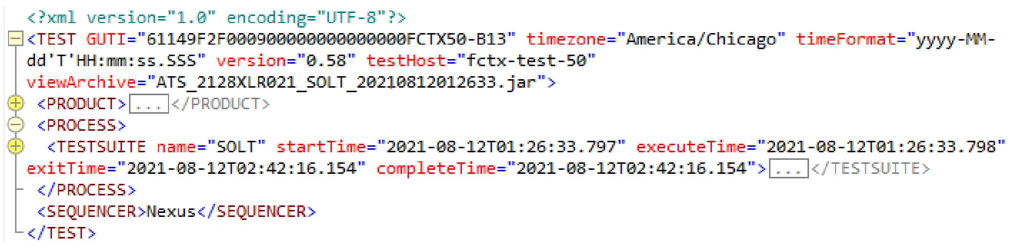

TEST
The encompassing container for the whole XML file.
PROCESS
The test process to be applied to the testable unit.
TESTSUITE
TESTSET
0..∞
ABORTED
The value contained within this element is the reason for aborting the test.
TESTSETType
attributes
PASS
FAIL
IGNORED
ABORTED
TESTEVENT
0..∞
RESULT
1..∞
OPAQUESET
0..∞
REF
0..∞

Illustration 1: The top level TEST element with Sub elements

Generating XML from the XSD, will produces a high level structure such as the following:


```
<?xml version="1.0" encoding="UTF-8"?>
<TEST GUTI="61149F2F000900000000000000FCTX50-B13" timezone="America/Chicago" timeFormat="yyyy-MM-
dd'T'HH:mm:ss.SSS" version="0.58" testHost="fctx-test-50"
viewArchive="ATS_2128XLR021_SOLT_20210812012633.jar">
 <PRODUCT>...</PRODUCT>
 <PROCESS>
  <TESTSUITE name="SOLT" startTime="2021-08-12T01:26:33.797" executeTime="2021-08-12T01:26:33.798"
exitTime="2021-08-12T02:42:16.154" completeTime="2021-08-12T02:42:16.154">...</TESTSUITE>
 </PROCESS>
 <SEQUENCER>Nexus</SEQUENCER>
</TEST>
```

In the above you can see that the key ELEMENTS are

- PRODUCT: detailing the actual physical things we are testing.
- PROCESS: detailing the steps that took place to test the product.
- SEQUENCER: the software used to execute the tests.

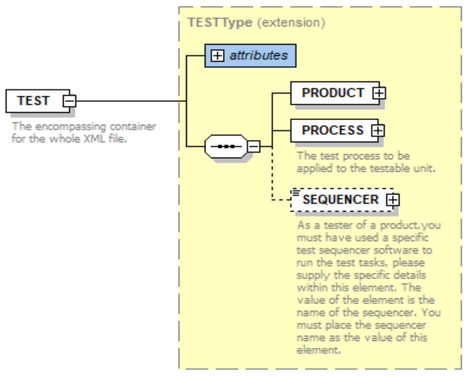

TESTType (extension)
attributes
TEST
The encompassing container for the whole XML file.
PRODUCT
PROCESS
The test process to be applied to the testable unit.
SEQUENCER
As a tester of a product, you must have used a specific test sequencer software to run the test tasks, please supply the specific details within this element. The value of the element is the name of the sequencer. You must place the sequencer name as the value of this element.

Figure 1 Test Schema Top level elements

If we then look at the top-level TEST element in a little more detail. Specifically, its attributes.

Note that the GUTI must be provided as this is our unique key for the test execution.

GUTI	The unique identifier.
timezone	The time zone where the test was carried out. This is required to allow TDMS to know the localtime of the test, based on the UTC time values encountered within the test result. Valid timezones can be found within the IANA resource https://www.iana.org/time-zones
timeFormat	Not required if you adhere to the ISO8601. If you should wish to alter the formatting, then this attribute would be required and should adhere to the Java data time formatting guidelines. Details of which can be found at: https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/text/SimpleDateFormat.html
version	Version of your XML code generating engine.
testHost	The name of the host that was used to carry out the test.
viewArchive	This is a JAR formatted file that contains additional files generated during the test. This file would contain details of the test and contain entries such as raw log files generated during the test run. The structure of the view archive should align with that of the test run. Ie if you have a TESTSET called TESTSET1 which has associated files generated during the test run, then create a subfolder TESTSET1 and place these files under this folder.

NOTE: It is important to note that it is mandatory to have the PRODUCT and PROCESS elements within your XML definition.

5.2 Top Level Test Elements in detail

We will now take each element in turn and explain its configuration.

5.2.1 PRODUCT element

The PRODUCT element provides us information that is non test process specific. But instead focuses on the product undergoing test and specifically the assignment of the Oracle Partnumber and Serialnumber. These are the numbers programmed into the eeprom of the product being manufactured.

An example of the XML produced from the XSD definition is as follows.

```xml
<PRODUCT>
    <UNIT name="465136N+2207RZ104H">
      <TESTLOCATION>B83ICT2</TESTLOCATION>
      <OPERATION>ICT</OPERATION>
      <FAMILY>X7-2L</FAMILY>
      <SERIALNUMBER>465136N+2207RZ104H</SERIALNUMBER>
      <PARTNUMBER>7347231</PARTNUMBER>
      <OPERATORID>G7165795</OPERATORID>
      <MANUFACTURERDETAILS>
                 <PARTNUMBER>12345</PARTNUMBER>
                 <SERIALNUMBER>16161M990+1</SERIALNUMBER>
      </MANUFACTURERDETAILS>
    </UNIT>
    <RELEASE isProductionRun="true" id="X7-2L_Agilent3070_8.30"/>
    <ADDITIONALDETAILS>
      <TARGET>ASM,MTHRBD, 2U</TARGET>
    </ADDITIONALDETAILS>
```

Figure 2 XML For Product

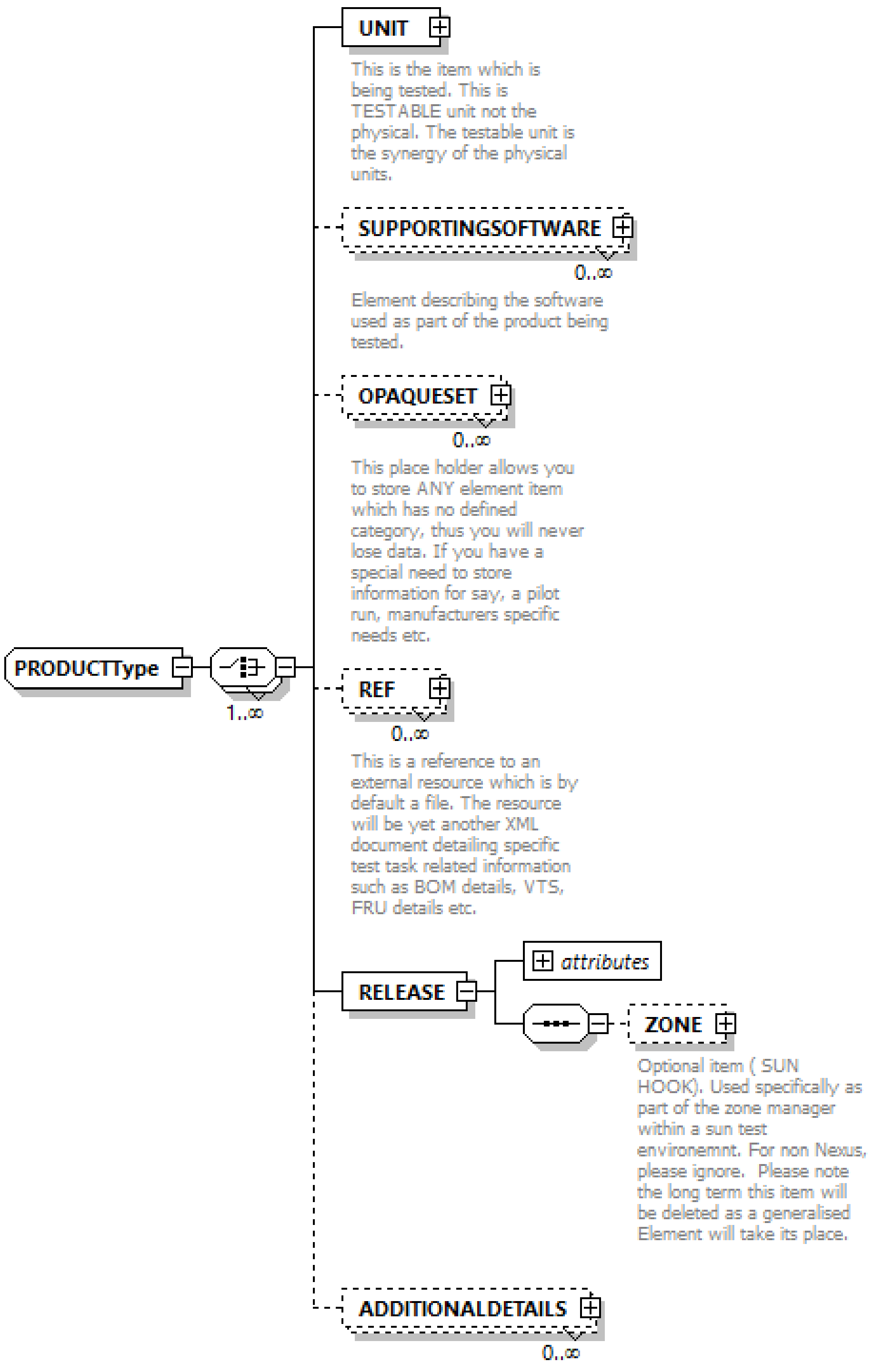

UNIT
This is the item which is being tested. This is TESTABLE unit not the physical. The testable unit is the synergy of the physical units.
SUPPORTINGSOFTWARE
0..∞
Element describing the software used as part of the product being tested.
OPAQUESET
0..∞
This place holder allows you to store ANY element item which has no defined category, thus you will never lose data. If you have a special need to store information for say, a pilot run, manufacturers specific needs etc.
PRODUCTType
1..∞
REF
0..∞
This is a reference to an external resource which is by default a file. The resource will be yet another XML document detailing specific test task related information such as BOM details, VTS, FRU details etc.
RELEASE
attributes
ZONE
Optional item (SUN HOOK). Used specifically as part of the zone manager within a sun test environemnt. For non Nexus, please ignore. Please note the long term this item will be deleted as a generalised Element will take its place.
ADDITIONALDETAILS
0..∞

Figure 3 TEST/PRODUCT element schema

The most important elements within this structure are contained within the UNIT element. These detail the unit we testing

Element Name	Description
//TEST/PRODUCT/UNIT/SERIALNUMBER	The serial number of the system under test. This value is also required at the UNIT/@name attribute level.
//TEST/PRODUCT/UNIT/SALESORDER	If known, this value should be provided as part of the UNIT element
//TEST/PRODUCT/UNIT/WORKORDER	If known, this value should be provided as part of the UNIT element.
//TEST/PRODUCT/UNIT/ PURCHASEORDER	The purchase order that related to this product. This is related to the SALESORDER. Whereby a SALESORDER would generate a PURCHASEORDER which in turn would generate the release of Work In Progress. I.e. the units being manufactured to satisfy this SALESORDER.
//TEST/PRODUCT/UNIT/FAMILY	The family of products to which this unit being manufactured belongs.
//TEST/PRODUCT/UNIT/POSITION	Although only relevant to specific types of test, this maybe within a rack.
//TEST/PRODUCT/UNIT/OPERATION	The logical operation e.g., IST SFT ASSSEMBLY being performed as part of this test result.
//TEST/PRODUCT/UNIT/PARTNUMBER	The part number that this system has been assigned.
//TEST/PRODUCT/UNIT/ TESTLOCATION	The location on the shop floor that this product is being tested
//TEST/PRODUCT/UNIT/OPERATORID	The responsible operator who is testing the system

Coupled with this, is the RELEASE element which contains //element(*,PRODUCTType)/RELEASE/@isProductionRun. This element is used to signify that a result is production or non-production. This is very important as this is used for yield calculations as all non-production results will be ignored. By default, this value is assigned to be Y (or true).

5.2.1.1 Part numbers and Serial numbers assignment

When producing a test result, it is assumed that the PRODUCT part number and serial number, will be defined by Oracle and will adhere to their specific format In addition. In addition, you should also provide part number and serial number details relevant to your business. These manufacturing part number and serial number , though optional , should be stored within the MANUFACTURERDETAILS element

5.2.1.2 Test result with multiple systems.

It is possible to create a result for a test result, which in itself had multiple sub level units. I.e. a test is executed on a top level system which has multiple children . All part of the same test. To detail the complete hierarchy of the test, you can utilize the TEST/PRODUCT/UNIT/PRODUCTINFORMATION element, and add multiple instances of TEST/PRODUCT/UNIT/PRODUCTINFORMATION/PRODUCTDETAIL elements. Each detailing the partnumber, serialnumber and location within the container.

5.2.2 The PROCESS Element

This section is where the actual test process details are defined.

As has been explained in a previous section. A test result is constructed from a TESTSUITE which can be further broken into many TESTSETs and within each testset, a set of TESTELEMENTs.

Within the TESTSUITE itself, it is possible to define various sub level XML files that are part of the test process. Files such the Bill Of Material, VTS results, log file details, field replaceable units can be added to the overall definition of your test result.

If we look at a sample PROCESS element as XML>

```
<PROCESS>
  <TESTSUITE name="SFT"
     startTime="2021-11-05T01:29:11.675"
     completeTime="2021-11-05T04:52:14.628">
   <FAIL/>
   <RESULT>
    <MESSAGE>ATO Check Fail: missing FRUs</MESSAGE>
    <EXECUTABLENAME>ato_check</EXECUTABLENAME>
    <CODE>150200</CODE>
    <FAILEDTESTEVENT>BTO_CHECK_DIAG</FAILEDTESTEVENT>
    <FAILEDTESTSET>POST_X64_1</FAILEDTESTSET>
    <FAILEDTESTSETS>2</FAILEDTESTSETS>
    <IGNOREDTESTSETS>20</IGNOREDTESTSETS>
    <PASSEDTESTSETS>10</PASSEDTESTSETS>
    <TESTSETS>32</TESTSETS>
    <DEVICEINFO>
        <MACADDRESS>a8:69:8c:14:ab:40</MACADDRESS>
    </DEVICEINFO>
   ………. 
```

You can see that in the above we have a failed test run, with the startTime and completeTime highlighted in red.

This is followed by the *RESULT* element containing <FAIL />.   Coupled with this is the message related to the failure. This message indicated what error was encountered, under this is the "executable name" used within the TESTSUITE to carry out the operation. In this case "ato_check".

Also within this block is an optional error code <CODE>, which can be used to provide categorization of the failure.
Finally, we identify the TESTEVENT and the TESTSET where the error occurred.

Within the next section of elements, or aggregate values of the number of TESTSET(s). This provides an indication of how many sets are within a suite and from this, how many were ignored or executed.

The last section before we look at test events is the DEVICEINFO element. This is used to store information such as MACADDRESS as shown in the sample above.
Note that although the MACADDRESS is an optional element in the TESTPROFILE XSD file, it is mandatory that it is captured in the ICT test process for ROT cards. The format the MACADDRESS can have, is variable. The hex chars can be upper or lower case, with or without the colon octet separator. Whatever format you choose, just be consistent.

Once we have the summary information, we can now produce the details for all test sets which were executed.

```
<TESTSET name="INITIALIZE"
startTime="2021-11-05T01:29:11.678" executeTime="2021-11-05T01:29:11.679"
exitTime="2021-11-05T01:42:57.944"
completeTime="2021-11-05T01:42:57.969">
    <PASS/>
    <MESSAGE>Part task passed</MESSAGE>
    <EXECUTABLENAME>myTaskRunner.exe</EXECUTABLENAME>
    <RESULT>
        <TESTEVENTS>40</TESTEVENTS>
    <IGNOREDTESTEVENTS>3</IGNOREDTESTEVENTS>
    <FAILEDTESTEVENTS>0</FAILEDTESTEVENTS>
    <PASSEDTESTEVENTS>37</PASSEDTESTEVENTS>
</TESTSET>
<TESTSET ……
```

In the above, you can see that most import attributes are the start and complete time coupled with the name of the test set and the result (in this case <PASS/>).

If more detail is required during the test operation, it is possible to detail the low level TESTEVENTs within the TESTSET. E.g.

```
TESTSET name="POST_X64_1" startTime="2021-11-05T04:20:52.302" executeTime="2021-11-
 05T04:20:52.306" exitTime="2021-11-05T04:47:08.209" completeTime="2021-11-
 05T04:47:08.226">
 <FAIL/>
 <RESULT>
  <TESTEVENTS>21</TESTEVENTS>
  <IGNOREDTESTEVENTS>1</IGNOREDTESTEVENTS>
  <FAILEDTESTEVENTS>2</FAILEDTESTEVENTS>
  <PASSEDTESTEVENTS>18</PASSEDTESTEVENTS>
  <EXITSTATUS>41</EXITSTATUS>
  <TESTEVENT type="test_case" name="GET_SP_STATUS"
        startTime="2021-11-05T04:20:52.306"
        executeTime="2021-11-05T04:20:52.442"
        exitTime="2021-11-05T04:20:56.445"
        completeTime="2021-11-05T04:20:56.447">
       <FAIL/>
       <RESULT>
              <EXITSTATUS>41</EXITSTATUS>
              <CODE>150200</CODE>
              <EXECUTABLENAME>checkAto</EXECUTABLE>
```

5.2.3 Defining Additional Items (OPAQUEITEM)

There may be times when it is necessary to define elements where there is no placeholder within the XSD. This may be for debugging, or for SEQUENCER specific needs as part of your process.

To achieve this, you can use the OPAQUESET and OPAQUEITEM elements. These can be placed anywhere within the document.

We should ensure that rather than have ad hoc entries throughout the document, you should group OPAQUEITEM elements within OPAQUESET elements.

```
<OPAQUESET name=”ARISTADETAILS”>
    <OPAQUEITEM name="ARISTA_10G_PORTS">28</OPAQUEITEM>
    <OPAQUEITEM name="ARISTA_25G_PORTS">1 25 26</OPAQUEITEM>
    <OPAQUEITEM name="ARISTA_CHANTYPE">ssh</OPAQUEITEM>
    <OPAQUEITEM name="ARISTA_CHECK_FAN_AIRFLOW_DIRECTION">(port-side exhaust|back-to-
front)</OPAQUEITEM>
    <OPAQUEITEM name="ARISTA_HAS_CUSTOMER_CONFIG">( GUTI == &quot;a&quot; and GUTI !=
&quot;a&quot; )</OPAQUEITEM>
    <OPAQUEITEM name="ARISTA_PROMPT">(localhost|$HOSTNAME|leaf-[ab])</OPAQUE>
</OPAQUESET>
```

In the above example, having the items grouped makes it easier to locate using the containing OPAQUESET.

Note: OPAQUEITEMS are not loaded into the database unless specifically identified for a particular product.

5.3 Example XML test Profile

We have looked at the various sections of the XSD,

```
<?xml version="1.0" encoding="UTF-8"?>
<TEST GUTI="000000000000000000290407061800032659" timezone="Asia-Bangkok" timeFormat="yyyy-MM-dd'T'HH:mm:ss.SSS" testHost="HIPOT02">
        <PRODUCT>
                <RELEASE isProductionRun="true" id="ODA_X7-2_PROD_2018.07.01" integrity="verified" />
                <ADDITIONALDETAILS>
                        <TARGET>SYSTEM</TARGET>
                </ADDITIONALDETAILS>
                <UNIT name="465769T+1820LN01D1">
                        <TESTLOCATION>HIPOT02</TESTLOCATION>
                        <OPERATION>HIPOT</OPERATION>
                        <FAMILY>T8-2</FAMILY>
                        <SERIALNUMBER>465769T+1820LN01D1</SERIALNUMBER>
                        <PARTNUMBER>7351432</PARTNUMBER>
                </UNIT>
        </PRODUCT>
        <PROCESS>
                <TESTSUITE name="HIPOT test Operation site" startTime="2018-06-07T04:31:29.565" completeTime="2018-06-07T04:32:37.268">
                        <PASS />
                        <RESULT>
                                <MESSAGE>Pass all Hipot tests</MESSAGE>
                        </RESULT>
                        <DEFINITIONS>
                                <BUILDID>RR</BUILDID>
                        </DEFINITIONS>
                        <TESTSET name="EBOND" startTime="2018-06-07T04:31:43.569" completeTime="2018-06-07T04:32:05.841">
                                <PASS />
                                <RESULT>
                                        <MESSAGE>Passed All tests</MESSAGE>
                                </RESULT>
                                <TESTEVENT type="e" name="EBOND_P0" startTime="2018-06-07T04:31:43.569" completeTime="2018-06-07T04:31:55.173">
                                        <PASS />
                                        <RESULT>
                                                <MESSAGE>PASS</MESSAGE>
                                        </RESULT>
                                </TESTEVENT>
                        </TESTSET>
                        <TESTSET name="HIGH-VOLT" startTime="2018-06-07T04:32:06.842" completeTime="2018-06-07T04:32:35.412">
                                <PASS />
                                <RESULT>
                                        <MESSAGE>PASS</MESSAGE>
                                </RESULT>
                                <TESTEVENT type="test_case" name="AC-HIPOT_P0-P1" startTime="2018-06-07T04:32:06.842" completeTime="2018-06-07T04:32:35.412">
                                        <PASS />
                                        <RESULT>
                                                <MESSAGE>PASS</MESSAGE>
                                        </RESULT>
                                </TESTEVENT>
                        </TESTSET>
                </TESTSUITE>
        </PROCESS>
        <SEQUENCER>MYSEQUENCER</SEQUENCER>
</TEST>
```

This test result has a single test suite containing 2 test sets. And the test was carried out at location HIPOT02.

Each test run would generate a result such as this, thus if we were to have a failed test run, we would bundle this result and send to Oracle, before restarting the test run. Ultimately, there could be N number of test results at an operation, with a system possibly moving in and out of an operation as it is debugged.

Within the PRODUCT element we have details of the product under test, and then within the PROCESS we have the details of all the various tasks that were carried out on the unit.

We now have a result and are ready to package this for processing by TDMS.

Let's move on to the next step. We will take this XML data file which we have named after the serial number e.g.

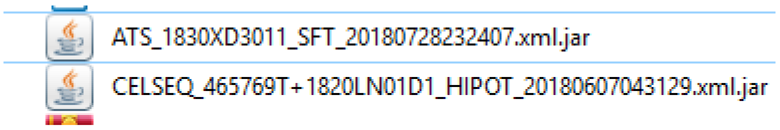
AK00225288.xml

And we will place it into a compressed JAR file .This JAR file should have the manifest removed and should be named in the following manner.

```
<SEQNAME>_<SERIALNUMBER>_<OPERATION>_<DATE+TIME>.xml.jar
```

Section	Description
SEQNAME	This is the name of your sequencer used to test your product. E.g. ATS ITS ISUOBJ etc. and should be an abbreviated name. The purpose of this prefix is to ease identification of the file when transferred to internal TDMS. The prefix will make it faster to classify a file by just looking at its name.
SERIALNUMBER	Self explanatory. The serial number of the system under test.
OPERATION	The operation that was carried out, to produce the test result. This should be an abbreviated of the operation i.e.. ICT, SFT, HIPOT, FVMI etc.
DATE+TIME	The date and time that the result was generated in the format: 4 char numeric -YYYY 2 char numeric - DD 2 char numeric - MM 2 char numeric – HH24 2 char numeric - MIN 2 char numeric - SEC

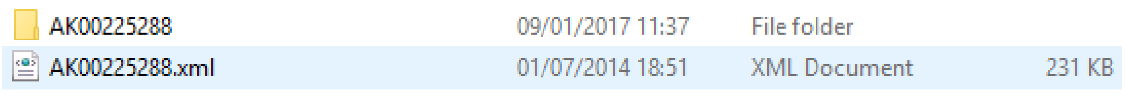
Here we have an example of 2 result files, for operations HIPOT and SFT, containing the XML data files :



ATS_1830XD3011_SFT_20180728232407.xml.jar
CELSEQ_465769T+1820LN01D1_HIPOT_20180607043129.xml.jar

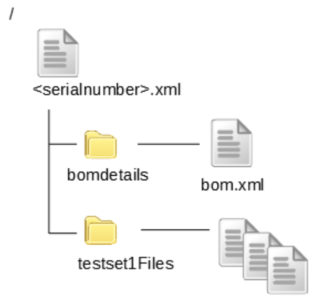
Within these files, you would place your XML at the top of the archive

e.g. We have a top-level file for the main TESTPROFILE xml, and then a sub folder where we can store further XML files.



AK00225288	09/01/2017 11:37	File folder	
AK00225288.xml	01/07/2014 18:51	XML Document	231 KB

This sub folder is optional of course and dependent on how much data you are archiving for delivery. It is recommended that the subfolders correspond to the testsets being executed.



For example, in the above, you can see that we have structured our contents to have a bom.xml file placed under a bomdetails folder, and further xml files placed into folder corresponding to the testset name that generated them.

5.4 Creating a Jar file.

To create a Jar file, there are no methods that Oracle recommends.

1- Using the Jar Command bundled within Java.
   > jar Mcf <nameOfJar> <input Files>

2- Using Java code
   Utilize the JarOutputStream class and JarEntry classes.

```
JarOutputStream target = new JarOutputStream(new FileOutputStream("myjarfile.jar"));
File source = new File(<file to add to jar>);
JarEntry entry = new JarEntry(source.getName());
entry.setTime(source.lastModified());
target.putNextEntry(entry);
in = new BufferedInputStream(new FileInputStream(source));
 byte[] buffer = new byte[1024];
     while (true)
     {
       int count = in.read(buffer);
       if (count == -1)
         break;
       target.write(buffer, 0, count);
     }
     target.closeEntry();
```

Figure 4 Example of creating Jar file with Java code

6 PACKAGING THE TEST RESULT

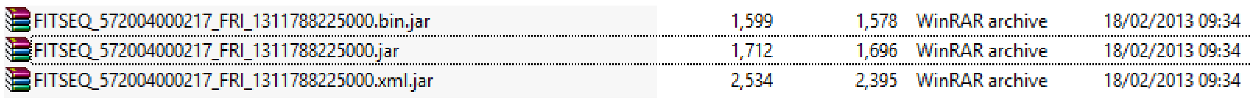
Now that we have the file containing the XML data used to load the test result into the database. We have to create a further JAR file. This is the **container** jar, referred to as the COLLECTED jar, which can contain a further 1 to many additional jar files.

Again, use the same naming convention as the xml jar, with the postfix name of the file slightly different.

```
<SEQNAME>_<SERIALNUMBER>_<OPERATION>_<DATE+TIME>_collected.jar
```

Note: It is optional as to what you decide to call this file. In our example we have called it "collected" jar, as its the collection of all the sub level jars. However, for ease of integration to the existing TDMS environment, we recommend the above syntax be adhered to.

If we now expand the collected jar file, we can see it is made up of 3 further jar files. One with the XML contents (that we have already explained above), one with files that can be read and indexed (the jar file with no postfix other than.jar), and one with binary data (the bin.jar file).



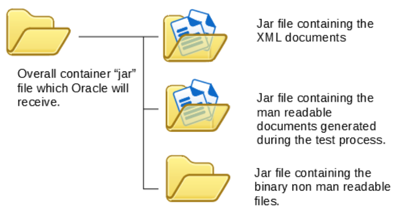
```
FITSEQ_572004000217_FRI_1311788225000.bin.jar    1,599    1,578    WinRAR archive    18/02/2013 09:34
FITSEQ_572004000217_FRI_1311788225000.jar        1,712    1,696    WinRAR archive    18/02/2013 09:34
FITSEQ_572004000217_FRI_1311788225000.xml.jar    2,534    2,395    WinRAR archive    18/02/2013 09:34
```

Explanation of each of the file types:

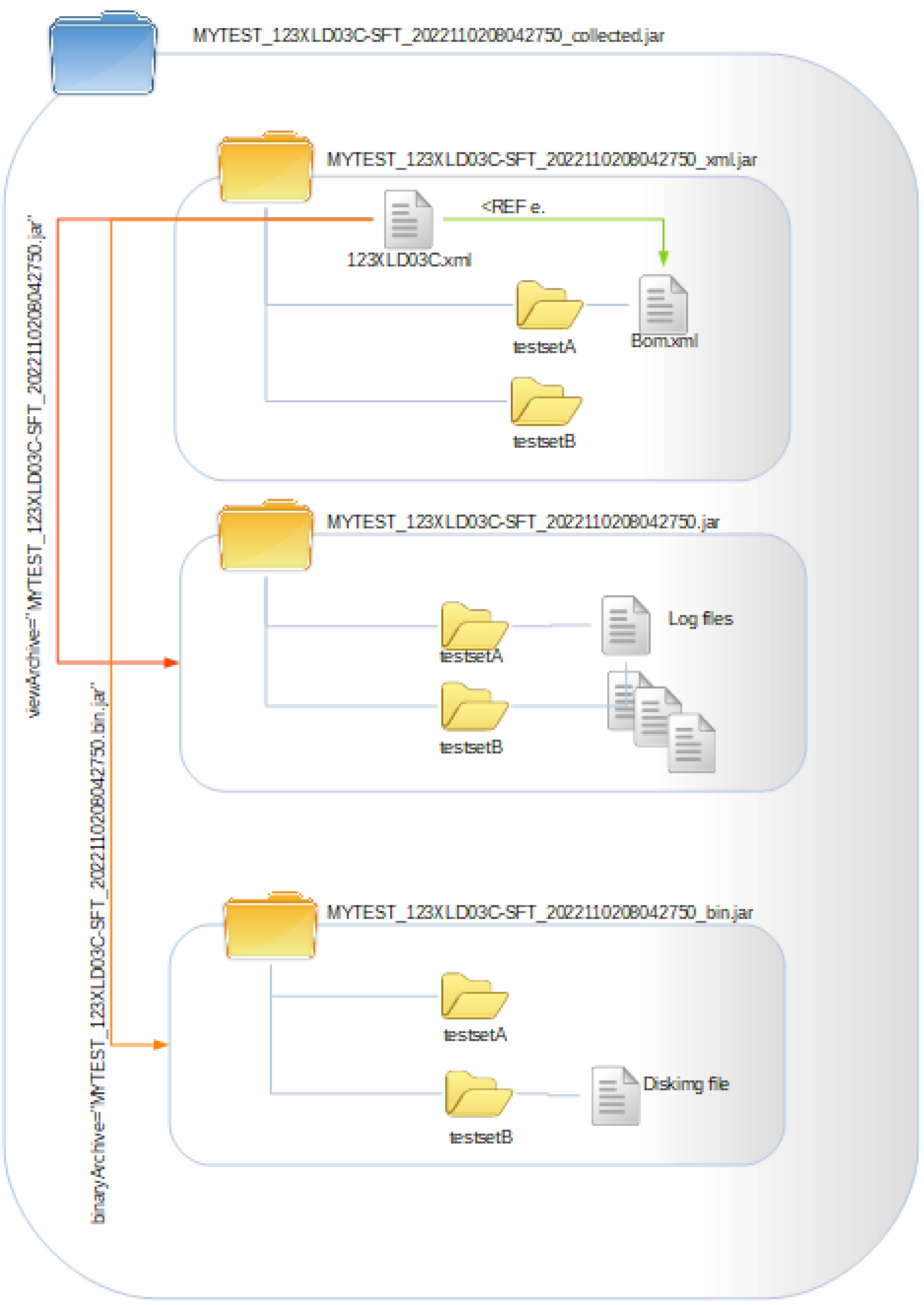
File	Description
<filename>.jar	This file is a compressed file which contains the audit of the actual test that was carried out. You are free to populate this file with whatever data is relevant to your test process. However, remember that this file must be browsable, either with the TDMS file browser or by a supplied plugin to TDMS to allow for custom browsing. As such, we recommend that the structure of the jar file has sub folders named after the test sets that were executed. We will refer to this file as the **viewarchive**.
<filename>. bin.jar	It is possible to provide a file that contains binary data, which is not man readable. This can accompany your test result and will be archived into TDMS for future reference. We will refer to this as the **binaryjarfile**
<filename>. xml.jar	This is the file that is used by TDMS to process your test result and insert into the database. All contents must adhere to the published set of schemas. We will refer to this file as the **xmljarfile**.
<filename>_collected.jar	To bundle all of these sub files into a single element which provides a single package for sending to Oracle, we create what is known as the **collected jar**.

This embedded structure gives us additional flexibility to embed binary data, images etc. within a single test result bundle.

Thus, in summary, the collected jar file is the single component containing all associated files from a test run.


Overall container "jar" file which Oracle will receive.
Jar file containing the XML documents
Jar file containing the man readable documents generated during the test process.
Jar file containing the binary non man readable files.

When we bundle all of this together, we have a single structure as follows.


MYTEST_123XLD03C-SFT_20221102080427 50_collected.jar
MYTEST_123XLD03C-SFT_20221102080427 50_xml.jar
123XLD03C.xml
<REF e.
Bom.xml
testsetA
testsetB
MYTEST_123XLD03C-SFT_20221102080427 50.jar
Log files
testsetA
testsetB
MYTEST_123XLD03C-SFT_20221102080427 50_bin.jar
testsetA
testsetB
Diskimg file
viewArchive="MYTEST_123XLD03C-SFT_20221102080427 50.jar"
binaryArchive="MYTEST_123XLD03C-SFT_20221102080427 50.bin.jar"

In the above you can see that we have the outer “collected” jar, with the 3 sub-level jar files, one with the XML details and the other 2 with man readable log files, where the other has binary specific data which would require reading by specific applications.

The XML jar file has its top-level serial number which points to the other 2 jar files using the **viewArchive** and **binaryjar** attributes.

Finally, the top-level xml file, named after the serial number, uses a REF element to point to the bom.xml file which is contained within a folder named after the test set which produced the details.

6.1 Adding META-INF

Due to the fact that these files you are generating are disparate from each other once extracted from the “collected” jar, the only means of knowing how they are related would be to read the top-level test result XML file. If this were to be corrupt in any way, or let’s say the files became orphaned during the extract process, we would be unable to identify how these files would relate to each other (other than by implementing a non-fool proof parsing algorithm to parse the names of the files). To mitigate this need, we require that custom attributes be added to a MANIFEST file placed into each Jar file. Details of the contents are as follows.

TDMS Custom Manifest values.

Value	Description
TDMS-FAMILY	Contains the family of the system under test. This should be the same family as used within the PRODUCT element of the XML.
TDMS-PARTNUMBER	Contains the part number of the system under test. This should be the same part number as used within the PRODUCT element of the XML.
TDMS-SERIALNUMBER	Contains the serial number of the system under test. This value should be the same as the serial number values within the PRODUCT element and also used on the naming of the file.
TDMS-OPERATION	Contains the logical operation that the test result pertains to.
TDMS-TESTID	This is a combination of the GUTI and the TESTSUITE start time. The value is constructed using GUTI+”:”+epoch time. Where Unix epoch time is the number of milliseconds since 1970.
Created-By	This should contain the name of your company/organization.
Built-By	You should put details of the software that was used to build the test result in here, along with the version if known.

6.1.1 Manifest File Example

META-INF/MANIFEST.MF

As stated, there is a requirement to include a manifest file in each of the sub-level jar files as well as the collected jar file. The MANIFEST.MF file within a JAR (Java Archive) file serves as a metadata file, providing essential information related to the JAR's contents.

The required format with example is as follows…

```
Manifest-Version: 1.0
Created-By: your-company-name
Built-By: your-software-name X.Y.Z
TDMS-FAMILY: family-name
TDMS-PARTNUMBER: part-number
TDMS-SERIALNUMBER: serial-number
TDMS-OPERATION: test-operation
TDMS-TESTID: guti:testsuitestart-epoch


e.g.

Manifest-Version: 1.0
Created-By: Oracle Corporation
Built-By: ATS+/Nebula 1.2.3
TDMS-FAMILY: E5-2L
TDMS-PARTNUMBER: 8215086
TDMS-SERIALNUMBER: 2512XL117B
TDMS-OPERATION: SFT
TDMS-TESTID: 67ED1EC100D8000000000000000FCTX33-B02:1743705906123
```

Note: The Manifest-Version is fixed at 1.0.

6.1.1.1 Manifest Using Command line.

The MANIFEST.MF file can be included at the point of jar file creation or as an update to an existing jar file. The example below shows how this would be done at the point of creation of the collected far file using the JAR command line executable.

The files to be included in the collected jar file

```
$ ls -1
ATS_2512XL117B_SFT_20250402062634.bin.jar
ATS_2512XL117B_SFT_20250402062634.jar
ATS_2512XL117B_SFT_20250402062634.xml.jar
```

The custom manifest file as described above

note that the last line must contain a linefeed – this is important

```
$ cat ../custom_manifest.mf
Manifest-Version: 1.0
Created-By: Oracle Corporation
Built-By: ATS+/Nebula 1.2.3
TDMS-FAMILY: E5-2L
TDMS-PARTNUMBER: 8215086
TDMS-SERIALNUMBER: 2512XL117B
TDMS-OPERATION: SFT
TDMS-TESTID: 67ED1EC100D800000000000000FCTX33-B02:1743705906123
```

Generate the jar file

note how the custom manifest file is in the parent directory to avoid it being included in the jarfile

```
$ jar -cvfm ATS_2512XL117B_SFT_20250402062634_collected.jar
../custom_manifest.mf .
added manifest
adding: ATS_2512XL117B_SFT_20250402062634.bin.jar(in = 2796053) (out=
2796876)(deflated 0%)
adding: ATS_2512XL117B_SFT_20250402062634.jar(in = 9688517) (out=
9404151)(deflated 2%)
adding: ATS_2512XL117B_SFT_20250402062634.xml.jar(in = 54579) (out=
52594)(deflated 3%)
```

Show the table of contents of the newly created jar file

note that the manifest has been placed under the META-INF folder, this is important as the manifest must reside under the META-INF folder to be valid

```
$ jar -tf ATS_2512XL117B_SFT_20250402062634_collected.jar
META-INF/
META-INF/MANIFEST.MF
ATS_2512XL117B_SFT_20250402062634.bin.jar
ATS_2512XL117B_SFT_20250402062634.jar
ATS_2512XL117B_SFT_20250402062634.xml.jar
```

Show the contents of the embedded manifest file

```
$ unzip -p ATS_2512XL117B_SFT_20250402062634_collected.jar META-
INF/MANIFEST.MF
Manifest-Version: 1.0
Created-By: Oracle Corporation
Built-By: ATS+/Nebula 1.2.3
TDMS-FAMILY: E5-2L
TDMS-PARTNUMBER: 8215086
TDMS-SERIALNUMBER: 2512XL117B
TDMS-OPERATION: SFT
TDMS-TESTID: 67ED1EC100D800000000000000FCTX33-B02:1743705906123
```

6.1.1.2 Inserting a manifest using java.

During the creation of your Jar files, the manifest can be created using the following logic.

```
Manifest manifest = new Manifest();

Attributes attributes = manifest.getMainAttributes();

attributes.put(Attributes.Name.MANIFEST_VERSION, "1.0");

attributes.put(“Created-By”, "Your company name/organization");

attributes.put(“Built-By, "Your software details");

attributes.put(new Attributes.Name("TDMS-FAMILY"), "E5-2L");

attributes.put(new Attributes.Name("TDMS-PARTNUMBER"), "8215086");

attributes.put(new Attributes.Name("TDMS-SERIALNUMBER"), "2512XL117B");

attributes.put(new Attributes.Name("TDMS-TESTID"),
"67ED1EC100D800000000000000FCTX33-B02:1743705906123");

try (JarOutputStream jarOutputStream =
new JarOutputStream(new FileOutputStream(jarFileName), manifest)) {
     ..... code here to actually put the jar together ........
```

Figure 5 Manifest using Java

6.1.2 Binary Jar file explained

During the test run, your test sequencer may have produced a set of additional files as part of the test process. These files are not intended to be loaded into the database but are there as a mechanism to provide engineering groups and hardware engineers access to component specific files when required.

These files can be associated with your test result by including in the Binary Jar file. (So named as it expects non man readable data as its content. e.g. image dumps from drives, diagnostic output from devices etc.)

To add these files, you simply bundle them into a file with the following naming convention.
`<SEQUENCERNAME>_<SERIALNUMBER>_<OPERATION>_<DATETIME>_bin.jar`

Note: ***Separator Character***

Within the filename you may notice that the separator character within TDMS is the underscore ‘_’. Please ensure that the various elements (SEQUENCERNAME, OPERATION etc.) do not within *themselves also contain this character i.e. please do not have an OPERATION with a value such as SFT_FA as this will cause confusion within the TDMS application and can result in errors when processing.*

Once created, you would reference the file using the binaryArchive attribute within the TEST element of the topmost XML file (i.e. the <serialnumber>.xml file).

Specifically, if we look at the XSD definition for a TEST element, we can see 2 optional attributes within the top level of the XML:

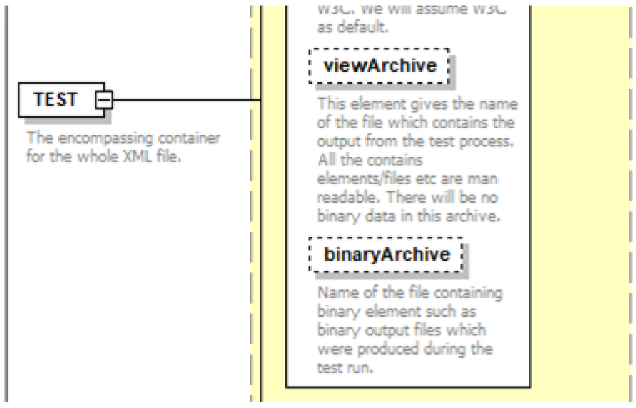

W3C. We will assume W3C as default.
viewArchive
This element gives the name of the file which contains the output from the test process. All the contains elements/files etc are man readable. There will be no binary data in this archive.
binaryArchive
Name of the file containing binary element such as binary output files which were produced during the test run.
TEST
The encompassing container for the whole XML file.

Figure 6 TEST element, JAR file links

Viewed as an XML implementation of the above XSD, you would define the following:

```
<TEST
     viewArchive=" SEQ_1111111_SFT_20181208112345.jar"
     binaryArchive="SEQ_1111111_SFT_20181208112345.bin.jar"
……. .
```

It's extremely important that you ensure the names of the files match up with the names specified within your XML elements, otherwise TDMS will not be able to locate the file and thus processing will fail.

6.1.3 Contents of the viewArchive Jar file

The viewArchive jar file would contain man readable details of the test that was executed and is in essence the raw audit of the test run. Files such as output files, logs etc. which are man readable should be placed within this archive file.

It is suggested that the structure of folders within this file marry to the test set names provided within the XML TESTSET elements such that there is a 1 to 1 relationship. This will make the traversing of the file somewhat easier for anyone cross referencing the files.

Although TDMS has a simple file browser for manipulating this file, it is possible to provide an HTML structured environment for the user. This is achieved by creating index.htm files throughout your document or creating a custom viewer which TDMS can utilize for a particular SEQUENCER. TDMS will interpret this instead of the raw jar content and attempt to present the HTML content. (all links relative to the file structure within the Jar file).

6.2 Illegal Characters

As is normally the case with archive files, the viewArchive and the **binaryArchive** will contain sub folders and each containing file entries. These folders and files names must not contain any of the following characters to maintain compatibility across all operating environments.

Illegal Characters for naming files and folders within an archive:

Character/Symbol	Description
#	Pound
<	Left bracket
>	Right bracket
$	Dollar
+	Plus
%	Percent
!	Exclamation
`	Back quote
&	Ampersand
*	Asterisk
'	Single quote
|	Pipe
{	Left curly bracket
}	Right curly bracket
?	Question Mark
“	Double quote
	Equals
:	Colon
\	Backslash
@	At sign
	Backspace
	Blank space
/	Forward slash.

Additional Naming Rules

Please also attempt to adhere to the following as a rule of thumb:

- Filename should not start nor end with a period, underline or hyphen character.
- Attempt to keep names of files under 31 characters.
- Full hierarchy paths should not be more than 260 characters in length, inclusive of the filename. As such please refrain from deeply nested files.
- Where possible please use lowercase, or camel case on your naming of files.

6.3 Associating Different XML Data types

Within the main test result, it is possible to associate additional XML files to be processed by TDMS and thus insert more details into the database. TDMS has many supported XML formats for various types of results during a test operation. Each with its own XSD/Schema.

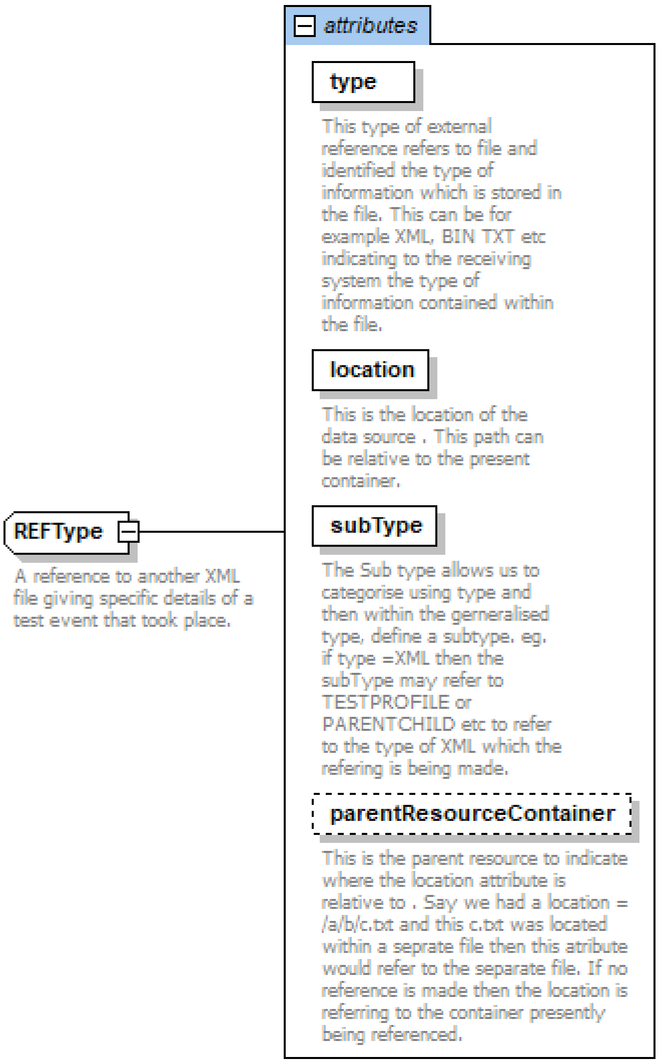
Ref Type	Description	XSD
BOM	Bill of material. Detailing the components which comprise the assembly.	bomSchema.xsd
FA	Failure Analysis data	FASchema.xsd
PARENTCHILD	Hierarchical structure of a solution. i.e. multiple systems, switches etc. To a solution, as to what a BOM is to a system.	parentSchema.xsd
DEVICES	Details of the various types of devices within the assembly	deviceDetailSchema.xsd

TESTPROFILE	Hierarchical sequence of test process data	mainTestProfileSchema-minimumSpecification.xsd
FIRMWARE	Oracle hardware stress test software output	firmware.xsd

To associate additional XML files with the test result, we utilize the **<REF> element** within our **<serialnumber>.xml** file. The REF element provides in essence a pointer to the file, relative to the enclosing Jar file.

The REF element should be placed within the TESTSUITE or TESTSET or TESTEVENT elements (where the enclosing TESTSET or TESTEVENT is the action where the reference was generated). Note that there can be N number of REF elements. Placing your REF element anywhere within the TEST xml is valid, however ad hoc placement will undoubtedly cause issues if debugging of XML is required.

Looking at the XSD again, here is the structure of the REF element.



This type of external reference refers to file and identified the type of information which is stored in the file. This can be for example XML, BIN TXT etc indicating to the receiving system the type of information contained within the file.

This is the location of the data source . This path can be relative to the present container.

The Sub type allows us to categorise using type and then within the gerneralised type, define a subtype. eg. if type =XML then the subType may refer to TESTPROFILE or PARENTCHILD etc to refer to the type of XML which the refering is being made.

This is the parent resource to indicate where the location attribute is relative to . Say we had a location = /a/b/c.txt and this c.txt was located within a seprate file then this atribute would refer to the separate file. If no reference is made then the location is referring to the container presently being referenced.

A reference to another XML file giving specific details of a test event that took place.

Figure 7 REF element

The definition of the attributes are as follows:

1. The **type**. This should be set to XML. In future releases JSON will be supported.

2. The **location**. This is the location within the archive file. By default, the system assumes you are referring to a file within the xml.jar container file. However, if you also set the parentResourceContainer to another file, located within your "collected.jar" file, then the search will be made there.

3. The **subType**. This is the XSD subtype that is defined within TDMS. Such as BOM VTS LOG etc.

4. **parentResourceContainer**. allows you to reference data in another file. Thus, it is possible to have additional files with more data within them. By default, though, TDMS will assume that the location attribute is relative the present file i.e. the xml.jar file.

Here is an example within a test result of inserting a REF element.

```
<PROCESS>
  <TESTSUITE name="SFT"
     startTime="2021-11-05T01:29:11.675"
     completeTime="2021-11-05T04:52:14.628">
   <FAIL/>
   <RESULTS ….

   <DEVICEINFO ….

  <REF location="2144XLF08K/test_suite_SFT/FRU.xml" type="XML" subType="FRU"/>
  <REF location="2144XLF08K/test_suite_SFT/bom.xml" type="XML" subType="BOM"/>
  <REF location="2144XLF08K/test_suite_SFT/firmware.xml" type="XML" subType="FIRMWARE"/>

 <TESTSET …..
```

In this example the reference is to BOM and FIRMWARE files located in a sub folders:

`<serialnumber>/<testSuiteName>/bom.xml`

We have chosen to produce the REF at the top level, however if data is generated within a testevent or testset , and the XML produced is relevant to the enclosing element , then the REF should be placed within the producing element.

Note: when you are adding additional files, it is preferred that only 1 occurrence of each type of file be referenced. I.e. if you are providing BOM information, you should only have 1 reference to a BOM.xml file. Although some XML file *types can indeed be sent to TDMS in multiple files, having disparate files throughout the XML Jar file of the same type can be confusing and hard to debug if errors should occur.*

7 ADDITIONAL XML DATA FILE TYPES

7.1 Bill Of Material (BOM)

As part of a test, it is required that you detail the parts used in the assembly of the system at the time of test operation. This is accomplished by producing a bill of material file (BOM.xml) and associating this with the top level test result (<serialnumber>.xml)

TDMS already has an XSD defined those details how the XML data should be structured, and you should use this as your reference when checking the syntax of your data. However, we will walk through a simple example of a BOM, to help you understand the basic structure and how you can then associate this with your test result.

7.1.1 XML Example

The BOM entries have various attributes used to describe the component being used as part of the overall system assembly.

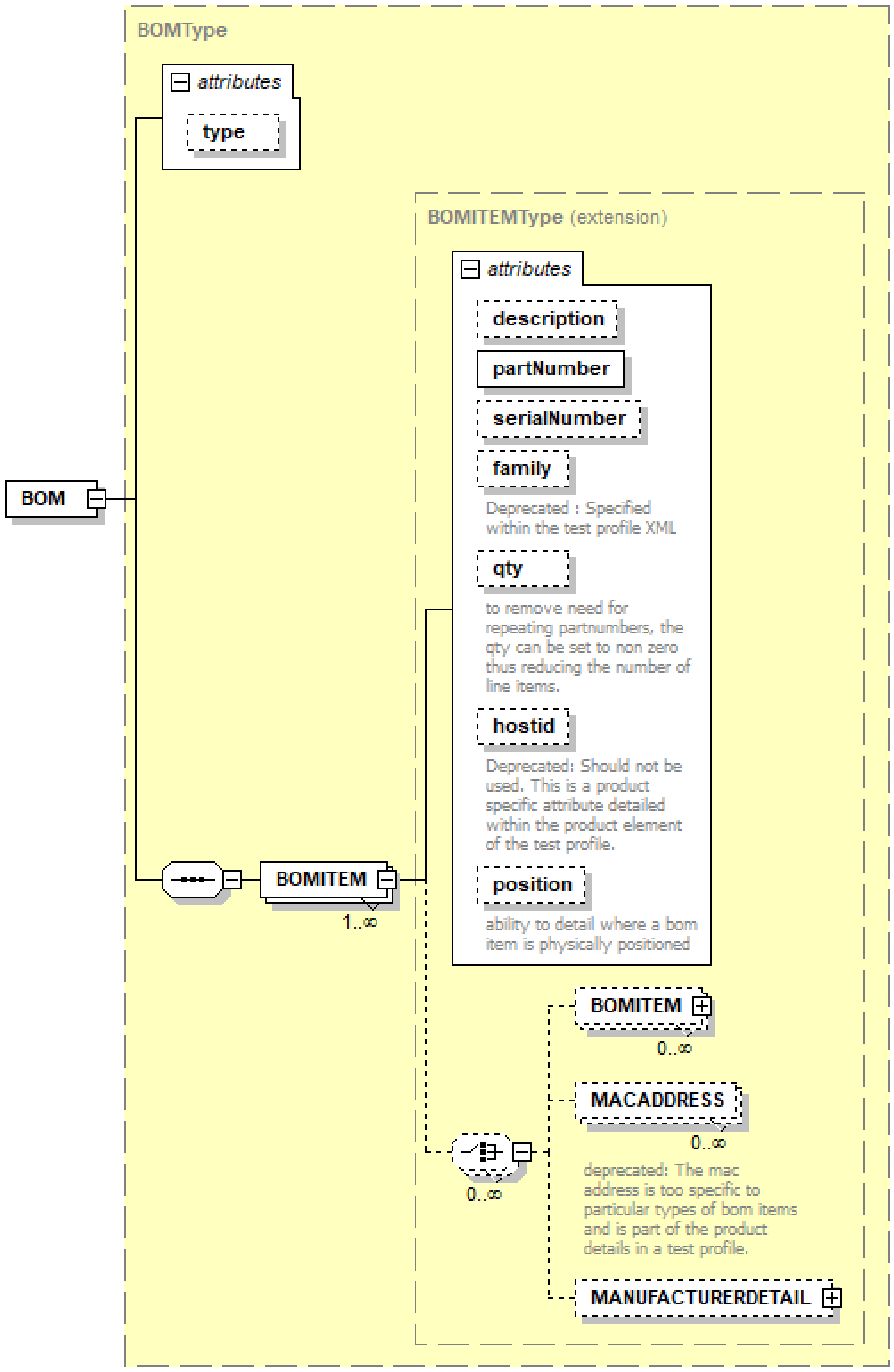

BOMType
attributes
type
BOMITEMType (extension)
attributes
description
partNumber
serialNumber
family
Deprecated : Specified within the test profile XML
qty
to remove need for repeating partnumbers, the qty can be set to non zero thus reducing the number of line items.
hostid
Deprecated: Should not be used. This is a product specific attribute detailed within the product element of the test profile.
position
ability to detail where a bom item is physically positioned
BOM
BOMITEM
1..∞
BOMITEM
0..∞
MACADDRESS
0..∞
deprecated: The mac address is too specific to particular types of bom items and is part of the product details in a test profile.
0..∞
MANUFACTURERDETAIL

Figure 8 BOM Schema

Using the above schema, we can define a BOM as follows

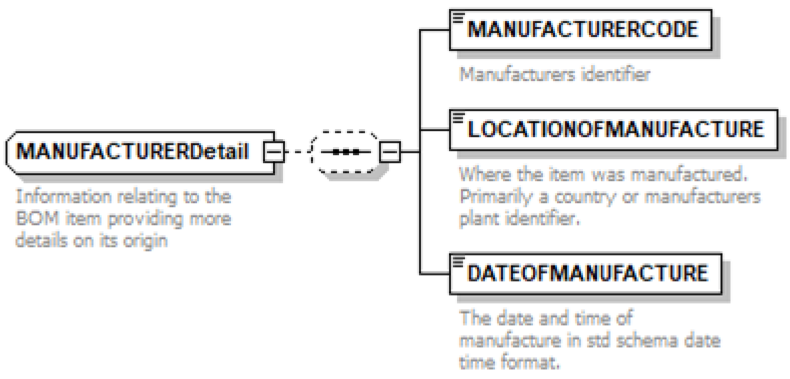
```
<BOM type=”asbuild”
   <BOMITEM partNumber=”11111” serialNumber=”222222”
            position=”slot1” description=”sub assembly 1” />

      <BOMITEM partNumber=”11111” serialNumber=”33333”
             position=”slot2” description=”sub assy 2” />

            ...
</BOM>
```

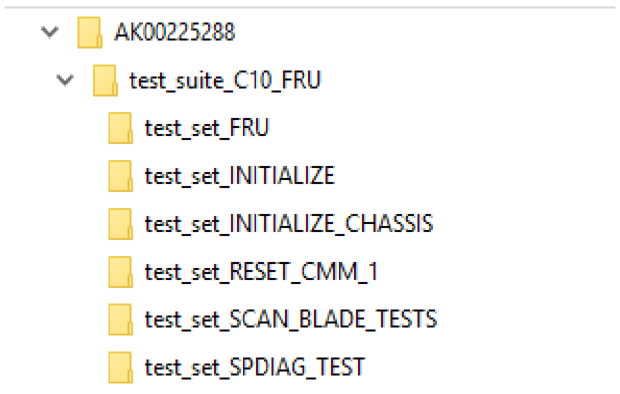
Of course, this simple example only shows a single level hierarchy, where each part has a quantity defaulted to 1. However, the position attribute which is highlighted should be present whenever the data is known as this information is essential for understanding the structure of what was actually shipped from a manufacturing site.

If it is required to provide supplier information for each item whenever possible, you can further define a BOMITEM to have MANUFACTURINGDETAIL.


MANUFACTURERDetail
Information relating to the BOM item providing more details on its origin
MANUFACTURERCODE
Manufacturers identifier
LOCATIONOFMANUFACTURE
Where the item was manufactured. Primarily a country or manufacturers plant identifier.
DATEOFMANUFACTURE
The date and time of manufacture in std schema date time format.

7.1.2 Associating BOM.xml into the top level XML

Now that you know the structure of the XML, you should create a file, let’s call it BOM.xml, and place it into a sub folder within our top-level XML jar file.



- AK00225288
  - test_suite_C10_FRU
    - test_set_FRU
    - test_set_INITIALIZE
    - test_set_INITIALIZE_CHASSIS
    - test_set_RESET_CMM_1
    - test_set_SCAN_BLADE_TESTS
    - test_set_SPDIAG_TEST

Figure 9 Structure of compressed result

```
>AK00225288/testsuite_C10_FRU/test_set_INITIALIZE/BOM.xml
```

.
We will now follow the rules defined in the previous section and add a REF element to the <SERIALNUMBER>.xml file

```
<REF
    type=”XML”
    subtype=”BOM”
    location=”AK00225288/test_suite_C10_FRU/test_set_INITIALIZE/BOM.xml” />
```

This can be placed anywhere within the TESTSUITE or a TESTEVENT XML element. But it is suggested that it be placed at a point in your structure where it reflects the process context i.e. the testset that is being executed to generate the XML. Create a folder which makes sense to the flow of the test being carried out.
Note: Again, it is important that there should only one BOM xml file per unit being tested.

The above example is defined with many folders, but this is only to be used as a guide to laying out the structure of your XML data. We recommend that the structure of your jar files folders reflect the test sets being executed as part of the test operation.

7.2 Devices

The devices XML is another mandatory data item required as part of the test result (when carrying out operations which are able to generate this data.)

The devices xml file defines details for specific device types.

I.e.

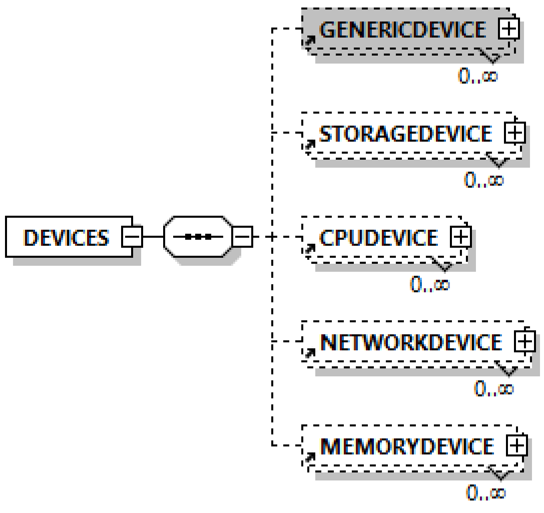


Figure 10 Devices structure

Having the DEVICES element allows for gathering of lower level, fine grained device details which is not possible with only a BOM xml file. Attributes such CPU speed, vendor part details, drive storage IDs can now be captured.

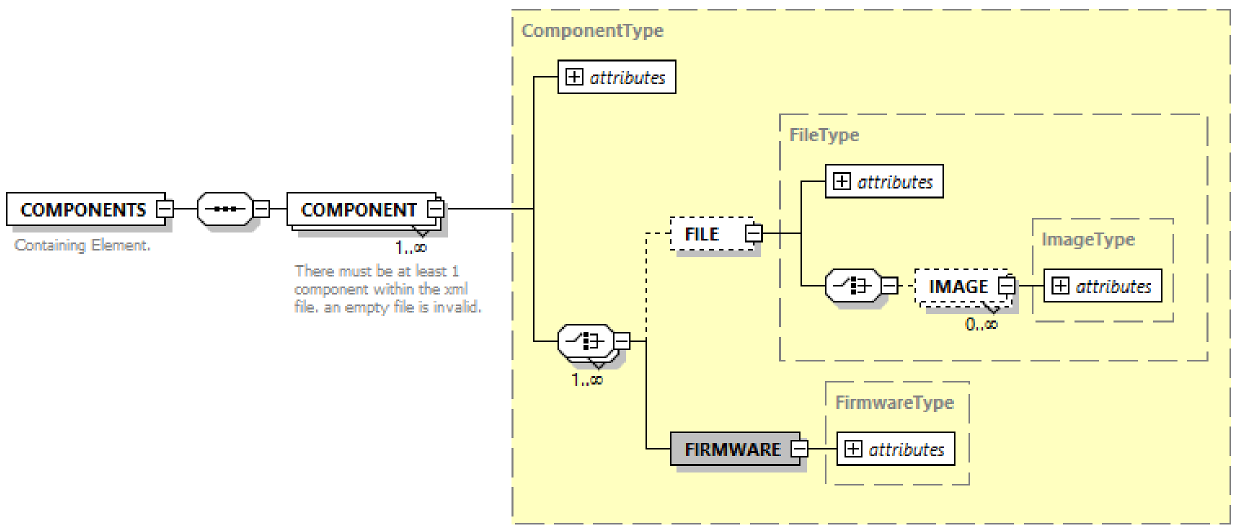
Although most devices fall under the main 4 categories, however, if a required device should not fall into any specific category (i.e. CPU, network, memory etc.) then the details should be placed under GENERICDEVICE element.

The most important usage of the DEVICES xml file is to associate electronic serial number information with label based serial numbers. When these 2 values are not the same, this file is the only means by which a 1 to 1 relationship can be made. This is more prevalent which we are dealing with DIMMs and SSDs,

7.3 Firmware

The firmware is also a mandatory data item which is used to detail the firmware which is installed on various components within the system. Creating this XML is best achieved using the Oracle executables which are made available during testing of products.

The basic structure of the firmware xml is as follows.



Each component (which there can be 1-N), would have the FIRMWARE element populated to detail the installed firmware revision, and the path within the operating system of the component.

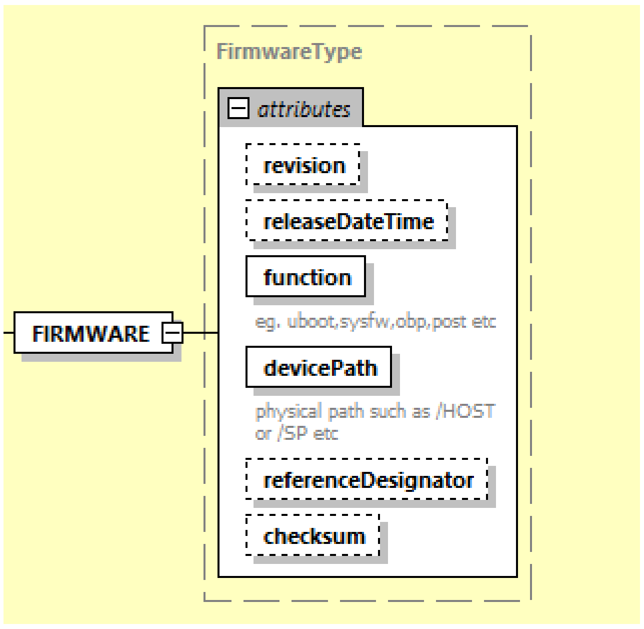


Figure 11 Firmware Schema

7.4 Parent Child

To Parent Child XSD provides the ability to associate disparate systems as an associated hierarchy, which together would constitute a solution. The hierarchy of the parent child should recurse to the point of the BOM definition within each system.

Note: You should not define the structure any further than to the system, or you will in essence be repeating the BOM structure at the system level unnecessarily.

The structure of the parent child schema is as follows

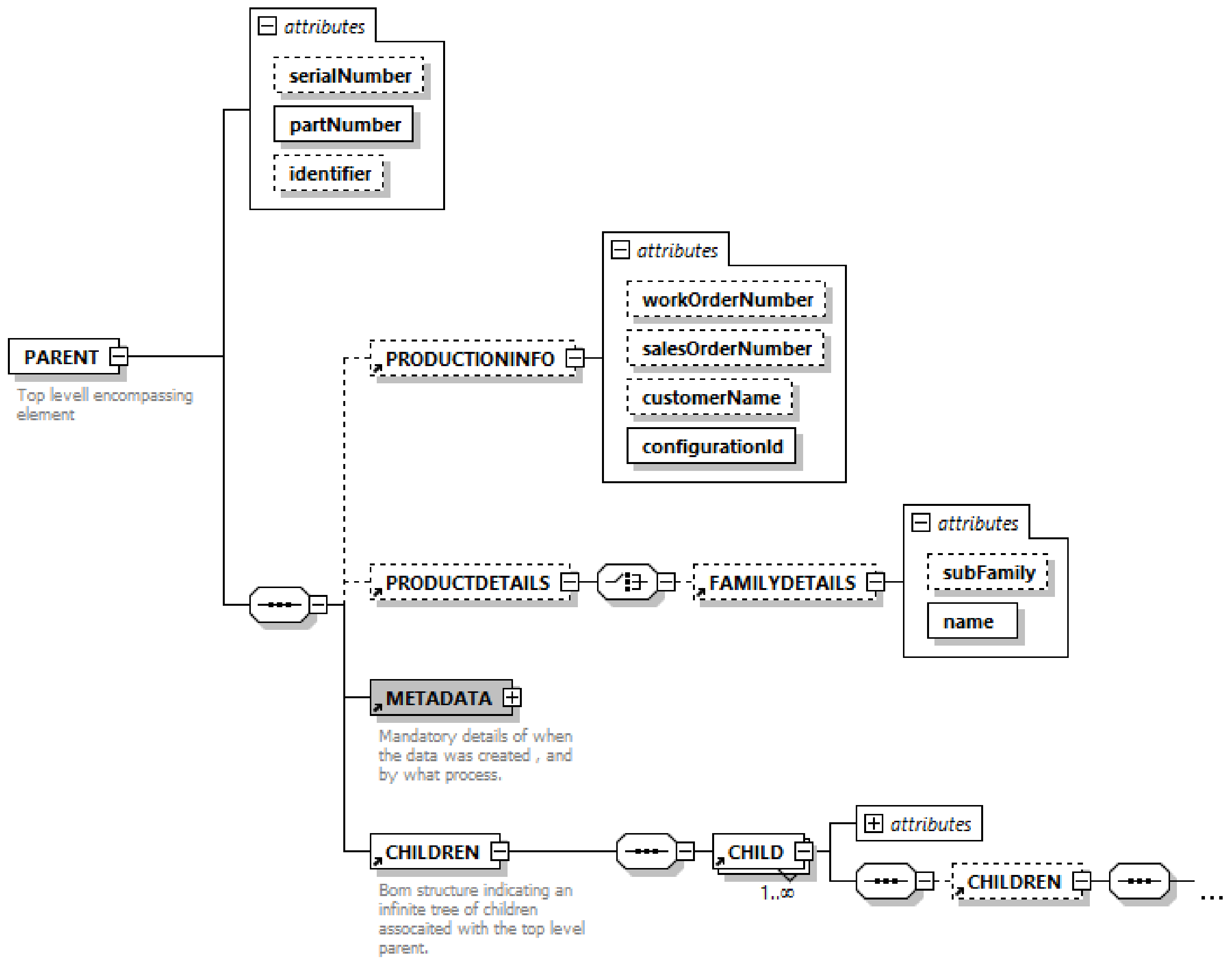


Figure 12 Parent child schema

The parent element should be used to model the topmost part number of the solution. If a serial number is associated, then this should also be provided.

Within the hierarchy, you can see that a PARENT element has a CHILDREN element which contains 1-N CHILD elements. And each child in turn can have more children associated. The levels of drill down required should be detailed by Oracle's product engineers based on each solution being manufactured.

From the above, the following XML would be produced for the CHILDREN element.

```
<CHILDREN>
        <CHILD serialNumber=”abc” partNumber=”11111” position=”slot1”
                description=”rack system”>
                <CHILDREN>
                       <CHILD serialNumber=”def” partNumber=”2222” position=”slot1-subslot1”
                          description=”sub part” />
                          ……
                </CHILDREN>
       </CHILD>
       <CHILD serialNumber=”xyz” partNumber=”33333” position=”slot2”
                decription=”rack system” />

………
</CHILDREN>
```

7.5 FRU XML

The FRU xml file can be generated using the following command

- prtfru -x

prtfru is an Oracle specific command used to obtain Field Replaceable Unit Identifier (FRUID) data from the system and any external I/O expansion units. It creates a hierarchical tree structure reflecting the path in the FRU tree to each container.

Please contact your product or test engineer for details of usage in respect to the product being tested.

8 ASSOCIATING FILES FROM TESTSETS AND/OR TESTEVENTS

During the execution of a test, there will inevitably be files generated as part of the normal operation of the test set and/or test event. These files are stored within the viewArchive and/or binary archive file. However, there are no direct correlation to indicate what files are associated with each testset.

As such it is advised that these files be linked to the testsets/testevents using the element:

`<ASSOCIATEDFILE />`

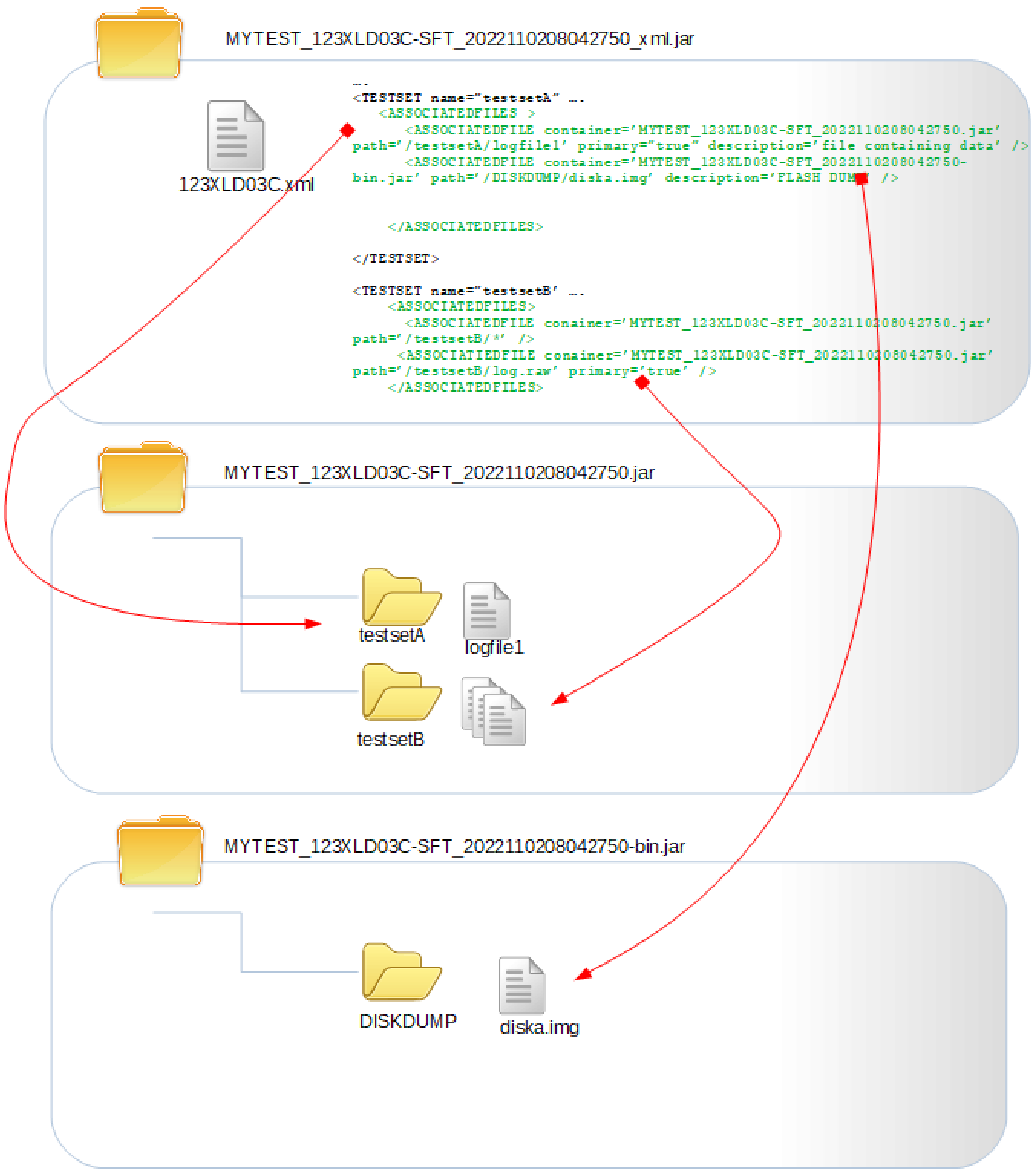
`This is contained within the TESTSET /TESTSUITE/TESTEVENT and wrapped with the <ASSOCIATEDFILES /> element.`

```
<ASSOCIATEDFILES >
      <ASSOCIATEDFILE container=”the jar file where the file is located”
               path=”relative path within container”
               primary=”true|false”
               description=”an optional summary indicating what the file contains”
               mimeType=”application/text” />
```

Each attribute can be described as follows:

Attribute	Description
container	This refers to either the viewArchive jar file or the binary jar file which accompanies your test result xml Note that within the ASSOCIATEDFILES it is possible to define a global container which applies to all of the contained ASSOCIATEDFILE elements.
path	The relative path within the container to the file. Additionally, you can specify the “*” character indicating that all files within this folder are to be included as part of the testevent/testset.
primary	Indicates if this is the primary file (i.e. this is the most important file generated from the testevent/testset . Showing the most salient information related to the testset/testevent being run.) There can only be one entry which is true within a list of files and having a single entry which points to the most relevant file is mandatory. The default will be false. Note that if there is no element containing the primary element, then the 1st child entry within the ASSOCIATEDFILES will be used as the most relevant document.
description	If required, you may add a small description to indicate what the file relates to and what it contains. This provides clarity to the engineers viewing the file within Oracle.
mimeType	If you are specifying a link to a single file and you know the content of the file, then you can specify the mime type. The default is application/text.

Note : If you are able to group files into folders which are named after the TESTSET or TESTEVENT being executed, you should at least indicate the top most folder for the TESTSET .


MYTEST_123XLD03C-SFT_20221102080427 50_xml.jar
123XLD03C.xml
MYTEST_123XLD03C-SFT_20221102080427 50.jar
testsetA
logfile1
testsetB
MYTEST_123XLD03C-SFT_20221102080427 50-bin.jar
DISKDUMP
diska.img

```
...
<TESTSET name="testsetA" ...
   <ASSOCIATEDFILES >
      <ASSOCIATEDFILE container='MYTEST_123XLD03C-SFT_20221102080427 50.jar'
path='/testsetA/logfile1' primary="true" description='file containing data' />
      <ASSOCIATEDFILE container='MYTEST_123XLD03C-SFT_20221102080427 50-
bin.jar' path='/DISKDUMP/diska.img' description='FLASH DUMP' />


    </ASSOCIATEDFILES>

</TESTSET>

<TESTSET name="testsetB' ...
    <ASSOCIATEDFILES>
      <ASSOCIATEDFILE conainer='MYTEST_123XLD03C-SFT_20221102080427 50.jar'
path='/testsetB/*' />
     <ASSOCIATIEDFILE conainer='MYTEST_123XLD03C-SFT_20221102080427 50.jar'
path='/testsetB/log.raw' primary='true' />
    </ASSOCIATEDFILES>
```

Figure 13 Example ASSOCIATEDFILES elements

One important point to note within the Figure above is that there are references to different containers within the single ASSOCIATEDFILES element. It is important that you only have 1 ASSICATEDFILES elements defined within the TESTEVENT/TESTSET or TESTSUITE.

9 TRANSFERRING RESULTS TO TDMS

There are 2 main architectures that can be used to get data into TDMS.

1. **PULL MODEL**
   TDMS will access a central repository storing your test results and will pull these to central TDMS for processing.

2. **PUSH MODEL**
   The responsibility will lie with the data producer to push the data using 1 of 2 possible mechanisms.

9.1 Pull Model

In this section we will detail how to set up your data for TDMS to pull and ultimately transfer to internal Oracle.

There are a variety of options available to sharing your results with TDMS:

1. **SFTP end point.**
   If you wish to have TDMS pull the test results and you have no local copy of TDMS, you can expose an end point connection to the Internet and have TDMS pull the results from your site .
   To implement this model you will require :
   a. An SFTP end point, where you have an SFTP daemon service running. There should be an internet facing IP address accessible by Oracle. On logging into this endpoint, the user credentials should allow access to a folder where the test result files would reside.
   b. The credentials for access should be set up accept SSH keys rather than basic username and password credentials. Public/private keys is the method endorsed by Oracle.
   d. The folder containing the compressed result files should be configured for read write access, to allow for reading of the files, and deletion on completion of the transfer.

2. **Cloud storage folder**
   Using the S3 cloud store to transfer data is really a hybrid model, whereby the client pushes and TDMS pulls the data. TDMS will provide the authentication and service details to allow for publishing of data to the cloud bucket.

3. **Nexus Integration**
   If you are using Nexus as your sequencer on a remote site, and you have a local TDMS instance. The test results will be pulled from Nexus and then stored locally within TDMS, and then ultimately pulled by the central TDMS server over the internet via SFTP.
   However, this does require that you allow the TDMS server to be seen and accessible over SFTP and the internet by Oracle based connections.
   TDMS will be required to know:
   a. The names and addresses of all “test domains” that are running Nexus.
   b. The network running your nexus environment should allow for ICMP PING as TDMS will check the

existence of a server using this method before any HTTP connections will be attempted. A successful ping verifies that the server is available to search for test results.

9.2 Push model.

Another method of sending data to TDMS is to use the PUSH model. Note, though that to use some of these proposed solutions, you must have a local copy of TDMS running.

There are 3 methods of PUSHING data to TDMS.

- **Manual method, using the COLLECTOR GUI web page.**
  (*requires local instance of TDMS*)
  The manual upload method requires that within TDMS there is a defined area for upload, which will then appear within the COLLECTOR application front end . From here it is a matter of uploading the result file by dragging and dropping to the web page.

- **REST**
  (*requires local instance of TDMS*)
  Using your own client software to push the data into the TDMS environment using REST services.
  Note: Please refer to section 9 of this document for details on REST services.

- **S3 bucket**
  It is possible to push your results to a predefined S3 cloud bucket. (as stated in previous section) Where the credentials and bucket name will be provided by Oracle. Utilizing this model will result in a JSON file being produced each day, within an S3 folder which Oracle will provide. This JSON object details the transfers and any issues encountered with the data.
  Note: To push your files, you should utilize the Python S3 libraries , or Java S3 client.

9.2.1 Cloud Store Transfer JSON Report

To provide an audit to a site utilizing that cloud storage service, (S3), Oracle will provide a daily summary report, formatted in Json. This report is placed on a named cloud bucket (provided by your Oracle TDMS engineers) each day to provide a summary for the previous days processing This includes overall aggregate totals and any errors which may have occurred with reference to the offending file and sublevel xml file.

The JSON file will be stored within a cloud bucket, with each file having a prefix of

```
/<BUCKETNAME>/<YYYY>/<MM>/<DD>/processingStats.json
```

The contents of the json file will be as follows :

```
{
 "summary": {
   "siteid": 99,
   "totals": {
           "totalDistinctResultsProcessed”: 100,
           "totalAborted”: 100,
           "totalResultsProcessed": 100,
           "loaded": 100,
           "partiallyLoaded": 100,
           "nonParsable": 100
 },
  "range": {
         "fromDate": "2022-02-28",
         "toDate": "2022-02-28"
   }
 },
   "partialLoadingErrors": [
                            { “serialNumber”: “12345”,
                              “guti”:”Pb7DQd9cuPMt1lRaPepMURVkhMrWfA939EEU”,
                              “testStartTime”: “2021-01-01T23:21:22.123”
                              “parentContainerPath”: “ABC_1234.xml.jar”
                              “referenceSubType”: “BOM”,
                              “relativePath”: “/testsets/testset1/myBom.xml”,
                              “errorDetail”: “Error line:1, col:5 illegal char found”
}
                          ],
   "nonParseableErrors": [ <Array Indicating files and error messages>
                            { }
                          ]
   }
```

Figure 14 Json Audit S3 file

9.3 Local REST services for upload

Only relevant when a local instance of TDMS is installed within your site.

9.3.1 Uploading a JSON formatted test result

Simple test results i.e. ones which have very little detail and no additional files, can push a JSON result to the following REST end point. TBD

If you would like to use the swagger specification, then the url:

```
        http://<LocalAddressOfTdmsServer>/COLLECTOR/rest/openapi.json
```

will provide the standard openapi specification.

Additional information: https://swagger.io/, https://swagger.io/specification/

End point for invoking the upload:

URL	https://<tdmsserver>/COLLECTOR/rest/testresults/result/{serialnumber}
Method	POST
ContentType	Application/json
Response Code	202 OK
	500 internal error
	400 invalid data within request

Json Body :

```
{
   “guti”: “global unique identifier of the test”,
   “timezone”: “The time zone that the test was executed in”,
   “productUnderTest”: {
                      “serialnumber”: “Serialnumber of the product tested, optional since already set”,
                      “partnumber”: “partnumber of system under test”,
                      “family”: “The family of product”
                      },
   “process”: {
                “operatorid”: “Operator who carried out the test”,
                “testlocation”: “The location where the test took place”,
                “operation”: “The operation being executed at this location”,
                “testsuite”: {
                               “name”: “Name of the test suite”,
                               “startTime”: “The start time as Iso date/time with milliseconds”,
                               “completeTime”: “The time completed “,
    “result”: “The result of the test as PASS FAIL ABORT”,
    “message”: “The message , summarizing  the result”,
    “testsets”: [
                                    {“name”: “Name of the testset”,
                                    “startTime”: “date/time to ms of the start time of the testset”,
                                    “completeTime”: “date/time to ms of the time the test set completed”,
                                    “result”: “The result ie PASS FAIL ABORT”,
                                    “message”: “Summary message of the result conditon”
                                    } ]
      }
     "billOfMaterial": [
         { “partumber”: “partnumber of the part”,
           “serilanumber”: “serialnumber of the part”,
           “position”: “Location part is placed within the assembly”,
           “description”: “man readable description of the part”,
           “bomItems”: [
                     ……
          ]
        }
        ]


}
```

9.3.2 Upload collected Jar using REST service

To upload a full collected jar file to TDMS, the following REST service is available.

URL	`https://<tdmsserver>/COLLECTOR/rest/dataprocessing//upload/receptionArea/{targetArea}`
Method	POST
Content Type	Multipart Form
Response	Application/json
Response Code	201 OK
	401 address blocked
	404 unknown target area.
	500 internal parsing error
	502 service offline
Params	targetArea: the defined area within TDMS that you have been instructed to upload your files to. This will map to an actual area within tdms that data is processed.

The body of the request will be a jar file adhering to the “collected” jar specification (see section ???) .

10 REAL TIME EVENTS

Though optional, it is requested that to provide, whenever possible, a real time remote view of progress that each product makes moving through the manufacturing environment. Real time events coupled with environmental readings should be transferred to TDMS during a product’s manufacturing lifecycle.

This real time data is sent to the central TDMS via an Oracle cloud based analytics streaming service, built around the Kafka streaming architecture (Note: off the shelf Kafka client libraries will be able to communicate to this service).

10.1 Process Events

When a test is being executed, it is recommended that visibility be provided as to the progress of the testing, in real time (or as real time as possible)

When defining an event that will be published using the REST services, the following JSON structure must be adhered to.

ATTRIBUTE NAME	DETAILS	VALUES
name	The name of the event, in the next sections, you will see various events. The name detailed within	The names of the events are defined within the previous sections.

	the docs should be used in this attribute.	
type	The type of event being published	The default for most events unless otherwise stated is PUBLIC. This scopes the events to being global within the system.
subType	Within the type there will be various subtypes	Each event will specify its sub type.

10.2 Name : DELIVERYIN
Type : PUBLIC
Subtype : SHOPFLOOROPERATION

Description : Whenever a unit is scanned into a location on the floor. Ready to start an operation.

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**Location of where we are scanning the item to.**
LOGOP	**String**	**False**	**The logical operation being carried out. Note that the LOGOP should be UPPERCASE cand can contain 0-9 or _ characters. It should not contain spaces. Refer to the XSD for the restriction details.**
FAMILY	**String**	**False**	**The family of product**
PARTNO	**String**	**False**	**The part number that the serial no belongs to.**

```
        {
             “type”: “PUBLIC”
             “subtype”: “SHOPFLOOROPERATION”
             “name” : “DELIVERYIN”
             “params”: {
                         “SERIALNO”: “1111111”,
                         “LOCATION”: “loc1”,
                         “LOGOP”: “IST”
             }
        }
```

10.3 Name : DELIVERYOUT
Type : PUBLIC
subtype : SHOPFLOOROPERATION

Description : Whenever a unit is scanned out of a location to be moved to another area

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**The operation that was being carried out.**
LOGOP	**String**	**False**	**The logical operation that was being conducted at this location. Note this is optional as there may be many operations.**

```
        {
              "type": "PUBLIC"
              "subtype": "TESTPROCESS"
              "name" : "DELIVERYOUT"
              "params": {
                             "SERIALNO": "1111111",
                             "LOCATION": "loc1",
                             "operation": "IST"
              }
        }
```

10.4 Name : TESTSUITE_START
Type : TESTPROCESS
subtype : PROCESS_UPDATE

Description : Whenever a test run within an operation begins

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**Location of where we are scanning the item to.**
NUMBEROFTESTSETS	**Integer**	**False**	**The number of test sets that are going to be executed within this operation.**
LOGOP	**String**	**True**	**The logical operation being performed**
GUTI	**String**	**True**	**The unique identifier of the test**
FAMILY	**String**	**False**	**Family of product**
TESTSUITE_STARTTIME	**Date/Time to ms**	**True**	**The start time of the test to ms level.** `UTC formatted:` 2018-11-07T00:25:00.073876Z

```
        {
              "type": "TESTPROCESS"
              "subtype": "PROCESS_UPDATE"
              "name" : "TESTSUITE_START"
              "params": {
                             "SERIALNO": "1111111",
                             "LOCATION": "loc1",
                             "LOGOP": "IST",
                             "NUMBEROFTESTSETS": 50,
                            "TESTSUITE_STARTTIME": "2018-11-07T00:25:00.073876Z",
                            "GUTI": "Pb7DQd9cuPMt1lRaPepMURVkhMrWfA939EEU"


              }
         }
```

10.5 Name : TESTSUITE_COMPLETE
Type : TESTPROCESS
subtype : PROCESS_UPDATE

Description: Whenever a test completes with a result of pass fail or aborted.

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION

SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**Location of where we are scanning the item to.**
LOGOP	**String**	**False**	**The operation being carried out.**
TESTSUITE_COMPLETE TIME	**DateTime**	**True**	**When the operation completed. UTC time** **Format: YYYY-MM-DDThh:mm:ssZ** **It is assumed that the time will be UTC however if required , a timezone can be added instead of the UTC ("Z") designator.**
RESULT	**String**	**True**	**Result of the testSuite:** **PASS** **FAIL** **ABORT** **NOOP**
TESTSUITE_STARTTIME	**DateTime**	**True**	**As per start time however this is the time when the test completed.**
GUTI	**String**	**True**	**Unique identifier**

```
{
    "type": "PUBLIC"
    "subtype": "TESTPROCESS"
    "name" : "TESTSUITE_COMPLETE"
    "params": {
                  "SERIALNO": "1111111",
                  "LOCATION": "loc1",
                  "LOGOP": "IST",
                  "TESTSUITE_COMPLETETIME": "2018-11-07T00:25:00.073876Z",
                  "TESTSUITE_STARTTIME":" 2018-11-06T23:25:00.073856Z
                  "RESULT": "PASS",
                  "GUTI": "Pb7DQd9cuPMt1lRaPepMURVkhMrWfA939EEU"
```

10.6 Name : TESTSET_START
Type : TESTPROCESS
Subtype : PROCESS_UPDATE

Description: Whenever a test begins

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**Location of where we are scanning the item to.**
TESTSET_NAME	**String**	**True**	**The name of the test set that is starting.**
TESTSET_NUMBER	**Integer**	**True**	**The test set number in the list of test sets. Provides a cursor position.**
TESTSET_STARTTIME	**DateTime**	**True**	**When the operation completed. UTC time** **Format: YYYY-MM-DDThh:mm:ssZ** **It is assumed that the time will be UTC however if required, a timezone can be added instead of the UTC ("Z") designator.**
GUTI	**String**	**True**	**The unique test identifier, indicating the test run that this event belongs.**
TESTSUITE_STARTTIME	**DateTime**	**True**	**The test suite start time to which this test set belongs.**

```
        {
                “type”: “PUBLIC”
                “subtype”: “TESTPROCESS”
                “name” : “TESTSET_START”
                “params”: {
                        “SERIALNO”: “1111111”,
                        “LOCATION”: “loc1”,
                        “TESTSET_NAME”: “memTest”,
                        “TESTSET_NUMBER”: 2,
                        “TESTSUITE_STARTTIME”: “2018-11-06T12:25:00.074376Z”,
                        “TESTSET_STARTTIME”, “2018-11-06T18:25:00.073876Z”,
                          “GUTI”: “Pb7DQd9cuPMt1lRaPepMURVkhMrWfA939EEU”
                }
         }
```

10.7 Name : TESTSET_COMPLETE
Type : TESTPROCESS
Subtype : PROCESS_UPDATE

Description: Whenever a test set completes within a TESTSET

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
SERIALNO	**String**	**True**	**The serial number that is being scanned into the location**
LOCATION	**String**	**True**	**Location of where we are scanning the item to.**
TESTSET_NAME	**String**	**True**	**The name of the test set which has completed.**
TESTSET_NUMBER	**Integer**	**True**	**The test set number in the list of test sets. Provides a cursor position.**
TESTSET_STARTTIME	**DateTime**	**True**	**The time when this test set started.**

TESTSET_COMPLETETIME	**DateTime**	**True**	**When the operation completed. UTC time** **Format: YYYY-MM-DDThh:mm:ssZ** **It is assumed that the time will be UTC however if required, a timezone can be added instead of the UTC ("Z") designator.**
GUTI	**String**	**True**	**The overall unique identifier.**
RESULT	**String**	**True**	**The result of the test set.**

```
{
        "type": "PUBLIC"
        "subtype": "TESTPROCESS"
        "name" : "TESTSET_COMPLETE"
        "params": {
                          "SERIALNO": "1111111",
                          "LOCATION": "loc1",
                          "TESTSET_NAME": "memTest",
                          "TESTSET_NUMBER": 2,
                          "RESULT": "PASS",
                          "TESTSET_COMPLETETIME": "2022-09-06T12:21:00.033876Z",
                          "GUTI": "Pb7DQd9cuPMt1lRaPepMURVkhMrWfA939EEU"
        }
}
```

10.8 Environment Events

When testing a product, it is desirable to keep track of the environment in which the test is taking place, such as humidity, temperature, and air particle quality as these could impact the products under test and ultimately affect the outcome of the test being performed.

Frequency of these events will be dictated by the Oracle product and process engineers, however the mechanism to publish these is the same as the test process events
These events are detailed as follows.

10.8.1 Name : TEMPERATURE
Type : PUBLIC
Subtype : ENVIRONMENT

Description: temperature reading at a frequency defined by the product and test engineers.

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
LOCATION	**String**	**True**	**Location within the shop floor area**
AREA	**String**	**True**	**The area within the shop floor where the reading pertains to. A factory may be divided into multiple areas such as building 1 , Assembly etc.**
DEVICE	**String**	**True**	**The device that the temperature was read from. Each device will be unique and could be your own naming convention or even the serialnumber of the device.**
DATETIMEOFREADING	**DateTime**	**True**	**When the operation completed. UTC time** **Format: YYYY-MM-DDThh:mm:ssZ** **It is assumed that the time will be UTC however if required, a timezone can be added instead of the UTC ("Z") designator.**
UNITOFMEASURE	**String**		**C (centigrade) or F (Fahrenheit)**
TEMPERATURE	**Float**		**Temperature reading, 2 decimal places.**

```
{
      “type”: “PUBLIC”
      “subtype”: “ENVIRONMENT”
      “name” : “TEMPERATURE”
      “params”: {
                    “LOCATION”: “LOC1-A1”,
                    “AREA”: “ORACLE-ASSEMBLY”,
                    “DEVICE”: “sensor1”,
                    “TEMPERATURE”: 22,
                    “UNITOFMEASURE”: “C”,
                    “DATETIMEOFREADING”: “2022-11-01T23:44:29.200Z”
      }
}
```

10.8.2 Name : HUMIDITY
Type : PUBLIC
Subtype : ENVIRONMENT

Description: Humidity reading from the particular area.

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
LOCATION	**String**	**True**	**Location within the shop floor area**
AREA	**String**	**True**	**The area within the shop floor that the reading pertains to.**
DEVICE	**String**	**True**	**The device that the temperature was read from**
DATETIMEOFREADING	**DateTime**	**True**	**When the operation completed. UTC time** **Format: YYYY-MM-DDThh:mm:ssZ** **It is assumed that the time will be UTC however if required, a timezone can be added instead of the UTC ("Z") designator.**
UNITOFMEASURE	**String**		**ABSOLUTE, RELATIVE, SPECIFIC**
HUMIDITY	**Float**		**Accurate to 1 decimal place.**

```
{
 “type”: “PUBLIC”
 “subtype”: “ENVIRONMENT”
 “name” : “humidity”
 “params”: {
             “LOCATION”: “LOC1-A1”,
             “AREA”: “ORACLE-ASSEMBLY”,
             “DEVICE”: “sensor1”,
             “HUMIDITY”: 22,
             “UNITOFMEASURE”: “ABSOLUTE”,
             “DATETIMEOFREADING”: “2022-11-01T23:44:29.200Z”
 }
}
```

10.8.1 Name : PARTICLECOUNT
Type : PUBLIC
Subtype : ENVIRONMENT

Description: Air quality reading for a particular area on the floor.

Parameters

NAME	TYPE	REQUIRED	DESCRIPTION
LOCATION	**String**	**True**	**Location within the shop floor area.**
AREA	**String**	**True**	**The area within the shop floor that the reading pertains to.**
DEVICE	**String**	**True**	**The device that the temperature was read from.**
DATETIMEOFREADING	**DateTime**	**True**	**When the operation completed. UTC time.** **Format: YYYY-MM-DDThh:mm:ssZ**

			It is assumed that the time will be UTC however if required, a timezone can be added instead of the UTC ("Z") designator.
PARTICLESIZE	**Float**	**True**	**Number of microns that the reading relates to.**
PARTICLECOUNT	**Long**	**True**	**Particles per million reading for the particular micron size.**
FLOWRATE	**Float**	**False**	
DURATION	**Float**	**False**	**Sample period in seconds.**
SUMASS	**Float**		
SUMM3	**Float**	**False**	**Sum in square meters.**
SUMFT3	**Float**	**False**	**Sum in square feet.**
SUM	**Float**	**False**	
DIFFERENTALMASS	**Float**	**False**	
DIFFERENTALM3	**Float**	**False**	**Differential Mass in square meters.**
DIFFERENTALFT3	**Float**	**False**	**Differential Mass in square feet.**

```
{
            "type": "PUBLIC"
            "subtype": "ENVIRONMENT"
            "name" : "PARTICLECOUNT"
            "params": {
                          "LOCATION": "LOC1-A1",
                          "AREA": "ORACLE-ASSEMBLY",
                          "DEVICE": "sensor1",
                          "PARTICLECOUNT": 300000,
                          "PARTICLESIZE": 0.5,
                          "DATETIMEOFREADING": "2022-11-01T23:44:29.200Z"


            }
}
```

11 SEND REAL TIME USING ORACLE TDMS REST SERVICES

It is possible to send real time events such as temperature / humidity etc. using TDMS public REST services. This provides a simpler mechanism to publish events.

Note that the following end points are detailed to the openApi 3 specification (formally known as swagger) and will be provided upon request.

11.1 API Gateway Host URL

The tdms api is accessible through the following url. By adding the servicePrefix (see each section below) , followed by the api end point, you have the ability to push real time data to Oracle TDMS.

At the time of writing this document, the following is the valid endpoint for the TDMS API.

https://cxualo5hojdi5lrzbqbvvz3czi.apigateway.us-ashburn-1.oci.customer-oci.com/<servicePrefix>

Please refer to the TDMS openAPI (swagger) specifiction(s) for the complete set of params, query and header params. Do not use this document when developing your communication software as this document may be slightly out of date whereas the openApi specn will reflect the present deployment on Oracle cloud.

11.2 Valid Path/Query Parameters

Please be aware that when you are constructing the URL path, you must ensure that you do not have any invalid characters and that you only have alphanumeric (a-z A-Z 0-9), and use – (dash) _ (underscore) as separators.

Additionally, when sending serial number path parameters, if you are using the plus + within the serial number, this must be encoded to %2B.

Dates and Times which occur within the query parameters do not need to be encoded (i.e. do not encode the : (colon) separator) . Sending as 2021-10-10T12:23:34.000Z without further manipulation is acceptable within the API.

11.3 Mandatory Header Fields:

All requests must contain the following header fields.

.

Name: x-tdms-apikey

Type: String

Description : This is your key for accessing the service. Without the key you have no permission to invoke any of the services. Please ensure that you keep this value safe at all times as this identifies your application to TDMS.

Name: x-tdms-applicationid

Type: String

Description: You must specify the application that is sending the request from your site. Again, without this the request will be rejected. You should ensure that you have no spaces but instead use _ or – where required. This applicationid is your software identifier. It is an arbitrary string which TDMS uses as the senderid. eg. MYPUBLISHER-rev1.0.

Name: x-tdms-siteid

Type : int

Description: You will be provided a unique siteid along with your x-tdms-apikey. These 2 values should be set on each request as it identifies you to the TDMS services.

11.4 Environmental End points

Service prefix: environmental

11.4.1 Temperature Endpoint

Method: POST

PATH: `/temperature/{area}/{location}/{deviceid}/{tempvalue}`

Headers:

- unitOfMeasure
  - C – centigrade (default)
  - F – Fahrenheit
- Required: false

11.4.2 Humidity Endpoint

Method : POST

PATH : `/humidity/{area}/{location}/{deviceid}/{humidityvalue}`

Headers :

- unitOfMeasure
  - RH – relative humidity (default)
  - AH – absolute humidity
  - SH – specific humidity
- Required: false

Particle Count Endpoint

Method: POST

PATH: undefined

11.4.3 Particle Endpoint

This end point should be used to send particle readings to TDMS.

Method: POST

PATH: `/particle/{area}/{location}/{deviceid}/{particleValue}`

`Param: microns enum: 0.3 0.5 1 5 10`

`Header: unitOfMeasure m3 ft3`

11.5 Manufacturing Process Events

Service prefix: process

Whenever you require to update Oracle as to the progress of a unit within the shop floor. You should use the following end points.

11.5.1 Unit Serialnumber created (released)

Whenever you have started a server or rack, this event would be sent. This indicates that the serialnumber has been released on your floor , as part of a work order .

Method POST

PATH : /unit/{serialnumber}

/rack/{serialnumber}

These 2 end points also details the Work order and Purchase order details.

11.5.2 Unit Serialnumber completed

Whenever you have completed a server or rack, this event would be sent. This indicates that the server or rack has completed the whole process within your shop floor.

Method DELETE

PATH : /unit/{serialnumber}

/rack/{serialnumber}

11.5.3 Unit has been placed into a physical location to undergo an operation

As a serialnumber moves around your shop floor, it will be placed into physcial areas known as locations. This event is sent as a serialnumber is placed into a physcial area on the floor.

Method POST

PATH : /location/{location}/{serialnumber}

11.5.4 Unit has been removed from a location after an operation has completed

Method DELETE

/location/{location}/{serialnumber}

11.5.5 Unit is starting a suite of tests to satisfy a particular operation at a location

Method POST

/testsuite/{location}/{operation}/{serialnumber}

11.5.6 Unit has completed the tests for the particular operation.

Method DELETE

/testsuite/{location}/{operation}/{serialnumber}

11.5.7 Unit is starting a testset within a testsuite.

Method PUT

/teststep/{location}/{operation}/{teststepname}/{serialnumber}

11.5.8 Unit has completed a teststep within a testsuite

Method DELETE

/testset/{location}/{operation}/{testsetname}/{serialnumber}

11.6 EVENT SERVICES

To send events to a local instance of TDMS installed within your site, you can take advantage of the TDMS EVENT REST services or TDMS EVENT SERVLET.

11.6.1.1 Sending to the Local Rest service

To send events to TDMS REST service, you use the following URL, and send an event to the URL as if you were sending to the KAFKA stream, i.e. it must be formatted in JSON. However, the formatting is slighting different, in that the name type and subtype are part of the service path.

URL	http://<TDMS server ip address>:<port>/EVENTHANDLER/event/{typeof event}/{subtype event}/{name of event}
Method	POST
Content Type	Application/json
Response	Application/json
Response Code	201 OK
	404 Event name not found.

Body	Within the body of the message the params should be set as json formatted attributes .
	E.g.
	{
	  “attributeA”: “valueA”
	}

Revision History

REVISION	MODIFIED BY	DATE	COMMENT
01	N/A	3 Apr 2007	Initial Release
02	N/A	05 Sept 2014	Updated from Sun to Oracle; removed Webdocs references and obsolete documents; and updated Apache FTP server link and FTP Protocol Related Documents link.
03	N/A	01 Apr 2019	Complete rewrite and update. More detail added with respect to the data requirements expected from an External Manufacturer. Changes include (but are not limited to): Added Section 1: Acronyms Section 4: Definitions Section 6: Test Result Delivery into TDMS Section 7: Transferring Results to TDMS Section 8: Real Time Events into TDMS Section 9: Failure Analysis XML Section 10: Purchase Order Section 11: Purchase Order Sub-Elements Section 12: Remote Connectivity Removed Section1: Transfer Overview Section 3.1.1: Virtual Private Network Section 3.3: Login and Password Account Credentials Section 3.5: Monitoring Schedule Section 4: Supported Protocols Section 5: Supported FTP servers Section 6: Backlog storage Section 7: Bandwidth Section 8: Data Corruption Appendix A: List of Responsibilities
04	N/A	12 July 2019.	Updated format, corrected revision, removed outdated reference documents and links, updated change history.
05	N/A	04 Apr 2022	Changes for Kafka services , final release before review All events now in sync with the event system definitions on internal TDMS.
06	N/A	08 June 2022	Corrected spelling errors. Added detail on iso8601 formatting. Real time service signatures corrected
07	N/A	08 June 2023	Changes made to the timezone definition to provide more information as to the legal time zone values.

			New element called DOCREF to indicate how to cross reference a document in another file. Added in reference to the oracle public API for sending real time process and environmental data. Modified the associated files to ensure that the diagrams are correct and also the specification details. Removed the kafka section as the API now replaces the deprecated API. Added in comments to indicate valid characters for query and path params.
08	N/A	05 Oct 2023	Update table on page 29
09	N/A	26 Aug 2024	Removed *Test Infrastructure Engineering* from title. Added a brief reference in the INTRODUCTION/OVERVIEW section to the method by which XSD files will be distributed to suppliers given their removal from the AQP matrix.
10	N/A	07 Apr 2025	Inclusion of specification for a META-INF/MANIFEST.MF file containing select metadata. Inclusion of requirement that the MACADDRESS is a mandatory data element to be captured in ICT for ROT cards.
11	N/A	25 July 2025	Inclusion of new element within the PRODUCT element called, MANUFACTURERDETAIL which allows for the definition of the SERIALNUMBER and PARTNUMBER . This is an optional element but something that we advise should be added to the overall structure. Minor tidy up of grammar, and inclusion of note to advise on GUTI.

- When Document Template is complete, email source file to eso_business_docs_us_grp@oracle.com
- All hard copies of this document are uncontrolled and are to be used for reference only.
- For questions or comments about this document, please send an email to: eso_business_docs_us_grp@oracle.com