# Contents

1 Quickstart Guide 3  
1.1 Preparations .................................................. 3  
1.2 Dashboard ....................................................... 3  
1.3 CLI .................................................................. 4  
1.4 Manual Installation ............................................. 5  

2 Protocol Conformance Testing 17  
2.1 General ........................................................... 17  
2.2 Test Suite Structure ............................................. 18  
2.3 Test Configurations ............................................. 18  
2.4 Test Purpose Catalogues ...................................... 18  
2.5 IoT-Testware Test Suites ..................................... 19  

3 ETSI and Eclipse Foundation 29  

4 Dashboard 31  
4.1 Introduction ...................................................... 31  
4.2 Overview .......................................................... 31  
4.3 Backend ............................................................ 32  
4.4 Frontend ............................................................ 32  
4.5 Integration .......................................................... 32  

5 Fuzzing 35  

6 About Eclipse IoT-Testware 37  
6.1 IoT-Testware Team ............................................. 37  
6.2 Objective ........................................................... 37  
6.3 Conformance Test Methodology and Framework ......... 37  
6.4 Implementation ................................................. 38  

7 Glossary 41  

8 References 43  

9 License 45  

10 MQTT Specification 49
11 CoAP RFC 7252 51
12 Indices and tables 53
Bibliography 55
Great that you considering the Eclipse IoT-Testware to test your implementations. We would like to guide you to start with the test suites.
The easiest way to get started is to use Docker. The provided Dockerfile hides the complexity of TTCN-3 from the user. Therefore the user has more time to test his systems.

1.1 Preparations

- Make sure you have a working Docker installation
- (optional) Set the following environment variables

```
TESTWARE=iot_testware
```

1.2 Dashboard

The easiest way to get started with the Eclipse IoT-Testware is to use the Dashboard Docker image.

1. Clone the IoT-Testware Dashboard project

```
git clone https://github.com/eclipse/iottestware.dashboard
cd iottestware.dashboard
```

2. Build the Docker image

```
docker build -t $TESTWARE .
```

3. Start the Docker container

```
docker run --network=host -ti $TESTWARE
```

4. Call the Dashboard in your browser from https://localhost:3001
1.3 CLI

If you are already familiar with concepts and the CLI of Eclipse Titan than this way is a light-weight solution to run the Testware without webserver and dashboard.

1. Clone the main IoT-Testware project

```bash
git clone https://github.com/eclipse/iottestware
cd iottestware
```

2. Build the Docker image

```bash
docker build -t TESTWARE .
```

3. Start the Docker container

```bash
docker run -ti TESTWARE /bin/bash
```

1.3.1 Run Test Campaigns

The IoT-Testware Docker image ships currently two test suites for MQTT and CoAP. We will show you quickly how to configure and run the test suites.

Starting test suites with TITAN

```
ttcn3_start [-ip host_ip_address] executable [file.cfg] [module_name[.testcase_name]]
```

Looks quite easy: in order to start a test campaign TITAN requires us to provide an executable test suite. As we want also be able to provide different kinds of configurations, we also need to provide a .cfg file. Fortunately, we already have all the components in Docker. Let’s see how we can run some conformance tests against your System Under Test (SUT).

1. Change directory to the MQTT playground and make yourself familiar with the provided files.

```bash
cd /home/titan/playground/mqtt; ls
```

2. (optional) By default, the public MQTT Broker iot.eclipse.org is set. If you want to change the configuration follow the instructions.

```bash
vi ./BasicConfig.cfg
```

3. (optional) To configure the TS for your SUT you can change the `hostName`, `portNumber` and `credentials` definitions.

```bash
tsp_addresses :=
{
  {
    id := "mqtt_server",
    hostName := "iot.eclipse.org",
    portNumber := 1883
  },
  {
    id := "mqtt_client",
    hostName := "0.0.0.0",
    portNumber := 45679,
    credentials :=
```

(continues on next page)
4.1 Run the whole test campaign given in BasicConfig.cfg

```
ttcn3_start iottestware.mqtt BasicConfig.cfg
```

4.2 Run a single test case TC_MQTT_BROKER_CONNECT_001 from MQTT_TestCases

```
ttcn3_start iottestware.mqtt BasicConfig.cfg MQTT_TestCases.TC_MQTT_BROKER_CONNECT_001
```

## 1.4 Manual Installation

The IoT-Testware is composed of several test suites from different repositories with once again several dependencies to the Eclipse TITAN runtime. Hence, the installation process can become quite complex. Therefore we provide several ‘flavours’ of installation.

### 1.4.1 install.py

This documentation helps you to understand and use the **install.py** script.

### Contents

- **install.py**
  - Prerequisites
  - help command
  - protocol command
  - build command
  - path command
  - executable command
  - verbose command

### Prerequisites

- Latest version of
- Latest version of . For installation details please consult the official
help command

You can refer to the help command if you don’t know how to continue at any point. Simply run one the following command:

```
python install.py -h
python install.py --help
```

The result gives you a first idea, what is possible with the script. The output looks like this:

- `h` show this help message and exit
- `p PROTO` available protocol test suites are `{mqtt, coap, opcua}` sets a protocol that will be cloned together with its dependencies
- `b` build the project and create a Makefile
- `--path PATH` specify optionally your root directory, where all dependencies will be stored
- `e NAME` set the name of the executable that will be generated
- `v` progress status output is verbose

protocol command

The protocol command is the a mandatory flag. It will scan your working directory to check whether all dependencies are met. In case there is something missing, the script will download the missing dependencies automatically. The command demands a parameter representing the interesting protocol.

Let’s assume you would like to run tests against a CoAP implementation. Run one of the following command to get the CoAP conformance test suite and all it’s dependencies:

```
python install.py -p coap
python install.py --protocol coap
```

Use the same procedure for any available protocol.

build command

This optional command can be used to build the IoT-Tesware. It builds a Makefile first and creates an executable afterwards. You can only build one protocol at a time. It is determined by the protocol command. To build the CoAP test suite for example, run one of the following commands:

```
python install.py -p coap -b
python install.py -p coap --build
```

path command

When you run the install script, it creates a folder structure under `~/Titan` by default. This is your base directory where the IoT-Testware and all its dependencies are stored:

**IoT-Testware** You find the test suites for the protocols you have chosen via the protocol command. It creates a folder for every protocol separately in the form of `iottestware.<PROTOCOL>`

**Libraries** Collection of libraries needed for the specific IoT-Testware protocol.

**ProtocolModules** The protocol modules, provided in , are included inside this directory. The subset of protocol modules are protocol dependent. They define the protocol types.
**TestPorts** To bridge the gap between the test suite and the system under test (SUT), test ports are needed. They are provided by Eclipse Titan project.

**executable command**

With this command it is possible to name the executable that is generated when calling the *build command*. In contrast, the install script chose a default name for the executable following the scheme:

```
iotestware.<PROTOCOL>
```

To set your own name for the resulting executable, let’s say “myExecutable” simply run one of this command:

```
python install.py -p coap -e myExecutable
python install.py -p coap --executable_name myExecutable
```

**verbose command**

If you set this command, the console output will be verbose and give you more information during the process. By default, the output is quite, meaning only important messages are shown. To switch the verbose output on, you add either “-v” or “--verbose” to your command like in the following examples:

```
python install.py -p coap --verbose -e myExecutable
python install.py -p coap -b -v
python install.py --verbose --protocol coap
```

### 1.4.2 Install with Eclipse IDE

These instructions will get you a clean IoT-Testware clone up and running on your local machine for development and testing purposes.

**Prerequisites**

- Latest version of **Java**
- Latest version of **Python (v2 or v3)**
- Latest version of **Eclipse Titan**. For installation details please consult the official [Eclipse Titan](https://eclipse.org/titan/).
Installing Quickstart

Note: Make sure all prerequisites are met. As we use Eclipse Titan (natively running under Linux) to compile and execute our test suite, the following instruction won’t cover other OS like Windows or MacOS. We recommend installing a Linux derivate in a virtual machine.

Set up

Firstly, you need to get all needed dependencies to run the test suites. To do so, simply run the python script install.py with your protocol of choice:

python install.py -p <PROTOCOL>

This is the most minimalistic way of getting the dependencies. For a more complete explanation of the installation script please refer to the documentation. With Eclipse Titan you are free to choose to work either from the Command Line (CLI) or from Eclipse IDE. Go ahead and read further instructions of your preferred way of working.

Eclipse IDE

In every of our protocol repositories you find a iotTestware.<PROTOCOL>.tpd file that you need to import. Open the Titan IDE in your desired workspace and use the import feature File -> Import -> TITAN -> Project from .tpd file
Klick Next and choose the iottestware.<PROTOCOL>.tpd from ${PATH_BASE}
Klick Next and choose importation options.
Klick Finish and the IDE will import all the required Projects and open the properties for each. Make sure each project is configured to **generate Makefile for use with the function test runtime**.

Right-click the `iottestware.<PROTOCOL>` project and select **Build Project**.
Note: Make sure you are in the TITAN Editing Perspective, otherwise the Build Project might be not available.

1.4.3 Install with Docker

For an easy deployment, you can use the shipped Dockerfile coming with the repository.

Docker is a computer program that performs operating-system-level virtualization. It uses the resource isolation
features of the Linux kernel to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.

Docker will perform all the heavy lifting of virtualization and running the IoT-Testware including Eclipse Titan. Therefore, an installed and functioning Docker is the only prerequisite.

**Note:** Although, it is possible to run Docker containers on different operating systems, Docker’s host networking driver only works on Linux hosts. Hence, it is recommended to run the Docker containers on a Linux machine.

### Preparations

- Make sure you have a working Docker installation
- (optional) Set the following environment variables
  ```
  TW_CONTAINER_NAME=iot_testware
  TW_NETWORK_NAME=iottestware_net
  TW_SUBNET=172.18.0.0/16
  TW_FIXED_IP=172.18.0.4
  TW_VOLUME_NAME=testware_volume
  TW_VOLUME_PATH=/home/titan/iottestware.webserver/backend/resources/history
  ```
- (optional) Create separated Docker network
  ```
  docker network create --subnet $TW_SUBNET $TW_NETWORK_NAME
  docker network ls
  ```
- (optional) Create persistent storage and docker volumes
  ```
  docker volume create $TW_VOLUME_NAME
  ```
- Build the Docker container
docker build -t $TW_CONTAINER_NAME .

Additional Docker commands

- Stop all running container
  \[\text{docker stop } $(\text{docker ps -aq})\]

- Delete all containers
  \[\text{docker rm } $(\text{docker ps -aq})\]

- Delete all images
  \[\text{docker rmi } $(\text{docker images -q})\]

- Force delete a specific image
  \[\text{docker rmi -f } <\text{IMAGE_ID}>\]

- open second bashwindow
  \[\text{docker exec -it } <\text{CONTAINER_ID}> /bin/bash\]

Start Docker container

Docker offers many options for starting and integrating containers. In this section we will show how the container can be started with persistend storage and how to attach the container to the previously created sub-network. Read the Docker networking overview for more information.

1. Most basic way to start a Docker container without persistend storage and using the host's network interface
   \[\text{docker run --network host } $TW_CONTAINER_NAME\]

2. Isolated Docker container which is attached to the sub-network with a fixed IP and without persistend storage
   \[\text{docker run --network } $TW_NETWORK_NAME --ip } $TW_FIXED_IP $TW_CONTAINER_NAME\]

3. Using host’s network interface and with persistend storage
   \[\text{docker run --network host } -v } $TW_VOLUME:$TW_VOLUME_PATH $TW_CONTAINER_NAME\]

4. Isolated Docker container which is attached to the sub-network with a fixed IP and with persistend storage
   \[\text{docker run --network } $TW_NETWORK_NAME --ip } $TW_FIXED_IP -v } $TW_VOLUME:$TW_VOLUME_PATH $TW_CONTAINER_NAME\]

Run the Dashboard

Note: This step requires that you have used the Dashboard Docker file
Once everything is correctly deployed and started you can access the IoT-Testware Dashboard from your browser. Dependent on your network configuration simply open one of the following URLs in your browser:

- If you used the host network for the container: https://localhost:3001
- If you deployed the container with a custom network and given fixed IP: https://$TW_FIXED_IP:3001

Useful
2.1 General

The purpose of conformance testing is to determine to what extent a single implementation of a particular standard conforms to the individual requirements of that standard. Please find additional and more detailed information about conformance testing at ETSI’s Center for Testing & Interoperability.

The ISO (International Organization for Standardization) standard for the methodology of conformance testing (ISO/IEC 9646-1 and ISO/IEC 9646-2) as well as the ETSI (European Telecommunications Standards Institute) rules for conformance testing (ETSI ETS 300 406) are used as a basis for the test methodology.

To implement this methodology we require several intermediary artefacts. Those single artefacts break down the whole complexity of conformance testing into smaller pieces, each with a specific perspective on the problem.
2.2 Test Suite Structure

In the first step we define a TSS (Test Suite Structure) for a specific IUT (Implementation Under Test).

The TSS reflects the coverage of the reference specification by the TS (Test System): it is a synopsis of “which tests are performed on which aspects of the reference specification”. The conformance requirements and the ICS (Implementation Conformance Statement) proforma of the base specification are an essential source of cross-reference to check that the coverage of the test suite specified by the TSS&TP (Test Suite Structure & Test Purposes) is acceptable.

2.3 Test Configurations

TODO: Why do we need Test configurations?

2.4 Test Purpose Catalogues

A TP (Test Purpose) (Test Purpose) is a formal description of a test case. A formal description in the form of a TP offers a possibility of describing the purpose of a test without having the later technical implementation in mind. Following the TSS the tester is supported in systematically covering the complete IUT specification.

The listing below shows a simple MQTT (MQ Telemetry Transport) TP specified in TDL-TO (Test Description Language - Structured Test Objective Specification).

```plaintext
Test Purpose {
  TP Id TP_MQTT_Broker_CONNECT_001

  Test objective
  "The IUT MUST close the network connection if fixed header flags in CONNECT Control
  Packet are invalid"

  Reference
}
```

(continues on next page)
"[MQTT-2.2.2-1], [MQTT-2.2.2-2], [MQTT-3.1.4-1], [MQTT-3.2.2-6]"
PICS Selection PICS_BROKER_BASIC

Expected behaviour
ensure that {
    when {
        the IUT entity receives a CONNECT message containing
        header_flags indicating value '1111'B;
    }
    then {
        the IUT entity closes the TCP_CONNECTION
    }
}

The example below shows a simplified tabular representation for the TP.

<table>
<thead>
<tr>
<th>TP-ID</th>
<th>TP_MQTT_BROKER_CONNECT_01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>PICS_Broker</td>
</tr>
<tr>
<td>Summary</td>
<td>The IUT MUST close the network connection if...</td>
</tr>
<tr>
<td>Reference</td>
<td>[MQTT-2.2.2-1], [MQTT-2.2.2-2]</td>
</tr>
<tr>
<td>Expected behavior</td>
<td>initial condition statement</td>
</tr>
<tr>
<td></td>
<td>ensure that statement</td>
</tr>
</tbody>
</table>

2.5 IoT-Testware Test Suites

This step focuses on a technical implementation of the TPs. We use TTCN-3 and Eclipse Titan to implement each TP into a TC (Test Case) and orchestrate to executable test suites.

2.5.1 MQTT Test Suite
MQTT Protocol

A very brief summary of MQTT from the FAQ MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks.

Note: We provide an annotated version of the official MQTT specification which can be directly referenced (e.g. and )

Test Configurations

From a general and abstract perspective MQTT has two basic architectures for testing. This architecture directly reflects the choice of your SUT (System Under Test). We will call the first architecture Broker Testing. A MQTT Broker is the SUT as shown in the figure below:

The second major architecture we will call Client Testing as now, the Client is in focus as the SUT.
Now we can start to extract different configurations from the test architectures. The image below depicts the step of retrieving test configurations from the architecture:

The output of this exemplary step is a test configuration (CF01) where the Broker is the SUT and the TS takes the role of a MQTT Client.

The MQTT test suite uses four test configurations in order to cover the different test scenarios. In these configurations, the tester simulates one or several MQTT clients or brokers implementing the MQTT protocol.

<table>
<thead>
<tr>
<th>ID:</th>
<th>MQTT_Conf_01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The MQTT Broker is the IUT and the TS takes the role of a MQTT Client</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID:</th>
<th>MQTT_Conf_02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The MQTT Broker is the IUT and the TS takes the role of multiple MQTT Clients.</td>
</tr>
</tbody>
</table>
ID: MQTT_Conf_03
Description: The MQTT Client is the IUT and the TS takes the role of a MQTT Broker. For this configuration an optional UT (Upper Tester) might be required.

ID: MQTT_Conf_04
Description: As well the MQTT Broker as the MQTT Client, each is an IUT in this configuration. The part of the UT from the previous configuration is here replaced by a concrete application.

Test Purposes

TODO: link to .tplan2 from GitHub and .pdf from ETSI

1 Test Purpose {
2   TP Id TP_MQTT_Broker_CONNECT_001
3   Test objective
4     "The IUT MUST close the network connection if fixed header flags in CONNECT Control Packet are invalid"
5   Reference
6     "[MQTT-2.2.2-1], [MQTT-2.2.2-2], [MQTT-3.1.4-1], [MQTT-3.2.2-6]"
7   PICS Selection PICS_BROKER_BASIC
}(continues on next page)
Expected behaviour
ensure that {
    when {
        the IUT entity receives a CONNECT message containing
        header_flags indicating value '1111'B;
    }
    then {
        the IUT entity closes the TCP_CONNECTION
    }
}

Test System

TODO: describe Test System -> TTCN-3 code

Test Cases

You can find all MQTT Tests on GitHub.

We will examine the procedure of a single TC in order to get the understanding of the code structure.

The code block below shows the TTCN-3 implementation of the TC TC_MQTT_BROKER_CONNECT_001 for the cohesive TP TP_MQTT_BROKER_CONNECT_01

```
/*
 * @purpose The IUT MUST close the network connection if fixed header flags in CONNECT
 \ Control Packet are invalid
 *
 * @reference [MQTT-2.2.2-2], [MQTT-3.1.4-1], [MQTT-3.2.2-6]
 */
testcase TC_MQTT_BROKER_CONNECT_001() runs on MQTT_Client
{
    if(f_init("mqtt_client", "mqtt_server"))
    {
        f_TC_MQTT_BROKER_CONNECT_001();
    }
    f_cleanUp();
}
```

Let’s have a deeper look into the details of a TC. The first block comment contains only two TTCN-3 documentation tags but these give us a direct connection between a TC, a TP, and the MQTT specification.

```
/*
 * @purpose The IUT MUST close the network connection if fixed header flags in CONNECT
 \ Control Packet are invalid
 *
 * @reference [MQTT-2.2.2-2], [MQTT-3.1.4-1], [MQTT-3.2.2-6]
 */
```

The signature of a TTCN-3 TC contains many information which help us to reflect the Test Architecture.

```
testcase TC_MQTT_BROKER_CONNECT_001() runs on MQTT_Client
```
We have a distinct name for the TC which can be easy mapped to it’s cohesive TP. (see in the MQTT tpl catalogue for TP_MQTT_BROKER_CONNECT_001) The signature tells us also, that this TC will be executed on a Client (runs on MQTT_Client).

The body of the TC is used to initialize the test configuration and start the TC behaviour which is wrapped into a single function.

```plaintext
if(f_init("mqtt_client", "mqtt_server"))
{
    f_TC_MQTT_BROKERCONNECT_001();
}
f_cleanUp();
```

**Test Case Functions**

How does a TC function look like?

```plaintext
function f_TC_MQTT_BROKER_CONNECT_001() runs on MQTT_Client
{
    var UTF8EncodedString v_clientId := f_getClientId();

    var template MQTT_v3_1_1_Message v_conMsg := t_connect_flags(p_client_id := v_clientId, p_flags := '1111'B);
    f_send(valueof(v_conMsg));

    if(f_receiveNetworkClosedEvent())
    {
        setverdict(pass, "IUT closed the Network Connection correctly");
    }
    else
    {
        setverdict(fail, "IUT MUST close the Network Connection");
        f_disconnectMqtt();
    }
}
```

While syntactically this function might appear confusing, though the semantic behind is quite powerful and simple. Let’s go through the implemented TC and the according TP.

**Translation Port**

*TODO: Explain translation port*

**Generic Functions**

*TODO: describe MQTT_Functions module*

**Test Templates**

*TODO: describe templates*
Default Behaviours

TODO: describe the default behaviours

See also:

MQTT Interop Test Day in Burlingame, CA - March 17, 2014 The goal was to have as many different MQTT client and server implementations participate in interoperability testing to validate the implementation of the upcoming OASIS MQTT standard.

MQTT Interop Test Day in Ottawa, Canada – April 8, 2014 MQTT Test Day Demonstrates Successful Interoperability for the Internet of Things

MQTT Interop Test Day in Burlingame, CA - March 9, 2015 The goal was to have as many different MQTT client and server implementations participate in interoperability testing to validate the implementation of the OASIS MQTT 3.1.1 specification.

2.5.2 CoAP Test Suite

CoAP Protocol

CoAP (Constrained Application Protocol) is a specialized Internet Application Protocol for constrained devices, as defined in RFC 7252.

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine- to-machine (M2M) applications such as smart energy and building automation.

Note: We provide an annotated version of the official RFC which can be directly referenced (e.g. or )
Test Configurations

The concrete CoAP test configurations are listed below:

<table>
<thead>
<tr>
<th>ID:</th>
<th>CoAP_Conf_01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The CoAP Server is the IUT and the TS takes the role of a CoAP Client</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID:</th>
<th>CoAP_Conf_02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The CoAP Server is the IUT and the TS takes the role of multiple CoAP Clients.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID:</th>
<th>CoAP_Conf_03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The CoAP Client is the IUT and the TS takes the role of a CoAP Server. For this configuration an optional UT might be required.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID:</th>
<th>CoAP_Conf_04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>As well the CoAP Server as the CoAP Client, each is a IUT in this configuration. The part of the UT from the previous configuration is here replaced by a concrete application.</td>
</tr>
</tbody>
</table>
Test Purposes

_TODO: link to .tplan2 from GitHub and .pdf from ETSI_

```plaintext
Test Purpose {
  TP Id TP_CoAP_MessageFormat_Header_Version_001

  Test objective
  "The IUT is responding on a correctly set version number."

  Reference
  "RFC7252#section-4.2 (b)", "https://tools.ietf.org/html/rfc7252#section-4.2"

  PICS Selection PIC_Server

  Expected behaviour
  ensure that {
    when {
      the IUT entity receives a request message containing
      version indicating value 1,
      msg_type indicating value 0, //Confirmable
      token_length indicating value 0,
      code indicating value 0.00, //Empty Message
      msg_id corresponding to MSG_ID1;
    } then {
      the IUT entity sends a response message containing
      version indicating value 1,
      msg_type indicating value 3, //Reset
      token_length indicating value 0,
      code indicating value 0.00, //Empty Message
      msg_id corresponding to MSG_ID1;
      or the client entity times_out //from section 4.2 (b)
    }
  }
}
```

Test System

_TODO: describe Test System -> TTCN-3 code_

Test Cases

You can find all CoAP Tests on GitHub.

2.5. IoT-Testware Test Suites
case TC_COAP_SERVER_001() runs on MTC_CT
{
    map(self:p, system:p);
    f_TC_COAP_SERVER_001();
    unmap(self:p, system:p);
}

Test Case Functions

TODO: describe a CoAP Test Case Function

function f_TC_COAP_SERVER_001() runs on MTC_CT
{
    f_sendMessage(m_coapPingMessage);
    f_receiveMessage(m_coapEmptyMessage);
}

Translation Port

TODO: Explain translation port

Generic Functions

TODO: describe MQTT_Functions module

Test Templates

TODO: describe templates

Default Behaviours

TODO: describe the default behaviours

See also:

CoAP Plugtests 1: Guide  ETSI CTI Plugtests Guide (First Draft V0.0.16 2012-03) for achieving interoperability

CoAP Plugtests 1: Report  The 1st CoAP Plugtest was held from 24 to 25 March 2012 in Paris, France and was co-located with IETF#83. This event was jointly organized by ETSI, IPSO Alliance and the FP7 Probe-IT project1.

2.5.3 OPC-UA Test Suite
Test projects currently cover MQTT, CoAP, LoRaWAN and foundational security IoT-Profile of IEC 62443-4-2. The work of ETSI MTS-TST is correlated with the IoT-Testware which is hosted by the Eclipse Foundation. The technical contributions from the Eclipse members are coordinated by several dedicated Eclipse committers. The work includes Test purposes in TDL (primarily TDL-TO which is an extension of TDL for Structured Test Objective Specification) but also TTCN-3 test code developments that is important for test campaign execution in the test labs. In particular, ETSI members from MTS-TST control the test purposes developments and are responsible for the utilization of the resulting TP definitions for the ETSI working items and technical specifications. This approach allows to get input from active developers from the Eclipse community and a fast implementation of the target test suites for the interested industry but also support a faster development of ETSI specifications.

The illustrates an overview regarding two development procedures and its relationships: (a) the definition and implementation of the target system (under test) that needs to address subjects, assets and requirements as well as threats, policies and assumptions, and (b) the test development including test architecture, test purposes and test suite structure. System and test engineers need to derive the test implementation to be executed and analysed, e.g. for certification purposes.
CHAPTER 4

Dashboard

Contents

• Dashboard
  – Introduction
  – Overview
  – Backend
  – Frontend
  – Integration

4.1 Introduction

The Eclipse IoT-Testware Dashboard is a collection of several tools and test suites which makes it extremely flexible and powerful. Firstly the Backend which runs in background and provides the core functionality. Secondly, the Frontend, a thin ReactJS Application which provides a convenient user interface. And last but not least, the IoT-Testware with the test suites itself.

4.2 Overview

The first picture gives a brief overview about the basic idea.
4.3 Backend

The backend component is a node.js application which is intended to abstract the “low-level” handling of testing tools and their configurations.

4.4 Frontend

The frontend component is a JavaScript application which serves as a user interface to the backend component. Through the nature of the architecture the frontend is intended to be a thin client.

4.5 Integration

The sequence diagram below gives an overview about the interaction of involved components, starting from the user and ending with the SUT.
4.5. Integration
TODO: What is fuzzing and how do we make use of fuzzing?
CHAPTER 6

About Eclipse IoT-Testware

6.1 IoT-Testware Team

Who’s involved

6.2 Objective

As stated in [MVQ18] communication protocols for the IoT are currently in a immature state and offer different kinds of attack vectors. We believe... TODO!

6.3 Conformance Test Methodology and Framework

The IoT-Testware test suites will have a well-defined test suite structure (TSS) and a set of protocol implementation conformance statements (PICS) as well as protocol implementation extra information for testing (PIXIT). The work will follow the standardized approach as defined in ISO “Conformance Test Methodology and Framework” ISO 9646 and the best practices as described by ETSI White Paper No 3 “Achieving Technical Interoperability – the ETSI Approach”.
6.4 Implementation

The Eclipse IoT-Testware project provides standardized Abstract Test Suite (ATS) for popular IoT protocols. For the implementation of the ATS for CoAP and MQTT we apply ETSI Test Methodology which is well-proven in standardizing and testing of telecommunication systems.

Such an ATS contains of several parts which are required to implement the Conformance Test Methodology and Framework. But ATS, as the name says, are abstract, which means we need a system which executes the ATS. Just like Java code requires the JVM to be executed, an ATS requires in our case a TTCN-3 runtime. As our TTCN-3 runtime we have chosen Eclipse Titan which can compile and run our ATS.
The Executable Test Suite (ETS) is, as the name states, is a test suite under execution, just like running Java code.
Glossary

ASP  Abstract Service Primitive
ATS  Abstract Test Suite
CoAP  The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks.
ETS  Executable Test Suite
ETSI  The European Telecommunications Standards Institute is an independent, not-for-profit, standardization organization in the telecommunications industry (equipment makers and network operators) in Europe, headquartered in Sophia-Antipolis, France, with worldwide projection.
IoT  see Internet of Things
Internet of Things  The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.
IUT  Implementation Under Test
MQTT  The MQ Telemetry Transport is a machine-to-machine (M2M) Internet of Things connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.
PCO  Point of Control and Observation
PDU  Protocol Data Unit
SUT  System Under Test
Test Case  A test case is...
Test Purpose  A test purpose is...
TC  see Test Case
TDL  Test Description Language (TDL) is a new language for the specification of test descriptions and the presentation of test execution results.
**Thing**  The Thing in the context of IoT is an entity which is connected to the IoT and consumes or provides digital services.

**TP**  Test Purpose

**TS**  Test System

**TSS**  Test Suite Structure

**TTCN-3**  Testing and Test Control Notation version 3 is a standardized, modular testing language specifically designed for testing communication systems.

**UT**  Upper Tester
Publications about the IoT-Testware are [HKR17] and [KKR17]
Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

   i) changes to the Program, and
   ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement,
2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants
Recipient a non-exclusive, worldwide, royalty-free copyright license to
reproduce, prepare derivative works of, publicly display, publicly
perform, distribute and sublicense the Contribution of such Contributor,
if any, and such derivative works, in source code and object code form.
b) Subject to the terms of this Agreement, each Contributor hereby grants
Recipient a non-exclusive, worldwide, royalty-free patent license under
Licensed Patents to make, use, sell, offer to sell, import and otherwise
transfer the Contribution of such Contributor, if any, in source code and
object code form. This patent license shall apply to the combination of
the Contribution and the Program if, at the time the Contribution is
added by the Contributor, such addition of the Contribution causes such
combination to be covered by the Licensed Patents. The patent license
shall not apply to any other combinations which include the Contribution.
No hardware per se is licensed hereunder.
c) Recipient understands that although each Contributor grants the licenses
to its Contributions set forth herein, no assurances are provided by any
Contributor that the Program does not infringe the patent or other
intellectual property rights of any other entity. Each Contributor
disclaims any liability to Recipient for claims brought by any other
entity based on infringement of intellectual property rights or
otherwise. As a condition to exercising the rights and licenses granted
hereunder, each Recipient hereby assumes sole responsibility to secure
any other intellectual property rights needed, if any. For example, if a
third party patent license is required to allow Recipient to distribute
the Program, it is Recipient's responsibility to acquire that license
before distributing the Program.
d) Each Contributor represents that to its knowledge it has sufficient
copyright rights in its Contribution, if any, to grant the copyright
license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under
its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and
b) its license agreement:
   i) effectively disclaims on behalf of all Contributors all warranties
      and conditions, express and implied, including warranties or
      conditions of title and non-infringement, and implied warranties or
      conditions of merchantability and fitness for a particular purpose;
   ii) effectively excludes on behalf of all Contributors all liability for
       damages, including direct, indirect, special, incidental and
       consequential damages, such as lost profits;
   iii) states that any provisions which differ from this Agreement are
       offered by that Contributor alone and not by any other party; and
   iv) states that source code for the Program is available from such
       Contributor, and informs licensees how to obtain it in a reasonable
       manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and
b) a copy of this Agreement must be included with each copy of the Program. Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must:

a) promptly notify the Commercial Contributor in writing of such claim, and
b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

(continues on next page)
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPENT NOR ANY
CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION
LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE
EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under
applicable law, it shall not affect the validity or enforceability of the
remainder of the terms of this Agreement, and without further action by the
parties hereto, such provision shall be reformed to the minimum extent
necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Program itself
(excluding combinations of the Program with other software or hardware)
infringes such Recipient's patent(s), then such Recipient's rights granted
under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to
comply with any of the material terms or conditions of this Agreement and does
not cure such failure in a reasonable period of time after becoming aware of
such noncompliance. If all Recipient's rights under this Agreement terminate,
Recipient agrees to cease use and distribution of the Program as soon as
reasonably practicable. However, Recipient's obligations under this Agreement
and any licenses granted by Recipient relating to the Program shall continue
and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in
order to avoid inconsistency the Agreement is copyrighted and may only be
modified in the following manner. The Agreement Steward reserves the right to
publish new versions (including revisions) of this Agreement from time to
time. No one other than the Agreement Steward has the right to modify this
Agreement. The Eclipse Foundation is the initial Agreement Steward. The
Eclipse Foundation may assign the responsibility to serve as the Agreement
Steward to a suitable separate entity. Each new version of the Agreement will
be given a distinguishing version number. The Program (including
Contributions) may always be distributed subject to the version of the
Agreement under which it was received. In addition, after a new version of the
Agreement is published, Contributor may elect to distribute the Program
(including its Contributions) under the new version. Except as expressly
stated in Sections 2(a) and 2(b) above, Recipient receives no rights or
licenses to the intellectual property of any Contributor under this Agreement,
whether expressly, by implication, estoppel or otherwise. All rights in the
Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the
intellectual property laws of the United States of America. No party to this
Agreement will bring a legal action under this Agreement more than one year
after the cause of action arose. Each party waives its rights to a jury trial in
any resulting litigation.
CHAPTER 10

MQTT Specification
CHAPTER 11

CoAP RFC 7252
CHAPTER 12

Indices and tables

• genindex
• modindex
• search


A
ASP, 41
ATS, 41

C
CoAP, 41

E
ETS, 41
ETSI, 41

I
Internet of Things, 41
IoT, 41
IUT, 41

M
MQTT, 41

P
PCO, 41
PDU, 41

S
SUT, 41

T
TC, 41
TDL, 41
Test Case, 41
Test Purpose, 41
Thing, 42
TP, 42
TS, 42
TSS, 42
TTCN-3, 42

U
UT, 42