WESTERN POWER DISTRIBUTION

OPENLV

OPENING UP THE SMART GRID

Developing with the LV-CAP Virtual Machine

Version 2622-MANUL-S0001-V02.03.01

ea technology

RIIO NIC NETWORK INNOVATION COMPETITION

Report Title	:	Developing with the LV-CAP Virtual Machine
Report Status	:	Issued
Project Ref	:	WPD/EN/NIC/02 - OpenLV
Date	:	07.02.18

Document Control

	Name	Date
Prepared by:	Richard Ash	06.02.2018
Reviewed by:	Richard Potter	07.02.2018
Approved by:	Daniel Hollingworth	07.02.2018

Revision History

Date	Issue	Status
07.02.2018	V02.03.01	Issued

1 Contents

DISCLAIMER

Neither WPD, nor any person acting on its behalf, makes any warranty, express or implied, with respect to the use of any information, method or process disclosed in this document or that such use may not infringe the rights of any third party or assumes any liabilities with respect to the use of, or for damage resulting in any way from the use of, any information, apparatus, method or process disclosed in the document.

2 Introduction

The Low Voltage - Common Application Platform (LV-CAP) is a hardware-agnostic software environment designed to accelerate the deployment of the Smart Grid on electricity networks. It does this by:

1. Allowing multiple "Smart" Applications to be deployed on a single set of hardware and sensors, eliminating costly duplication.
2. Decoupling the challenging but application-agnostic areas of communications and data storage from the development of innovative monitoring and control algorithms.
3. Making it easy to deploy new and updated algorithms to existing equipment without expensive field visits.
4. Providing a common software abstraction for a range of hardware implementations, so that the optimum hardware and software can be procured separately, rather than as a compromise bundle.
5. Allowing a wide range of parties to compete to supply Applications to run on a single set of hardware owned by the system operator.

The interface to LV-CAP and the standards required is set out in the API document 2383-MANUL *LV Common Application Platform Public API*.

This document details how to start developing for the LV-CAP platform using a Virtual Machine image supplied by EA Technology. This Virtual Machine image gives developers a representative environment on which to test and debug their Applications on and ensure their Application integrates with the rest of the system.

3 Platform Overview

The Low Voltage - Common Application Platform (LV-CAP) provides a framework for measurements to be made, processed through algorithms, and actions taken based on the results (Figure 1). All of these functions may be undertaken by Applications developed by EA Technology or third parties. LV-CAP provides a number of core services for third party Container developers to utilise. These are:

1. Container management (installation, configuration, starting and running of Applications, including multiple copies and versions.).
2. A Data Marketplace which allows all Applications on the platform to communicate with each other in a uniform manner.
3. A Data Storage mechanism which allows Application outputs to be stored for future use.

All other functionality is provided by Applications, but using standard interfaces so that different implementations can be swapped in and out without affecting other Applications. To achieve this Applications do not communicate directly but rather via the Data Marketplace as shown in Figure 2. Typically one or more Sensor Applications will be used to ingest measurement data, which will pass to Algorithm Applications. The outputs of these Algorithm Applications will then be used to drive Output Applications or uploaded to central data stores by Communications Applications.

More detail on the working of LV-CAP can be found in document 2383-MANUL *LV Common Application Platform Public API*, hereafter "the LV-CAP API".

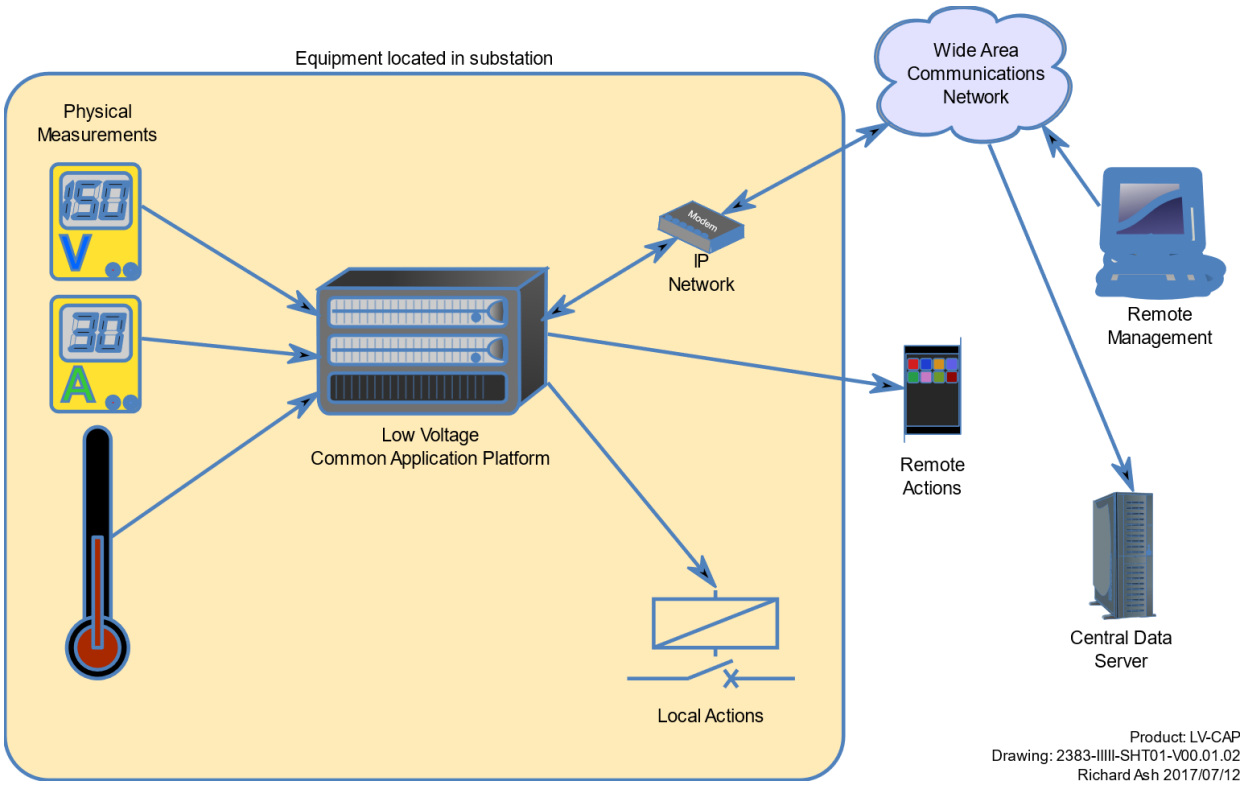


Figure 1 - LV-CAP System Concept

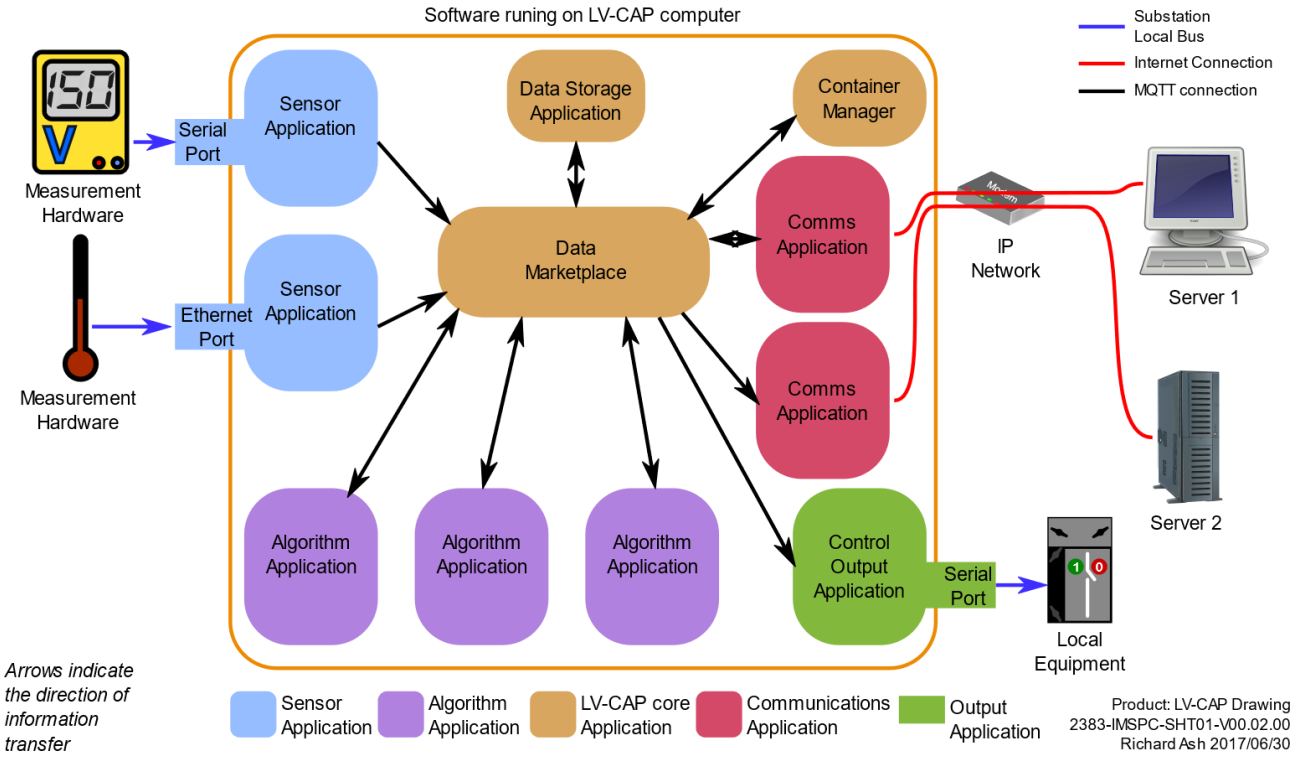


Figure 2 - LV-CAP Software Architecture

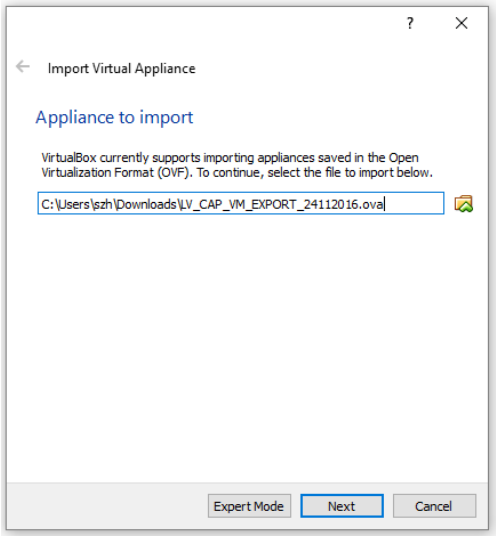
4 Running the Virtual Machine

The Virtual Machine image supplied by EA Technology is in the Open Virtualization Format 2.0 (.ova) file format. It is recommended that Oracle's VirtualBox (https://www.virtualbox.org) version 5 is used for importing and running the Virtual Machine, as this is where testing has been undertaken.

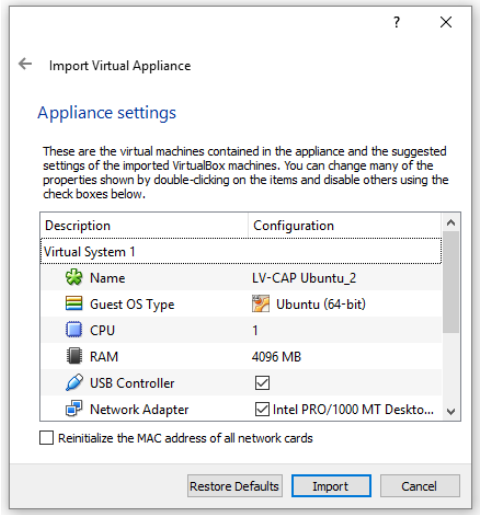
4.1 Importing the Virtual Machine

The Virtual Machine .ova file requires importing into VirtualBox before it can be used. Once downloaded follow the below steps to import the .ova file:

1. Launch Virtual Box then Navigate to "File" -> "Import Appliance"
2. Once the "Import Virtual Appliance" window has launched select the LV-CAP Virtual Machine .ova file then click "Next".



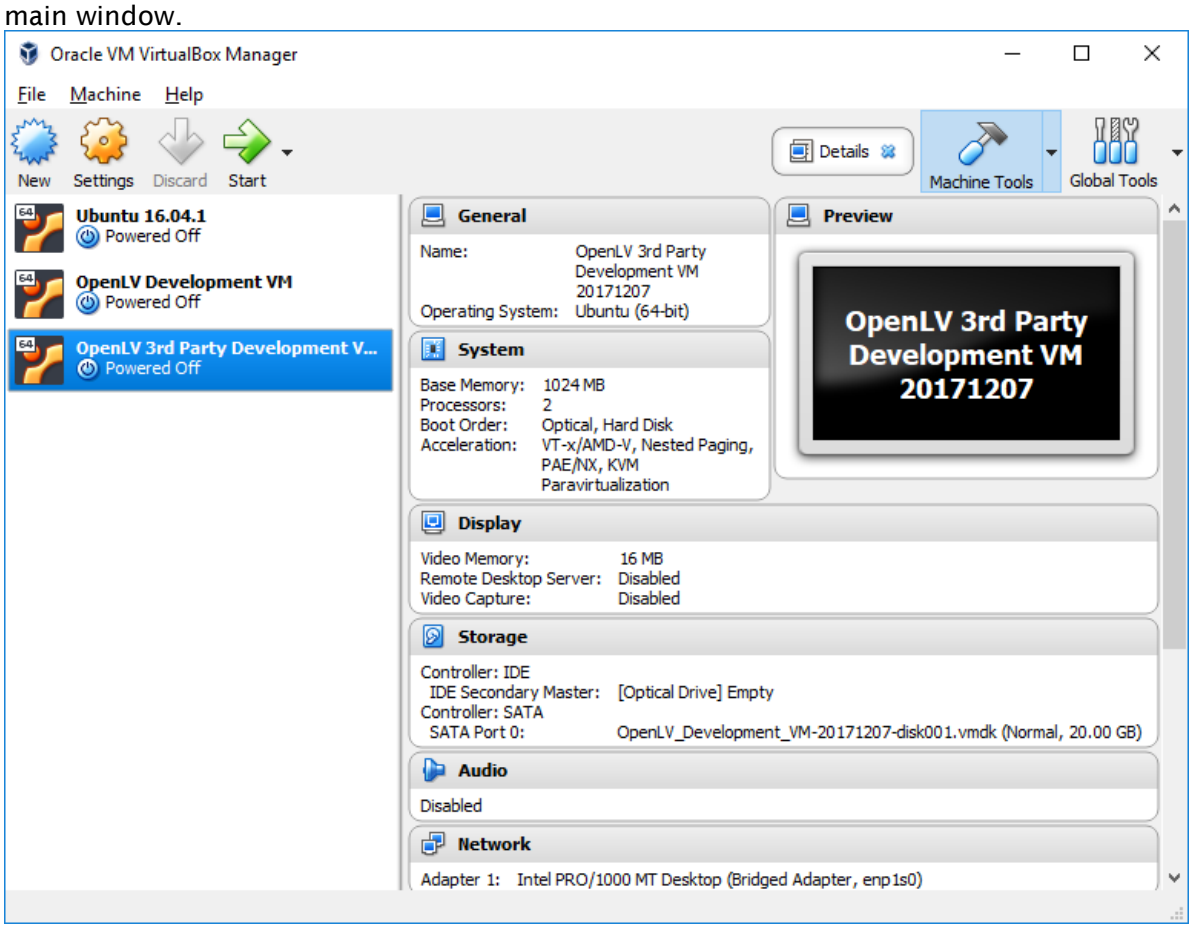
3. The Virtual Machine settings will then be displayed. This allows any of the configuration parameters to be modified to suit the computer it is to be run on such as RAM, CPU or Network Adapters. The Virtual Machine is supplied configured for Bridged networking and to request an IPv4 network address on boot using DHCP. This will generally work on wired networks but may require adjustment if you are on



a Wireless network.

4. It is recommended to tick the box "Reinitialize the MAC address of all network cards" to ensure there are no conflicts with other Virtual Machines on your system.

5. Once happy with the configuration, click Import. A progress dialog will display and then once complete the Virtual Machine will show in the machine list in VirtualBox's main window.



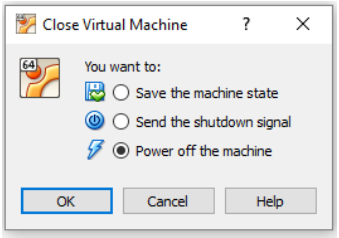
4.2 Starting the Virtual Machine

To start the Virtual Machine, select it in the list of available machines then click the green "Start" arrow at the top of the VirtualBox window. Upon starting the Virtual Machine, a new window will open to display its output.

4.3 Shutting down the Virtual Machine

To shut down the Virtual Machine, click the "X" icon in the top right of the Virtual Machine output window. Three options will be presented:

1. "Save the machine state" – This will store the current state of the Virtual Machine and power if off. This allows the Virtual Machine to be powered back on and pick up from its previous state.
2. "Send the shutdown signal" – This will send the shutdown signal to the Virtual Machine's OS, equivalent to shutting down from the command line.
3. "Power off the machine" – This kills the Virtual Machine without any shutdown of the OS. This is not recommended, and data loss may result.



Select the desired option, and then click “OK” to perform the action.

5 Accessing the Virtual Machine

Like the production LV-CAP environment, the virtual machine does not have a graphical user interface installed. There is no point to a graphical interface on a system which is installed in a locked, unmanned substation!

There are a number of ways the LV-CAP platform can be accessed for editing/copying files on the platform, and viewing the status of the platform. These are covered in the sections below.

5.1 User Account

A default development account exists on the system. This has sudo rights and so can be used to create other user accounts as required. On first login you will be forced to change the password to keep the virtual machine secure. The first login should be made using the following credentials:

Username: *development*
Password: *OpenLVdevelopment*

5.2 Terminal access to the host OS

Once the Virtual Machine has booted, a terminal prompt will be displayed in the VirtualBox window. This allows login into the host OS of the platform in text mode. This requires no further settings, but the terminal is not sophisticated, and there is no means to copy files.

5.3 SSH Access to the host OS

The host OS can be accessed using an SSH session from host machine of the Virtual Machine, or another computer on the same network (depending on your network settings). To do so the IPv4 address of the Virtual Machine must be determined. This can be found by running the command `ip addr sh enp0s3` once logged into the host OS terminal (as detailed in section 5.1). This will produce an output like that shown in Figure 3.

```
richard@openlv-lucy:~$ ip addr sh enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:18:b1:3f brd ff:ff:ff:ff:ff:ff
    inet 172.16.2.122/16 brd 172.16.255.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe18:b13f/64 scope link
       valid_lft forever preferred_lft forever
richard@openlv-lucy:~$
```

Figure 3 - Determining the Virtual Machine IP Address

On the third line of output can be seen the label "inet" followed by the value "172.16.2.122/16". The numbers will be different on your system. The part of the value before the "/" is the IPv4 address, i.e. "172.16.2.122" in this case.

An SSH client (e.g. the open source Putty client) can be used to connect to this IP address. When prompted, log in using the user account credentials listed above.

5.4 Copying files using SCP to the host OS

Files can be copied to the host OS using an SCP session from the host machine of the Virtual Machine or another computer on the same network. To do so the IP address of the Virtual

Machine must be determined as described in section 5.3 above. Once the IP address is known, a SCP client (e.g. the open source WinSCP or Filezilla clients) can be used to connect to this IP address. When prompted, log in using the user account credentials listed above.

6 Running a new Application on the Platform

The development Virtual Machine is, unlike a production system, not connected to a central management control server. This server would normally be the only route by which the Platform can be updated or reconfigured. For the development Virtual Machine the normal Management Communications Application is replaced by a dummy communications Application. This does not in fact communicate to anywhere, but watches certain local directories for new files. When new files are detected, the same MQTT messages are passed into the rest of the system as would be produced by the normal Management Communications Application.

6.1 Authorising the Application

The LV-CAP system has a process of Authorisation in order to control what applications can run on the system, and in particular:

- Authorising Applications to connect to the Data Marketplace.
- Controlling what topics each Application can publish and subscribe to.

The development Virtual Machine is shipped with a configuration file which includes Authorisation for the known Applications at the time the Virtual Machine image was created. However, it will be necessary to manually authorise new Applications which are being developed before they can connect to the MQTT broker. Once an Application has been accepted for deployment on LV-CAP the Authorisation will be added to the system so that production systems do not need manual Authorisation.

The process for Authorising a new Application on the development Virtual Machine is as follows:

1. Make sure that the Instance ID and Application ID of the new Application are known.

2. Inspect the Data Marketplace Application Container to determine where its configuration files are being stored (this path is different on different machines). This can be done with the command
   docker inspect dd7e7e53-7de2-456e-9e9b-01a136f1cd84 | jq '.[0]."Mounts"[] | select(.Destination == "/etc/mosquitto") ."Source" '

3. Edit the containers.acl located in the directory given by the above command
   sudoedit $(docker inspect dd7e7e53-7de2-456e-9e9b-01a136f1cd84 | jq -r '.[0]."Mounts"[] | select(.Destination == "/etc/mosquitto") ."Source" ')/containers.acl

4. Scroll down to the end of the file. In this file, lines starting with a '#' are treated as comments. You will find a commented out block of template rules as shown below:

```
# user <APID>
# API Section 8.2.1
#topic read  config/response/<IID>
#topic write config/request/<IID>
# API Section 8.2.2
#topic read  status/request
#topic write status/response/<IID>
# API Section 8.2.3
#topic read  command/<IID>
# API Section 8.2.4
#topic write storage/data/error/<IID>
# API Section 8.3 Sensor Data
#topic write sensor/data/<IID>/#
#topic read  sensor/data/#
# API Section 8.4 Algorithm output
#topic write algorithm/data/<IID>/#
#topic read  algorithm/data/#
# API Section 8.5 Data Upload
#topic write storage/request/newdata/<IID>
#topic read  storage/response/newdata/<IID>
#topic write storage/uploaded/<IID>
# API Section 8.6
#topic write storage/data/<IID>
#topic write storage/request/<IID>
#topic read  storage/response/<IID>
```

5. Copy these rules and paste them to form the Authorisation rules for your new Application.

6. Add a comment before the pasted rules to say which Application they authorise.

```
# rules for EATL Skeleton Application
```

7. Remove the '#' from the 'user' line. Replace '<APID> with the new Application's Application ID.

```
user eatl_skeleton
```

8. Enable the core topic access defined in the LV-CAP API section 8.2. To do this remove the '#' from the 'topic' lines labelled with API Section 8.2.x. Replace <IID> each time it occurs with the new Application's Instance ID.

```
# API Section 8.2.1
topic read  config/response/eatl_skeleton_00
topic write config/request/eatl_skeleton_00
# API Section 8.2.2
topic read  status/request
topic write status/response/eatl_skeleton_00
# API Section 8.2.3
topic read  command/eatl_skeleton_00
# API Section 8.2.4
topic write storage/data/error/eatl_skeleton_00
```

9. Set up access to sensor data (API Section 8.3). For Algorithm Applications this means read access only, as shown below. If a Sensor Application is being developed, then it should have write access to only its own Instance area of the sensor topics.

```
# API Section 8.3 Sensor Data
#topic write sensor/data/<IID>/#
topic read  sensor/data/#
```

10. Set up access to Algorithm data (API Section 8.4). Applications have read access to all topics, and write access to only their own Instance area.

```
# API Section 8.4 Algorithm output
topic write algorithm/data/eatl_skeleton_00/#
topic read  algorithm/data/#
```

11. This is all that is normally required for Algorithm Applications. Applications which fulfil the Data Upload role, or which need access to values stored in the Data Storage Application, will need other lines to be uncommented in line with the API documentation. The completed rules look like this:

```
# rules for EATL Skeleton Application
user eatl_skeleton
# API Section 8.2.1
topic read  config/response/eatl_skeleton_00
topic write config/request/eatl_skeleton_00
# API Section 8.2.2
topic read  status/request
topic write status/response/eatl_skeleton_00
# API Section 8.2.3
topic read  command/eatl_skeleton_00
# API Section 8.2.4
topic write storage/data/error/eatl_skeleton_00
# API Section 8.3 Sensor Data
#topic write sensor/data/<IID>/#
topic read  sensor/data/#
# API Section 8.4 Algorithm output
topic write algorithm/data/eatl_skeleton_00/#
topic read  algorithm/data/#
```

12. Save the file and exit the editor.

13. Re-start the Data Marketplace Application so that it re-reads its configuration: **docker restart dd7e7e53-7de2-456e-9e9b-01a136f1cd84**

14. The other core Applications will re-connect to the Data Marketplace, and the system is now ready to deploy the new Application. It is recommended to keep a copy of your modified containers.acl file for future reference.

15. If the Data Marketplace is upgraded, then the modifications to the file will have to be re-done.

6.2 Deploying the Application

Once the development system has been Authorised to run the new application, the Application can be deployed.

As described in the LV-CAP API, the Container Manager is responsible for starting and stopping of all Applications. The Container Manager is informed of the Applications which should be running via it's configuration file. Details of the format of this configuration file can be found below. It is important that the steps described below are performed in the correct order, in order to emulate the operation of the normal Management Communications Application and central management server.

The process for deploying a new Application to the platform is as follows:

1. Package the Application as a Docker image file with a *.tar* extension, named with the Application ID as specified in section 4.2 of the LV-CAP API.

2. Copy the running Container Manager configuration file from */home/CM/LVCAP_config/75e81145-e85f-42ff-b992-d9d12c865c0e/config.json* to a local file named *75e81145-e85f-42ff-b992-d9d12c865c0e.json.*
3. Edit the configuration to add the JSON object shown in Figure 4 below into the "Containers" JSON Array.

```
{
    "containerName": "<Instance ID>",
    "File": "<Application ID>.tar",
    "imageTimestamp": <timestamp of the created
container>,
    "DockerParams": {
        "containerName": "<Instance ID>",
        "imageID": ""<Application ID>:<Version>"
    }
},
```

Figure 4 - Example Application definition

- *containerName* – The Instance ID (for legacy Applications, GUID) of this Application.
- *File* – name of the image file for the Application. For legacy application <GUID>.tar.
- *imageTimestamp* – The Unix time (in seconds) when the image was created.
- *DockerParams*
  - *containerName* – The same as the *containerName* above.
  - *imageID* – The Docker tag (including version) of the image, as supplied to the -t option of the *docker build* command.

An example Container Manager configuration is contained in 11 of this document.

4. Once modified, copy the Container Manager configuration *75e81145-e85f-42ff-b992-d9d12c865c0e.json* to the */tmp/LVCAP_config/* directory. It is important that no other files are copied at this time!
5. At the next check for local updates (1-minute update frequency) the dummy Communications Application will notice the file and notify the Container Manager.
6. The Container Manager will load its new configuration and add the new Application to the list of files to be downloaded from the server. It cannot yet be run because the new Application image and configuration have not yet been downloaded.
7. Now copy the Application image .tar to */tmp/LVCAP_images/* on the Virtual Machine, and the Application's JSON configuration file to */tmp/LVCAP_config/*. This simulates the files being downloaded from the central management server on request.
8. On the subsequent check cycle, the Application image and configuration file will be announced to the Container Manager, which will process them.
9. The Application will then be started by the Container Manager.

7 Updating an Application running on the Platform

As described in the LV-CAP API, the Container Manager is responsible for updating containers and their configuration. When updates exist, the Container Manager must be informed so they can be deployed.

The process for updating the configuration of a running Application is simplest:

1. Copy the Application configuration to */tmp/LVCAP_config/* on the Virtual Machine. The file name must be the same as the original, i.e. <Instance ID>.json
2. At the next check for local updates (1-minute update frequency) the updated configuration will be sent to the running container.

The process for updating an Application image on the platform is as follows:

1. Package the Application as a Docker image file with a *.tar* extension, named with the Application ID as specified in section 4.2 of the LV-CAP API.
2. Copy the running Container Manager configuration file from */home/CM/LVCAP_config/75e81145-e85f-42ff-b992-d9d12c865c0e/config.json* to a local file named *75e81145-e85f-42ff-b992-d9d12c865c0e.json.*
3. Edit the configuration by updating the Application's "*imageTimestamp*" to the created timestamp of the updated image file, and setting the correct ImageID for the new Application version.
4. Once modified, copy the Container Manager configuration to the */tmp/LVCAP_config/* directory. It is important that no other files are copied at this time!
5. At the next check for local updates (1-minute update frequency) the dummy Communications Application will notice the file and notify the Container Manager.
6. The Container Manager will load it's new configuration and add the updated image to the list of files to be downloaded from the server. It cannot yet be run because the new Application image has not yet been downloaded.
7. Upload the Application image .tar to */tmp/LVCAP_images/* on the Virtual Machine
8. At the next status check (1-minute update frequency) the out of date Application will be shut down, the new image loaded, and the new Application started in its place.

8 TLS Certificate Creation

All Applications running on LV-CAP are required to connect to the MQTT broker using TLS certificates, as described in the LV-CAP API (EA Technology drawing 2383-MANUL *LV Common Application Platform Public API*). Each Application must have its own secure key and TLS certificate. In order to develop Applications a simple "self-service" certificate issuing system has been created. For production deployment these certificates are not permitted, and must be replaced with certificates issued by EA Technology from a controlled issuer.

8.1 Certificates for Development

The development Virtual Machine includes a set of certificate authority files which can be used by developers to create TLS certificates signed by the Development Certificate Authority. These are stored in a compressed archive */home/CM/2726_TLS_certificate_authority.tar.xz* (also available separately). To start work this must be unpacked into the developer's private workspace.

Once unpacked there is a directory *OpenLV_TLS_certificate_authority* which contains the following files:

- broker-ca.pem – The TLS certificate used by the MQTT broker. Applications need to use this certificate to authenticate the MQTT broker when connecting to it.
- rootCA.key – The private key for the development certificate authority. The passphrase for this key is "OpenLV5cert!", which has to be entered when creating certificates.
- rootCA.pem – The root certificate for the development certificate authority, which all the other certificates are signed by.
- rootCA.srl – Serial number tracking file used to automatically give each certificate a unique serial number.
- Makefile – Automation file which automates the process of signing certificates without typing cryptic commands.

The first steps in obtaining a certificate set for a container are to create a private key and a certificate signing request. This can either be done manually, or automated with a Makefile template provided by EA Technology.

8.1.1 Creating Key and Signing Request Manually

To create a key and Certificate Signing Request (CSR) using the OpenSSL tools:

1. Generate a 2048 bit RSA private key. This key is for a specific Application image. It is important that this key is not disclosed because anyone in possession of this key can impersonate your Application. The name of this file is not important, but your Application will need to have access to this key.
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:2048 -out <filename.key>
2. Create a certificate signing request. This explains to the certificate authority exactly what you want the certificate to say.
   openssl req -new -key <filename.key> -out <filename.csr>
3. This will prompt for a number of pieces of information about the organisation requesting the certificate (i.e. the Application author). The most important field is the Common Name. This must be set to the Application ID as described in the LV-CAP API documentation.
   a. For legacy Applications this is the assigned 48-byte GUID string.
   b. For new Applications it is the combination of the Vendor, Application Name and Application Version separated with underscores, e.g. eatl_testcontainer_0.1
   c. The email, challenge password and optional company name can be left blank.

4. This will produce the Certificate Signing Request *<filename>.csr*. The name of this file is not important.

8.1.2 Creating Key and Signing Request using the Template Makefile

The TLS Template is provided as the compressed archive *2745-SWREL-<version>-TLS_template.tar.xz*. This unpacks to produce a top-level directory *template/*, which in turn contains two directories named *LV-CAP/* and *2745_tls_template/*. The former contains supporting files which should not be modified, the latter a Makefile to create TLS Certificates for LV-CAP.

To use the Template Makefile:

1. Unpack the template archive into the developer's workspace.
2. Edit file *template/2745_tls_template/Makefile* to set the variables at the top:
3. The five variables which start with "TLS" are used to populate the TLS certificate fields with information about the organisation developing the Application.
4. The variable VENDOR must be set to the **Vendor** string as defined in section 4.2 of the LV-CAP API. This will be assigned to the organisation developing the Application.
5. The variable APP_NAME must be set to the **Application Name** string as defined in section 4.2 of the LV-CAP API. This is chosen by the organisation developing the Application.
6. The variable APP_VER must be set to the **Version** string as defined in section 4.2 of the LV-CAP API. This is chosen by the organisation developing the Application.
7. Change directory to the *2745_tls_template/* directory and run the command **make**
8. This will generate a key for the Application, named *Vendor_Application Name.key,* and a certificate signing request named *Vendor_Application Name.csr*.

8.1.3 Signing Development Certificates

Whichever way the certificate signing request is made, the end result is a *.csr* file which needs to be converted into a certificate file. This can be done manually using the OpenSSL tools, but the commands are complex so a Makefile has been created to automate the process when using the

1. Unpack the certificate authority archive *2726_TLS_certificate_authority.tar.xz* into the developer's private workspace.
2. Copy the Certificate Signing Request (*.csr* file) into the *OpenLV_TLS_certificate_authority/* directory.
3. Change working directory into the same directory.
4. Run the command: **make <name>.crt**
5. This will prompt for the passphrase for the private key, which is “OpenLV5cert!”
6. The result is to create the certificate <name>.crt from <name.csr>.
7. Copy the certificate <name>.crt back to the Application's build directory.
8. If a certificate is required with validity other than the default 90 days, then the DAYS variable may be set on the make command line: **make <name>.crt DAYS=30**

8.2 Certificates for Production Deployment

The process for obtaining production certificates starts the same as for development, with the generation of a private key and Certificate Signing Request. However instead of signing using the Development Certificate Authority, the Certificate Signing Request must be sent to the organisation operating the production Certificate Authority for signing (currently EA Technology). They will then return the signed certificate to be used in the production Application.

9 Viewing MQTT Message Traffic

To view traffic on the TLS MQTT broker it is necessary to authenticate as a user with sufficient rights when connecting to the broker. A set of suitable TLS certificates and scripts for doing this are contained within the development tools bundle 2746-SWREL-Vxx.xx.xx-OpenLV_TLS_devtools.tar.xz

9.1 Terminal

Messages can be viewed on the text mode terminal of the Virtual Machine, via any of the methods described in section 4.

1. Unpack the tarball containing the development tools bundle.
2. Change directory into `2746-SWREL-Vxx.xx.xx-OpenLV_TLS_devtools/2746_devtools/`
3.
    a. To view all messages on the MQTT broker, run `make sub-all`
    b. To view sensor messages on the MQTT broker, run `make sub-sensor`
    c. To view algorithm data messages on the MQTT broker, run `make sub-alg`
4. When you want to stop, press CTRL+C to return to the command prompt.

Each message is output on a new line, which starts with the name of the topic it was published on, followed by a space and then the payload of the message.

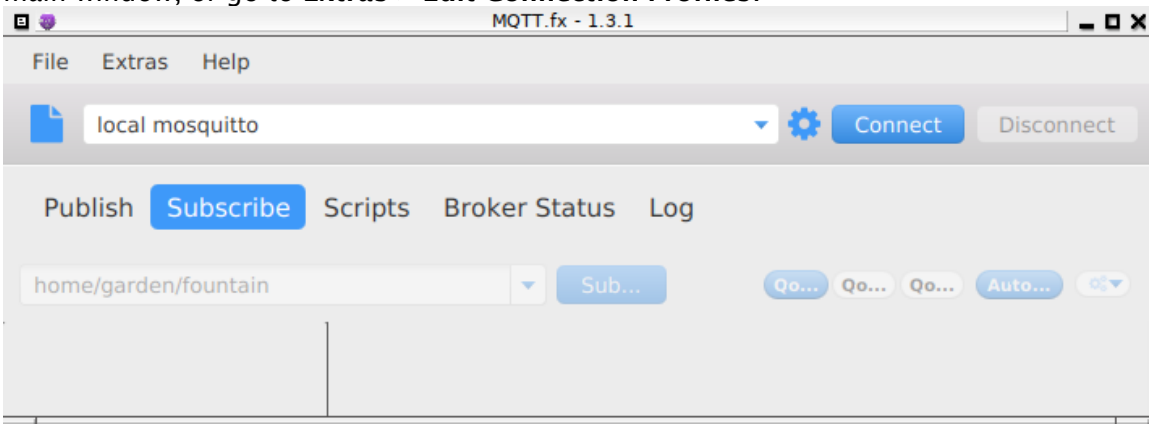
9.2 MQTT.fx

MQTT.fx is a graphical MQTT client which allows interactive subscription and publishing of messages on a MQTT broker. Because it is a graphical application, it cannot be run on the Virtual Machine itself. Instead it must be run on the host machine of the Virtual Machine or another computer on the same network.

MQTT.fx for Windows, Linux or Apple Mac can be downloaded from http://mqttfx.org/.

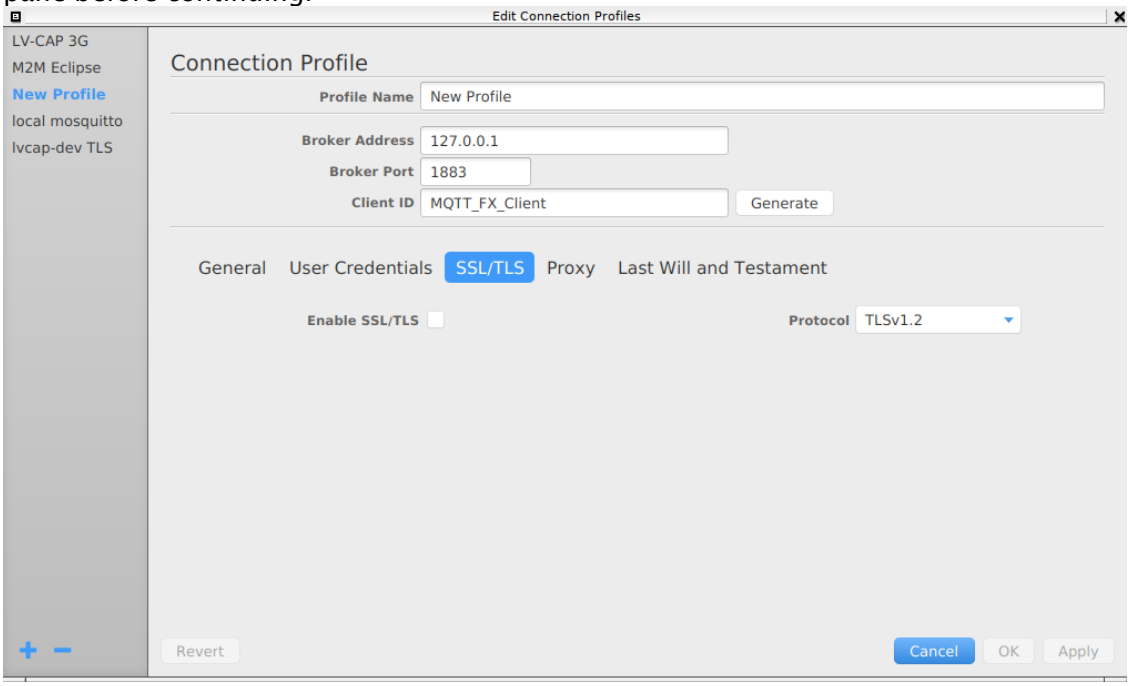
To connect to the MQTT broker on the Virtual Machine the IP address of the Virtual Machine must be determined as described in section 5.3 above. Once the IP address is known, MQTT.fx can be configured to connect to the TLS MQTT broker.

1. Click on the cog-wheel button to the left of the **Connect** button at the top of the main window, or go to **Extras > Edit Connection Profiles**.

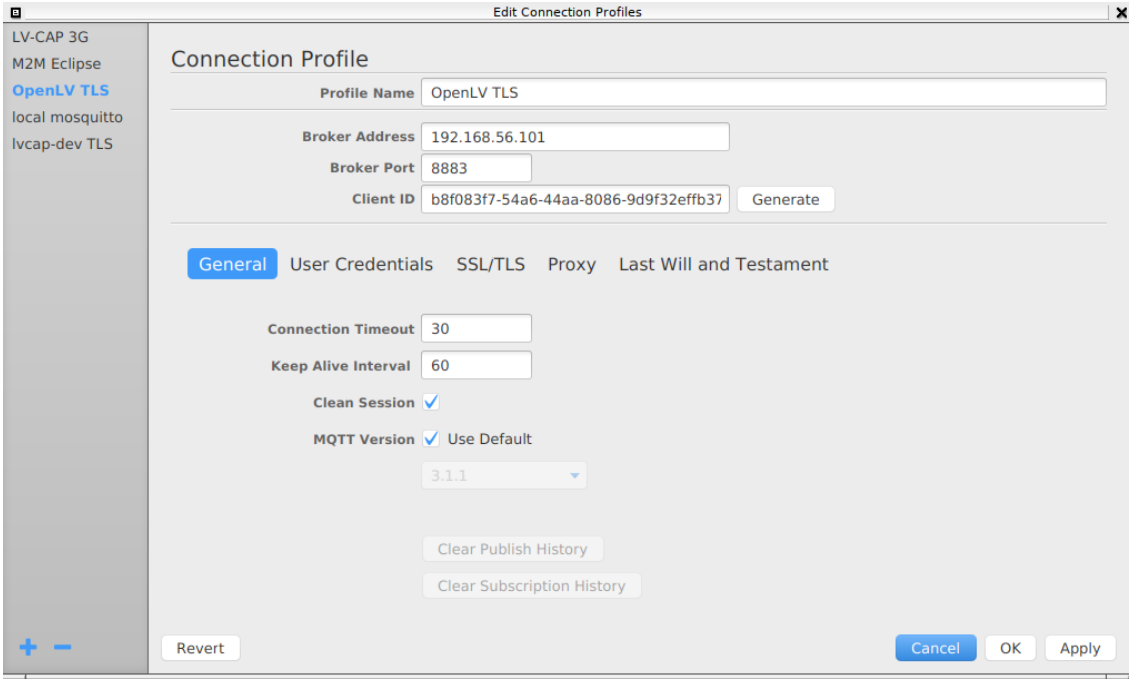


2. This will open the Connection Profile editor window. Click the blue + button at the bottom left to create a new profile. Make sure it is selected (blue) in the left hand

pane before continuing:

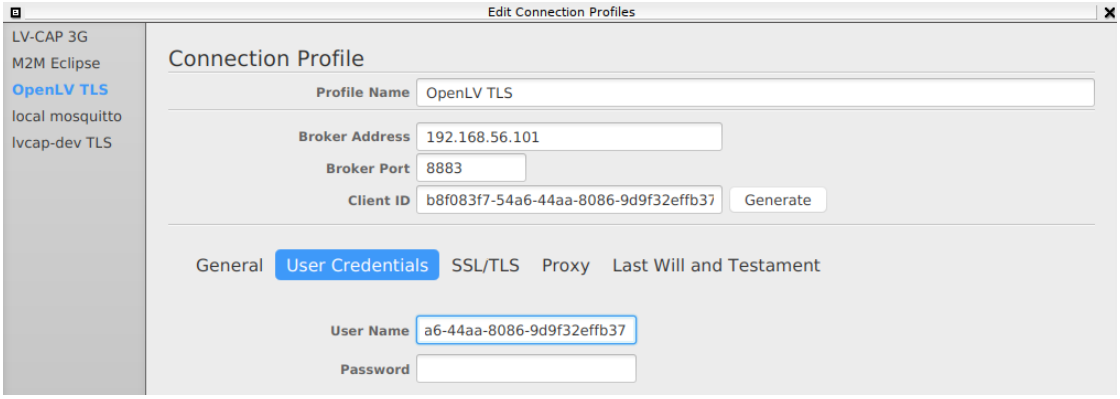


3. Fill out the four fields at the top of the window as follows:
Name: *OpenLV TLS* (can be anything you want it to be)
Broker Address: Virtual Machine IP Address you have identified
Broker Port: *8883*
Client ID: *eatl_tlsdevtools*

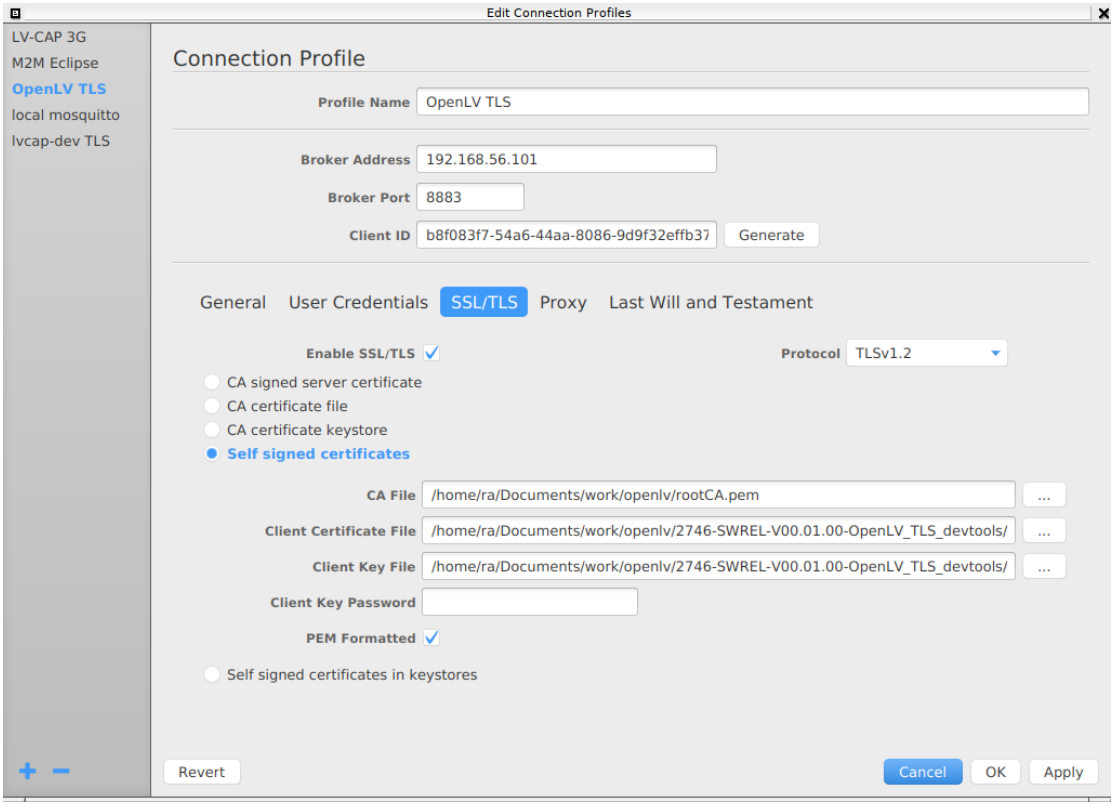


4. In the lower part of the window, leave the **General** tab settings as defaults:
Connection Timeout: *30*
Keep Alive Interval: *60*
Clean Session: *Ticked*
MQTT Version: **Use Default** *Ticked*

5. Move on to the **User Credentials** tab. Set **User Name** to *eatl_tlsdevtools* and leave **Password** blank.



6. Move on to the **SSL/TLS** tab. Tick the **Enable SSL/TLS** box and more controls will appear. Select the **Self signed certificates** radio button.
Browse for the **CA File** and select the *broker-ca.pem* file from the development tools.
Browse for the **Client Certificate File** and select the file *eatl_tlsdevtools.crt*
Browse for the **Client Key File** and select the file *eatl_tlsdevtools.key*
Leave **Client Key Password** blank.
Tick the **PEM Formatted** box.



7. Do not set anything on the **Proxy** or **Last Will and Testament** tabs.
8. Click **OK**.
9. Back in the main window, select the **OpenLV TLS** entry from the list and click **Connect**.

You are now connected and authenticated over TLS and can publish and subscribe on any topic of the MQTT broker.

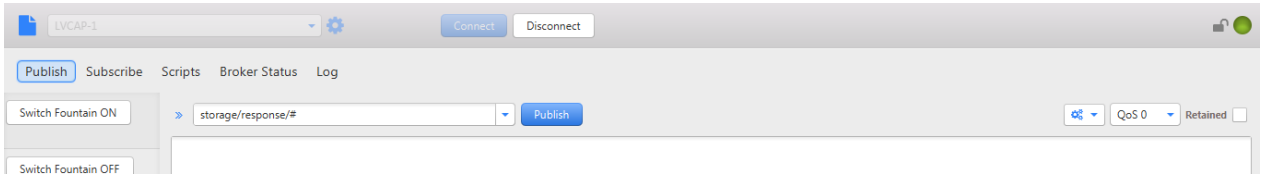


Figure 5 - MQTT.fx after connection

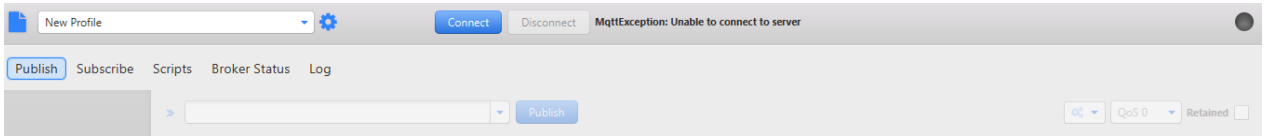


Figure 6 - MQTT.fx failed connection

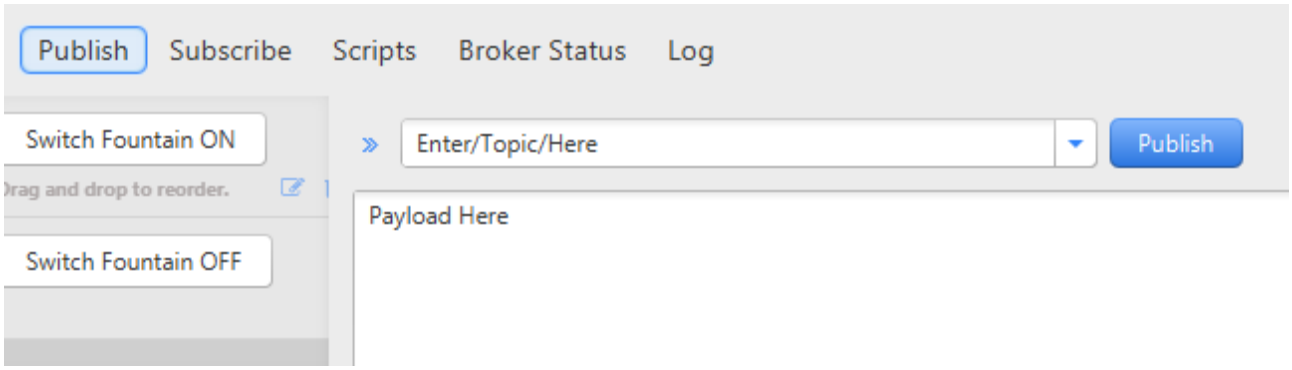


Figure 7 - MQTT.fx publish topic and payload

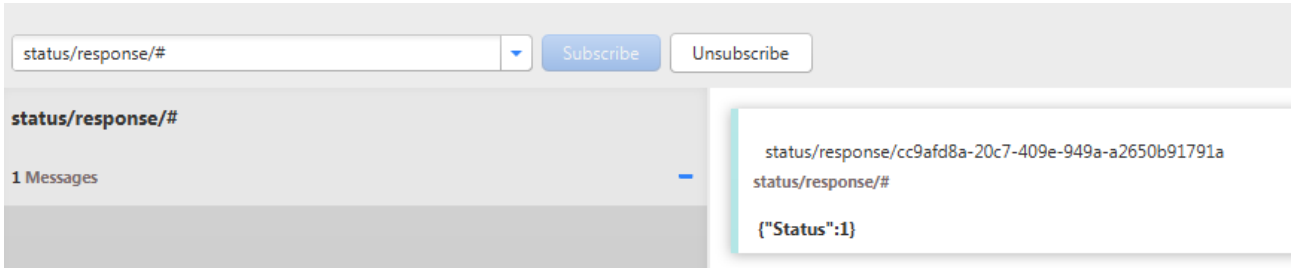


Figure 8 - MQTT.fx Subscription with incoming message

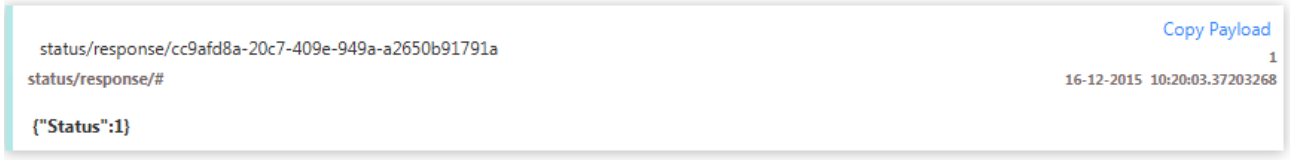


Figure 9 - MQTT.fx incoming payload package

10 Input Data for Testing

Real LV-CAP systems will have one or more Sensor Applications running, whose purpose is to receive data from measurement hardware and make it available on the Data Marketplace for applications to receive. The LV-CAP virtual development environment of course does not have any measurement hardware available to it. In order to test applications, a tool has been developed which accepts Comma Separated Variable (CSV) format data and publishes it onto the Data Marketplace either in real time or faster than real time.

By using the replay tool, it is possible to provide reproducible conditions for testing software, which can be re-run as many times as required and much more quickly than using real input hardware. The format of the CSV files is the same as that written by the EA Technology CSV Data Recorder Application (EATL Dwg. 2660), so that it is possible to record data from real hardware installations and then replay it on the virtual development environment.

10.1 LV-CAP Data Replay Tool

The LV-CAP Data Replay tool is part of the TLS Development Tools package, released as `2746-SWREL-S011-Vxx.xx.xx-OpenLV_TLS_devtools.tar.xz`

The Data Replay tool itself is a Python 3 script named `play_csv.py`.

The necessary dependencies to run the script are already installed on the virtual development environment, so the package archive simply needs to be unpacked. If it is necessary to run the tool on other systems, then installation instructions can be found below.

10.2 Data format

CSV files to be played back must adhere to the following structure to be played correctly:

- The first row of the file is a header.
- The heading of the second column sets the topic on which the data from the second and third columns will be published, the heading of the fourth column sets the topic on which the data from the fourth and fifth columns will be published and so on for all the columns.
- The first column must contain the timestamp for the row, in UTC. The format should be YYYY/MM/DD HH:MM:SS.
- There is no requirement for regular time intervals between rows, but the timestamp must be monotonically incrementing (i.e. each row's timestamp must be greater than the row before!).
- The second column contains the Value member of the JSON to be sent on the first topic, and so on for all the other even-numbered columns.
- The third column contains a logical value which sets the Valid flag of the JSON to be sent on the first topic, and so on for all the other odd-numbered columns.
- Empty fields will be skipped, with no payload sent.

10.3 Playing back data

10.3.1 Command Line Options

The replay tool accepts the command line options described below (see also the -h / --help option output for a full list of available options).

Several of the options are related to connecting to the Data Marketplace MQTT broker. These will normally be set to use the certificates in the TLS Development Tools package as described in Section 9.

`--host HOST`	The host name of the MQTT broker to connect to.
`-p PORT` `--port PORT`	The TCP port number to connect to MQTT broker on.
`--cafile CAFILE`	Path to a file containing PEM encoded CA certificates which are trusted. Used to prove the MQTT broker is authentic.
`--cert CERT`	Path to a file containing a PEM encoded certificate for this client to prove its identity to the MQTT broker.
`--key KEY`	Path to a file containing the PEM encoded private key used by this client to prove its identity to the MQTT broker.

For the purpose of debugging the script, the -l / --log / --no-log options can be used to control loge messages from the underlying MQTT library. None of these options should need to be set for normal use.

The other options control the data which is to be played back:

`-i INFILE` `--infile INFILE`	The name of an input CSV file containing data to replay. This must be specified!

`-s`
`--store`
`--no-store`
	Control the ToStore element of the output JSON payloads sent, controlling whether the data replayed is stored by the Data Storage Application.

`-f TIME_FACTOR`
`--time-factor`
`TIME_FACTOR`
	Set a time scaling factor for the replay.

The most powerful option is setting TIME_FACTOR. If this is set to a factor of 10, then the data will be replayed at 10 times real time. This means that if the input file has data at 10 minute intervals, the MQTT messages will be sent at 1 minute intervals. The time stamps in the payloads will still be the original file timestamps, i.e. 10 minutes apart.

10.3.2 Examples

The examples assume you have logged in to the virtual development environment and changed directory to the "2746_devtools" directory created when the TLS development tools archive is unpacked. To connect to the Data Marketplace on the virtual development environment, using the certificates from the TLS development tools, the following options are needed:

```
--host marketplace --port 8883 --cafile broker-ca.pem --cert
eatl_tlsdevtools.crt --key eatl_tlsdevtools.key
```

To play back the file "input-28day.csv" from the current directory, at a rate of one hour per second, not storing the data in the Data Storage Application, use the following command:

```
./play_csv.py --host marketplace --port 8883 --cafile broker-ca.pem --cert
eatl_tlsdevtools.crt --key eatl_tlsdevtools.key --no-store -i input-
28day.csv -f 3600
```

To play back the file "input-1day-w5d1.csv" from the current directory, at a rate of one minute per second, storing the data in the Data Storage Application, use the following command:

```
./play_csv.py --host marketplace --port 8883 --cafile broker-ca.pem --cert
eatl_tlsdevtools.crt --key eatl_tlsdevtools.key --store -i input-1day-
w5d1.csv -f 60
```

10.3.3 Preparing Input files

CSV files to be used as input to the data replay tool can be obtained from the output of the EA Technology CSV Data Recorder Application (EATL Dwg. 2660), or created off-line using any suitable tools.

An Open Document Spreadsheet file which creates 7 days' worth of load currents is included in the TLS Development Tools package named `load-current-generator.ods`. The column headers as supplied are for load current measurements made using the Lucy GridKey Sensor Application, but can easily be changed for other data points. The load currents produced are based on the standard daily load curves used in underground cable ratings, repeated for each of the 7 days in succession.

To use this spreadsheet to produce input data for the replay tool:

1. Open the spreadsheet in LibreOffice Calc
2. Adjust settings in the blue cells on the "Control" worksheet to set:
    - The desired load curve shape for each phase.
    - The peak load current for each phase.
    - The start date/time (in UTC).
    - The time interval between data points (should be an integer factor of 1 hour for correct operation).
3. Switch to the "export" worksheet.
4. Click File > Save a Copy …
5. Change Save as type to "Text CSV"
6. Enter file name and click Save
7. In export dialogue:
    - Character set: Western Europe
    - Field delimiter: ,
    - Text delimiter: "
    - Tick Save cell content as shown and untick the others
    - Click OK
8. When it warns only the selected sheet was saved, click OK.

The resulting file always has exactly 7 days of data with a header, but the files can easily be modified to produce other duration data sets.

10.4 Installing the replay tool

10.4.1 Installing on Ubuntu

Most of the required packages can be installed on the target operating system, Ubuntu 16.04 LTS, from the Ubuntu repositories with the following command:

```
sudo apt install python3-pip
```

The "pause" and "paho-mqtt" libraries must be installed using the Python PIP package management tool with the command:

```
sudo -H pip3 install paho-mqtt pause
```

Both of these commands assume the system has internet access to download packages.

10.4.2 Installing on other Linux Systems

The replay tool is a Python script, written for Python 3. The only direct dependencies outside of the Python standard library are:

- pause https://pypi.python.org/pypi/pause/ which is used to control the timing of output.
- paho-mqtt https://pypi.python.org/pypi/paho-mqtt used to connect to the MQTT broker.

The names of these packages vary between Linux distributions, or both can be installed using Python's own PIP package management tool. Note that care must be taken to install the Python 3 (as opposed to Python 2) packages!

11 Appendix 1 Example Container Manager Configuration

This is an example Container Manager Configuration for running an Application with the following information:

- Vendor: "eatl" (see Section 4.1 of the LV-CAP API document)
- Name: "skeleton" (see section 4.1 of the LV-CAP API document)
- Application version: 00.02.00-developement

From this we know that the Application's Instance ID will be *eatl_skeleton_00* and its Application ID will be *eatl_skeleton*. By substituting these into the structure given in Section 6 of this document, we get the configuration below:

```
{
  "StatusUpdatePeriod": 60,
  "Containers": [
  {
    "containerName": "eatl_skeleton_00",
    "File": "eatl_skeleton_00.tar",
    "imageTimestamp": 1484838320,
    "DockerParams": {
        "containerName": "eatl_skeleton_00",
        "imageID": "eatl/skeleton:00.02.00-developement"
    }
  }
  ]
}
```

To run this application, we will need three files:

- The above Container Manager configuration, in a file named *75e81145-e85f-42ff-b992-d9d12c865c0e.json*
- The Application image file, named eatl_skeleton_00.tar
- The Application configuration file, named eatl_skeleton_00.json

See Section 6 of this document for the procedure to start the Application.

To run multiple Applications, place similar elements in the "Containers" array for each Application.