



# Open Source oneM2M Conformance Testing Tool - oneM2MTester User Guide

Version:	<b>V2.2.0</b>
Editor:	<b>KETI</b>
Organization:	<b>KETI, Ericsson, EGM, Sejong University</b>
Date:	<b>2018 September</b>
Document type:	<b>User Manual</b>
Dissemination level:	<b>Public</b>

## Copyright and Disclaimer of Liability

This document may contain technical inaccuracy or typing errors, and the author does not have any responsibility on this matter.

The contents of this document can be changed or added regularly, and the relevant corrected version will be added to the document under the title named “New Edition” in consecutive order. The product or program mentioned in this document may be changed or modified without any prior notice.

## Content

1. Introduction to oneM2MTester Project.....	7
1.1. Aim of oneM2MTester Project .....	7
1.2. oneM2MTester Architecture .....	7
1.2.1. Functionalities .....	7
1.2.2. Software Architecture.....	7
1.3. oneM2M Abstract Test Suite .....	8
2. oneM2MTester Runtime Environment.....	9
2.1. Java JDK Installation .....	9
2.2. Eclipse and plugins install .....	9
2.2.1. Eclipse Titan Plugins.....	9
2.2.2. Eclipse Titan Core .....	12
2.3. Configurations .....	13
2.3.1. TITAN Core configuration.....	13
2.3.2. Eclipse Plugins configuration .....	14
3. Start using oneM2MTester .....	15
3.1. oneM2M Abstract Test Suite .....	15
3.2. oneM2MTester System Adapter .....	15
3.2.1. Download and Import System Adapter Source .....	15
3.2.2. Run Testcases against Implementation Under Tests .....	21
3.2.3. Check Testcase Execution Result.....	25
4. External References.....	27
4.1. Eclipse TITAN Documentation .....	27
References.....	27
5. Appendix.....	28
Appendix-C. Run Executable Test Suite using TITAN CLI.....	28

## Figure

Figure 1. oneM2MTester architecture .....	8
Figure 2. Checking the version of java .....	9
Figure 3. Screen from Eclipse Titan download homepage.....	10
Figure 4. Installing plugins .....	11
Figure 5. Installing plugins .....	11
Figure 6. Add Repository screen .....	12
Figure 7. Installing plugins .....	12
Figure 8. Installing plugins .....	12
Figure 9. Eclipse TITAN Preferences .....	14
Figure 10. Import from File System Option .....	16
Figure 11. Import downloaded oneM2MTester package.....	17
Figure 12. Switching on Titan project nature .....	17
Figure 13. Import from Existing Projects into Workspace.....	18
Figure 14. Select the unzipped oneM2MTester package and import into the workspace.	18
Figure 15. Importing project with Titan Project Descriptor file.....	19
Figure 16. Selecting oneM2M Tester project descriptor file .....	19
Figure 17. oneM2MTester project structure .....	20
Figure 18. TITAN Runtime Configurations.....	22
Figure 19. Run configuration button in Eclipse TITAN button bar.....	22
Figure 20. Customize Perspective menu .....	23
Figure 21. TITAN Execution Controller.....	24
Figure 22. Log view for executable test suite executionz.....	25

Figure 23. Generated log file for executable test suite execution..... 26

Figure 24. Capture of Running Main Controller..... 28

Figure 25. Create a MTC and enter listening state ..... 29

Figure 26. Capture of established connection between MTC and HC ..... 30

Figure 27. Running log of test executable test suite..... 31

Table

Table 1 Eclipse TITAN Log options ..... 26

# 1. Introduction to oneM2MTester Project



## 1.1. Aim of oneM2MTester Project

The oneM2MTester project is an open source oneM2M conformance testing project which was initialized by Korea Electronics Technology Institute (KETI) in January 2016. The project aims to develop and distribute an open source conformance testing tool for oneM2M implementations' evaluation (including CSE and AE). The project is a collaboration with nine international members including Sejong University (SJU), Telecommunications Technology Association (TTA), European Telecommunications Standards Institute (ETSI), Sensinov, Easy Global Market (EGM), InterDigital, Ericsson, LAAS-CNRS and DTNC.

## 1.2. oneM2MTester Architecture

### 1.2.1. Functionalities

The oneM2MTester development is based on Eclipse Titan, which is an open source testing tool providing TTCN-3 testing system environment, and extended with system adapter and codec for oneM2M protocol bindings (i.e. HTTP, CoAP and MQTT) and serializations (i.e. XML and JSON).

The oneM2MTester implements system adapters and codec within a oneM2M port to enable the communication between TTCN-3 test system and the implementation under test (IUT), i.e. encoding the oneM2M request primitive messages into protocol binding-specific messages before sending the messages to the IUT and vice versa.

The oneM2MTester makes full use of the TTCN-3 test system environment provided by Eclipse TITAN, including TITAN Designer, TITAN Executor, and TITAN LogViewer to handle the oneM2M port development, testcase execution and test result evaluation. The oneM2MTester is designed to be able to select and execute a selection of testcases or the complete test set automatically and generate a trusted verdict for the corresponding testcase execution. In addition, the TITAN LogViewer plugin provides graphical log presentation to assist users in debugging the IUT. A number of logging options are provided to let users choose their preferences for logs. oneM2MTester also reuses the TITAN configuration file which includes runtime parameters for the IUT (such as host address, CSE name, CSE ID etc.) to load the runtime configuration of IUT into test system prior executing testcases against the IUT.

### 1.2.2. Software Architecture

As shown on Figure 1, oneM2MTester architecture is built based on that of TITAN so that oneM2MTester can use TITAN system components including TITAN compiler, component handling, test management and test logging. In fact, component handling and test management are implemented within the Main Controller and Host Controller for handling one Main Test Component and multiple Parallel Test Components. TITAN compiler compiles TTCN-3 testcase code (oneM2M Abstract Test Suite) into a Executable Test Suite (ETS).

The oneM2M port implements both a system adapter through abstract socket which handles to establish TCP connections between TTCN-3 test system and the IUT, and a codec for encoding oneM2M request primitives to a protocol-binding specific message represented in XML or JSON serialization.

The TITAN test system loads the IUT runtime configurations first and executes the Executable Test Suite (ETS) against the IUT after establishing a connection (such as TCP and UDP etc.) with IUT. The running logs are saved in logfile and displayed in the Console panel during testcase execution while the execution result is generated with verdict and displayed in the TITAN test result panel at the end of the execution.

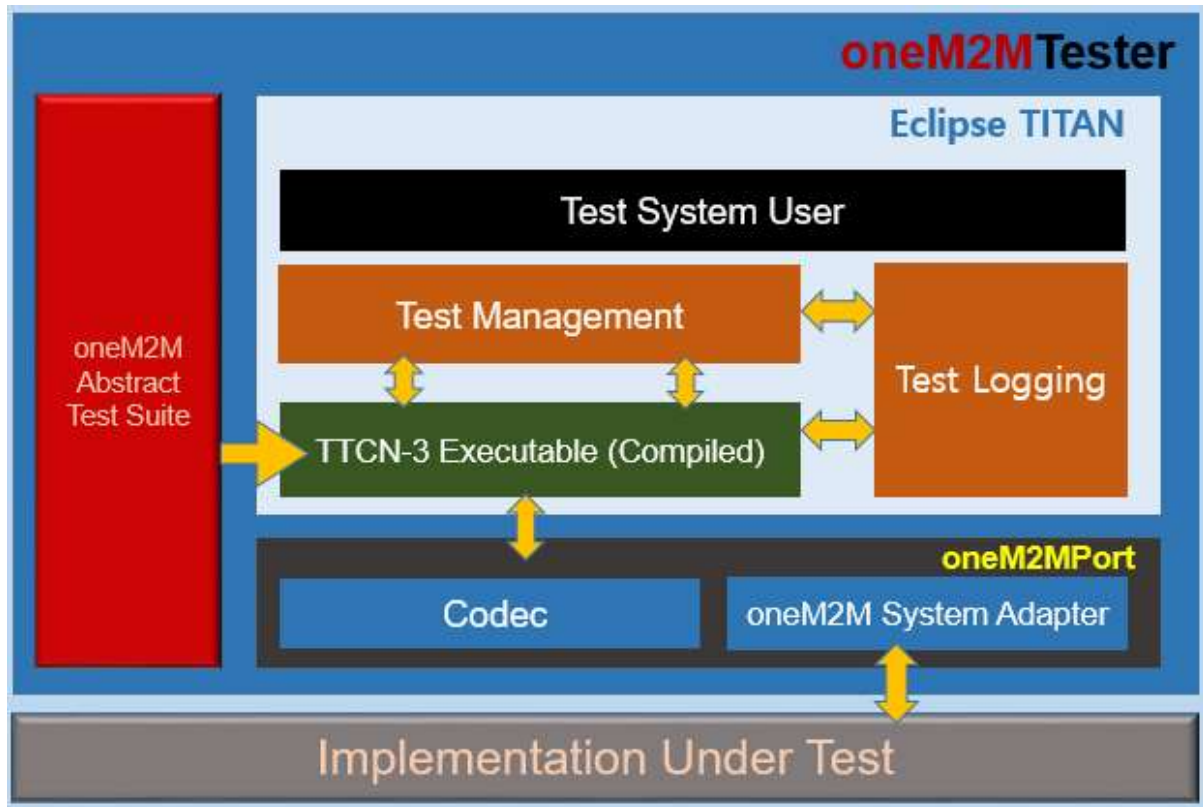


Figure 1. oneM2MTester architecture

### 1.3. oneM2M Abstract Test Suite

A oneM2M Abstract Test Suite, abbreviated to ATS, is a oneM2M standard document for conformance testcases which are designed for CSE and AE and written in TTCN-3. The ATS is designed and used for oneM2M certification testers to evaluate and certify oneM2M implementations and/or developers to self-test their own oneM2M implementations. The ATS development work is conducted by the ATS Task Force-001 while managed by oneM2M Testing Work Group. For the time being, the development work of ATS is not complete and it only covers functions for Registration, Data management and repository. In addition, the current ATS only contains testcases target to CSE implementations, i.e. IN-CSE, ASN-CSE and MN-CSE.

The ATS defines abstract structure for oneM2M resourcePrimitives as well as requestPrimitives and responsePrimitives which are sent through oneM2M reference points such as mca and mcc to the IUT.

In order to enable the communication between the TTCN-3 test system and the IUT, the system adapter which is oneM2M protocol binding specific is required to convert oneM2M requestPrimitives to protocol-specific messages.



## 2. oneM2MTester Runtime Environment

To set the Testing environment, following 4 steps are needed.

- A. Java JDK\_installation
- B. Eclipse and plugins install
- C. Titan Core install
- D. Configurations

Detailed descriptions are below.

### 2.1. Java JDK Installation

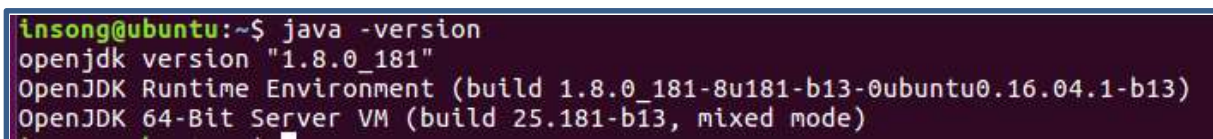
The Java JDK can be obtained from:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

In other ways, if you are using Ubuntu, you can easily install JDK using Terminal.

- `sudo add-apt-repository ppa:openjdk-r/ppa`
- `sudo apt-get update`
- `sudo apt-get install openjdk-8-jdk`

After type these three code, you can check version of java by typing it: `java -version`.



```
insong@ubuntu:~$ java -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-8u181-b13-0ubuntu0.16.04.1-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
```

Figure 2. Checking the version of java

JDK is successfully installed, if you can see the JDK version.

The Eclipse IDE from the Eclipse Foundation<sup>1</sup> is used in oneM2MTester because the tool that compiles and runs the tests, Eclipse TITAN, is based on Eclipse. The oneM2MTester tries to support the latest version of Eclipse. We recommend users to check and download the latest Eclipse IDE release from <https://www.eclipse.org/downloads/packages/release/Neon/3>. Note that Eclipse IDE Neon has been successfully evaluated with Eclipse TITAN. The Eclipse installation is pretty straightforward using the Eclipse Installer. It will guide the user through the installation. The only things to set up are the installation directory and the Eclipse package(s) to install, and the Eclipse Installer will take care of the rest. For more details on installation of Eclipse IDE, users may refer to general Eclipse IDE installation guide, which is out of scope of this user guide.

### 2.2. Eclipse and plugins install

#### 2.2.1. Eclipse Titan Plugins

---

<sup>1</sup> [www.eclipse.org](http://www.eclipse.org)

The TTCN-3 TITAN plugins provide efficient way to create the TTCN-3 test suites, based on the Eclipse Platform. There are 4 different plugins: TITAN Designer, TITAN Executor, TITAN LogViewer and Titanium. TITAN Designer provide the usual IDE features for TTCN-3, ASN.1 and the Titan runtime configuration file. Also, using TITAN Core, it can build TTCN-3 code and control its execution. TITAN Executor enables the execution of test cases, TTCN-3 control parts or test sets defined by the user. It also gives a quick overview of test execution progress, including test case verdicts. TITAN LogViewer can merge TITAN log files and analyse them either in a tabular-textual or in a graphical view. It also allows filtering the log files content. Titanium helps in code structure and code quality analysis.

The Titanium plugin requires support from Titan Designer, Apache POI 3.9.0, JUNG (Java Universal Network/Graph Framework), and Commons-Collections with Generics. Users don't need to bother the installation of those plugins and search for the required libraries, because TITAN provides a plugin package containing all those plugins and required libraries in one archive which can be downloaded from Eclipse Titan download page shown as below by clicking the **Eclipse plug-ins x.x.x plus dependencies** package.

<https://projects.eclipse.org/projects/tools.titan/downloads>

Note that the plugins that are used in OneM2MTester are TITAN Designer feature, TITAN Executor feature and TITAN Log Viewer. The Titanium feature is optional for oneM2MTester. In addition to the system variables definition, the user shall configure the TITAN Plugins in Eclipse after the installation of Eclipse. When you go to the URL above, you will see screen like Figure 3.

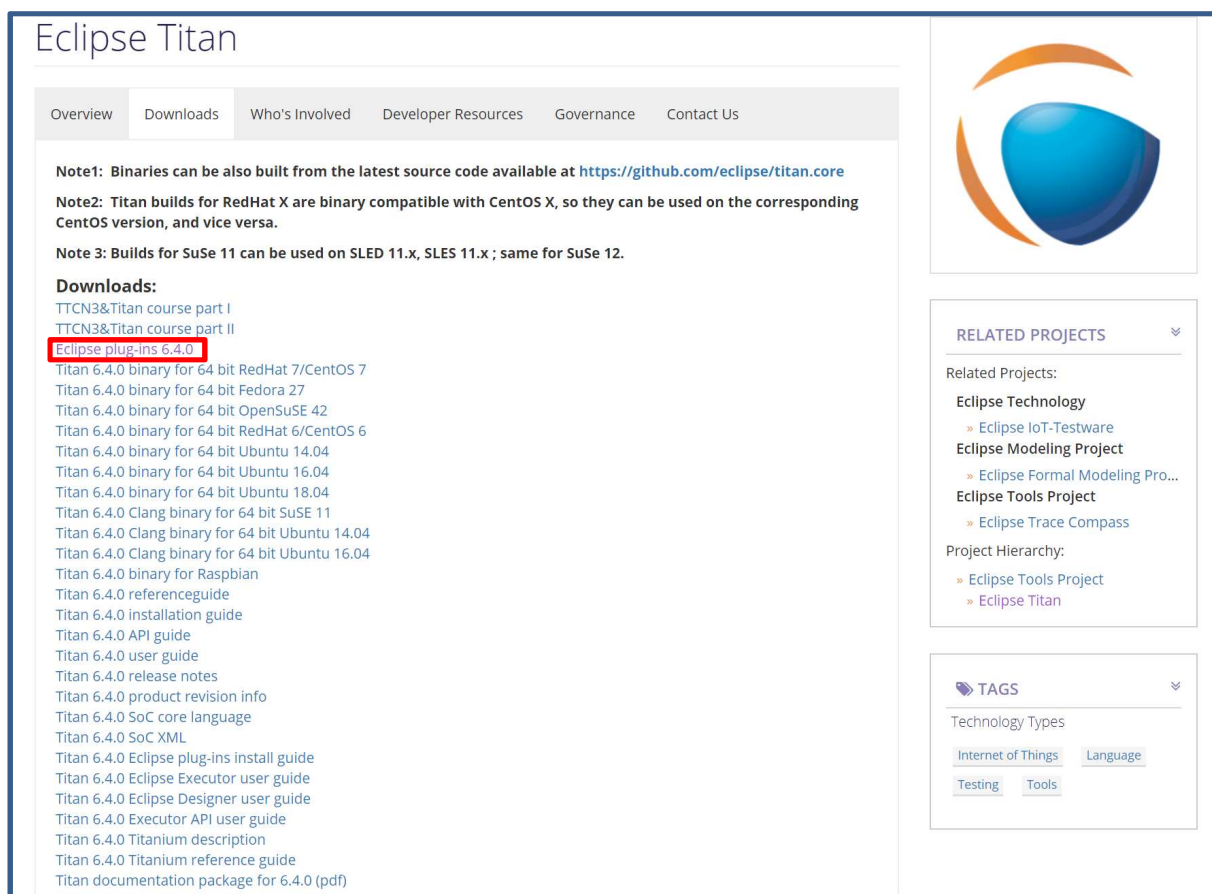


Figure 3. Screen from Eclipse Titan download homepage

After downloading Eclipse plug-ins x.x.x from the website, install it in Eclipse.

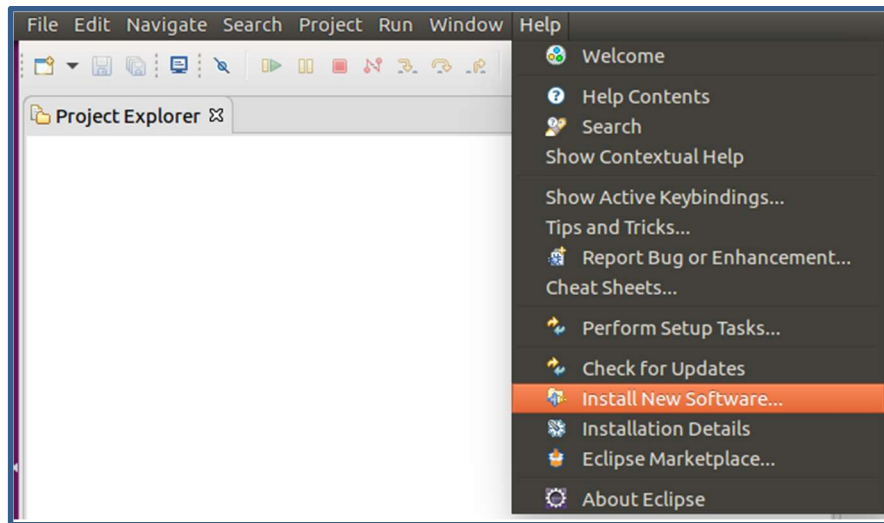


Figure 4. Installing plugins

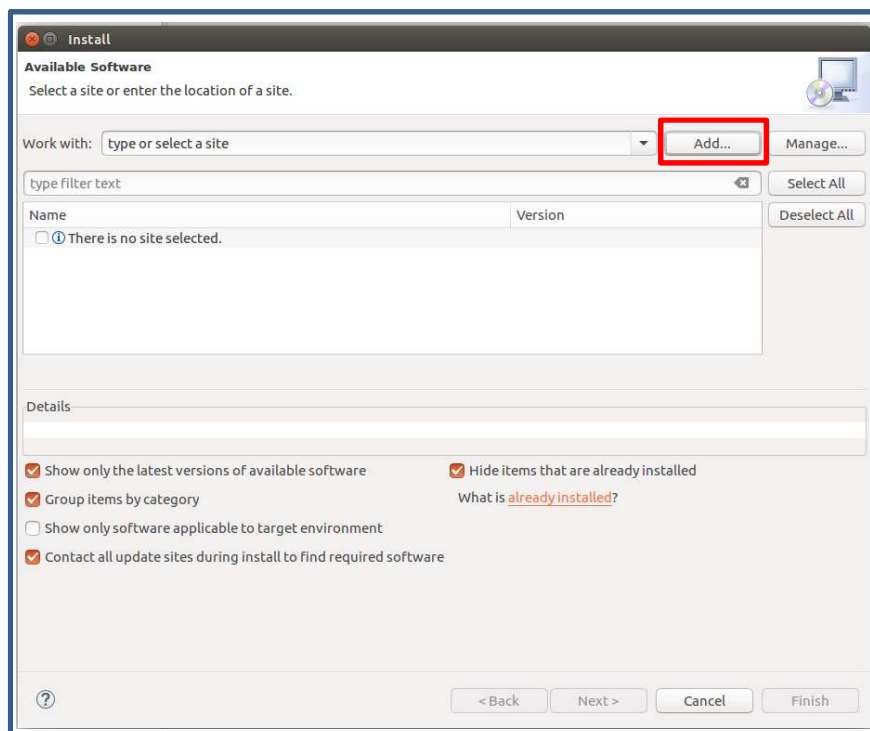


Figure 5. Add button in install menu

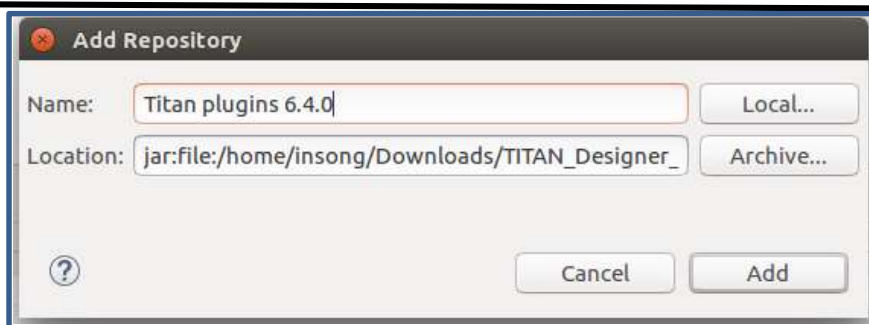


Figure 6. Add Repository screen

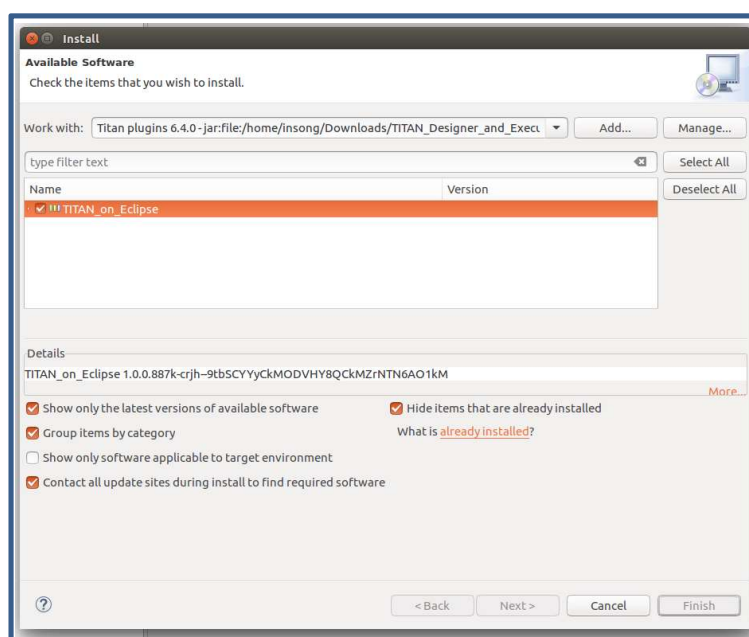


Figure 7. Installing plugins

If you face the security warning message, just click the OK button.

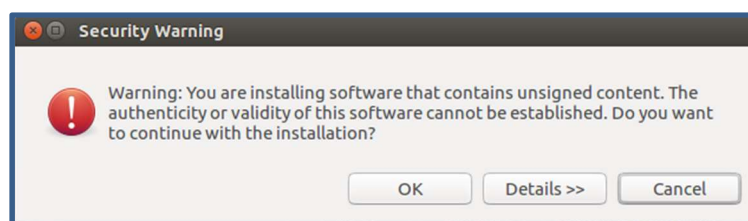


Figure 8. Installing plugins

### 2.2.2. Eclipse Titan Core

The Titan compiler builds an executable test suite from the TTCN-3, the test port code and the TITAN runtime library. It is written in C++ and is available for download in binary form from the Eclipse site: <https://projects.eclipse.org/projects/tools.titan/downloads>. Please, choose the version that is compiled for your Linux distribution. The oneM2MTester has been successfully worked on Ubuntu 16.04. The source code is available for download from its GitHub repository: <https://github.com/eclipse/titan.core>.

According to the TITAN Installation Guide (1), there are few prerequisites for TITAN Core on Linux:

- Openssl-0.9.8k
- Libxml-2.7.1
- JDK 1.5.0\_10 or later(JDK 1.8.0 or later is recommended)

In addition to the normal libraries, we need the development packages of these libraries in order to compile TTCN-3 test suites. These test suites should be called openssl-dev (or -devel) or libopenssl-dev (or -devel) and libxml2-dev (or -devel) respectively.

The downloaded binary files for the corresponding distribution of Linux are extracted to an empty directory. Usually this directory is inside the user directory (/home/<userId>/TITAN) or, if the user has administrator privileges and wants to make it available for all users, it can be /usr/local/TITAN, or /opt/TITAN. The archive extraction is done with the command *tar*, for example "*tar xvzf archive-file.tgz*". The details about the installation can be found in the section 2 of (1). The users are always recommended to download and use the latest version of Titan core to work with oneM2MTester. The releases of Eclipse TITAN can be found here

<https://github.com/eclipse/titan.core/releases>

If you can't use the binary version of TITAN, you can always compile it using the source code. The walkthrough for Linux is explained in the README.linux file in the GitHub repository.

<https://github.com/eclipse/titan.core/blob/78a82971bdd606e92e522b9808e27991ddad7dc9/README.linux>

## 2.3. Configurations

### 2.3.1. TITAN Core configuration

In order to use TITAN properly in the command line, we have to tell the system where its binaries are located. Because oneM2MTester is mainly oriented to the Eclipse IDE, this step is optional.

This is done by defining system variables that are needed by TITAN:

- `TTCN3_DIR`: This variable contains the absolute location of the TITAN installation.
- `LD_LIBRARY_PATH`: Here the user needs to specify the OpenSSL shared library, that is provided with TITAN. The path to add is "`$TTCN3_DIR/lib`".
- `PATH`: This variable should contain the location of TITAN binaries. The path is: "`$TTCN3_DIR/bin`".

The other two variables, `MANPATH` and `TTCN3_BROWSER`, are optional:

- In `MANPATH` should be specified the manual files that are provided with TITAN. This usually is "`$TTCN3_DIR/man`".
- In `TTCN3_BROWSER` we should specify the web browser that we want to use for browsing the help files, if Netscape isn't available or isn't preferred by the user.

Under Linux the variables should be declared in the login script. Different shells support different commands in order to add/modify the variables. This, and all other details about the process are explained in section 3 *Setting the User Environment* of (1).

For the beginners, type 'nano .bashrc' in the Ubuntu Terminal. Go to end of the bashrc file and type this information.

```
export TTCN3_DIR=/home/<user id>/titan.core/Install  
export PATH=/home/<userid>/titan.core/Install/bin/:${PATH}  
export LD_LIBRARY_PATH=/home/<userid>/titan.core/Install/lib:${LD_LIBRARY_PATH}
```

Save it after typing it.

### 2.3.2. Eclipse Plugins configuration

In addition to the system variables definition, the user shall configure the TITAN Plugins in Eclipse after the installation.

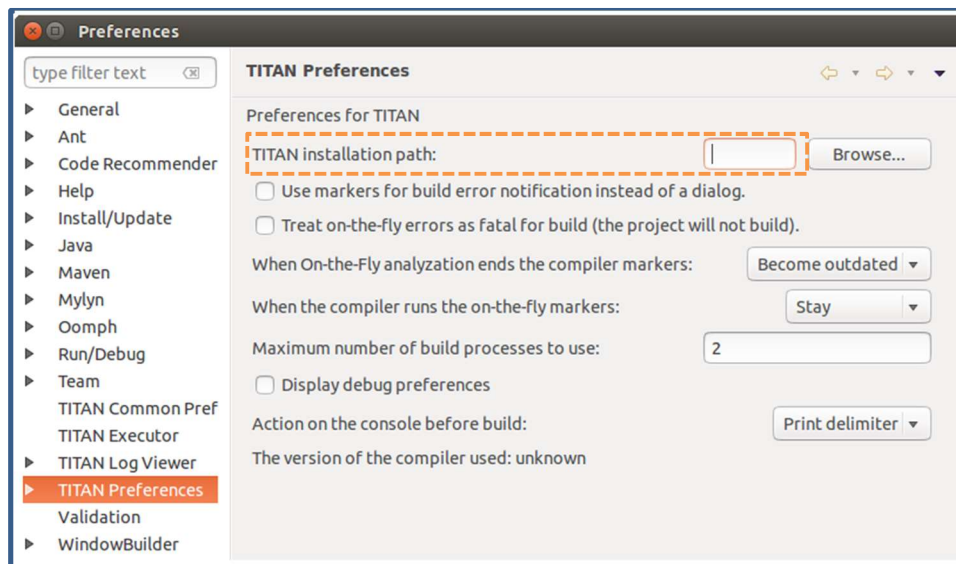


Figure 9. Eclipse TITAN Preferences

Figure 9. Eclipse TITAN Preferences Figure 9 shows Eclipse TITAN preference settings. Here user only needs to configure the **TITAN installation path** to directory of Eclipse TITAN CORE (i.e. the field value of variable `TTCN3_DIR`) you downloaded or cloned from Eclipse TITAN github. For more details, see the section 5 of (2).



## 3. Start using oneM2MTester

This section provides information to guide users to use oneM2MTester (version 2.1) to run testcases against oneM2M implementations (i.e. CSE) one step by step.

### 3.1. oneM2M Abstract Test Suite

The oneM2M test suite contains testcases designed for oneM2M conformance testing. The version of the ATS supported by oneM2MTester version 2.1 is based on oneM2M Release 1.0. The scope of the testcases covers functions as following:

- Registration for AE
- Data Management and Repository for resources including accessControlPolicy, Container, PollingChannel, Schedule, Group, Subscription,
- Access control policy management,
- LocationPolicy management etc.

The testcase in the ATS consists of three parts, a preamble, behavior test body and a postamble, which are designed to be executed in the correct order against an IUT.

- A preamble indicates the pre-conditions that a certain testcase has to meet before running, for instance, AE registration in order to test a container base creation operation.
- The behavior test body includes test details for a certain testcase, for instance, test container creation under the created AE resource which is created during AE registration in preamble step.
- The postamble resets the initial state by deleting the registered AE and created resources underneath it, for example.

The behavior test body is executed only when the preamble is successfully executed, i.e. if the execution of preamble in a certain testcase fails then the test system will set the verdict of this testcase execution to "FAIL". The verdict of a testcase execution will be set to "PASS" only when all the preamble, behavior test body, and postamble are successfully executed in order.

The ATS files are included in the distributed oneM2MTester package and users don't need to bother to download the ATS files separately.

Note that the users are not recommended to modify **any** part of the ATS TTCN-3 code. Instead they can input their server platform (IUT) configurations such as host IP address, CSE name and CSE URI of the IUT, by editing TITAN configuration file (e.g. `OneM2MTesterConfig.cfg`). The ATS code is part of oneM2M standard and not permitted to be modified without permission of oneM2M Testing Work Group.

### 3.2. oneM2MTester System Adapter

#### 3.2.1. Download and Import System Adapter Source

oneM2MTester system adapter is implemented to enable the communication between the test system and the IUT. For the time being, oneM2MTester version 2.1 supports HTTP, CoAp binding and XML, JSON serialization, which are implemented inside of a oneM2M port. MQTT binding will be supported by Tester soon.

The full package of oneM2MTester system adapter is uploaded into GitLab which can be accessed on the link below:

<http://203.253.128.150:7591/onem2m/conformancetesting/OneM2MTester.git>.

GitLab provides two different ways for users to use the source package, one way is to directly download the package from gitlab by clicking the download button in the oneM2MTester project home page. Also, users can also use the following command to clone the oneM2MTester project into local repository:

```
"git clone http://203.253.128.150:7591/onem2m/conformancetesting/OneM2MTester.git"
```

After downloading or cloning the oneM2MTester package, users have to import it into a Titan project. There are three ways to import the oneM2MTester project shown as below.

- A. Create a new Titan project first and then right click the created Titan project and select the import option, then in the pop-up window as shown as Figure 10, expand the option "General" and select "File System". Browse the downloaded oneM2MTester package, select all sub-folders, and finish the importing by clicking "Finish" as shown on Figure 11. After importing, by right clicking on the project's name in the Project explorer window, switch on the Titan project nature as shown Figure 12.

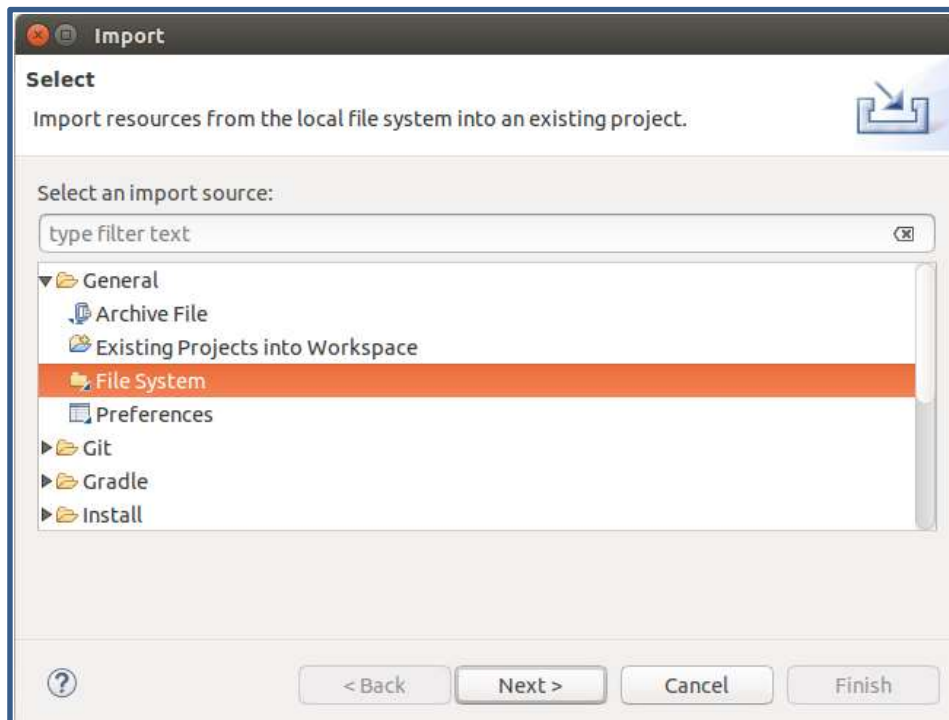


Figure 10. Import from File System Option



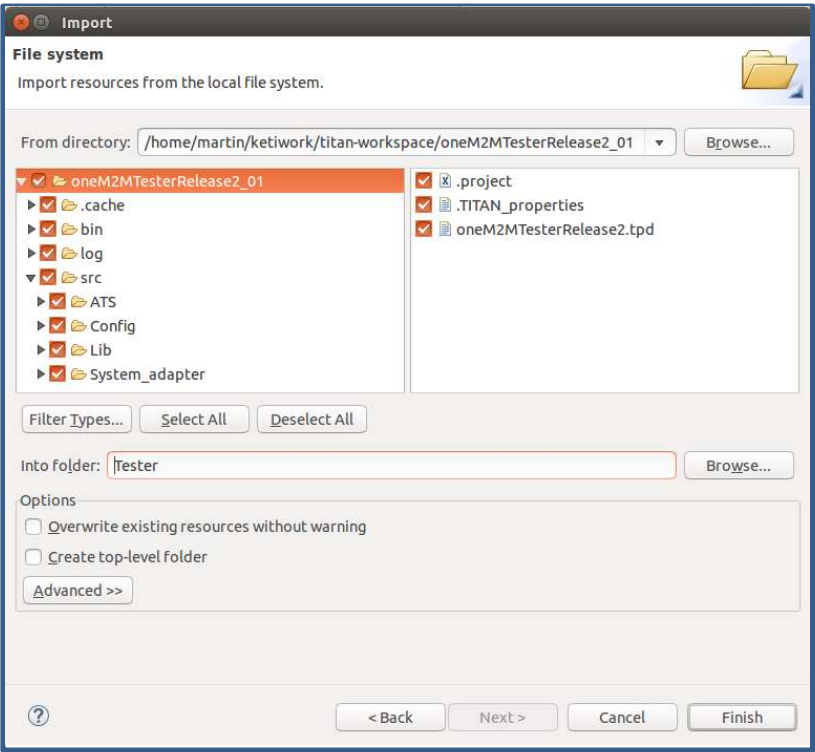


Figure 11. Import downloaded oneM2MTester package

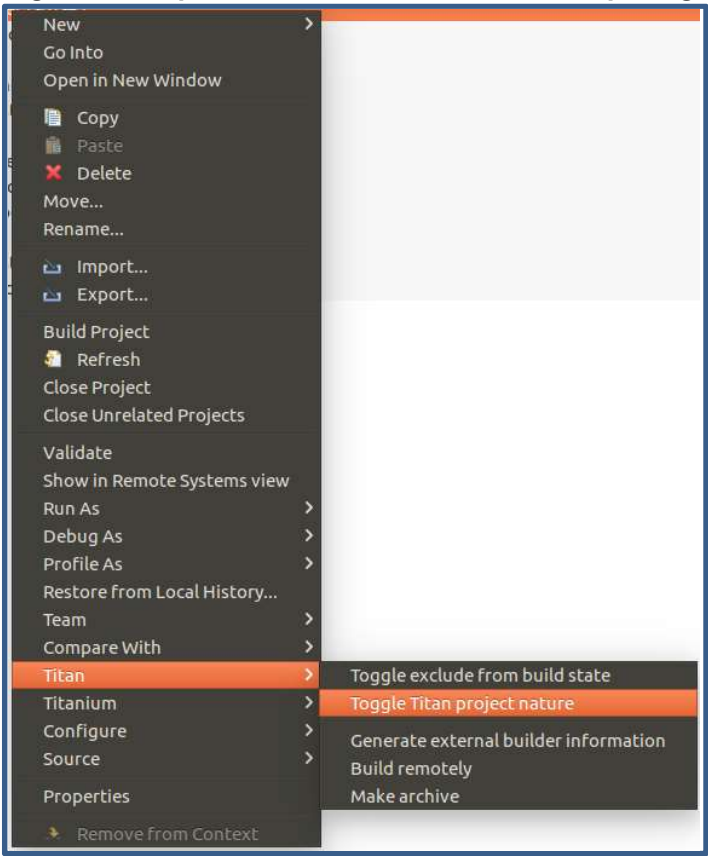
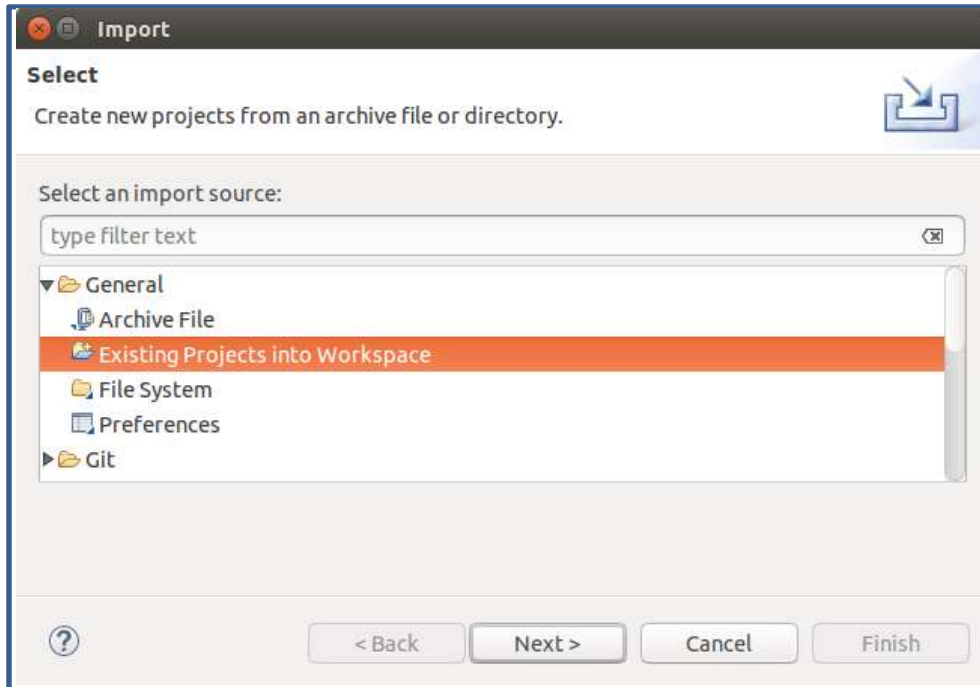
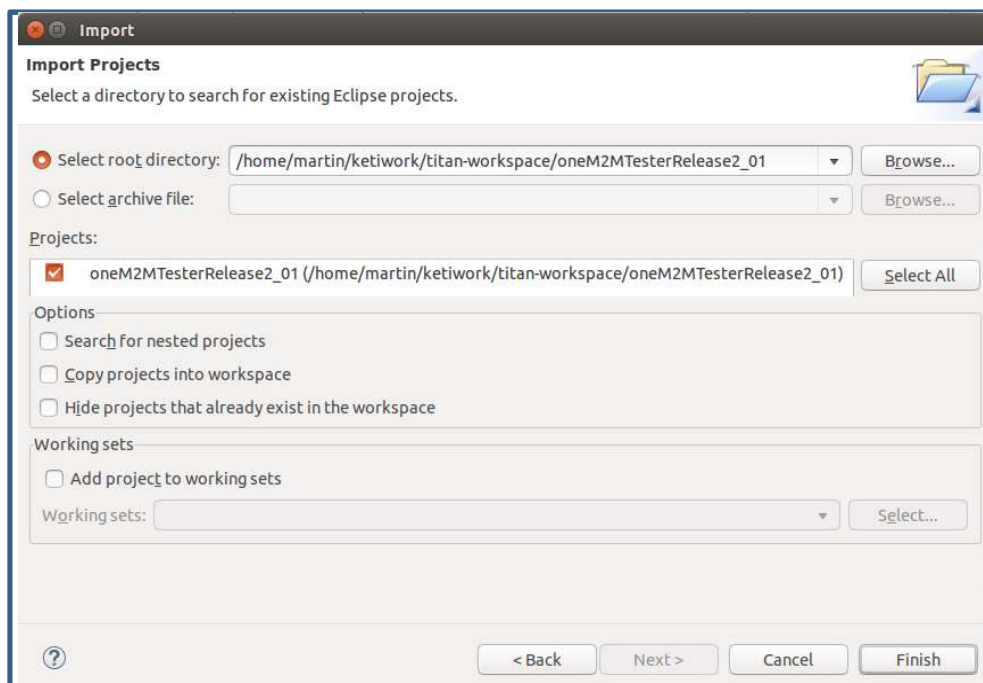


Figure 12. Switching on Titan project nature

- B. In Titan Project Explorer panel, right click and select the “Import” option, then in the dialog window, expand “General” option and select “Existing Projects into Workspace” as shown on Figure 13. Browse the unzipped oneM2MTester package and import the project into the workspace shown a Figure 14.



**Figure 13. Import from Existing Projects into Workspace**



**Figure 14. Select the unzipped oneM2MTester package and import into the workspace**

- C. In the Eclipse File menu select Import and in the Import window select TITAN/Project form (.tpd file) as shown on Figure 15. Select the file oneM2M\_Test.tpd from the directory the project has been stored previously and click Finish (see Figure 16). For more details, see section 4.9.5 of (3).

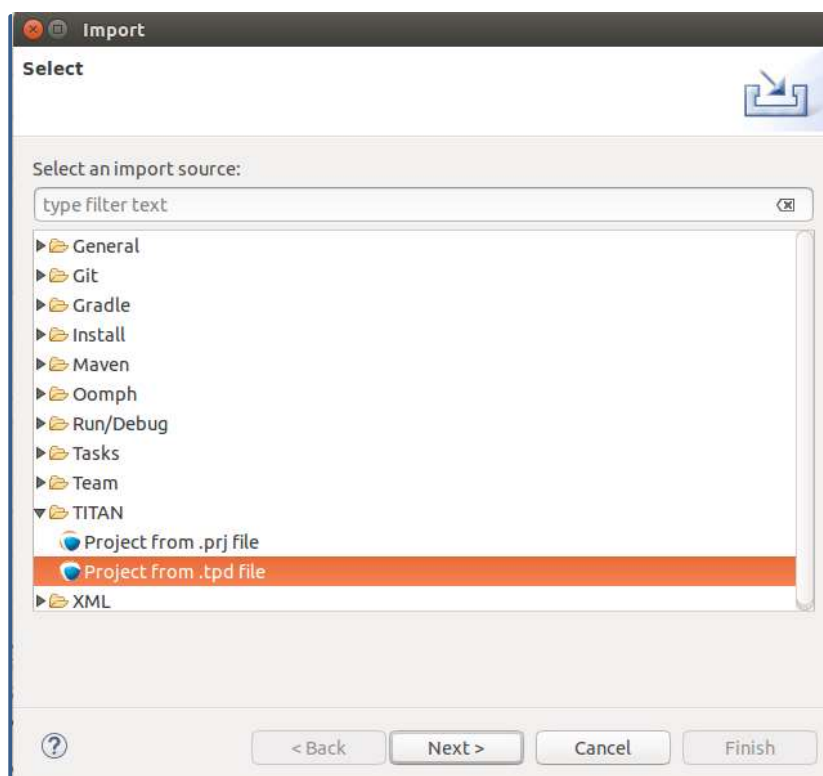


Figure 15. Importing project with Titan Project Descriptor file

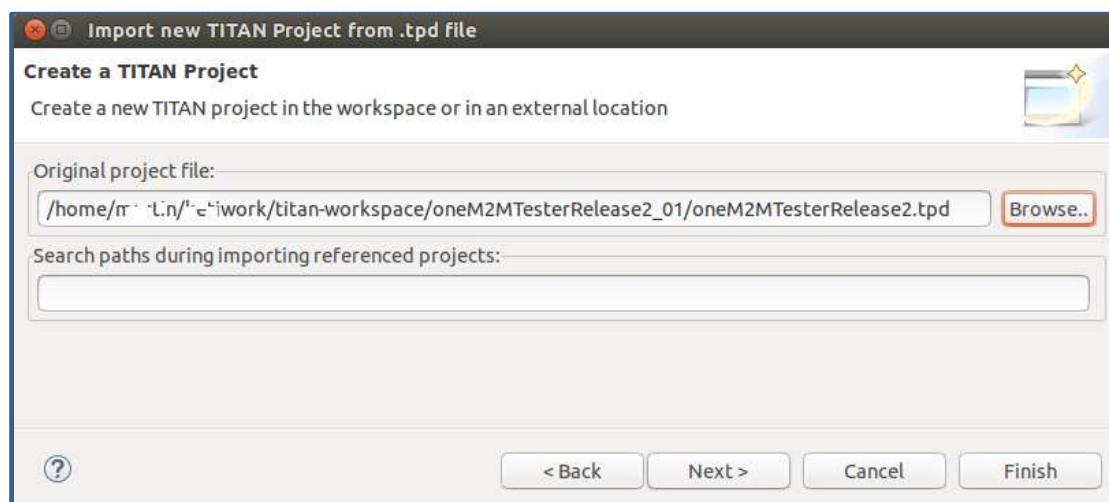


Figure 16. Selecting oneM2M Tester project descriptor file

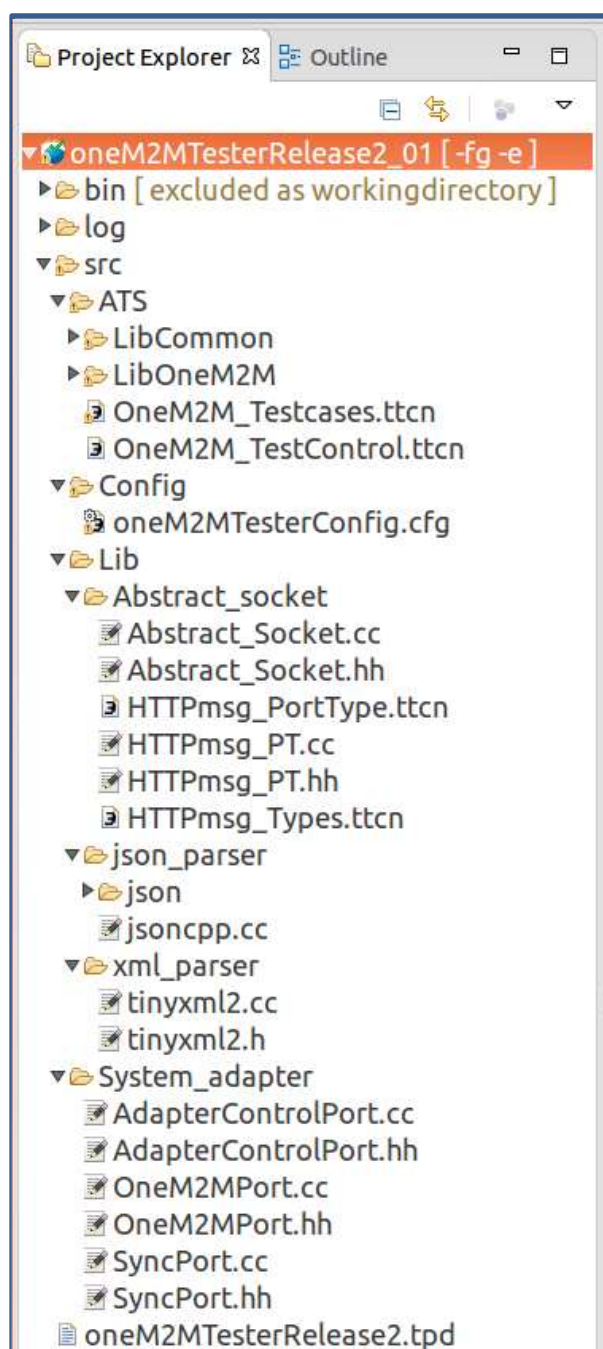


Figure 17. oneM2MTester project structure

### 3.2.2. Run Testcases against Implementation Under Tests

After importing the oneM2MTester project, expand src folder, you will see same as structure as shown on Figure 17.

- The “Config” folder includes TITAN configuration files with module parameters for runtime configurations, logging options and test case execution configurations etc,
- The “ATS” folder contains TTCN-3 code designed for oneM2M conformance test,
- The “Lib” folder includes external library that are using e.g. abstract socket, jsoncpp and tinyxml etc, and
- “System\_adapter” folder includes implemented ports that are defined in TTCN-3 modules, e.g. oneM2MPort, AdapterControlPort, SyncPort etc. Note that the the system adapter and codec for oneM2M binding protocols are implemented in those ports.

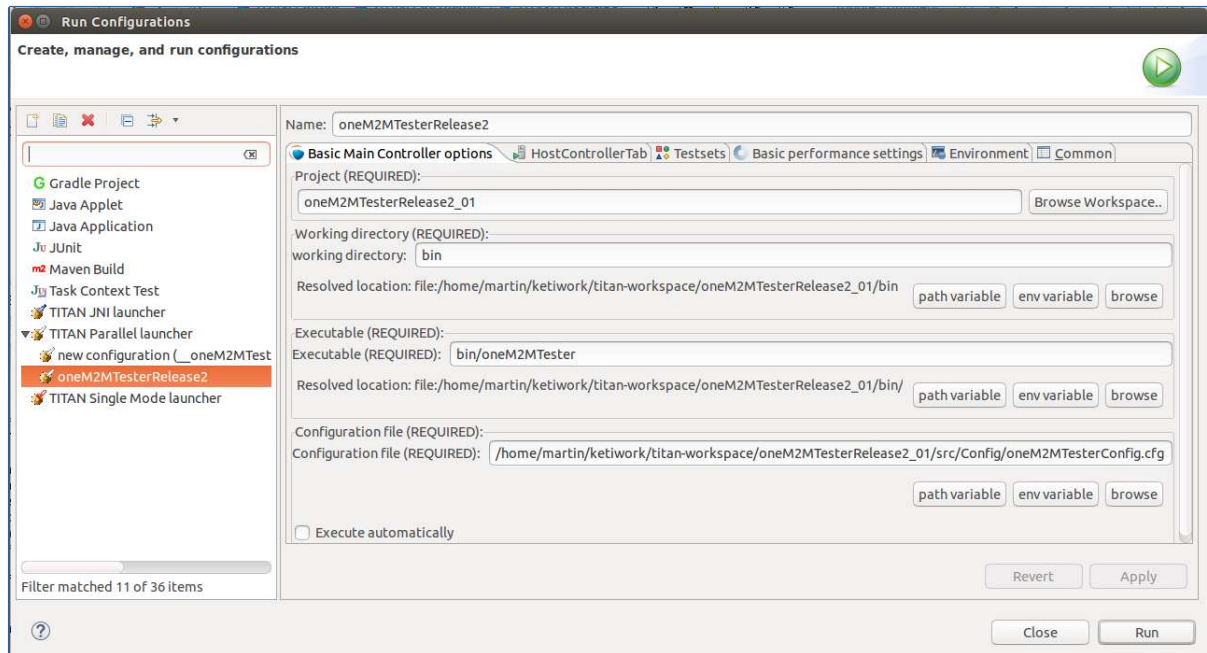
There are several configurations users have to modify in order to run testcases against their Implementation Under Test (IUT) as following:

- 1) Customize Titan configuration file: src > Config > OneM2MTesterConfig.cfg by modifying module parameters as following:

- Set `PX_SUT_ADDRESS` to the IP address and port of the IUT e.g. “222.90.69.7:65057”,
- Set `PX_CSE_NAME` to the CSE name of the IUT e.g. “csebase-name”,
- Set `PX_URI_CSE` to the CSE\_ID of the IUT e.g. “~/ID-CSE-01” in case of spRelative address format is used,
- Set `PX_APP_ID` to the application ID used by IUT,
- Set `PX_AE_ID_STEM` to the AE-ID-Stem to be used by the IUT for AE registration case e.g. “Cadmin-ae” in case I have received a AE-ID-Stem assigned by the hosting CSE before otherwise you can set as a “C” or “S” for an initial AE registration case,
- Set `PX_SERIALIZATION` to a serialization value i.e. either “XML”, “JSON”, “xml” or “json”: when the IUT wants to test against with XML serialization, set this parameter to either uppercase “XML” or lowercase “xml” while for JSON serialization, set it to either uppercase “JSON” or lowercase “json”.
- Set `PROTOCOL_BINDING` to a binding protocol to be tested, i.e. either “HTTP”, “CoAP” or “MQTT”. Note that currently the oneM2MTester only accepts HTTP protocol binding. Other protocol bindings including CoAP and MQTT will be supported soon.
- Set `PX_ADDRESSING_FORMAT` to either value `e_cseRelative`, `e_spRelative` or `e_absolute` indicating the resource addressing format of CSE relative, SP relative or absolute one, respectively.
- Set `PX_UNSTRUCTURED` to Boolean `false` or `true` to indicate user preference on the *unstructured* or *structured* oneM2M resource format, respectively.

- 2) Switch to the TITAN Executing prespective on the Eclipse toolbar.
- 3) Create **Run Configurations** at your first time using oneM2MTester: User can easily create a one run configuration by right clicking the OneM2MTester project and select “Run As” option. Expand the right arrow and select “Run Configurations” and a dialog window will show as Figure 18.
  - Basic Main Controller options
  - Host Controller Tab: By clicking the “Init” button, this tab will be initialized to the “localhost”; if other host is used to run the tests than the host running the Eclipse IDE, these settings can be changed easily.

- Testsets
- Basic Performance settings
- Environment (check PATH and TTCN\_DIR variables)
- Common



**Figure 18. TITAN Runtime Configurations**

The Run configuration is loaded by Eclipse Titan when the executable test suite (ETS) is executed. After creation of run configurations, users can simply select any one of created run configurations by click the **Run button** in the button bar of **Eclipse TITAN Editing** user interface as highlighted in Figure 19 and then run the selected run configurations.



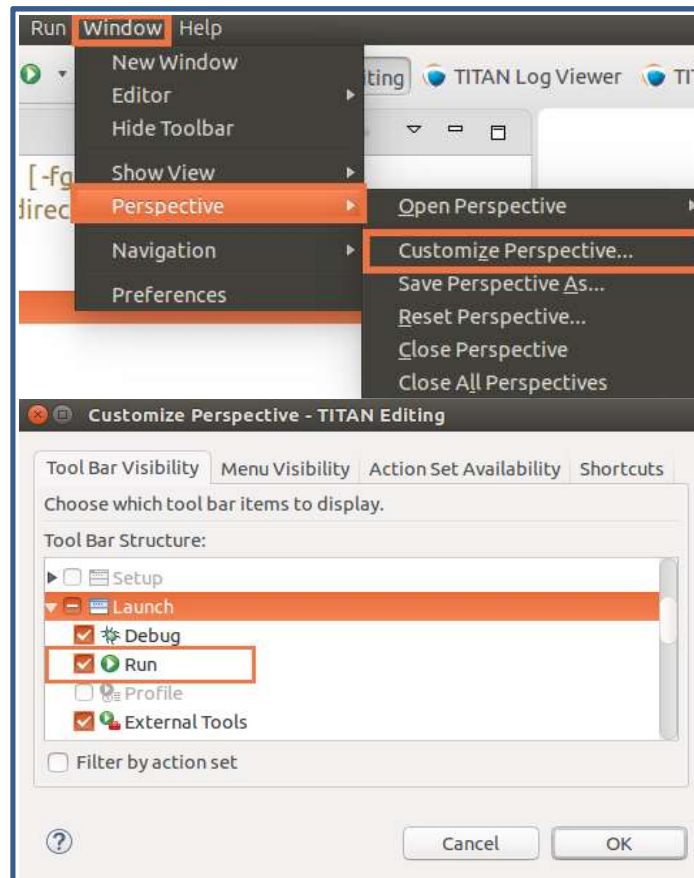
**Figure 19. Run configuration button in Eclipse TITAN button bar**

For details on how to create a launch configuration, please go to section of 4.2 of (4).

In case that if the Run configuration button doesn't show in your Eclipse TITAN Editing user interface, change the Perspective setting as following:



- Open the **Customize Perspective** dialog from the **Window** menu at the top of Eclipse TITAN user interface and enable the option of Run button as Figure 20 shows.



**Figure 20. Customize Perspective menu**

When the Titan executor is successfully launched, the TITAN Editing user interface will be switched to TITAN Executing user interface. Users may also launch the TITAN executor by right clicking the OneM2MTester project in Project Explorer panel and select “TITAN Parallel launcher” in “Run As” option. We prefer to run the TITAN Parallel launcher for execution of test suites in distributed and parallel manner.

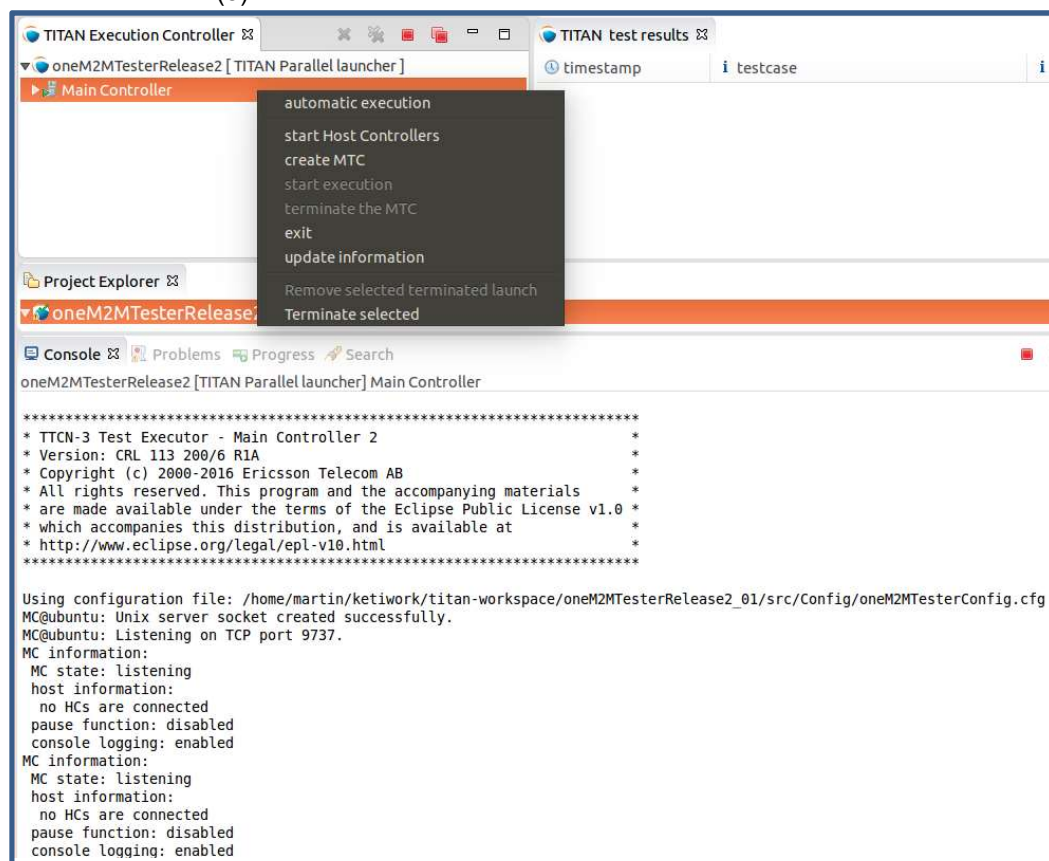
In the TITAN Executing user interface, TITAN Execution Controller panel displays the configuration user set for certain TITAN launch mode shown as Figure 21. Users have to follow steps as following to run parallel test suites:

- Start Main Controller (MC),
- Start Host Controller (HC),
- Create MTC (main test component),
- Start the control part or a selection of testcases of TTCN-3 module on MTC.

Alternatively, users may run Main Controller automatically by select “automatic execution” which will automatically create a MTC first and establish a control connection between MC and MTC, and then start Host Controllers (“localhost” in our case) to connect to the MC using configured parameters (“localhost” in our case) in Titan runtime configuration panel.

Where Main Controller is started manually by users and runs in one instance during entire test suite

execution and it arranges the creation and termination of MTC on user request and the execution of module control part on MTC, the Host Controller in an instance of the executable test program, i.e. the translated test suite linked with Test Ports and Base Library. Exactly one Host Controller should be run on each computer that participates in distributed TTCN-3 test execution and started by users manually on all participating computers. For more details about Parallel TTCN-3 execution architecture, please refer to section 4.4.1 of (5).



**Figure 21. TITAN Execution Controller**

The testcases imported from oneM2M ATS are currently designed only for testing and evaluation of oneM2M common service entities (CSEs) implementations, such as ASN-CSE, MN-CSE and IN-CSE etc, in terms of oneM2M Common Service Functions (CSFs) and oneM2M resource primitives.

Regarding the selection of testcases of TTCN module, testcases are categorized as following:

Note that a single testcase is a compound of sub-testcases written in time order:

- Test preamble: test pre-conditions before execution of a certain testcase, for instance, ensure that an AE resource has been created prior to test a container resource creation operation.
- Test behavior body: the actual test content written for testing one single certain oneM2M CSF (create, retrieve, update, or delete operation etc.).
- Test postamble: an opposite action of test-preamble, i.e. remove all pre-conditions set for the current testcase when a testcase finishes execution to ensure another testcase will not be impacted with current testcase execution condition.

In addition, Eclipse TITAN also provides a Command Line Interface (CLI) to run executable test suite. The details on how to use CLI to run executable test suite are present in Appendix-C.



### 3.2.3. Check Testcase Execution Result

After execution of a certain testcase, the execution result either PASS, FAIL or INCONC is displayed in *TITAN test results* panel in TITAN Executing user interface. The running log of a certain testcase execution is displayed in real-time manner during the testcase execution in Console panel shown as Figure 22.

Eclipse TITAN MTC saves all running logs into a .log file named OneM2MTestLog-MTC.log (whose name can be set by `LogFile:= "../log OneM2MTesterLog-%n.log"`) in the log folder directly under OneM2MTester project as shown on Figure 23. Double clicking the generated log file will open a new tab displaying a full testcase running logs which contains more detailed information than the information displayed in Console panel provided you have enabled the DEBUG mode for ConsoleMask option in Titan configuration file (OneM2MTesterConfig.cfg). The log file would be helpful for users to check the test result especially when the execution of a testcase is set to a verdict of FAIL.

Figure 23 also shows logs in category of testcase, e.g. TC\_CSE\_DMR\_CRE\_BV\_001\_01. Right click the log for a certain testcase, you will see there are three view mode, "Open Statistical View", "Open MSC View", and "Open Text Table View", which represents extra log information with statistics table, graphical call flow, and test table, respectively. Those three log view modes are regarded as a complement to the OneM2MTesterLog-MTC log file.

TITAN provides a group of log options listed at Table 1 to deal with different requirements for the Console log and generation of the log files. Logging parameters can be configured in TITAN configuration file by setting ConsoleMask and FileMask options, where ConsoleMask defines event types that are output to the console while FileMask specifies event types that are written to log file (i.e. oneM2MTesterLog). When FileMask is omitted from the configuration file, its default value (LOG\_ALL) is applied while when ConsoleMask is omitted, default value (ERROR|WARNING|ACTION|TESTCASE|STATISTICS) is applied. For more details, please refer to section 7.2.4 and 7.2.5 of (6).

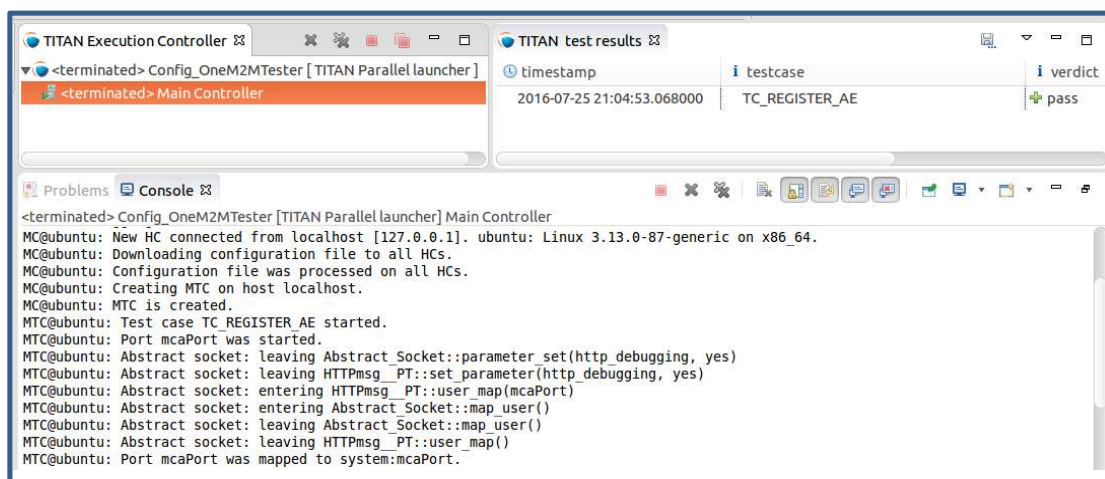


Figure 22. Log view for executable test suite execution

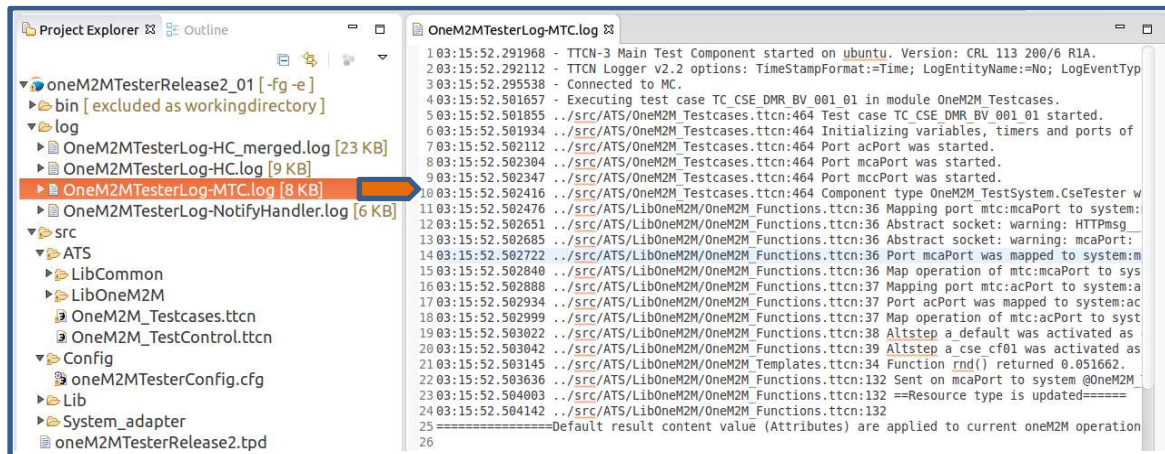


Figure 23. Generated log file for executable test suite execution

Table 1 Eclipse TITAN Log options

Logging option	Description
<b>DEBUG</b>	Logging debug messages in Test Ports and external functions. If omitted, only operation messages related to MTC is logged in Console. We recommend users to enable this option in case that the execution of a testcase is set to verdict of FAIL.
<b>PORTEVENT</b>	Logging port events such as send, receive, for instance, MTC@ubuntu: Sent on mcaPort to system @OneM2M_TestSystem.M2MRequestPrimitive: {requestPrimitive := {...}}
<b>TESTCASE</b>	Logging the start, the end operation and the final verdict of certain execution of testcase, for instance, MTC@ubuntu: Test case TC_REGISTER_AE started. MTC@ubuntu: Test case TC_REGISTER_AE finished. Verdict: fail reason: Error while registering application
<b>STATISTICS</b>	Logging statistics of verdicts at the end of execution, for instance, MTC@ubuntu: Verdict statistics: 0 none (0.00 %), 1 pass (100.00 %), 0 inconc (0.00 %), 0 fail (0.00 %), 0 error (0.00 %).
<b>MATCHING</b>	Logging analysis message of template matching failures in port receive operations, for instance, MTC@ubuntu: Matching on port mcaPort int4004 (4004) with (int2000 (2000), int2001 (2001), int2002 (2002), int2004 (2004)) unmatched: First message in the queue does not match the template: MTC@ubuntu: Matching on port mcaPort succeeded: matched
<b>PARALLEL</b>	Logging parallel test execution and test configuration related operations (Create, done, connect, map etc), for instance, HC@ubuntu: MTC finished. Process statistics: {process id: 7426, terminated normally, exit status: 0, user time: 0.000000 s, system time: 0.007691 s, maximum resident set size: 3428.....}
<b>VERDICTOP</b>	Logging verdict operations (setVerdict, getVerdict), for instance, MTC@ubuntu: Setting final verdict of the test case. MTC@ubuntu: Local verdict of MTC: pass reason: "Application registered successfully"
<b>VERDICTOP_SETVERDICT</b>	Logging as soon as the TTCN-3 setVerdict operation is executed, for instance, MTC@ubuntu: setverdict(pass): none -> pass reason: "Application registered successfully", new component reason: "Application registered successfully"

Note: For more logging options, please refer to Table 14~Table 25 in (6).

## 4. External References

### 4.1. Eclipse TITAN Documentation

The Eclipse TITAN documentation in oneM2MTester Gitlab project is available on the following link:  
[http://support.iotocean.org:7591/onem2m/conformancetesting/oneM2MTester/tree/master/Documents/Eclipse\\_TITAN\\_Documentation](http://support.iotocean.org:7591/onem2m/conformancetesting/oneM2MTester/tree/master/Documents/Eclipse_TITAN_Documentation).

### References

1. **Ericsson.** *Installation Guide for the TITAN TTCN-3 Test Executor.* 2016.
2. —. *Installation Guide for the TITAN Designer and TITAN Executor for the Eclipse IDE.* 2016.
3. —. *User Guide for the TITAN Designer for the Eclipse IDE.* 2016.
4. —. *User Guide of the TITAN Executor for the Eclipse IDE plugin.* 2016.
5. —. *Ericsson. User Guide for TITAN TTCN-3 Test Executor.* 2016 : s.n.
6. —. *Programmer's Technical Reference Guide for the TITAN TTCN-3 Toolset.* 2016.

## 5. Appendix

### Appendix-C. Run Executable Test Suite using TITAN CLI

Users could run executable test suite through the TITAN Execution interface as we introduced in section 3.2.2 and if users are familiar with Linux CLI, they can also run executable test suite using TITAN CLI. Note that we assume user has created a run configuration .cfg file. Here are the procedures to use TITAN CLI to run the executable test suite:

1. Start a main controller with run configurations:
  - Open a terminal (T-1) and go to the configuration file directory in your Eclipse TITAN workspace. Here we assume we have a configuration file `oneM2MTTesterConfig.cfg` located in directory  
`/titan-workspace/oneM2MTTesterRelease2_01/src/Config/oneM2MTTesterConfig.cfg`.
  - Run the main controller via command `mctr_cli oneM2MTTesterConfig.cfg`

If the main controller is successfully started, you will see the log similar as Figure 24.

```
r | n@ubuntu:~/titan-workspace/IoP3Test/src/Config$ mctr_cli oneM2MTes
terConfig.cfg

*****
* TTCN-3 Test Executor - Main Controller 2                               *
* Version: CRL 113 200/6 R1A                                             *
* Copyright (c) 2000-2016 Ericsson Telecom AB                           *
* All rights reserved. This program and the accompanying materials      *
* are made available under the terms of the Eclipse Public License v1.0 *
* which accompanies this distribution, and is available at              *
* http://www.eclipse.org/legal/epl-v10.html                             *
*****

Using configuration file: oneM2MTTesterConfig.cfg
MC@ubuntu: Unix server socket created successfully.
MC@ubuntu: Listening on TCP port 9635.
MC2> |
```

Figure 24. Capture of Running Main Controller

## oneM2MTester User Guide

2. Create a Main Test Component (MTC)
  - Run command `cmtc` in terminal T-1 to create a MTC as shown as Figure 25. Then the system starts listening on port 9635 to wait for connections from Host Controllers (HC).

```
*****
* TTCN-3 Test Executor - Main Controller 2                               *
* Version: CRL 113 200/6 R1A                                           *
* Copyright (c) 2000-2016 Ericsson Telecom AB                         *
* All rights reserved. This program and the accompanying materials     *
* are made available under the terms of the Eclipse Public License v1.0 *
* which accompanies this distribution, and is available at             *
* http://www.eclipse.org/legal/epl-v10.html                           *
*****

Using configuration file: oneM2MTesterConfig.cfg
MC@ubuntu: Unix server socket created successfully.
MC@ubuntu: Listening on TCP port 9635.
MC2> cmtc
Waiting for HC to connect...
```

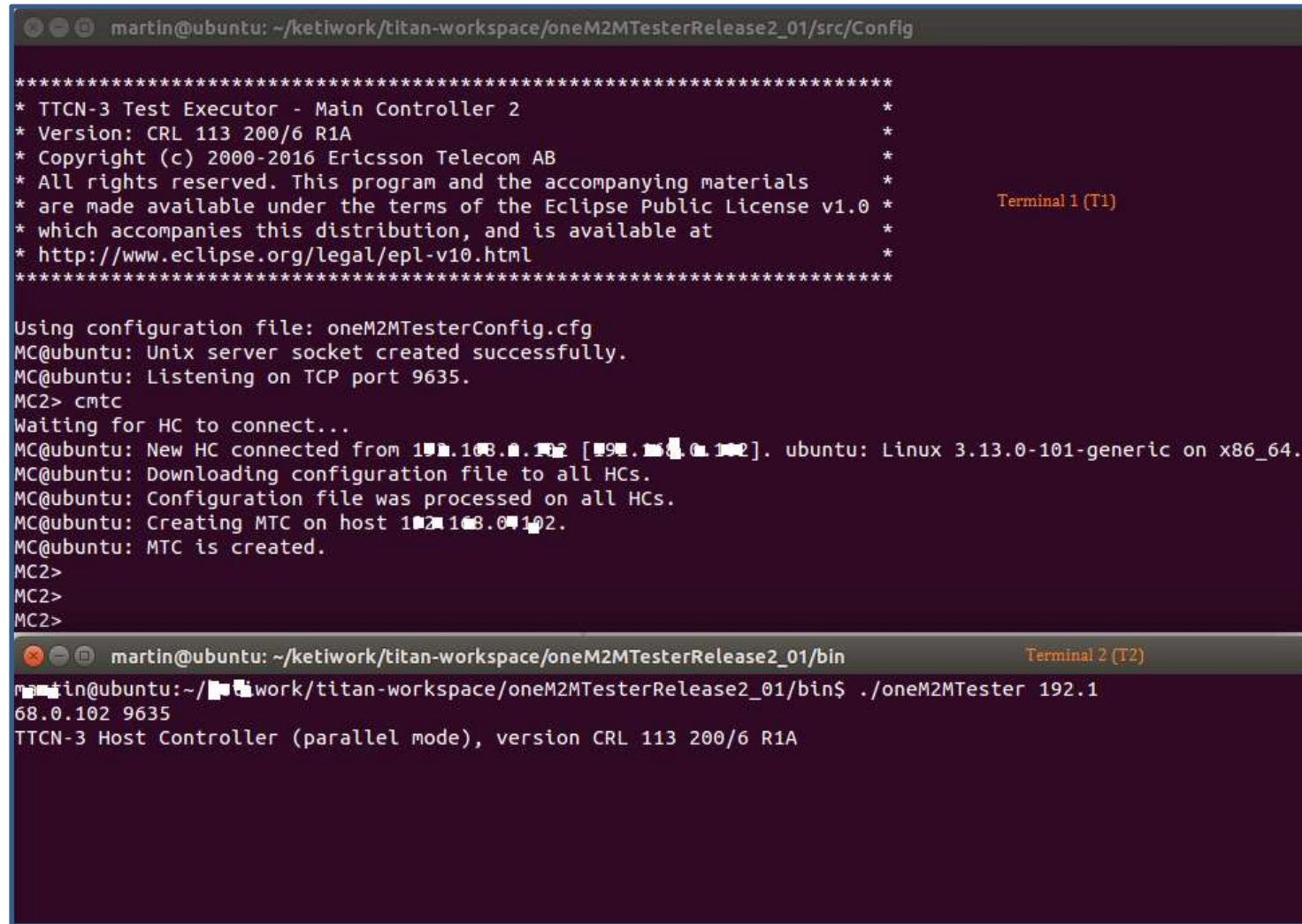
Figure 25. Create a MTC and enter listening state

3. Start a Host Controller (HC)
  - Open another terminal (T-2) and go to the directory of the executable test suite. Here we assume we have generated an executable test suite `oneM2MTester` which is located in `~/titan-workspace/oneM2MTesterRelease2_01/bin`.



## oneM2MTester User Guide

- Start a HC by running command `./oneM2MTester 127.0.0.1 9035` Note that the port `9035` is the port that MTC is open for listening. If the HC is started successfully, messages shown as Figure 26 will be received, including the HC is connected and configuration file was processed on all HCs as well as the MTC is created.



The image shows two terminal windows. The top window, titled 'Terminal 1 (T1)', shows the MTC (Main Controller 2) running. It displays copyright information for Ericsson Telecom AB and the Eclipse Public License v1.0. It then shows the MTC using the configuration file 'oneM2MTesterConfig.cfg', creating a Unix server socket, listening on TCP port 9635, and waiting for an HC to connect. A new HC is connected from 192.168.0.102, and the configuration file is processed on all HCs. The MTC is then created. The bottom window, titled 'Terminal 2 (T2)', shows the HC (Host Controller) running. It displays the command `./oneM2MTester 192.168.0.102 9635` and the output 'TTCN-3 Host Controller (parallel mode), version CRL 113 200/6 R1A'.

```
martin@ubuntu: ~/ketiwork/titan-workspace/oneM2MTesterRelease2_01/src/Config

*****
* TTCN-3 Test Executor - Main Controller 2                               *
* Version: CRL 113 200/6 R1A                                             *
* Copyright (c) 2000-2016 Ericsson Telecom AB                           *
* All rights reserved. This program and the accompanying materials      *
* are made available under the terms of the Eclipse Public License v1.0 *
* which accompanies this distribution, and is available at              *
* http://www.eclipse.org/legal/epl-v10.html                             *
*****

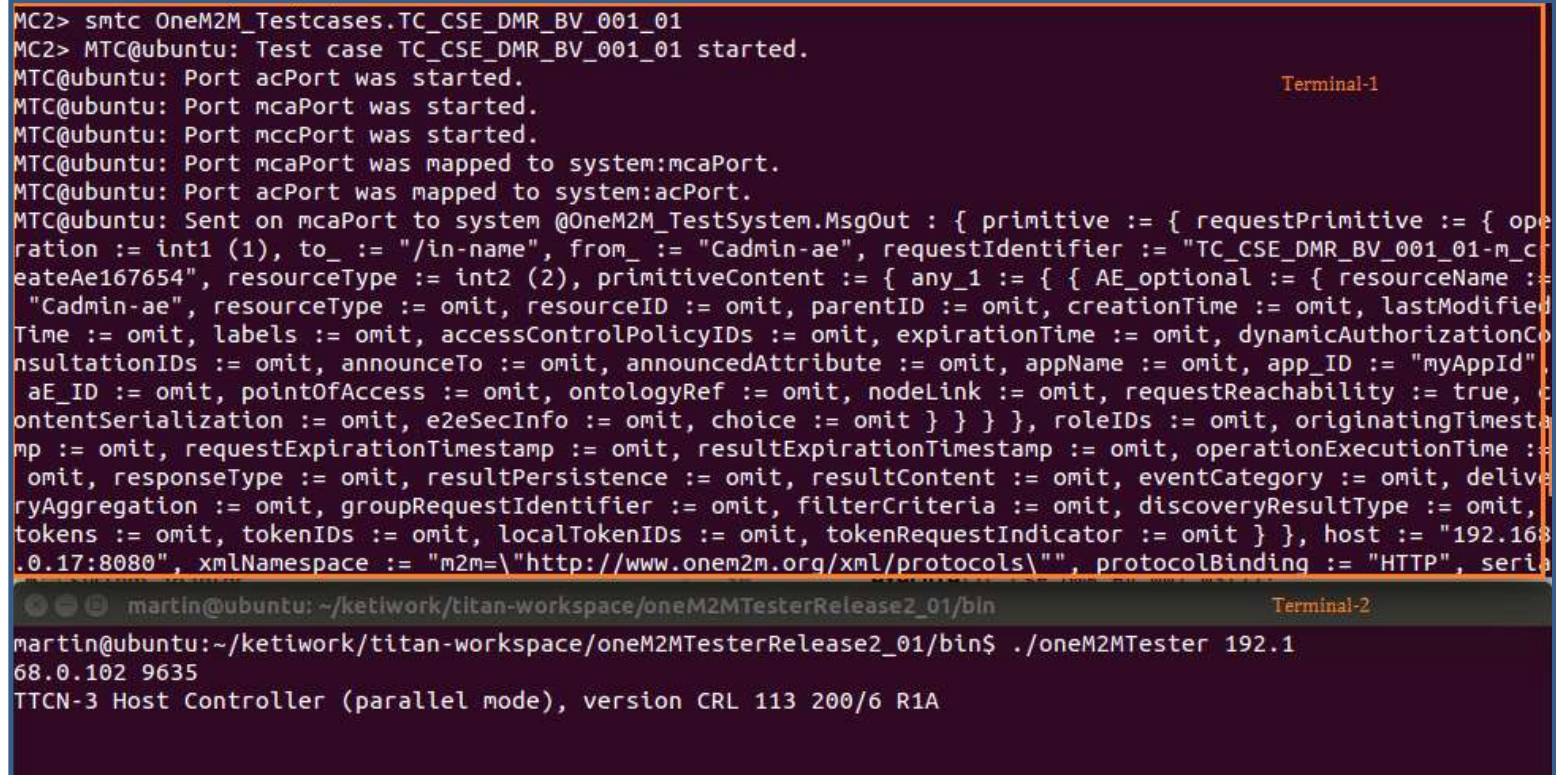
Using configuration file: oneM2MTesterConfig.cfg
MC@ubuntu: Unix server socket created successfully.
MC@ubuntu: Listening on TCP port 9635.
MC2> cmtc
Waiting for HC to connect...
MC@ubuntu: New HC connected from 192.168.0.102 [192.168.0.102]. ubuntu: Linux 3.13.0-101-generic on x86_64.
MC@ubuntu: Downloading configuration file to all HCs.
MC@ubuntu: Configuration file was processed on all HCs.
MC@ubuntu: Creating MTC on host 192.168.0.102.
MC@ubuntu: MTC is created.
MC2>
MC2>
MC2>

martin@ubuntu: ~/ketiwork/titan-workspace/oneM2MTesterRelease2_01/bin
martin@ubuntu: ~/ketiwork/titan-workspace/oneM2MTesterRelease2_01/bin$ ./oneM2MTester 192.168.0.102 9635
TTCN-3 Host Controller (parallel mode), version CRL 113 200/6 R1A
```

Figure 26. Capture of established connection between MTC and HC

## oneM2MTester User Guide

4. Execute the executable test suite
  - After the MTC is created and connected with HCs, we can run executable test suites now by using command `smtc` `[MODULE_NAME.TESTCASE_NAME]` e.g. `smtc OneM2M_Testcases.TC_CSE_DMR_BV_001_01` then you will see the live running log in the log control panel as shown as Figure 27.



```
MC2> smtc OneM2M_Testcases.TC_CSE_DMR_BV_001_01
MC2> MTC@ubuntu: Test case TC_CSE_DMR_BV_001_01 started.
MTC@ubuntu: Port acPort was started.
MTC@ubuntu: Port mcaPort was started.
MTC@ubuntu: Port mccPort was started.
MTC@ubuntu: Port mcaPort was mapped to system:mcaPort.
MTC@ubuntu: Port acPort was mapped to system:acPort.
MTC@ubuntu: Sent on mcaPort to system @OneM2M_TestSystem.MsgOut : { primitive := { requestPrimitive := { operation := int1 (1), to_ := "/in-name", from_ := "Cadmin-ae", requestIdentifier := "TC_CSE_DMR_BV_001_01-m_createAe167654", resourceType := int2 (2), primitiveContent := { any_1 := { { AE_optional := { resourceName := "Cadmin-ae", resourceType := omit, resourceID := omit, parentID := omit, creationTime := omit, lastModifiedTime := omit, labels := omit, accessControlPolicyIDs := omit, expirationTime := omit, dynamicAuthorizationConsultationIDs := omit, announceTo := omit, announcedAttribute := omit, appName := omit, app_ID := "myAppId", aE_ID := omit, pointOfAccess := omit, ontologyRef := omit, nodeLink := omit, requestReachability := true, contentSerialization := omit, e2eSecInfo := omit, choice := omit } } }, roleIDs := omit, originatingTimestamp := omit, requestExpirationTimestamp := omit, resultExpirationTimestamp := omit, operationExecutionTime := omit, responseType := omit, resultPersistence := omit, resultContent := omit, eventCategory := omit, deliveryAggregation := omit, groupRequestIdentifier := omit, filterCriteria := omit, discoveryResultType := omit, tokens := omit, tokenIDs := omit, localTokenIDs := omit, tokenRequestIndicator := omit } }, host := "192.168.0.17:8080", xmlnsNamespace := "m2m=\"http://www.onem2m.org/xml/protocols\"", protocolBinding := "HTTP", serial...
```

```
martin@ubuntu: ~/ketiwork/titan-workspace/oneM2MTesterRelease2_01/bin
martin@ubuntu:~/ketiwork/titan-workspace/oneM2MTesterRelease2_01/bin$ ./oneM2MTester 192.168.0.102 9635
TTCN-3 Host Controller (parallel mode), version CRL 113 200/6 R1A
```

Figure 27. Running log of test executable test suite

# Thank you.

## Contact:

**NakMyoung Sung** (diesnm201@gmail.com)  
**JaeYoung Hwang** (forest62590@gmail.com)  
**InSong Lee** (insong3535@gmail.com)

**oneM2MTester developing team**

**2018 September**