WESTERN POWER DISTRIBUTION

OPEN LV

OPENING UP THE SMART GRID

Requirements Specification

ea technology

RIIO NIC NETWORK INNOVATION COMPETITION

Report Title:	OpenLV Solution Requirements Specification
Report Status:	Issued
Project Ref:	WPD/EN/NIC/02 – OpenLV
Date:	20 September 2017

Document Control

	Name	Date
Prepared by:	Tim Butler	August 2017
Reviewed by:	Richard Ash	August 2017
	Richard Potter	August 2017
Recommended by:	Dan Hollingworth	20 September 2017
Approved (WPD):	Mark Dale	20 September 2017

Revision History

Date	Issue	Status
20 September 2017	1.0	Issued pre-FATs Part 2
15 August 2017	0.1	Draft issued for comment pre-FATs Part 1.

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.

Telephone +44 (0) 1332 827446. E-mail wpdinnovation@westernpower.co.uk

Contents

Table of figures

Table of tables

1 Introduction

1.1 Purpose

The Purpose of this document is to provide a record of the requirements for the overall OpenLV solution that will be required to support Project trials for the three Methods outlined in the Full Submission Proforma (OpenLV Bid document).

1.2 Background

The FSP provides a high-level description of the overall OpenLV solution that will be required to support Project trials for the three Methods.

In order to define the requirements for the overall OpenLV Solution, the key hardware and software components have been defined, then the requirements for each component have been identified.

In order to prioritise requirements, the MoSCoW approach has been utilised. This approach is a well-known technique used in business analysis and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement.

Each requirement has been identified and prioritised using the MoSCoW approach, that stands for Must, Should, Could and Will not:

- M – Must have this requirement to meet the business needs;
- S – Should have this requirement if possible, but project success does not rely on it;
- C – Could have this requirement if it does not affect anything else in the project; and
- W – Will not deliver this requirement at the current time, but it could be delivered at a later date.

1.3 OpenLV Solution Overview

The OpenLV solution to be trialled within the OpenLV project, utilises a distributed intelligence device, built on a software platform, LV-CAP™, developed to enable multiple applications and solutions to be deployed in a single enclosure. The trials will, across three 'Method' areas, demonstrate that the platform can:

- monitor the network in real-time;
- process the collected data to determine the need for action to manage the network;
- implement that action if necessary;
- provide this data to third party companies and equipment; and
- provide this data to community groups.

1.3.1 Method 1: LV network capacity uplift

Method 1 will demonstrate the capability of the LV-CAP™ platform to perform measurements and control from within an HV/LV substation, in 60 substations (30x pairs).

To demonstrate this, the deployed trial hardware will utilise monitored data to predict future network load and when necessary, automatically share the feeder load between two transforms. This will be demonstrated through direct control of ALVIN Reclose™ circuit breakers installed in the substations at either end of the utilised feeder in a subset (5x pairs) of Method 1 installations.

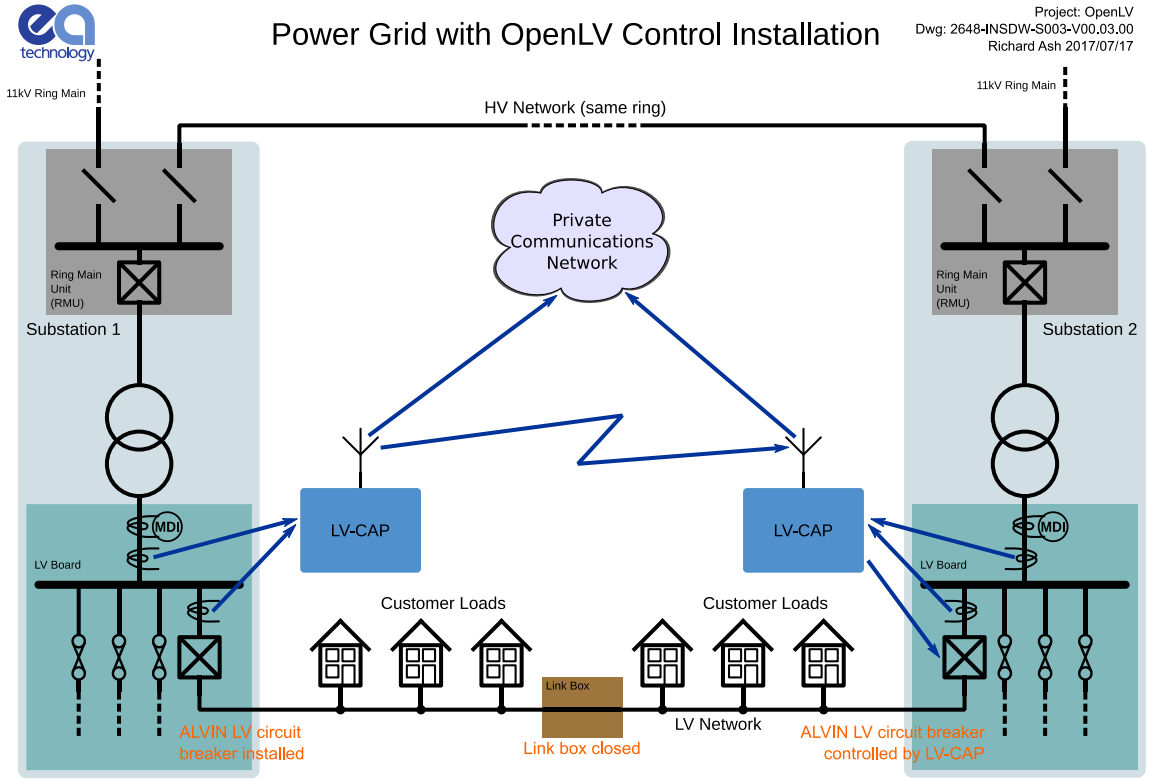

Figure 1: OpenLV Method 1 Overview

1.3.2 Method 2: Community engagement

Once deployed, the LV-CAP™ platform can be used to provide data to community groups or individual customers. LV-CAP™ platforms deployed for Method 2 implementation will be identical to those deployed for Method 1 but will not, in any situation, have ALVIN Reclose™ devices installed as well.

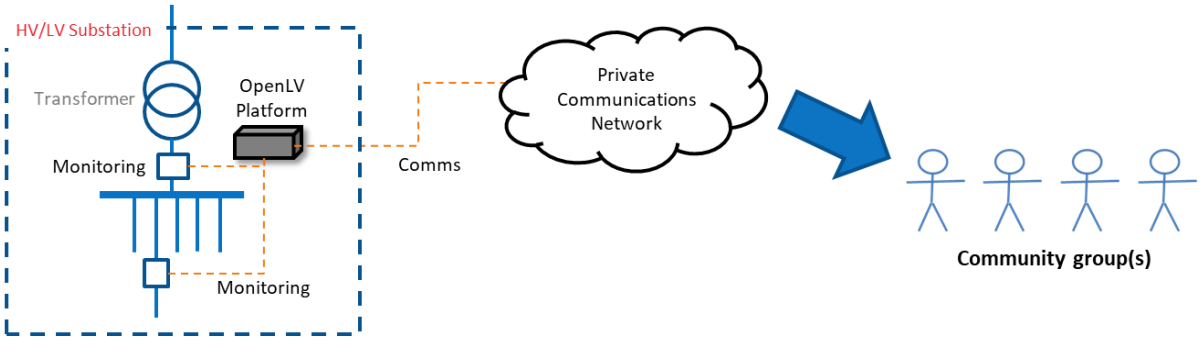

Figure 2: OpenLV Method 2 Overview

10 LV-CAP™ devices have been allocated for the implementation of Method 2 activities.

1.3.3 Method 3: OpenLV extensibility

Once deployed, the LV-CAP™ platform can be used to provide a secure platform for third parties to provide services to the DNOs, customers, and wider industry. This may take the form of pure Applications, or a combination of Applications and connected external devices. As with Method 2, however, Method 3 installations will not, in any situation, have ALVIN Reclose™ devices installed as well.

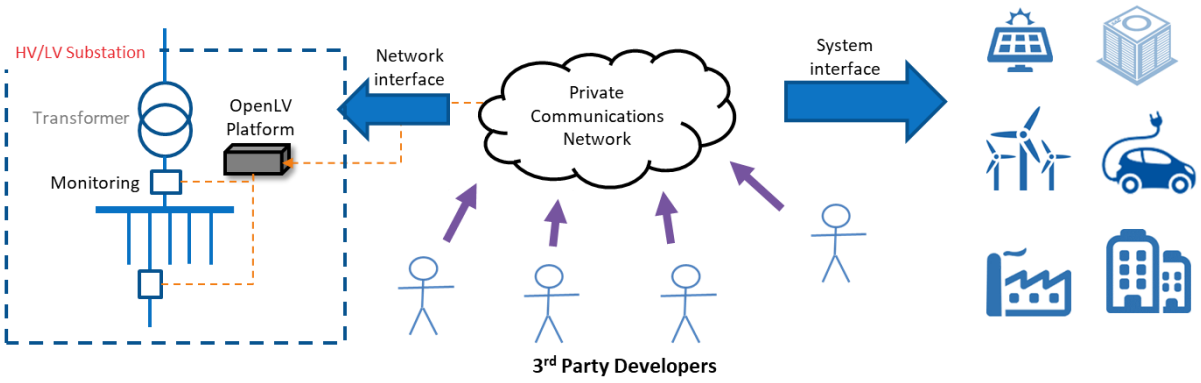

Figure 3: LV-CAP™ Method 3 Overview

10 LV-CAP™ devices have been allocated for the implementation of Method 3 activities.

1.4 LV-CAP™ Software Platform Overview

The LV-CAP™ software platform is designed to enable a single hardware deployment to monitor the network and make the gathered data available to multiple software Applications running on the platform. These Applications could be developed by multiple manufacturers and control various unrelated network assets without any application being influenced or affected by another.

This enables a single investment in the hardware to support deployment of multiple solutions to benefit the network.

The figure below demonstrates the deployment of software applications within the LV-CAP™ platform under the OpenLV Project.

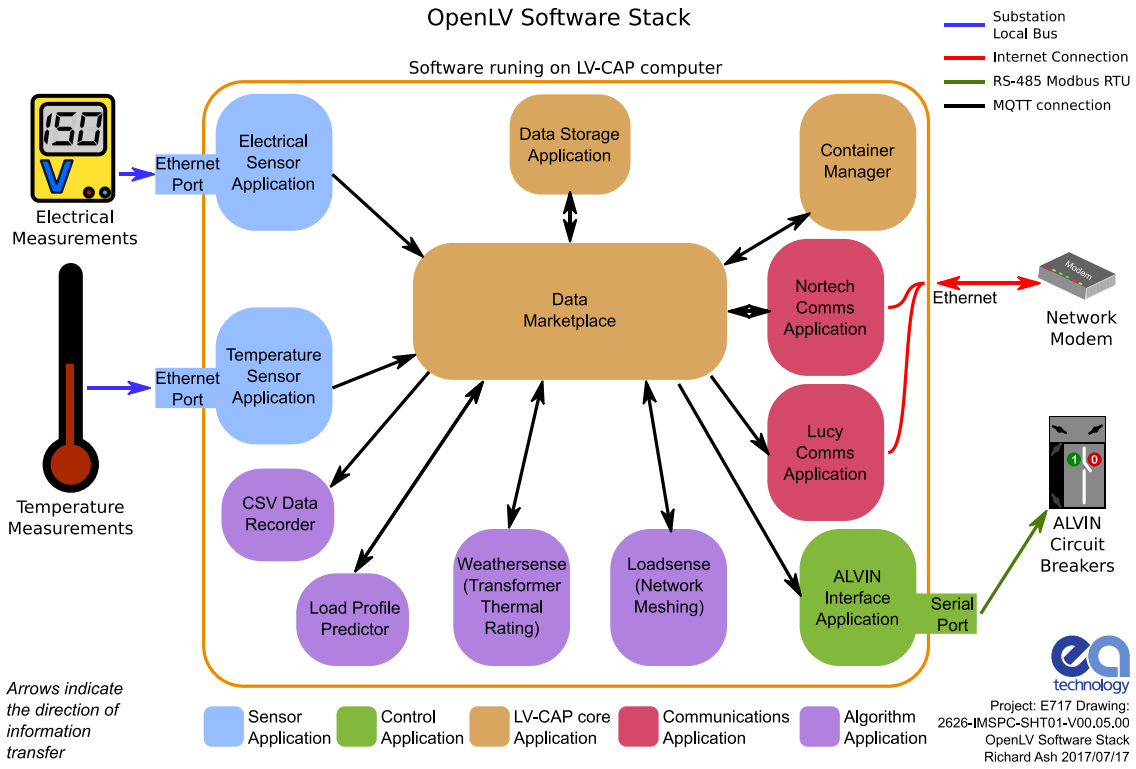

Figure 4: LV-CAP™ Software Platform Overview

1.5 Report Structure

The structure of this document is as follows:

- **Section 2: Grouping of the OpenLV Solution Requirements –** Provides an overview of the key components of the overall system.
- **Section 3: OpenLV Solutions Requirements –** Outlines the requirements for each component of the overall OpenLV solution.
- **Section 4: The Way Forward –** Outlines how the requirements will be used within the Project.

2 Grouping of the OpenLV Solution Requirements

The key components of the overall OpenLV solution have been assessed and requirements have been grouped under the titles outlined in

Table 1: Grouping of Requirements

Group	Description
Intelligent Substation Devices	The Intelligent Substation device is an enclosure, containing a ruggedised PC capable of being installed in harsh environments and interface connections to receive data from external sensors. Applications installed within the LV-CAP™ Software Platform gather data, store it locally, process if necessary and send requested information back to central servers.
LV-CAP™ Software Platform	The LV-CAP™ software platform runs on the intelligent substation devices. This is an operating system that enables multiple Applications to be installed in software containers on a single device. The software also provides Apps with access to data provided from the sensors which are installed in each LV substation.
LV Monitoring Equipment	LV network monitoring provides the core data to be utilised by all Apps that will be deployed on the Intelligent Substation Devices.
Temperature Sensing	Temperature sensors are required to monitor the temperature of the LV transformer and ambient air temperature. This data is provided to the relevant container 'Apps' within the LV-CAP™ software platform and can be used for Dynamic Thermal Rating (DTR) of LV transformers to release additional capacity from existing LV network assets.
LV Network Meshing	This section details the hardware to be installed and the associated software to enable meshing of individual feeders between two LV substations. This has the potential to release additional capacity from existing LV network assets.
Load profile predictor application	A load profile predictor application is required to utilise historical load on both the transformer and specific LV feeder and predict the likely load profile for the future.

Group	Description
CSV data recorder application	Storage of all data captured by the system, information generated by any applications and a record of any actions implemented are required to be stored on non-volatile memory within the ISD.
LoadSense	Loadsense is an application designed to respond to outputs from Weathersense relating to real time and predicted network loading. These outputs may trigger an immediate (relatively) response or be a prediction alert, effectively stating that "unless network load is less than predicted, the transformer is going to exceed its RTTR rating in x-hours." Loadsense would schedule a LV network meshing to occur prior to that time slot, if the networks to be connected also have sufficient capacity.
Communications	This section covers the applications relating to communications to and from the deployed trial devices. Communications are required between deployed devices and separate, centralised data servers and between individual LV-CAP™ platforms.
Centralised System Requirements	Centralised systems are required to enable 'Apps' to be deployed on the intelligent substation devices and to store data that is required for the assessment of Project trials.
Overall System Requirements	This group covers all other requirements not captured under the groups defined above.

The numbering system used to identify individual requirements within Section 3 maintains traceability of the individual requirements identified. The numbering system is as follows:

- I:XXX: Provides relevant information regarding the overall solution;
- M:XXX: Outlines a 'must have' requirement, these are requirements that are needed to ensure the success of the Project;
- S:XXX: Outlines a requirement that 'should' be provided, if possible, but project success does not rely on it;
- C:XXX: Outlines a requirement that 'could' be provided, but does not affect anything else in the project;
- W:XXX: Outlines requirements that 'will not' be delivered at the current time, but could be delivered at a later date.

3 OpenLV Solution Requirements

EA Technology undertook an InnovateUK Energy Catalyst project with the University of Manchester and Nortech Management Ltd to develop a Common Application Framework for LV Network Management. The core software platform developed under the project is called Low Voltage Common Application Platform (LV-CAP™) and consists of a number of Docker containers communicating with each other using MQTT.

This section outlines the functional requirements and necessary interfaces with 3rd party equipment and software for LV-CAP™. The OpenLV Project will deploy 80 LV-CAP™ devices to test the platform as outlined above.

The final 'product' delivered will be referred to as the OpenLV Solution, acknowledging that the specific combination of hardware and software that results from this requirements specification will not be implemented again outside of the OpenLV Project, although the requirements will likely be utilised as a base of core functionality for future deployments.

3.1 Overall System Requirements

The purpose of the OpenLV Project is to test the LV-CAP™ platform's capabilities and demonstrate its ability to provide benefits to the industry, communities and third-party companies. Therefore, the trial system deployed by the project must demonstrate that it meets the below requirements.

- gather voltage and current data relating to the LV network;
- store this data for a period of time suitable to meet the requirements of the applications running on the platform;
- make this data available to multiple applications running on the platform;
- enable multiple applications to process and manipulate this data;
- transmit the pertinent, generated information back to central location(s), without requiring the raw data to be transmitted as well;
- be able to transmit the raw data to a data repository if required;
- enable deployed applications to make decisions regarding the LV network, based on the monitored data;
- implement those decisions autonomously if deemed appropriate for the network based on decision processes agreed with WPD;
- Make data and information, not considered operationally sensitive by the DNOs, available to communities, academia and third-party businesses.

In order to achieve these above requirements, a number of applications are necessary with specific, additional requirements detailed in the Annexes to this document.

3.2 Intelligent Substation Devices (ISDs)

3.2.1 Overall Statement

I:001. The Intelligent Substation device is an enclosure, containing a ruggedised PC capable of being installed in harsh environments and interface connections to receive data from external sensors.

I:002. Applications installed within the LV-CAP™ Software Platform gather data, store it locally, process if necessary and send requested information back to central servers.

I:003. The ISDs are responsible for providing the LV-CAP™ platform and associated software containers with a computing environment suitable for implementation for the duration of the OpenLV Project.

I:004. The Operating System for LV-CAP is GNU/Linux. The LV-CAP™ core software will be installed on the operating system along with the following containers, designed to deliver the required OpenLV solution functionality.

- LV-CAP™ Software Platform
- LV Monitoring
- Temperature Sensing
- Data Upload Application
- Dynamic Thermal Ratings Application
- Load Profile Predictor
- LoadSense
- ALVIN Reclose™ Interface Application
- CSV Data Recorder
- Management Comms Application
- Peer to Peer Communications Application

I:005. The requirements for each element of the ISDs are detailed later in this section.

3.2.2 Services required

I:006. The services detailed here refer to those required to be provided by the ISD platform as a whole, rather than individual elements, either hardware or software, within the platform.

Processing capability

M:001. The computational hardware must be based upon an industrial PC architecture.

M:002. The computational hardware within the ISD must be capable of running LV-CAP™, the additional application containers detailed below and up to three others to be written by community groups and / or third-party companies.

I:007. Applications to be developed by community groups or third parties will be subject to ‘reasonable complexity’ restrictions, as determined solely by the LV-CAP™ team at EA Technology to ensure the hardware is capable of delivering all project requirements.

I:008. The system within each enclosure must be capable of ensuring that all applications defined in this document can operate simultaneously in order to meet their individual requirements.

Communications

M:003. The ISD must have a modem / router installed capable of providing the below functionality.

I:009. The ISD must be capable of communication, both incoming and outgoing, with a control server based on Nortech's iHost platform located at EA Technology's Capenhurst offices.

I:0010. The ISD must be capable of outgoing communications (i.e. initiated without external request) for the transmission of data and information to the iHost platform located at EA Technology's Capenhurst offices.

I:0011. The ISD must be capable of outgoing communications (i.e. initiated without external request) for the transmission of data and information to a separate cloud based data storage server owned and operated by Lucy Electric.

I:0012. The ISD must be capable of communication, both incoming and outgoing, with other ISD's located in a local, (geographically similar) location to transfer data between the two platforms in support of the network automation functionality.

I:0013. The ISD must be capable of receiving data pertaining to the monitoring of the LV network from monitoring hardware installed in the same substation as the ISD.

I:0014. The ISD must be capable of receiving data pertaining to the temperature of the associated transformer and ambient air temperature.

Storage Capacity

M:004. The internal, non-volatile storage of the ISD must be of sufficient capacity to store all data captured by the sensors, generated by installed application containers and received from other ISDs, for the duration of the OpenLV Project (minimum period of 18 months).

Security (Physical)

M:005. The enclosure must be physically secured from unauthorised access. It cannot be assumed that the trial equipment will be always installed at indoor, secured substations.

Environmental

I:0015. The ISDs will be installed in a mixture of indoor and outdoor substations and the enclosure must be suitable for use in either instance.

M:006. The enclosure therefore, must be suitably IP rated to adequately protect the internal equipment in all potential substation environments and installation methods.

Protection

M:007. In the situation where network meshing hardware (ALVIN Reclose™) is installed and connected, it must be possible for a maintenance engineer to isolate the LV-CAP™ platform from the Reclose™ devices enabling manual operation of the network meshing capability.

M:008. This communication isolation must be capable of being 'locked' in a given state, to ensure automated operation cannot resume unexpectedly.

M:009. The enclosure must be non-conductive to avoid potential earthing issues.

Watchdogs & reset capability

I:0016. It is essential that the requirement for manual (in person) resets in the event of loss of communications or loss or responsiveness is avoided wherever possible due to the range of locations in which the trial equipment is to be installed.

M:0010. It must be possible to remotely reset the platform without requiring physical access to the ISD through establishing remote access to the router to trigger a reset of the computational hardware.

I:0017. The ISD should have appropriate 'Watchdogs' to ensure the individual devices within the overall ISD reset automatically if necessary.

W:001. There should be a Watchdog to ensure the router / model is reset if it enters a non-responsive state.

S:001. There should be a Watchdog to ensure the computational hardware is reset if it appears to have entered a non-responsive state, as indicated by a lack of network communication within the ISD.

3.2.3 Mounting arrangements

M:0011. The enclosure must be capable of multiple mounting options, for example:

- direct wall mounting;
- magnetic mounting bracket on the side of switchgear equipment; or
- bolting in place on the floor.

I:0018. Individual arrangements on site will determine which approach shall be used.

3.2.4 Products required

M:0012. The ISD must include an enclosure for the necessary hardware (industrial PC, modem and ancillary connection elements for monitoring devices.

I:0019. The ISD must include an industrial, ruggedised PC capable of running the LV-CAP™ platform and associated application containers.

I:0020. The ISD must include LV monitoring equipment for the gathering of information about the associated LV network.

I:0021. The ISD must include a modem enabled for two-way data transmission, capable of communicating over multiple mobile networks.

M:0013. The PC hardware must be installed with the LV-CAP™ platform.

3.2.5 Dependencies

I:0022. The system must be capable of installation within electrical safety standards required by WPD for deployment of LV monitoring equipment on their network, specifically:

- Standard Technique: SP2KD

3.2.6 Performance measurement

I:0023. The ISDs must enable the LV-CAP™ platform, including subsidiary application containers and associated connected hardware, to meet all requirements as defined in sections below.

I:0024. The ISDs must be capable of performing these functions for at least the duration of the OpenLV Project trials.

M:0014. The overall system must include the necessary monitoring equipment to gather necessary data for each software application.

M:0015. The overall system must be capable of communicating with physically separate network monitoring hardware providing voltage and load data.

M:0016. The overall system must be capable of measuring incoming signals from the temperature monitoring hardware.

M:0017. The overall system must be capable of communicating with ALVIN Reclose™ devices (x3) if they are installed in the substation.

I:0025. ALVIN Reclose™ devices will only be installed in 10 locations within the trials.

I:0026. Every ISD installed must be capable of communication with ALVIN Reclose™ devices as sites determined as suitable for their installation will not be identified until later in the project.

M:0018. The overall system must be capable of communicating with other ISDs via non-dedicated cable connection methods such as the use of a modem-to-modem connection.

I:0027. This communication link will be used to share information relating to the LoadSense application between adjacent, linked, substation LV-CAP™ platforms.

3.3 LV-CAP™ Software Platform

3.3.1 Overall statement

I:0028. The purpose of the OpenLV Project is to demonstrate the LV-CAP™ platform is a capable of operating as a non-specific distributed intelligence platform for the LV network.

I:0029. The LV-CAP™ software platform runs on the ISDs. This is an operating system that enables multiple Applications to be installed in software containers on a single device.

I:0030. The software also provides Apps with access to data provided from the sensors which are installed in each LV substation.

All applications developed for and deployed to the LV-CAP™ platforms must conform to the LV-CAP™ API document (

I:0031. Appendix A – LV-CAP™ API).

I:0032. The exceptions to 0, are below and will not be implemented as part of the OpenLV Project:

W:002. Individual message signing (see section 8.1.2 for further information).

W:003. Signing of Docker Image files.

W:004. Only one instance of each Application will be run (see section 4.2) on LV-CAP.

As a result, Applications may continue to use legacy GUID identifiers.

W:005. To simplify TLS implementation, TLS keys and certificates will be built into Docker Image files. The end date of TLS certificates should be set beyond the end of the OpenLV project trials in September 2019. TLS implementation is mandatory.

W:006. The Priority feature of the data storage APIs will not be implemented, with all queries returning messages of all priorities. Applications are free to output Priority data, but it will not be parsed yet. Similarly, requests may be made with Priority key values, but the key will be ignored.

3.3.2 Services required

I:0033. The ISD requires a number of 'services' to be provided by the LV-CAP™ platform. For example, these include monitoring of the LV network, storing the associated data and making predictions based on the data gathered.

I:0034. Each service required by the ISD is provided by an individual, specific software application.

I:0035. Rather than providing directly measurable outputs, the LV-CAP™ platform enables the operation of the various applications, and makes the data gathered and generated available.

I:0036. The LV-CAP™ platform will be subjected to a cyber-security assessment, including penetration testing, architecture evaluation and code review.

I:0037. The LV-CAP™ platform must demonstrate an appropriate level of security within the system. This will be informed by the cyber-security review to be undertaken by NCC Group.

M:0019. The LV-CAP™ platform must ensure communications between applications and the message broker are encrypted and authenticated to prevent application impersonation.

3.3.3 Products required

Hardware environment

I:0038. The OpenLV project hardware consists of an industrial PC based around a dual-core Intel Core i3 processor with 8GB of RAM and a 512GB SSD.

I:0039. This PC provides the processing power and storage for the whole LV-CAP solution.

I:0040. It has two Ethernet ports for network communications, one of which is utilised to connect a stand-alone 4G router which provides wide area network communications.

Base operating system

I:0041. The ruggedised PC within the ISD will be running 64-bit Ubuntu Server 16.04 LTS with current updates applied.

3.3.4 Dependencies

I:0042. The LV-CAP™ software platform, as deployed within the OpenLV project is managed and controlled via an instance of Nortech's iHost server.

I:0043. The LV-CAP™ platform deployed within the OpenLV Project requires PC hardware, with an installed operating system, as defined above.

3.3.5 Performance measurement

M:0020. The LV-CAP™ software platform must be demonstrated to run all software applications deployed to the platform if those applications conform to the API documentation provided.

3.4 LV Monitoring Equipment

3.4.1 Overall statement

I:0044. LV network monitoring provides the core data to be utilised by all Apps that will be deployed on the Intelligent Substation Devices.

I:0045. The ISDs must have connected monitoring equipment for the collection of data pertaining to the LV Network.

I:0046. The equipment must provide the ISD with voltage and current measurements at sufficient resolution and granularity to enable the effective operation of each application on the platform to meet the requirements specified in this document.

3.4.2 Services required

M:0021. The complete ISD system must be capable of measuring the following from appropriate sensor hardware for all phases.

Voltage Measurements

I:0047. RMS Voltage phase to neutral (x3) at the substation busbars.

Current Measurements

I:0048. For each circuit measured:

- RMS current in each phase
- Power factor for each phase
- Real and Reactive power flow each phase (including direction, so reverse power is read as negative current)

Temperature Measurements

I:0049. Outdoor ambient air temperature must be measured.

I:0050. Indoor ambient air temperature (indoor substations) must be measured. In multiple room substations, this will be in the transformer chamber.

I:0051. Transformer top oil temperature (or as close an approximation as can be managed).

3.4.3 Products required

I:0052. Lucy Electric GridKey's MCU520 system must be utilised as the monitoring platform.

I:0053. 6x Flexible Rogowski Coils or Current Transformers, compatible with the GridKey MCU520 system must be provided, to monitor the total transformer load and the specific feeder connecting to an adjacent substation.

I:0054. Modified fuse carriers are the preferred method for connecting the GridKey platform to LV network.

I:0055. 3x modified fuse carriers are required for substations where there is sufficient capacity, (i.e. at least one empty fuse holder per phase), within the fuse board.

I:0056. G-Clamps are required for connection of the GridKey platform to the substation neutral busbar.

I:0057. In the event that sufficient capacity on the fuse board is not available for all phase connections, (I:0055), then G-Clamps should be used as with I:0056.

I:0058. An application container (GridKey Sensor Container) is required to provide the interface capabilities between the MCU520 and the LV-CAP™ platform.

I:0059. The GridKey Sensor Container will communicate directly with the GridKey MCU520 via a local Ethernet port.

W:007. The GridKey Sensor Container will not have access to the wide area communications network.

I:0060. The requirements for this application, enabling communication between the LV-CAP™ platform and Lucy Electric GridKey's MCU520 platform are detailed in Appendix C – Lucy Electric Application Container.

3.4.4 Dependencies

I:0061. For the application to function, an MCU520 must be procured from Lucy Electric and connected to the ISD hardware via the ethernet port.

3.4.5 Performance measurement

M:0022. The LV-CAP™ platform must be provided with timestamped readings of voltage and current readings from the GridKey Sensor Application.

S:002. It is desirable that synchronous sampling is implemented to aid in analysis of the gathered data.

I:0062. These readings must be provided at a frequency of once per minute sufficient to meet the requirements of other applications running on the platform.

M:0023. It must be demonstrated that data readings from the GridKey Sensor Application can be acquired at a rate of once every minute for an indefinite period.

M:0024. It is essential that the system demonstrate the capability of continuous data capture at a rate of once every ten (10) seconds for a period of at least one hour.

3.5 Temperature Sensing

3.5.1 Overall statement

I:0063. Temperature sensors are required to monitor the temperature of the LV transformer and ambient air temperature. This data is provided to the relevant container 'Apps' within the LV-CAP™ software platform and can be used for Dynamic Thermal Rating (DTR) of LV transformers to release additional capacity from existing LV network assets.

I:0064. It is necessary for the operation of the DTR application that the ambient temperature and specific temperatures relating to the transformer are collected and made available.

3.5.2 Services required

M:0025. The ISD must have the means to collect thermal readings as defined in I:0064, receive and store this data in a format readable by other applications.

3.5.3 Products required

I:0065. This specification requires, at a minimum:

- physical means for detecting the ambient temperature and specific transformer temperatures;
- software compatible with the LV-CAP™ platform for receiving and managing the data from these sensors.

I:0066. Therefore, the ISDs must be equipped with the necessary thermocouples to monitor the range of temperatures required by the DTR application.

I:0067. The ISDs must be equipped with the necessary interface equipment to connect the temperature monitoring equipment to the LV-CAP™ hardware.

M:0026. The temperature sensing application container must take the values provided by the thermocouple(s) and pass them to the LV-CAP™ system for storage in non-volatile memory.

I:0068. The temperature readings must be recorded at a rate of once every minute for an indefinite period.

3.5.4 Dependencies

I:0069. For the data to be provided, the application requires an appropriately sensitive thermocouple to be connected to the ISD via a suitable data port.

3.5.5 Performance measurement

M:0027. The LV-CAP™ platform must be provided with timestamped temperature readings from the Temperature sensing application at a rate of once per minute.

I:0070. These readings must be provided at one-minute intervals throughout the duration of the OpenLV Project.

I:0071. It is noted that future business-as-usual deployments may require the ability to vary the rate of data capture.

S:003. Therefore, it is desirable to demonstrate that the rate of data capture can be varied between 10-second and 10-minute intervals, in 10-second stages.

3.6 LV Network Meshing

3.6.1 Overall statement

I:0072. This section details the hardware to be installed and the associated software to enable meshing of individual feeders between two LV substations. This has the potential to release additional capacity from existing LV network assets.

I:0073. The OpenLV Project must demonstrate that autonomous control of network assets, based on pre-defined logic, is possible via a distributed intelligence platform (the ISDs).

I:0074. Within the OpenLV Project, this is to be demonstrated through direct control of ALVIN Reclose™ devices to mesh and de-mesh adjacent LV networks.

3.6.2 Services required

M:0028. The LV Network Meshing Application must enable communication capabilities between the LV-CAP™ platform and ALVIN Reclose™.

M:0029. The application must read the desired information from the ALVIN Reclose™ devices, and pass it to the LV-CAP™ platform for storage in non-volatile memory through the CSV data recorder.

- MIR_BUS_VOLTAGE_RMS
- MIR_CABLE_VOLTAGE_RMS
- MIR_LINK_CURRENT_RMS
- MIR_OPEN_OPERATIONS
- MIR_CLOSE_OPERATIONS
- MIR_WATCHDOG_FAULTS_DETECTED
- MIR_CHIP_TEMPERATURE
- MIR_REACTIVE_POWER
- MIR_ACTIVE_POWER
- MIR_UPTIME_HIGH
- MIR_SWITCH_TEMPERATURE
- MHR_SHADOW_FAULT_STATUS

M:0030. It must be demonstrated that data readings from the LV Network Meshing Application can be acquired at a rate of once every minute for a period of at least one hour.

I:0075. The variable MHR_SHADOW_FAULT_STATUS reads the current state of the circuit breaker within the connected ALVIN Reclose™ devices.

I:0076. LV Network Meshing Application must be able to trigger an opening or closing of the ALVIN Reclose™ device's circuit breaker, meshing, or de-meshing the network as applicable.

I:0077. For the purposes of the OpenLV Project trials, it is preferred that a record of the process is stored for project evaluation.

M:0031. The application must therefore store a record of reacting to a command, whether to initiate or break a network mesh, in non-volatile memory.

M:0032. The application must also store a record of the state of the connected ALVIN Reclose™ devices both before and after implementing of the command, i.e. open / closed.

I:0078. If there are no attached circuit breakers, an appropriate 'error code' must be provided instead.

M:0033. All control communications, whether acknowledged by a connected ALVIN Reclose™ device or not, must be stored in memory on the LV-CAP™ platform for later analysis if required.

M:0034. The ISD must be electrically isolated from ALVIN Reclose™ devices installed within the substation.

3.6.3 Products required

I:0079. This specification requires, at a minimum, provision of an application capable of communicating with ALVIN Reclose™ devices to deliver the above requirements

I:0080. The ALVIN Reclose™ devices will be procured by the OpenLV Project.

I:0081. The interconnection cable to interface the ISD with the ALVIN Reclose™ devices will be provided by EA Technology's LV Solutions team, in collaboration with EA Technology's HV59s team.

3.6.4 Dependencies

I:0082. The application requires a control input from another application (LoadSense) to determine whether to open or close attached circuit breakers.

M:0035. The LV Network Meshing Application must only respond to instructions to mesh or de-mesh the network through opening and closing of ALVIN Reclose™ circuit breakers from the LoadSense application.

3.6.5 Performance measurement

M:0036. The ALVIN Reclose™ devices must respond to an instruction to initiate an open or close operation.

M:0037. It must be demonstrated that the control signals for transmission to the ALVIN Reclose™ devices from the LV Network Meshing application are triggered whether an ALVIN Reclose™ devices is installed within the substation or not.

3.7 Load Profile Predictor Container 'App'

3.7.1 Overall statement

I:0083. A load profile predictor application is required to utilise historical load on both the transformer and specific LV feeder and predict the likely load profile for the future.

3.7.2 Services required

I:0084. This application must utilise historical load values to generate a forecast of future load on the transformer and individually monitored LV feeder.

I:0085. It must not utilise all the data available on the trial platform as future systems will not have access to 'unlimited' historical data due to local storage limitations.

I:0086. The duration of historical data utilised by the application should be confirmed with explanation of why that duration has been selected.

3.7.3 Products required

I:0087. This specification requires, at a minimum, an application that utilises the historical load data to create a predictive forecast for the network and asset in question.

3.7.4 Dependencies

I:0088. In order to predict future load profiles, the Load Profile Predictor application requires the historical data gathered by the GridKey MCU520 and stored in non-volatile memory.

3.7.5 Performance measurement

I:0089. This application must utilise a sufficient period of historical data to provide sufficient predictive assurance for the calculated outputs.

M:0038. The Load Profile Predictor application must output a load forecast at half-hourly intervals for the next 24-hour period.

3.8 CSV Data Recorder Application

3.8.1 Overall statement

I:0090. Storage of all data captured by the system, information generated by any applications and a record of any actions implemented are required to be stored on non-volatile memory within the ISD.

3.8.2 Services required

M:0039. This application must store all data output by each application container on the platform.

M:0040. All data must be timestamped such that raw data, and processed information derived from that data can be reconstructed at a later date if required.

M:0041. All data must be attributable to the application that created and published it.

3.8.3 Products required

I:0091. This specification requires, at a minimum, an application that monitors all communications traffic within the LV-CAP™ platform and stores it with a timestamp, and provides a record of which application published that item of data.

3.8.4 Dependencies

I:0092. This application requires other applications to be running on the LV-CAP™ platform to provide data and processed information for storage.

I:0093. The application must be granted sufficient authorisations within the platform to enable access to all data and information for storage.

3.8.5 Performance measurement

M:0042. It must be demonstrated that accurate data values are stored in non-volatile memory for each application on the LV-CAP™ platform that is providing measured or calculated data.

M:0043. This data must be stored at a frequency that matches the outputs of the individual applications.

3.9 LoadSense Container 'App'

3.9.1 Overall statement

I:0094. Loadsense is an application designed to respond to outputs from Weathersense relating to real time and predicted network loading.

I:0095. These outputs will trigger an immediate response to outputs from the Dynamic Thermal Rating application.

I:0096. The LoadSense application implements network meshing through the ALVIN Reclose™ devices and associated LV Network Meshing application.

3.9.2 Services required

I:0097. At present, the operational characteristics of the LoadSense application have not been agreed with WPD; consequently, this section will be updated in the future once the requirements have been determined.

3.9.3 Products required

M:0044. This specification requires, at a minimum:

- Provision of an application capable of utilising the outputs from the applications listed below in combination with decision processes agreed with WPD to determine if initiating a network meshing event is appropriate; and
- If such an event is required, the application must instruct the ALVIN Reclose™ interface application to commence network meshing procedures.

M:0045. The application must also determine when it is appropriate to de-mesh the networks, again based on decision processes agreed with WPD, and instruct the ALVIN Reclose™ interface application accordingly.

3.9.4 Dependencies

I:0098. Input is required from the below applications:

- Load profile predictor
- WeatherSense
- Peer-to-Peer communications

3.9.5 Performance measurement

M:0046. The application must be demonstrated to arrive at the correct decision given specific inputs and initiate the appropriate action of the ALVIN Reclose™ interface application as a result.

I:0099. In the event that ALVIN Reclose™ devices are installed as part of the project trials, it is possible to determine the condition of the device (i.e. circuit open or closed) from the visual indicator on the front.

3.10 Dynamic Thermal Ratings Application

3.10.1 Overall statement

The Dynamic Thermal Ratings (DTR) Application utilises the current transformer temperature, along with the forecast load profile from the Load Profile Predictor Application, to determine the temperature of the transformer asset being monitored over the next 24 hours.

3.10.2 Services required

M:0047. The DTR Application must determine up-to-date thermal ratings for the associated transformer.

M:0048. Based on the forecast load profiles generated by the Load Profile Predictor Application the DTR Application must determine the forecast temperature profile for the transformer.

I:00100. In both instances, this information must be output to the main LV-CAP™ platform.

3.10.3 Products required

I:00101. This specification requires, at a minimum, provision of an application, compatible with the LV-CAP™ platform, containing transformer DTR algorithms.

3.10.4 Dependencies

I:00102. The DTR application requires load data, temperature data and load profile predictions from the LV-CAP™ platform in order to operate.

3.10.5 Performance measurement

M:0049. The DTR application must generate outputs once each subsequent predicted load profile is available, based on that profile.

M:0050. Therefore, the DTR application must output a thermal rating forecast at hourly intervals for the next 24-hour period.

3.11 Centralised Systems

3.11.1 Overall Statement

I:00103. Within the OpenLV Project there is a requirement for two 'centralised' systems to enable management of the trial platforms and delivery of the project requirements.

I:00104. The trial system utilises a Nortech iHost server to manage the deployed hardware and store the gathered and processed data.

I:00105. A second server, to be provided by Lucy Electric will store the data gathered by the platform, and processed data generated by applications deployed under Methods 2 and 3.

I:00106. In both cases, separate communication applications are required although both will utilise the 4G modem within the ISDs.

3.11.2 Security

I:00107. In all cases, both for the iHost based control server and the Lucy Electric cloud based server, security must be paramount in keeping with the OpenLV Project's Data Protection Strategy.

M:0051. User authentication via unique login and password must be enabled.

S:004. Two-factor authentication should be utilised wherever possible.

M:0052. Mutual authentication must occur for all communication between platforms.

I:00108. An independent cyber-security evaluation of the LV-CAP™ platform and associated control systems will be undertaken as part of the OpenLV Project.

I:00109. The system must implement any recommendations from this evaluation to ensure the safety of WPD's network assets.

3.11.3 Application deployment and management server (Nortech)

Services required

I:00110. The OpenLV Project's deployed ISDs require a central management and control system, this is provided by a Nortech iHost server.

M:0053. This system must be capable of deploying a new application container to a single device, a subset of devices, or all devices.

M:0054. This system must be capable of removing an application container from a device, a subset of devices, or all devices.

M:0055. This system must be capable of updating the application containers on a device, a subset of devices, or all devices.

M:0056. This system must be capable of changing configuration settings for any individual container on a specific device, a subset of devices, or all devices.

S:005. Identifying when a deployed platform, that has not been decommissioned, has not connected to the server for more than one (1) day, three (3) days and five (5) days, and trigger notification alerts in each instance.

M:0057. The Nortech Comms Application and the iHost server must mutually authenticate each other so that only authorised data uploads occur, and Man-in-the-Middle attacks are prevented. (This requirement is linked with I:00125.)

M:0058. Measures must be taken to ensure that the data uploaded remains confidential in transit, to comply with the OpenLV Project Data Protection Strategy. (This requirement is linked with I:00126.)

Products required

I:00111. Nortech's iHost server is utilised as the central command and control system for the LV-CAP™ platforms.

M:0059. The iHost server for the OpenLV Project must be installed behind a firewall to restrict unauthorised access as far as reasonably practicable.

Dependencies

I:00112. The server is installed at EA Technology and requires access to communications outside of the EA Technology corporate network to enable communications with the deployed LV-CAP™ platforms.

I:00113. The server requires each deployed platform to have a functional router modem and Nortech Communications application as defined in this document.

Performance measurement

M:0060. The iHost server must demonstrate the ability to receive all data uploaded from each connected LV-CAP™ platform.

M:0061. The platform must demonstrate the ability to deploy a new application container to a connected LV-CAP™ platform;

M:0062. The platform must demonstrate the ability to update an application container on a connected LV-CAP™ platform;

M:0063. The platform must demonstrate the ability to change configuration files for a software container on a connected LV-CAP™ platform;

M:0064. The platform must demonstrate the ability to remove an application container from a connected LV-CAP™ platform;

3.11.4 Cloud Based Hosted Platform (Lucy)

Services required

I:00114. The OpenLV Project's deployed ISDs require a public facing data management system to enable community groups and third-party companies access to network data, and outputs generated by their own applications.

M:0065. The Cloud Based Hosted Platform system must be capable of receiving data from the Data Upload Application installed on each LV-CAP™ enabled device deployed within the project.

M:0066. The Cloud Based Hosted Platform system must be capable of sharing this data with appropriate individuals via an API interface.

M:0067. The Cloud Based Hosted Platform system must be capable of sharing this data with appropriate individuals via a web-portal viewer interface.

M:0068. The (Lucy Electric) Data Upload Application and the associated Cloud Based Server must mutually authenticate each other so that only authorised data uploads occur, and Man-in-the-Middle attacks are prevented.

M:0069. Measures must be taken to ensure that the data uploaded remains confidential in transit, to comply with the OpenLV Project Data Protection Strategy.

Products required

M:0070. Lucy Electric to provide a separate, instance of their cloud based data server for use by the OpenLV Project.

Dependencies

I:00115. The server requires each deployed platform to have a functional router modem and the Lucy Electric Communications application as defined later in this document.

Performance measurement

I:00116. The platform must demonstrate the ability to receive a selected subset of data from each connected LV-CAP™ platform;

I:00117. The platform must demonstrate the ability to allow authorised individuals to access the information stored within the server, on a location (LV-CAP™ platform) basis.

3.12 Communications

3.12.1 Overall Statement

I:00118. The LV-CAP™ platform, as being deployed as part of the OpenLV Project, requires three separate communication applications, each to meet specific communication requirements for project delivery.

I:00119. It is necessary for each platform to have the capability to communicate with:

- The application deployment and management server;
- Cloud based, public facing data storage server; and
- Adjacent LV-CAP™ platforms for data sharing purposes.

3.12.2 Security

I:00120. In all cases, both for the iHost based control server and the Lucy Electric cloud based server, security must be paramount in keeping with the OpenLV Project's Data Protection Strategy.

I:00121. User authentication via unique login and password must be enabled.

I:00122. Mutual authentication must occur for all communication between platforms.

3.12.3 Management Comms Application

I:00123. The Nortech communications container is considered a core-element of the LV-CAP™ platform as Nortech's iHost server is utilised to manage and control all LV-CAP™ platform's (ISD's) deployed within the OpenLV Project.

Services required

M:0071. The application container must facilitate two-way communication between the LV-CAP™ platform and the iHost server.

I:00124. This must enable transfer of all desired data from the platform back to the iHost server. This data may be all monitored and calculated values or a selected subset thereof. In either case, the container must be capable of transferring the desired data.

M:0072. The data from each LV-CAP™ system running the Nortech Comms Application must be uploaded as a separate RTU (or multiple virtual RTUs) within the iHost server.

M:0073. Once successfully uploaded to the iHost server, data must be marked as 'uploaded' within the LV-CAP™ platform to prevent retransmission.

M:0074. The application must receive and implement new application containers for installation onto the LV-CAP™ platform.

M:0075. The application must receive and implement configuration files for the installed applications.

M:0076. The application must receive and implement instructions to remove application containers from the LV-CAP™ platform.

M:0077. The application must be capable of managing loss of communications during file upload and download, resuming once communications are restored.

I:00125. The Nortech Comms Application and the iHost server must mutually authenticate each other so that only authorised data uploads occur, and Man-in-the-Middle attacks are prevented. (This is linked with S:005.)

I:00126. Measures must be taken to ensure that the data uploaded remains confidential in transit, to comply with the OpenLV Project Data Protection Strategy. (This is linked with M:0058.)

I:00127. The volume of mobile data transferred must be managed to reduce the operating costs of the OpenLV system.

I:00128. The application must be configured via the standard LV-CAP™ configuration mechanism (see sections 8.2.1 and 9.5 of the LV-CAP™ API). The configuration is likely to be altered in the course of the OpenLV Trials, so the configuration settings available must be documented alongside the Application.

I:00129. The configuration is expected to cover the following areas:

- iHost server settings (included where to send the data, and authentication settings).
- Data Selection settings, i.e. which topics are to be uploaded to the iHost server.
- (Optionally) Where data is to be placed in the iHost structure.

I:00130. The requirements document provided to Nortech for this application container is located in Appendix B – Nortech Application Container.

I:00131. As part of the OpenLV Project, a Cyber-Security review of the LV-CAP™ platform and Applications deployed within the project is to be undertaken. The Cyber-Security supplier will be undertaking an audit of the LV-CAP™ platform and it should be expected that this will include an audit of the software Application and associated documentation created by Nortech as part of the project.

Products required

I:00132. This specification requires, at a minimum, a software container to manage the communication link between the LV-CAP™ platforms and the central iHost server in line with the required services above.

Dependencies

I:00133. The application will require access to:

- the connected router modem;
- the data stored on the platform.

Performance measurement

M:0078. The application must enable communications between an individual LV-CAP™ platform and the iHost command and control server.

I:00134. The application must demonstrate transfer of all data stored on the platform to the server.

I:00135. The application must demonstrate receipt and installation of a new application container.

I:00136. The application must demonstrate receipt and application of a revised configuration set for an application container.

I:00137. The application must demonstrate removal of an application container.

I:00138. The application must demonstrate ability to resume a download when communications are restored.

3.12.4 Data Upload Application

Services required

M:0079. This must enable transfer of all desired data from the platform back to Lucy Electric's cloud based data centre. This data may be all monitored and calculated values or a selected subset thereof. In either case, the container must be capable of transferring the desired data.

M:0080. The data from each LV-CAP system must be uploaded as a separate RTU (or multiple virtual RTUs) within the server.

M:0081. Once successfully uploaded to the server, data must be marked as 'uploaded' to prevent retransmission.

I:00139. The application must be capable of being configured via instructions received from the iHost platform control server.

M:0082. The GridKey Upload Container and the GridKey Data Centre must mutually authenticate each other so that only authorised data uploads occur, and Man-in-the-Middle attacks are prevented.

M:0083. Measures must be taken to ensure that the data uploaded remains confidential in transit, to comply with the OpenLV Project Data Protection Strategy.

I:00140. The requirements for the GridKey Data Upload container, to be provided by Lucy Electric are defined in a separate document located in Appendix C – Lucy Electric Application Container.

I:00141. As part of the OpenLV Project, a Cyber-Security review of the LV-CAP™ platform and containers deployed within the project is to be undertaken. The Cyber-Security supplier will be undertaking an audit of the LV-CAP™ platform and it should be expected that this will include an audit of the software container and associated documentation created by Lucy Electric as part of the project.

Products required

I:00142. This specification requires, at a minimum, a software container to manage the communication link between the LV-CAP™ platforms and the GridKey Data Centre in line with the required services above.

Dependencies

I:00143. The application will require access to:

- the connected router modem;
- the data stored on the platform.

Performance measurement

M:0084. The application must enable communications between an individual LV-CAP™ platform and the GridKey Data Centre.

M:0085. The application must demonstrate transfer of selected data stored on the platform to the server.

M:0086. The application must demonstrate ability to resume a download when communications are restored.

3.12.5 Peer to Peer Comms Application

Services required

I:00144. The LoadSense application in each ISD requires data relating to the status and operation of the linked transformer in order to ensure safe and effective operation.

I:00145. This data must include voltage and current, and the outputs from the load predictor and WeatherSense applications.

M:0087. This application must enable the transfer of the necessary data between the linked, adjacent devices.

M:0088. The data to be transferred between devices may change over the course of the project and consequently, the data must be configurable.

Products required

I:00146. This specification requires, at a minimum, an application to enable the transfer of information to allow the decision of whether to initiate network meshing. It must also be able to respond to equivalent requests.

Dependencies

I:00147. The application will require access to:

- the connected router modem;
- the data stored on the platform.

Performance measurement

I:00148. The Peer-to-Peer Communications application must demonstrate the ability to send and received the configured datasets with the assigned ‘partner’ LV-CAP™ platform.

3.13 Overall System

3.13.1 Overall Statement

I:00149. A magnetic mounting arrangement is preferred by the client DNO with wall or floor mounting acceptable as an alternative.

M:0089. Therefore, the enclosure must be capable of multiple mounting arrangements, including magnetic attachments, wall mounting bolts or ground placement.

3.13.2 Loss of Power

S:006. In the event of loss of power, the platform must, on completion of a successful reboot, determine from data logs how long it was offline, and consequently how much data has been lost.

This information must be stored within the system log files and must include:

- ISD serial number;
- Location;
- Time of last successful data record;
- Time of successful system restoration.

A notification should be issued to the OpenLV Project team, either via the iHost server or direct notification such as an e-mail.

M:0090. The ISD must be capable of self-restoration following a loss of power during boot-up-sequence.

S:007. The ISD is ideally required to withstand up to three loss-of power events within a period of five (5) minutes without suffering unrecoverable errors.

S:008. The ISD should respond appropriately to a loss of power during download of software or configuration updates, ensuring that the download is resumed / restarted and completed once the system is running.

W:008. The ISD must respond appropriately to a loss of power during an update procedure to the LV-CAP™ platform.

I:00150. The system should complete the process with the update / setting changes applied.

3.13.3 Controlled Access

M:0091. The ISD enclosure must be capable of being securely locked with a padlock.

I:00151. WPD will provide padlocks to restrict access only to those staff competent and authorised for LV Switching operations.

S:009. Access to the system software through direct ethernet connection must be restricted through methods such as digital signing, communication encryption and require a password to access the device.

3.13.4 Network deployment

M:0092. The cable connections (power, thermocouple and communications) must be suitable for implementation on WPD's network.

4 Appendix A – LV-CAP™ API

LV Common Application Platform Public API

UNRESTRICTED

Product: LV-CAP
Drawing: 2383-MANUL-V04.03.00
Date: July 2017

Version History

Date	Version	Author(s)	Notes
2015/11/06	1	James Slater/Siôn Hughes	First version of Third Party Developer API document
2015/11/12	2	James Slater/Siôn Hughes	Amendments made to first release of document
2017/07/26	04.03.00	Richard Ash, James Slater	Re-organise to reduce duplication, incorporate changes to meet 2362-RQSPC. Add MQTT security information and error reporting API. Updates from meeting with Nortech. Change Application identifiers and terminology, add concept of Priority for messages (future proofing). New release for OpenLV project.

EA Technology Limited, Capenhurst Technology Park, Capenhurst, Chester, CH1 6ES;
Tel: 0151 339 4181 Fax: 0151 347 2404
http://www.eatechnology.com
Registered in England number 2566313

Contents

Figures

Tables

1. Introduction

The Common Application Platform for LV Networks (LV-CAP) is a software environment which facilitates the implementation of the Smart Grid at the lower distribution voltages. To drive down the cost of deploying Smart Interventions, the platform allows multiple algorithms to be deployed to one set of measurement and data processing hardware. The platform allows these algorithms to be designed and produced by independent third-party developers and packaged as stand-alone Applications which can be easily deployed by the distribution network operator without requiring bespoke software development.

This document details the Application Programming Interface (API) for developers intending to write Applications to run on LV-CAP. LV-CAP uses Docker to overcome dependency problems for third party developers, and helps to maintain and manage containers. It uses a MQTT messaging system for the communication of running containers and has a data storage functionality to persist data. This document has details on how a third-party Application can be set-up, run and interact with the core services on the platform.

2. Glossary

Term	Description
ACL	Access Control List, a list of the resources which a specific client may access. Used to control access to topics on the **MQTT broker**.
API	Application Programming Interface – a set of defined interfaces to be used by application developers.
APID	See **Application ID**.
Application	A **Docker Container** suitable for use with LV-CAP in accordance with this API document. All LV-CAP **Applications** are Docker Containers, but not all Docker Containers are suitable for use as LV-CAP Applications.
Application ID	The unique identifier for a specific version of an Application, by combining the **Vendor, Application Name** and **Application Version**. See Section 4.2.
Application Name	This is a string which identifies an **Application**. This is chosen at will by the Application developer. See Section 4.2.
Application Version	A string which indicates the version of **Application** in a **Docker Image**. Decimal points may be used to separate version numbers, e.g. 1.2.3. This is chosen by the Application developer. See Section 4.2.
BLOB	Binary Large OBject, a SQL database field which can store an arbitrary array of binary data.
Container Manager	The main process which controls all LV-CAP **Applications**.
Docker	Open source program that allows Linux applications and their dependencies to be packaged as a **Docker Image**.
Docker Container	An isolated environment in which a **Docker Image** is run under **Docker**. Multiple Docker Containers may be created from a single Docker Image and run simultaneously.

Term	Description
Docker Image	A file system image containing a packaged Linux program (with its dependencies) which can be deployed to run on a **Docker** system.
GUID/UUID	A Version 4 GUID/UUID is a universally unique 48-byte identifier which is generated using random numbers. Example:- 821b8e33-4eaa-480e-b205-30fa9572af1a
IID	See Instance ID
Instance	One running copy of an **Application**, which is separate from any other copy of the Application, and has its own independent configuration. See Section 4.2.
Instance ID	String identifying a specific **Instance** of an **Application**, which is unique only on a given LV-CAP system. See Section 4.2.
LV	Low Voltage. Used in this context to refer to the Low Voltage electricity distribution network which delivers power to domestic and commercial customers at 400/230V AC.
LWT	Last Will and Testament, in **MQTT** a message to be sent when to subscribers when a publisher disconnects unexpectedly.
MQTT	(Message Queue Telemetry Transport) a publish subscribe based lightweight messaging protocol, used on top of the TCP/IP protocol.
MQTT Broker	Process which is responsible for distributing messages to interested clients based on the topic of a message. The LV Common Application Platform runs a private instance of an MQTT Broker
MQTT Topic	Identifier within an **MQTT** message used by the broker to allow filtering and direction of messages. All messages are published to a topic, and clients receive them if they are subscribed to the topic.
Vendor	A string which identifies the developer of an Application. These are allocated by EA Technology to each party creating **Applications** to run on LV-CAP.

Table 1 - Glossary of Terms

3. Platform Overview

The LV 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 using the messaging API described in Section 8. This is shown in Figure 2.

A key piece of the provided framework is the Container Manager. The Container Manager has ultimate control over the entire system ensuring everything runs as expected. Apart from the Container Manager, all core services on the platform run as Docker containers which the Container Manager is responsible for starting, stopping and updating. All Applications are packaged within Docker containers which the Container Manager will again start, stop and manage. A Docker container contains a GNU/Linux application and all its library dependencies except the Linux kernel itself. This allows each Docker Image created to be portable, easily updated and independent.

The Container Manager utilises the functionality within Docker to limit and share resources of a running container. This control allows the Container Manager to manage platform resources, giving Applications their requested resources and preventing them from consuming excess resources and starving others of resource. Developers of third party Applications must be aware that their application cannot use the entire resources of the system and that it must share processor, RAM and storage with other Applications running on the system.

As well as managing the start-up and shutdown of Applications, the Container Manager is responsible for ensuring that updated configuration files are delivered to the relevant containers, and that updates to containers are applied. Finally, it checks that Applications are still running correctly, handling any errors returned from Applications and dealing with Applications that have ceased to operate correctly.

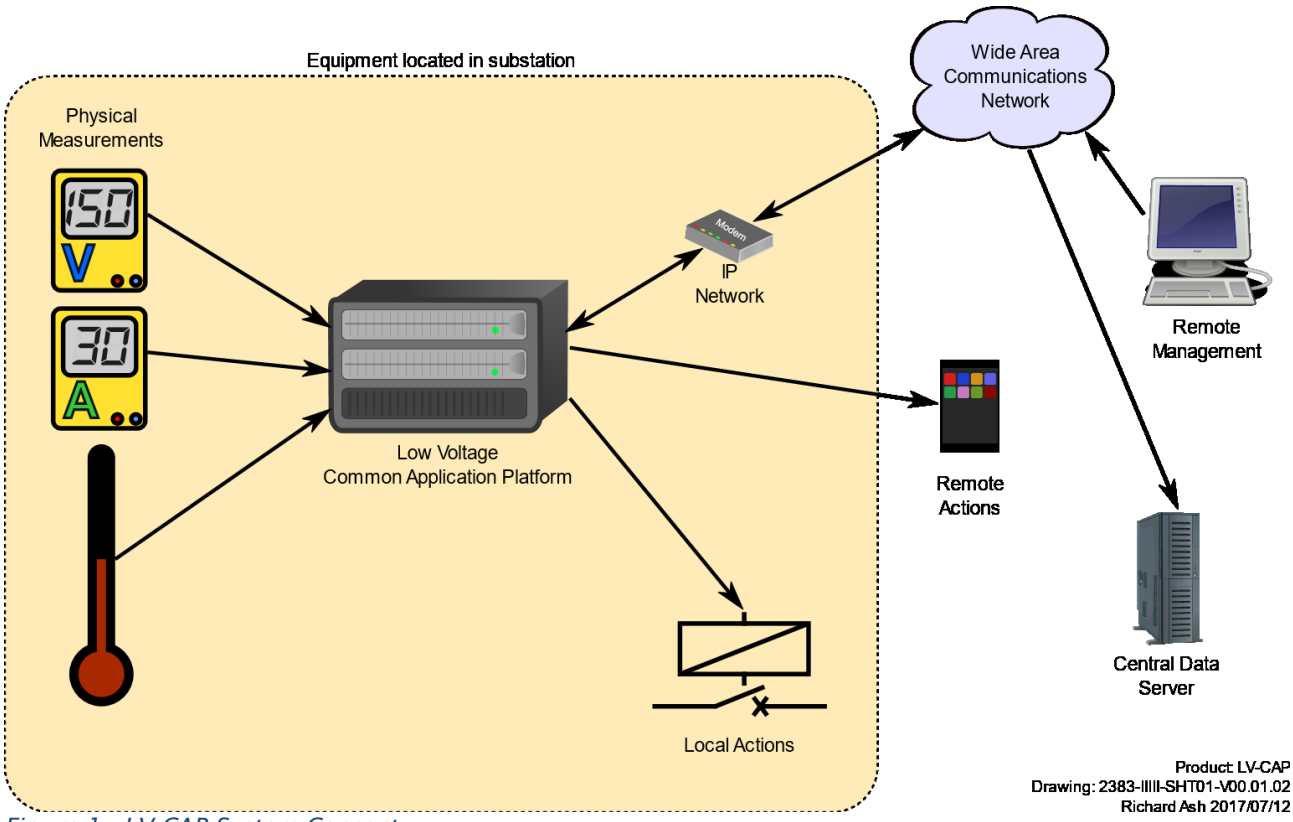

Figure 1 - LV-CAP System Concept

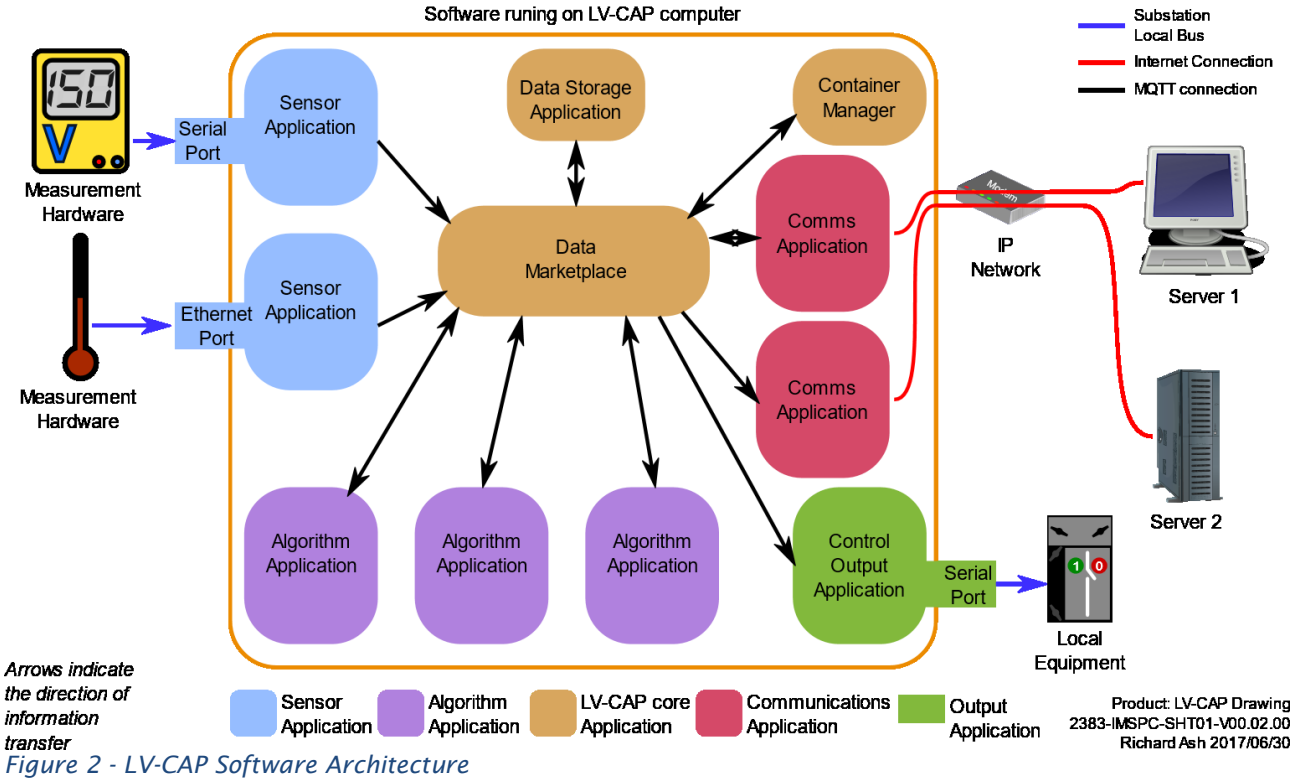

Figure 2 - LV-CAP Software Architecture

All communications between Applications in LV-CAP take place through the Data Marketplace (Figure 2). This uses Message Queue Telemetry Transport (MQTT) to transport messages. An MQTT broker, Mosquitto, is supplied as part of LV-CAP and is used by both core services and third-party containers. The message protocol for communicating on the MQTT broker and connection

settings to the broker are documented in Section 8 of this document. Access Control Lists (ACLs) are used on the MQTT broker to secure it, preventing Applications from publishing on and subscribing to topics they should not. The ACLs are automatically configured by LV-CAP when a new Application is added to the system.

LV-CAP systems are configured with an internal IP network. The Data Marketplace operates on this internal network and all Applications are automatically connected to it. Applications are only connected to external networks when there is a clear requirement for such a connection, and the system administrator has permitted it.

The Data Storage Application provides a database connected to the Data Marketplace. As well as being used by the Container Manager, it stores the outputs of Applications so that they can be subsequently retrieved for external communication or further processing.

Applications running on LV-CAP will generally fulfil one of four roles. Some Applications may fulfil more than one role at the same time.

1. Sensor Applications are responsible for reading data from physical sensor hardware. The data read is sanity checked and published to the Data Marketplace in a standard format. The data is then available to any other Application to subscribe to. The set of sensors provided for any given LV-CAP installation, and hence the Sensor Applications required, will vary depending on the user's requirements. The data format is independent of the measurement hardware so that different supplier's hardware can be used without software alterations outside the related Sensor Application.
2. Algorithm Applications consume data from one or more sensors and perform calculations upon it, for instance calculating the real-time temperature of a Transformer or forecasting the localised demand for energy. The Applications read from the Data Marketplace and publish their outputs back to the Data Marketplace.
3. Output Applications are the mirror image of Sensor Applications. They respond to information on the Data Marketplace (created by Algorithm Applications) by controlling physical hardware connected to the LV-CAP system, for instance carrying out network switching or energy storage.
4. Communications Applications connect the LV-CAP platform to the outside world. LV-CAP provides an IP communications link to the outside world, which Communications Applications use to upload and download data. A Communications Application uploads selected data values from the Data Marketplace to a central data server, or downloads Application images and configuration files from a central management server.

The default Communications Application is provided by Nortech Management Ltd. to communicate with their iHost server product.

4. General Principals

This section includes some principals which have driven the design and operation of the LV-CAP system. An understanding of these will make it easier to navigate and comprehend the rest of this specification.

4.1 Architecture

LV-CAP is designed to work as a loosely-coupled data processing pipeline, in which measurement data from Sensor Applications feeds into one or more Algorithm Applications. The outputs of these Algorithm Applications may feed other Algorithm Applications. Ultimately data reaches either a Communications Application to be sent to an external system, or an Output Application to take local actions on the Smart Grid (Figure 3).

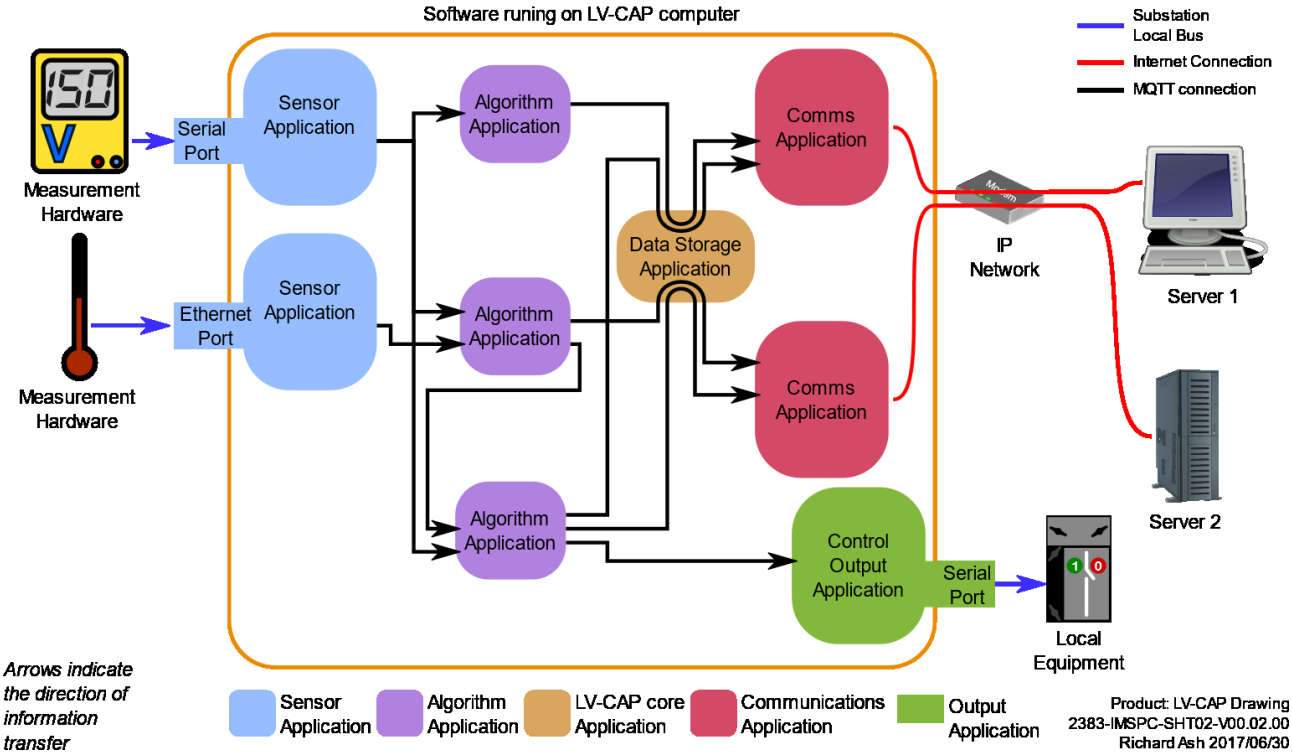

Figure 3 - Data Flow through an example LV-CAP system

Data is pushed through the pipeline from the Sensor Applications towards the outputs. The pipeline runs in approximately real time, although this is not enforced as in a true real-time system. If the workload of the LV-CAP system temporarily exceeds the available processing power then the system will lag behind before catching back up when resources allow.

The pipeline forms a tree structure, with each node being an input or output on a single topic in the Data Marketplace. Any Application may subscribe to any topic to make use of the data found there, with the delivery of messages to the various destinations handled by the MQTT broker. The expectation is that Applications will generally output onto fixed topic names (within their allocated sub-tree), whilst being freely configured (via the Configuration API) to read from whichever input topics the system operator requires.

The MQTT broker forming the Data Marketplace will only buffer a single message on each topic, so Containers must handle their input messages sufficiently quickly to be ready when the next message arrives on a topic. The Data Marketplace does not perform any rate adaption, so Applications need to be prepared to receive their input messages at whatever intervals the upstream Application produces them.

4.2 Application Identification

Applications must have uniquely defined identifiers. These fulfil a number of roles:

- To ensure that Applications will never encounter a name “clash” with another Application.
- To allow multiple copies of the same Application to run simultaneously, with separate configuration settings.
- To allow different versions of the same Application to be installed and run simultaneously.
- To allow system operators to unambiguously specify what Applications are to be run on any given system.

In this section, each word in **bold** is defined in the Glossary at the start of this document. To facilitate the selection and operation of **Application**s, each **Application**'s **Docker Image** has three unique pieces for information associated with it:

1. A **Vendor** string. This is a string which identifies the developer of the **Application**. These are allocated by EA Technology to each party creating **Application**s to run on LV-CAP.
2. An **Application Name** string. This is a string which identifies the **Application**. This is chosen at will by the **Application** developer. It should not container version information.
3. An **Application Version**. This is a string which indicates the version of the **Application**. Decimal points may be used to separate version numbers, e.g. 1.2.3. This is chosen by the **Application** developer.

This information enabled a system operator to specify exactly what **Application** they wish to run on LV-CAP. There are a number of constraints on the above fields which must be satisfied when they are chosen:

- The **Vendor** and **Application Name** must be valid Docker Names (see Reference 4):
  - Composed of valid ASCII characters.
  - Restricted to lower case letters, digits, periods and hyphens (no underscores).
  - May not start with a period or a dash.
- Each release or update of an Application must have a unique combination of **Vendor, Application Name** and **Application Version.**
- For an Application to be successfully updated, the update must have an **Application Version** which Docker considers to be different to the existing Application's **Application Version.**
- For compatibility, the total length of the **Vendor, Application Name** and **Application Version** must be less than 44 characters.

The **Vendor, Application Name** and **Application Version** are combined to form the **Application ID <APID>**. When used as a file name or **Topic Name** then these sections are separated with an underscore:

```
<Vendor>_<Application Name>_<Application Version>
```

When used as a tag for a **Docker Image** then they are combined according to the usual docker convention:

```
<Vendor>/<Application Name>:<Application Version>
```

The <**APID**> identifies a specific Application executable in a globally unique manner. In the future this will be enforced through the digital signing of **Application Images** and their <**APID**>.

When creating the *Docker Image*, these fields are specified to the -t option of the docker build command as follows:

```
docker build -t <Vendor>/<Application Name>:<Application Version>.
```

When an **Application** is to be executed on LV-CAP, a **Docker Container** is created from the **Docker Image**. Each **Docker Container** must have a unique name, and we must support creating multiple Containers from one Image. To enable this a fourth field is used, which is the **Instance**. Instance is a two digit number which is unique to this Instance of an **Application** on the LV-CAP system. **Instance** values are set up in the **Container Manager** configuration by the system operator. The **Instance** value of "00" is special and reserved for use by Applications which cannot have more than one instance running. A single instance of any other Application may use any

other value between "01" and "99". There is no requirement for **Instance** numbers to be contiguous, and their numeric value has no significance.

The **Vendor, Application Name** and **Instance** are combined to form the **Instance ID** (abbreviated in this document as <**IID**>) in the form

`<Vendor>_<Application Name>_<Instance>`

The <**IID**> identifies a specific instance of an **Application**, which is unique only on a given LV-CAP system, and may use any (specified) version of the **Application**. This is set up through the **Container Manager** configuration file.

Each Application Instance **Docker Container** created on LV-CAP will have the container name in Docker set to the <IID>.

The **Application Version** is deliberately omitted from the **Instance ID** so that the name does not change when newer versions of the **Application** are deployed. The **Instance ID** <IID> of a container is used as its handle, to identify the Container's area of file system space, MQTT topic namespace and so on.

4.2.1 Legacy Applications

Applications developed against older versions of this **API** (and the Innovate UK project) were identified by a single **GUID**. This 48-byte opaque string served as both **Application ID** <APID> (although it lacked version information) and **Instance ID** <IID> (although it lacked instance numbers). Applications using this form can still run one instance, using their legacy identifiers.

For these legacy containers, two instances of the same Container with the same GUID will never be run on the same LV-CAP installation. The GUID of a container was used as its handle, to identify the Container's area of file system space, MQTT topic namespace and so on.

4.3 Message Serialisation

All messages transmitted via the Data Marketplace are serialised in JavaScript Object Notation (JSON). Adding additional white space to JSON payloads to 'pretty print' them is discouraged. All messages sent via the MQTT Broker must be valid JSON.

Standardised JSON object structures are used wherever possible to maximise interoperability. These are defined in Section 9 of this document.

4.4 Topic Names

All messages exchanged through the Data Marketplace are published on MQTT topics. Whilst this API sets out specific topics for some purposes (e.g. interactions with the Container Manager) it is up to Application authors to choose suitable topics (and especially sub-topics) for the messages which their container produces. In order to use the standard JSON object structures defined in Section 9, unchanging information about the value has to be encoded in the topic name, rather than in the JSON payload itself. This also reduces the transmission of redundant (invariant) information where communications bandwidth is limited.

The MQTT standard itself places few restrictions on the choice of topic names, apart those specified in Section 4.7 of the standard. When choosing topic names however, the following guidelines should be borne in mind to make development and administration easier:

- Avoid spaces in topic names, as they are prone to confuse parsers of all sorts.
- Avoid non-ASCII characters in topic names, as they are prone to confuse users or their tools.
- Use topic levels to separate sections of your topic name. E.g. "output/transformer/forecast/4h/capacity" not "output/transformer-forecast_4hCapacity".
- Design in extensibility – it will be disruptive to change existing topic names to allow additional data to be published (e.g. more channels or intermediate calculation results).

To make it easier for system operators to understand what messages on a topic mean, the following general form of topic names is recommended:

<subtree>/<asset>/<parameter group>/<time>/<parameter>

Not all components will be required for a given topic name and may be omitted. This results in topic names like:

algorithm/data/0ca2eadb-b128-4dff-9bd7-cbb15e21b8b1/Number1Tx/state/hst

sensor/data/eatl_modbusrtusensor_01/Number1Tx/load/A

4.5 Units

All messages transmitted should have a timestamp (as shown in the preferred JSON formats in section 9). These timestamps are 64-bit UNIX timestamps, defined as the number of seconds since 1st Jan 1970 UTC. Where sub-second resolution is required, the fractional value should be stored as a separate field.

Wherever possible, Applications should use the time stamp fields from incoming messages in preference to referencing system time (explicitly or implicitly). This will make it much easier to test Applications in a reproducible manner by simply replaying a fixed sequence of input messages, regardless of the relationship between message time stamps and system time.

Values are always given in the base SI unit for the quantity being measured or calculated. For example, current is always given in Amps, never in milliamps or kiloamps. Temperatures are given in degrees Centigrade rather than Kelvin (in accordance with common engineering practice).

Metadata for the display of values may be passed between containers via Data Series Metadata Objects described in Section 9.4.

4.6 Text Encoding

UTF-8 is the preferred method of encoding text.

When including non-English text in JSON strings bear in mind that that the double-quote character must be escaped with a backslash, and other escape sequences are used for newline etc. control characters, as per the JSON specification.

4.7 Data Persistence

Applications have two options for persisting data:

1. Data which is output to the Data Marketplace can be stored in the Data Storage Application (Section 0). Any Application can then retrieve this data in the future (up to a time limit imposed by the removal of old data values).
2. Each Application is assigned a filesystem volume. This file system is private to the Application and not visible to any other container on the system. Data can be stored here by the Application, e.g. to save system state or history.

The rest of the Docker Container environment is ephemeral and will be lost when the Application is re-started, either by the Container Manager or because the whole LV-CAP platform is rebooted.

4.8 Data Flow and Valid Flags

LV-CAP is designed to work on the basis that data keeps flowing through the processing pipeline at all times. To support this, the standardised JSON object structures in Section 9 all contain a Valid key. When correct data is not available or cannot be calculated, Applications should continue to output messages to the Data Marketplace in the normal manner, but with the Valid key set to false. Applications subscribed to the topics will then be made aware that time is moving on, but that there is a problem with the data source.

Containers which fulfil the Sensor Application role (Section 0) should continue to output messages at the configured interval under all circumstances. This includes if the sensors are disconnected or producing out-of-range values. When data is not available or out-of-range, the "Valid" member in the output should be set to false. The actual sensor value sent in this case does not matter,

because subscribed containers should not use the value when "Valid" is false. The timestamp field must be updated so that the subscribed containers can keep track of time.

Algorithm containers receiving input messages with Valid set to false should not use the Value in the received message, but may rely upon the time stamps. When the input timestamps reach the point that the algorithm is due to provide output, it must do so. It is up to the Application author to decide if there is sufficient Valid data to produce an output or not. If there is insufficient Valid data to produce a new result then the container should output, setting Valid to false and using the timestamp from the most recent input message (whether that message is valid or not).

4.9 Data Priority

The standard JSON formats described in Section 9 provide for a Priority field. This allows the upload of certain messages to be prioritised by Data Upload Applications, based on policy set by the system operator and priority information from Application authors.

Valid Priority values are integers between 1 and 5. A Priority value of 1 is the highest priority and 5 is the lowest priority. Any other Priority value is not valid and is treated the same as if Priority is not specified. These messages with no specified Priority have lower Priority than all messages with a specified Priority.

The data APIs in sections 8.4 and 8.5 support query by Priority.

5. Start-up Procedures

5.1 LV-CAP System Start

5.1.1 Start-up of Core Services

As discussed in section 0, the LV Common Application Framework consists of a number of core services for third party containers to use. These have a defined start-up order and only once these have all started up will any other container be started. The Core Services are started in the following order:

1. Container Manager
2. Data Marketplace
3. Data Storage Application

5.1.2 Start-up of Remaining Applications

The remaining Applications installed on a given LV Common Application Platform will be started automatically, once the platform's framework has successfully started up and entered the running state. All other Applications must be independent of each other (there is no concept of inter-Application dependencies) so that Applications can start in any order. Applications will be started by the Container Manager in order of their installation date (i.e. order of when they were added to Docker's available image list).

5.2 Application Start

Each Application on the LV Common Application Platform must perform certain actions when it is started by the Container Manager. Failure to do so is likely to result in the Application being shut down by the Container Manager.

Upon starting, a third-party Application must perform the following actions in the given order:

1. Connect to the Data Marketplace (see Section 8.1 for the MQTT Broker connection details).
2. Subscribe to topics listed in Table 2
3. Send a configuration request message to the Container Manager via the Data Marketplace.
4. Wait until a response is sent back by the Container Manager. This response will either contain the Application's configuration, or will include an error message if the Container Manager is not aware of any configuration for the Application.
5. If configuration data is received, the Application should process the configuration and apply it internally.
6. If the configuration is valid, the Application can start operating, sending a status update to the Container Manager indicating all is OK.
7. If no configuration is available or there is an error in the configuration, the Application must send a status update to the Container Manager indicating an error:
    - Sending STATUS_INITIAL will result in the Container Manager re-sending the configuration file, the Application should stay in a non-operating state awaiting configuration.
    - Sending STATUS_ERROR and will cause the Container Manager to restart the Application. See Section 8.2.2 for more information on status messages.

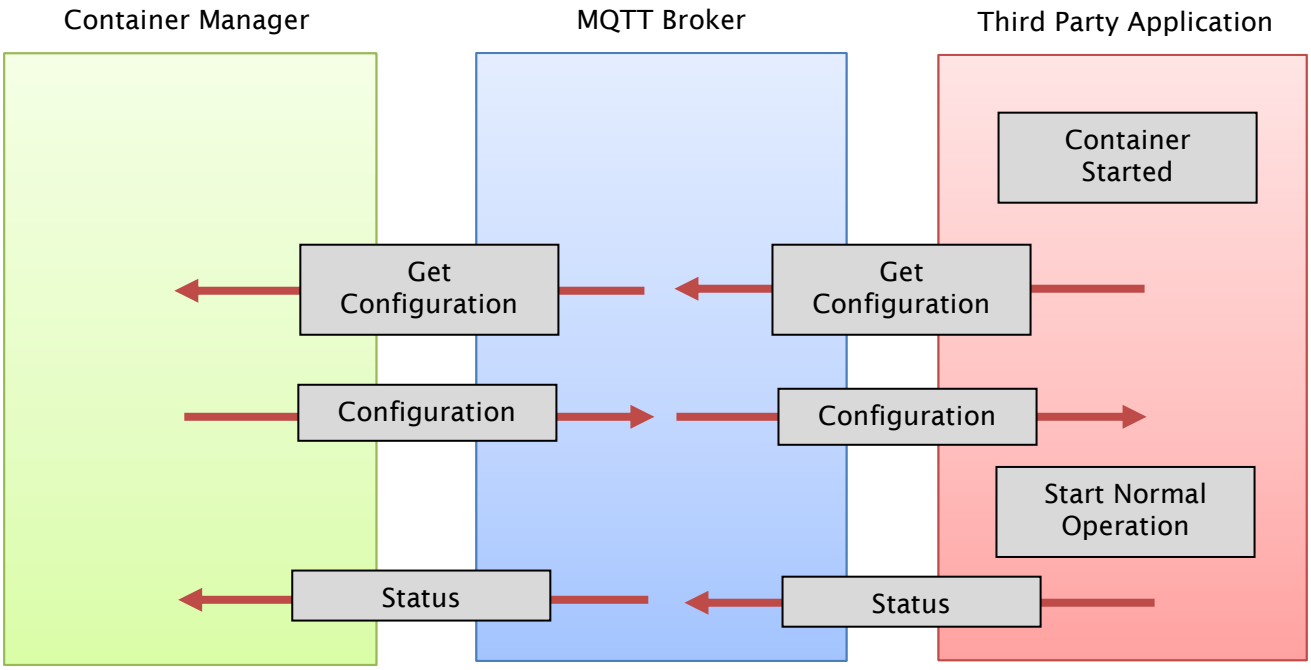

Figure 4 - Application start-up procedure

The above diagram shows the Application start procedure for a successful start.

Due to the Application start order (Section 5.1.2) it is possible that once a container starts its 'normal operation', other Applications it may want to communicate with may not yet be operating. In this situation the Application will have to wait until any dependencies are running. This is not normally a problem because the MQTT protocol allows publishing and subscription to occur in any order, with no requirement for topics to be configured or created in advance.

5.3 Required Subscriptions for all Applications

All Applications must subscribe to the following MQTT message topics in order to interact correctly with the Container Manager and remain running on the platform.

Topic	Purpose
status/request	Receives status requests from the Container Manager. Non-response to two consecutive status requests will lead to the Application being restarted without notice by the Container Manager.
config/response/<IID>	Receives Application configuration sent by the Container Manager. The IID is the Application's own unique IID as in section 4.2
command/<IID>	Integer, the command to execute.

Table 2 - Required Subscriptions by Third Party Applications

6. Shutdown Procedure

Similar to the start-up procedure (Section 5), the LV Common Application Platform has a defined shutdown procedure. This shutdown procedure is designed to allow Applications to shut down in a safe manner and avoid any data loss or corruption. The shutdown procedure not only applies to shutting down of the entire platform, but also occurs when an updated Application image is applied. Applications can request their own shutdown if required, but all Applications shall respond to a shutdown request from LV-CAP.

When LV-CAP requests shutdown of an Application, the procedure is defined as:

1. Container Manager sends a shutdown command to the Application via the MQTT broker (see section 8.2.3).
2. The Application handles the notification and performs its own internal shutdown procedure, which may include writing any data to disk, stopping all MQTT subscriptions including that of status requests, and any other work to perform a clean shutdown. Once completed, the Application must respond to the Container Manager using a *status/response* message with the STATUS_SHUT_DWN status.
3. The Container Manager will then shut down the container. If the Container Manager does not receive a status message from the Application to be shutdown which includes the STATUS_SHUT_DWN status for more than 1 minute, the container will automatically be shut down.

If the Application fails to shut down within the “status/request” (default 2 minutes) interval then Container Manager will shut it down forcibly by terminating the process.

In the event of an Application requesting its own shutdown by the Container Manager, the procedure is defined as:

1. The Application performs its own internal shutdown procedure, which may include writing any data to disk, stopping all MQTT subscriptions including that of status requests, and any other work to perform a clean shutdown.
2. Once completed, the Application must send a status to the Container Manager using a *status/response* message with the STATUS_SHUT_DWN status.
3. The Container Manager will then shut down the Application, and added to the stopped Application list. It will not be run again until the Container Manager configuration is altered or the Container Manager is re-started.

If an Application needs to be re-started by the Container Manager, the procedure is:

1. The Application prepares for being restarted, which may include writing any data to disk, stopping all MQTT subscriptions including that of status requests, and any other work to perform a clean restart.
2. Once completed, the Application must send a status to the Container Manager using a *status/response* message with the STATUS_RESTART status.
3. The Container Manager will then shut down the Application and start it back up again immediately.

7. Data Storage

The Data Storage Application allows persistence of data from Applications. Data stored in the Data Storage Application can also be configured by the system administrator to be uploaded by one or more Communications Applications. In this role, the Data Storage Application acts as a buffer so that data is uploaded to its destination reliably, even in the face of unreliable communications links.

The data stored on the platform will be placed in a database which can be accessed via the Data Marketplace (see section 8.3). The output of each Application Instance <IID> will be stored in its own table. This table is created when the Application Instance is first created by the Container Manager. All tables created for Applications will store records with the format documented in Table 3.

Field	Type	Description
ID	Opaque Integer	Each record stored will be assigned an ID by the Data Storage Application. This integer will be unique amongst the records currently stored in the Data Storage Application, but may be re-used over the life of the LV-CAP system as old data is purged from the database and new records added. There are no guarantees about the numeric value of this identifier.
Timestamp	Integer	The Unix timestamp at which the record was added to the Data Storage Container.
SubTopic	String	Part of the MQTT topic string on the Data Marketplace from which the record came will be stored in this field. Because tables are allocated by the Application Instance ID, the topic string up to the IID would be the same for all records. The common string is not stored, leaving only the Application's sub-topic string. If no sub topics are present this field will be null.
Data	BLOB	This is the MQTT JSON message sent to be stored. It is stored as a BLOB so that no alterations are made to the JSON object.

Table 3 - Containers Table schema

The payload an Application sends to be stored will retrieved unaltered from the Data Storage Application. The Data Storage Application will set the values of the other fields automatically. The API for retrieving records is documented in Section 8.5. Applications are strongly encouraged to output their data in one of the standard JSON payload formats described in Section 9.

8. Data Marketplace API

The main form of communication between Applications and the LV Common Application Framework is via the Data Marketplace. This section documents the MQTT message topics, their associated payload and includes examples of message payload. The API is broken up into sections according to the Application roles (Section 0) expected to use them. Application may (and will) use methods from more than one section of the API.

8.1 MQTT Broker

Each container wishing to operate on the LV Common Application Framework must connect and communicate using the provided MQTT broker. The LV-CAP system uses a secured MQTT broker, in order to support authentication of Application when they connect to the Data Marketplace. The connections settings required are shown in Table 4.

Setting	Value
Hostname	marketplace
Port Number	8883
Encryption	TLS v1.2 or higher
Authentication	X509 client certificate
Username	Set to the Application's Application ID
Client ID	Set to the Application's Application ID

Table 4 – Secured MQTT Broker Settings

EA Technology will operate a TLS Certificate Authority for the LV-CAP system. All client SSL certificates must be signed by this Certificate Authority, which will be trusted by the Data Marketplace. This Certificate Authority certificate will be issued to Application developers for inclusion in Application at build time, so they can authenticate the Data Marketplace.

Client certificates will be signed on request by the certificate authority, with the Common Name (CN) of the certificate set to the Application ID <APID> of the Application they are to be used by (see Section 4.2). This client certificate should be embedded in the Application so that it can be used to connect to the Data Marketplace. The client certificate and associated private key need to be embedded in the Application so that it can connect to the Marketplace. The private key should be encrypted to minimise the risk of it being extracted from the Application by a third party. These is no reason for EA Technology, or any other Application author, to know the Application's private key.

When the Application connects to the Data Marketplace it's certificate will be checked. If valid, and not revoked by the system operator, it will be allowed to connect. Access control lists will then allow the Application to publish on the topics set out in this API. In general, subscriptions will not be restricted.

A new certificate should be obtained whenever an updated version of the Application is produced. This both mitigates the fixed expiry date of certificates, and allows the certificate of specific Application versions to be revoked if the keys are compromised. This will also have to be done when an updated Certificate Authority Root Certificate is required.

8.1.1 Payload Descriptions

JSON does not have a concept of fixed-size (bit width) integers, however implementation in strongly typed languages is made much easier by defining the maximum size of integer fields wherever possible. In this documentation:

- Any key which is shown with type “Integer” will always fit into a 32-bit signed integer.
- Any key which is shown with type “Int64” will always fit into a 64-bit signed integer.

8.1.2 Security and Signing

The present implementation of LV-CAP provides only limited security between Applications, and so requires a high degree of trust in Application authors. To improve this situation in the future, an optional “signature” object has been added to all JSON payloads specified in this API. This member is reserved for the definition (in a future version of this API) of a mechanism for cryptographically signing each JSON payload.

The signing scheme is intended to use public (asymmetric) key cryptography. The source Application will sign all outgoing messages with a private key, which must be kept secret. Destination Applications receiving these messages can use the source Application’s public key (which does have to be kept secret) to verify that the messages received are indeed from the correct source container. It is intended that the public keys will be distributed to the relevant Applications via their configuration data.

A Application which does not implement signature verification will be able to receive future signed messages without modification, because it will ignore the signature object. Applications with signature validation implemented will have to decide on their policy for messages received without signatures.

At some future date, it may become mandatory to sign messages on some critical API topics when communicating with the LV-CAP core components. It will be up to other Application authors at what point they require signed input messages.

The signing of Docker Images will also be added in future to ensure that when system operators specify a particular Application ID <APID> (see section 4.2) only that specific version can be run.

8.1.3 Last Will and Testament

The MQTT broker supplied by the framework supports the Last Will and Testament (LWT) feature. This can be used to define, upon connection, a message which will automatically be sent by the broker to subscribers of the set topic upon the non-clean disconnection of a client. In order to manage the platform all Applications must provide a LWT on their status response topic (Section 8.2.2). The status response sent as the LWT must include the FAILED state within the payload. Applications may also set LWT’s on any topic they desire to inform others of their failed state.

8.1.4 Quality of Service

MQTT provides a Quality of Service (QoS) level feature, which defines how hard a broker or client will work to ensure a message is delivered. More details can be found in section 4.3 of the MQTT Standard.

MQTT QoS is a property of both the publishing and subscribing of a message, so a client can publish a message at any QoS and a client may subscribe to a topic at any QoS. The implemented QoS will be the lowest of the publishing and subscribing QoS levels. There are 3 QoS levels defined in MQTT:

- QoS 0 - At most once. The status request topic has a QoS of 0 as this regular heartbeat is not critical, and must be sent regularly.
- QoS 1 - At least once. The Container Manager sends out commands at QoS 1 as Containers can easily handle receiving the same command more than once.
- QoS 2 - Exactly once. This is used when querying the Data Storage Container, as multiple message delivery could have complex and undesirable affects upon the database.

8.2 LV-CAP Core API

The Core API is responsible for management of Applications. All Applications will need to use the Core API to register with and run on LV-CAP.

8.2.1 Configuration

The Configuration message topic is used to request and distribute configuration to Applications.

QoS: Messages on this these topics must be sent and received with QoS = 1. Applications must cope with multiple copies of their configuration information being delivered.

Retention: Messages sent on these topics must have the retention flag set to false.

Topic	Description	Sender	Receiver	Payload	Notes
config/request/<IID>	Message containing a request from a container to the Container Manager requesting it's configuration	Any Application	Container Manager	{ "Timestamp": <Int64>, "Signature": {} }	No required payload. **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the configuration was requested. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section 8.1.2.

Topic	Description	Sender	Receiver	Payload	Notes

config/response/<IID>	Message containing updated configuration for a specific Application Instance. Can be a response to a request, or a new set of configuration pushed to a Application Instance.	Container Manager	Application Instance with IID specified in topic name	<pre>{
				"Configuration":
				{
				"<Key_1>": <Value_1>,
				"<Key_2>": <Value_2>,
				"<Key_n>": <Value_n>
				},
				"Timestamp": <Int64>,
				"Signature": {}
				}</pre>
					Configuration: JSON Object read directly from the Application Instance configuration file. The structure will be different for each Application, as described in Section 9.5.
					Timestamp: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the configuration was requested. Required in signed payloads to protect against replay attacks.
					Signature: (Reserved) See Section 8.1.2.

Table 5 - Configuration MQTT topics

8.2.2 Status

The Status topic is used by the Container Manager to request the status of running Applications. The Container Manager will periodically request the status, and running Applications must respond to the request to confirm that they are operating correctly.

If an Application does not respond or responds with a status other than STATUS_MSG_OK or STATUS_INITIAL (see Table 7), it is considered to have failed the request. After three successive failed status requests the Application will be restarted by the Container Manager. If the Container still fails further status requests to reach a total of 5 consecutive requests, it will be permanently shut down, and this error logged in the database.

QoS: Messages on this these topics must be sent and received with the QoS shown in Table 6.

Retention: Messages on this these topics must have the retention flag set to false.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
status /request	0	Message to request status from all running containers.	Container Manager	All Applications	{ "Timestamp": <Int64>, "Signature": {} }	No required payload. **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the status was requested. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section 8.1.2.
Status /response/ <IID>	1	Message containing a status update from the Application Instance identified by <IID>.	Any Application	Container Manager	{ "Status": <Integer>, "Message": "<message>" "Timestamp": <Int64>, "Signature": {} }	**Status**: (Required) One of the values from Table 7. **Message** (Optional): If the Message string is present it will be sent to the error Database. **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the status was requested. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section 8.1.2.

Table 6 - MQTT Status Topic

The valid status response values are shown in Table 7.

Status Value	Meaning
1	STATUS_MSG_OK – the Application is running normally.
2	STATUS_MSG_FAIL – the Application has failed. The Container Manager will restart the container. If the key “Message” is present in the JSON object it will be stored in the Data Storage Application as an error message.
3	STATUS_MSG_ERR – the same as STATUS_MSG_FAIL for backwards compatibility.
4	STATUS_SHUT_DWN – the Application has completed its shutdown procedures and is ready to be stopped by the Container Manager. The container will not be restarted unless the Container Manager configuration is altered or the Container Manager is re-started.
5	STATUS_INITIAL – the Application is waiting to receive its configuration (and can do nothing until it does). The Container Manager will resend the Application's configuration.
6	STATUS_RESTART – the Application wishes to be re-started. It has completed any shutdown procedures and saving of state and is ready to be stopped and started again by the Container Manager.

Table 7 - Status Field Values

Status values other than STATUS_MSG_OK and STATUS_INITIAL are regarded as failure conditions. If the key “Message” is present in a JSON object with a failure status, the Message string will be stored in the Data Storage Application as an error.

8.2.3 Command

The MQTT command topic allows the Container Manager to send instructions to any running Application.

QoS: Messages on this these topics must be sent and received with QoS = 1

Retention: Messages sent on this topic must have the retention flag set to false.

Topic	Description	Sender	Receiver	Payload	Notes
command/<IID>	This is a command sent from the Container manager for the container to execute	Container Manager	Any Application	{ "Command": <Integer>, "Timestamp": <Int64>, "Signature": {} }	**Command**: (Required) One of the command values shown in Table 9 below. **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the command was issued. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section 8.1.2.

Table 8 – Commands MQTT

The command values in Table 9 are currently specified. In the future, more values may be added, so all LV-CAP Applications must check the payload of the message received is the expected value.

Command Value	Command
1	Shut Down. Currently the only implemented command. All Applications must implement this command. This command is used when an updated Application is deployed. The Container Manager will send a shutdown command for the running Application to stop everything it is doing before re-starting the Application. See Section 6 for more details.

Table 9 - Command Topic Command Values

8.2.4 Error

The MQTT error topic allows all containers to log any issue or internal error.

QoS: Messages on these topics must be sent and received with QoS = 1.

Retention: Messages sent on this topic must have the retention flag set to false.

Topic	Description	Sender	Receiver	Payload	Notes
storage/data /error/<IID>	Topic to log any external or internal errors to storage.	All Applications	Data Storage Application	{ "Errno": <Integer>, "Message": "String", "Timestamp": <Int64>, "Signature": {} }	**Errno: (**Required**)** One of the errno values shown in the **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the command was issued. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section

Table 10 - Report Error Topic Table

Command Value	Errno Name	Description
1	ERRNO_JSON_INVALID	Payload from MQTT failed to Parse. Invalid JSON.
2	ERRNO_IO	Input/output Error
3	ERRNO_ACCESS	Permission denied
4	ERRNO_NO_DEVICE	No device found
5	ERRNO_FILE_DIRECTORY	Directory not found
6	ERRNO_MQTT_SUBSCRIPTION	Failed subscription to MQTT Topic
7	ERRNO_MQTT_PUBLISH	Failed Publish, this is only used when trying to publish a payload to. If failed to publish an error message use std::out. This will be saved by the Docker Log files and can be accessed later by Admin.
8	ERRNO_APPLICATION	Process failed due to Application error. The message to accompany this Errno is mandatory.

Command Value	Errno Name	Description
9	ERRNO_CONFIGURATION	Failed processing the incoming Config. This is if the contents of the configuration expected does not match or has the wrong types. (This could also be ERROR_JSON_INVALID if it's not valid JSON)
10	ERRNO_MQTT_CABLLBACK	Error occurred in the MQTT Call back. This can be when setting up the call back or an error within the call back with an incoming message
11	ERRNO_SENSOR	This can have two applications. The first, for any Sensor Container that has an error with reading a sensor it can output this Errno with the relevant message. The second is for any Algorithm Container reading in the Sensor Payload and the Payload is valid but any of the Key types is incorrect.
12	ERRNO_NETWORK	Error accessing the network.
13	ERRNO_PORT	Error opening or accessing a port.
14	ERRNO_PROFILE	Any Algorithm Application expecting a Profile Payload, the Payload is valid but any of the Key types is incorrect.

Table 11 - Errno Description Table

8.3 Sensor Data API

Applications which fulfil the Sensor Application role (see Section 0) will publish on the topics in the Sensor Data API. Applications in the Algorithm Application role will often subscribe to these topics to obtain their inputs. This data will not normally be stored.

8.3.1 Sensor Readings

Topics for transferring sensor reading data collected and published by Sensor Applications.

QoS: Messages on this these topics must be sent and received with QoS = 1.

Retention: Messages on this these topics must have the retention flag set to false.

Topic	Description	Sender	Receiver	Payload	Notes
sensor/data/<IID>/<sensorname>	New sensor readings	Sensor Applications	Any Application	Standard Scalar Object Format, Series Object Format or Co-ordinate Object Format.	See Section 9 for details of standard JSON formats.

Table 12 - Sensor Reading MQTT messages

See Section 4.4 for guidelines on choosing intelligible topic names for message output. Sensor Applications are responsible for publishing data and setting the Valid flag in messages (Section 9) in accordance with the guidelines set out in Section 4.8.

Readings will often be published at fixed time intervals. These intervals will start when the sensor Application receives its configuration, and so may not be aligned to "clock face" times. For instance, if the configuration was received at 09:05:00, setting a time interval of 20 seconds. The Sensor Application will output at 09:05:20 then at 09:05:45 and so on.

Depending on the properties of the sensor Application, there is a possibility that if many sensors have the same interval time and one sensor takes longer to read that this would delay the next sensor and so on. The start of the normal operation for the Sensor Application is most susceptible to this, however after a short time this will reach an equilibrium and each output will be at the prescribed interval. Applications consuming the messages must be equipped to cope with these timing variations.

8.3.2 Sensor Metadata

Topics for transferring sensor metadata published by Sensor Applications. Publishing on these Metadata topics is optional.

QoS: Messages on this these topics must be sent and received with QoS = 1.

Retention: Messages on this these topics must have the retention flag set to true.

Topic	Description	Sender	Receiver	Payload	Notes
sensor/data/ <IID>/ <sensorname>/ metadata	Sensor reading metadata	Sensor Application	Any Application	Standard Data Series Metadata. Object Format.	See Section 9 for details of standard JSON formats.

Table 13 - Sensor Reading MQTT messages

8.4 Algorithm Data API

Applications which fulfil the Algorithm Application role (see Section 0) will publish on the topics described in the Algorithm Data API. Application in the Algorithm Application role may subscribe to these topics to obtain inputs. Applications in the Output Application role will normally subscribe to one or more of these topics to obtain inputs.

Data published on these topics may be stored in the Data Storage Application, depending on the latter's configuration and the "ToStore" flag set by the publishing Application. Only stored data will be available for upload by Communications Applications.

QoS: Messages on this these topics must be sent and received with QoS = 1.

Retention: Messages on this these topics must have the retention flag set to false.

Topic	Description	Sender	Receiver	Payload	Notes

algorithm/data/<IID>/<subtopic>	The main topic an algorithm Application will publish its data on	Algorithm Application	Any Application	Any valid JSON object.
				
				Applications are strongly encouraged to use one of the standard JSON Object Formats to improve interoperability.
				{
				  <Valid JSON Payload>
				}
					Algorithm Applications may output on any sub-topic starting with “algorithm/data/<IID>” (where <IID> is the Application Instance's assigned identifier).

Table 14 - Algorithm Data Table

When choosing the sub-topic on which to output data, authors are encouraged to use a descriptive topic name (Section 4.4). This makes configuring systems easier and less error prone. For example, transformer capacity forecasts for the available capacity in transformer T1 over the next 4 hours might be output on topic

algorithm/data/<IID>/T1/forecast/4h/capacity

If the JSON payload is to be stored in the Data Storage Application it must have a KEY “ToStore” and the value set to true. If this is not present or is set to false then the data will not be stored. Only stored data will be available for upload by Communications Applications.

Payloads should have a KEY “Timestamp” containing the Unix timestamp the calculation refers to. Where the calculation covers a range of time, this should be the time stamp of the most recent time covered by the calculation.

Algorithm Applications may use optional metadata subtopics in exactly the same way as Sensor Applications, as documented in Section 8.3.2.

8.5 Data Upload API

The Data Upload API provides a means to access the data queued for upload in the Data Storage Application. Applications which fulfil the Communications Application Upload role (see Section 0) will use this API extensively. Applications using this API must be explicitly authorised by the system operator in the Data Storage Application configuration. A separate (virtual) queue is maintained for each Upload Application of data

which is waiting for upload. Once a message has been uploaded the Upload Application must notify this fact back to the Data Storage Application via this API so that the queues can be updated.

This API operates on a pattern of separate topics for requests and response messages. When using this API, Applications should always subscribe to the response topic before publishing a request. This avoids a race between the response and the subscription which may cause the container to miss response messages.

All methods in this API work with the per-Application database tables described in Section 0. The SubTopic and Data columns are set from the received message. The other columns in the table will be set automatically by the Data Storage Application.

QoS: Messages on this these topics must be sent and received with the QoS shown in Table 15.

Retention: Messages on this these topics must have the retention flag set to false.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
storage/request /newdata/<IID>	2	A request for new data to be uploaded by Upload Application <IID>. The request will search all tables in the database which the Application is permitted to upload from.	Upload Application with identifier <IID>	Data Storage Application	{ "MaxLength": <Integer>, "StartTime": <Int64>, "EndTime": <Int64>, "PreferOldest": <Boolean>, “InstanceID”: <IID> "SubTopic": <String>, "MinPriority": <Int>, "MaxPriority": <Int>, "Timestamp": <Int64>, "Signature": {} }	The request Payload keys are documented in Table 16.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
storage/response /newdata/<IID>	2	The response to the above request.	Data Storage Application	Upload Application with identifier <IID>	{ "Status": <Integer>, "Response": [{ "TableName": <IID>, "TableRows": [{ "ID":<Integer>, "Timestamp": <Int64>, "SubTopic": <string>, "Data": <JSON Object>, }, {rowN}] }, { "TableName": <IID>, “TableRows”: [{ "ID":<Integer>, "Timestamp": <Int64>, "SubTopic": <string>, "Data": <JSON Object>, }, {rowN}] }] "Timestamp": <Int64>, "Signature": {} }	The response Payload keys are documented in Table 17. If an error occurs then the Payload will still have Status and Response members, but the Response array will be empty.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
storage/uploaded /<IID>	1	Indicates that messages have been uploaded by <IID> and they should be removed from the upload queue.	Upload Application with identifier <IID>	Data Storage Application	{ "NewData": [{"<IID>": [<Integer>, <Integer>]}, {"<IID>": [<Integer>, <Integer>]},], "Timestamp": <Int64>, "Signature": {} }	**NewData**: (Required) Array of objects, one for each table to be updated. **<IID>:** (Required) Array of opaque integer identifiers of the messages which have been uploaded. **Timestamp**: (Optional) Standard LV-CAP timestamp (see Section 4.5) when the update was sent. Required in signed payloads to protect against replay attacks. **Signature**: (Reserved) See Section 8.1.2.

Table 15 – Communications Upload Container MQTT

Key	Status	Description
`MaxLength`	Optional	The maximum number of records to be returned from the database. This is subject to an upper limit set in the Data Storage Container configuration (see 8.5.1 below). If no value is given then the default value is 100 records.
`StartTime`	Optional	A UNIX timestamp. Only records added to the Data Storage Application after this time will be returned. If not supplied then records from the start of the database will be returned, unless the operator has imposed a tighter restriction.
`EndTime`	Optional	A Unix timestamp. Only records added to the Data Storage Application before this time will be returned. If not supplied then records up to the present time are returned.
`PreferOldest`	Optional	Flag indicating that if there are more than MaxLength records available, the oldest data should be returned rather than the default of returning the newest data.

Key	Status	Description
InstanceID	Optional	String identifying the Application Instance which data should be returned for. Only records which exactly match the given topic will be returned (no wildcards). This constrains the query to only return results from the specified table in the database. If this member is an empty string or omitted from the JSON then data from all Application Instances is returned.
SubTopic	Optional	String giving the sub-topic data is required for. This is the sub-topic below data/algorithm/<IID>. Only records which exactly match the given topic will be returned (no wildcards). To retrieve data from all sub-topics, do not include this key in the JSON payload. To request data only from the top-level topic (no sub-topics) then this key must be included in the JSON payload with an empty string value.
MaxPriority	Optional	Integer defining what priority messages are to be returned. If this JSON key is supplied, messages with priority equal to or numerically less than the value only will be returned. The special value of 6 can be used to return only messages which had no Priority value when stored. If neither this JSON key nor MaxPriority is specified then messages of all priorities will be returned.
MinPriority	Optional	Integer defining what priority messages are to be returned. If this JSON key is supplied, messages with priority equal to or numerically greater than the value only will be returned. If neither this JSON key nor MaxPriority is specified then messages of all priorities will be returned. If both keys are supplied then only messages which meet both criteria will be returned.
Timestamp	Optional	A UNIX timestamp when the request was sent. Required in signed payloads to protect against replay attacks.
Signature	Reserved	See Section 8.1.2

Table 16 – Request Object Keys

Key	Status	Description
Status	Always Present	Integer indicating whether the query succeeded or not. See Table 18.
Response	If Status = DSC_QUERY_OK	An array of objects containing data from different tables to be uploaded. Always an array even if data is only from one table.
Response/**TableName**	Always Present	Name of the table the data in this object is from.

Key	Status	Description
Response/**TableRows**	Always Present	An array of selected rows from the table (array even if only one row is selected). Each object in the array is an individual message from the source table.
Response/TableRows/**ID**	Always Present	Opaque integer identifier for the message. These have no meaning except as a handle to be passed back to the Data Storage Application when the message has been uploaded. ID values are only unique within a single table, and may be recycled after the database has been cleaned.
Response/TableRows/**Timestamp**		UNIX timestamp when the message was added to the Data Storage Application (see Section 0).
Response/TableRows/**SubTopic**		The subtopic (below algorithm/<IID>) on which this message was published.
Response/TableRows/**Data**		The original message JSON object stored in the Data Storage Application.
Timestamp	Optional	A UNIX timestamp when the response was sent. Required in signed responses to protect against replay attacks.
Signature	Reserved	See Section 8.1.2

Table 17 – Response Object Keys

Status Value	Code	Description
0		Never sent, an unanticipated error.
1	DSC_QUERY_OK	Query succeeded, the length of the complete results set is less than or equal to MaxLength. The result is returned in the TableRows array. See also DSC_QUERY_MORE.
2	DSC_QUERY_EMPTY	The query was valid, but found no records. The TableRows array will be empty.
3	DSC_QUERY_TABLE_DENIED	The query is against a table (Application Instance) which the sending container is not allowed to access.
4	DSC_QUERY_TOO_LONG	The query requested more data than the Data Storage Application is willing to provide, because the MaxLength field value was too large (see 8.5.1 below).
5	DSC_QUERY_TOO_BIG	The data requested by the query is too big to fit into the MQTT payload length restriction.
6	DSC_QUERY_TOO_OLD	The data requested in the query is from further in the past than the Data Storage Application is willing to provide.

Status Value	Code	Description
7	DSC_UNAVAILABLE	The Data Storage Application is unable to respond to this request, either because it is too busy or is in the process of shutting down.
8	DSC_QUERY_MORE	Query succeeded, there are more than `MaxLength results.` The first MaxLength results are returned in the `TableRows` array, but another query is needed to get more values.
9	DSC_QUERY_INVALID	The JSON query object is empty or not valid JSON and cannot be parsed.

Table 18 – Response Status Values

The Data Upload API is not designed to be re-entrant. After a request has been made, the container should wait for the response (there may need to be an exceptional time-out in case the Data Storage Application suffers an error). If a second request is made whilst the response is being produced, the response is undefined. The request does not modify the database at all, so if a second identical request is made after the first response is received, the same data will be returned.

The response status value is used to show whether there is more data available than was sent or not. There is no concept of a database cursor or response pagination. As a result, API users who need to upload all the available data must:

1. Request data for upload.
2. Upload the received messages (if any).
3. Update the database to mark the messages as uploaded.
4. Continue querying until a response of DSC_QUERY_EMPTY is received, at which point there is no more data to upload.

Although the JSON format for the "storage/response/newdata/<IID>" topic allows for messages from multiple tables to be sent in one message, this is not guaranteed. The Data Storage Container may opt to return data from only one table or topic in the response (where there is data to retrieve), and return data from other tables/topics when subsequent requests are received.

8.5.1 Limits

The Data Storage Application is a shared resource and excessively large queries have the potential to degrade the performance of LV-CAP for all users. To mitigate this risk, limits are imposed on the queries which will be accepted.

- **Maximum number of records requested in one query**. If `MaxLength` is not set then a limit of 100 records will be applied. A query for 100 records will always be permitted. This limit may be increased (up to a maximum of 15 000) by the LV-CAP operator, but Application should not depend upon larger queries being allowed on any given system. Note that the limit is on the requested size, not the actual number of records found (which is not known when the query is set up). Thus a request for 16 000 records will always fail (DSC_QUERY_TOO_LONG from Table 18), even if the table is empty.
- **Maximum query size**. Because the query result is sent as an MQTT message via the Data Marketplace, it is limited to a maximum of 256MB (268,435,455 bytes), as documented in section 2.2.3 of the MQTT 3.1.1 specification. If the query results in a message which is longer than this limit, then an error (DSC_QUERY_TOO_BIG from Table 18.) is returned instead. Applications must request fewer messages to reduce the returned message size below the limit.

- **Maximum data age**. The Data Storage Application will not be able to store data going back in time indefinitely. Old records will be purged by the Data Storage Application to control the database size. To manage database performance the LV-CAP operator may also impose a maximum age on queries. Any request for data older than this age will fail with DSC_QUERY_TOO_OLD from Table 18.

8.5.2 Examples

An example query payload requesting the oldest available data for upload, from all Applications, is shown in Figure 5. The query is not signed. Up to 100 records will be returned as there is no maximum length given. This query may fail:

- With status DSC_QUERY_TOO_OLD if the Data Upload Application does not allow queries indefinitely into the past.
- With status DSC_UNAVAILABLE if the Data Upload Application is shutting down or overloaded.
- With status DSC_QUERY_TOO_BIG if the results will not fit in a MQTT packet.

If it succeeds it could give status:

- DSC_QUERY_EMPTY if there is no data to be sent.
- DSC_QUERY_OK if there are between 1 and 100 messages to be sent.
- DSC_QUERY_MORE is there are more than 100 messages to be sent.

```
{
    "PreferOldest": True
}
```

Figure 5 – Example query payload

An example query payload requesting the latest available data from a specific Application Instance is shown in Figure 6. The query is not signed. Up to 50 records will be returned as requested. This query may fail:

- With status DSC_QUERY_TABLE_DENIED if the Data Storag Application does not allow this Data Upload Application to upload data from this Application Instance.
- With status DSC_UNAVAILABLE if the Data Storage Application is shutting down or overloaded.
- With status DSC_QUERY_TOO_BIG if the results will not fit in a MQTT packet.

If it succeeds it could give status:

- DSC_QUERY_EMPTY if there are no messages from this Application Instance.
- DSC_QUERY_OK if there are between 1 and 50 messages to be sent.
- DSC_QUERY_MORE is there are more than 50 messages to be sent.

```
{
    "MaxLength": 50,
    “InstanceID”: "eatl_profiler_04"
}
```

Figure 6 – Example query payload for a specific Application Instance

An example query payload requesting the latest, highest priority, data from a specific topic is shown in Figure 7. The query is not signed. Up to 10 records will be returned as requested. This query may fail:

- With status DSC_QUERY_TABLE_DENIED if the Data Storage Application does not allow this Data Upload Application to upload data from this Application Instance.
- With status DSC_UNAVAILABLE if the Data Storage Application is shutting down or overloaded.
- With status DSC_QUERY_TOO_BIG if the results will not fit in a MQTT packet.

If it succeeds it could give status:

- DSC_QUERY_EMPTY if there are no messages on this topic with priority equal to 1.
- DSC_QUERY_OK if there are between 1 and 10 messages on this topic with priority 1.
- DSC_QUERY_MORE is there are more than 10 messages on this topic with priority 1

```
{
    "MaxLength": 10,
    "InstanceID": "eatl_profiler_04",
    "SubTopic": "alarm/highhigh",
    "MaxPriority": 1
}
```

Figure 7 – Example query payload for a specific topic and priority.

8.6 Data Storage API

The Data Storage API provides a means to access the data persistently stored by the Data Storage Application. This API provides a mechanism for Applications to access data previously stored by Applications, e.g. where a system history is required.

This API operates on a pattern of separate topics for requests and response messages. When using this API, Applications should always subscribe to the response topic before publishing a request. This avoids a race between the response and the subscription which may cause the container to miss response messages.

All methods in this API work with the per-Application Instance database tables described in Section 0. The SubTopic and Data columns are set from the received message. The other columns in the table will be set automatically by the Data Storage Application.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
storage/data/<IID>/	2	Insert data into the Data Storage Application, in the <IID> table.	Any Application	Data Storage Application	{ "Key": Data, "KeyN": DataN }	Data messages on this topic can hold anything the sender wishes. The message payload will be stored unaltered as a BLOB. If messages are sent on a sub-topic below storage/data/<IID>/ then the sub-topic will be stored in the SubTopic column of the table. This is equivalent to publishing data on algorithm/data/<IID> with the ToStore flag true.
storage/request/<IID>	1	Request data by the Application Instance <IID>.	Any Application	Data Storage Application	{ "MaxLength": <Integer>, "StartTime": <Int64>, "EndTime": <Int64>, "PreferOldest": <Boolean>, "InstanceID": <IID> "SubTopic": <String>, "MinPriority": <Int>, "MaxPriority": <Int>, , "Timestamp": <Int64>, "Signature": {} }	The Data Storage Application will return the requested data on the storage response topic below. The request Payload keys are documented in Table 16.

Topic	QoS	Description	Sender	Receiver	Payload	Notes
storage/response/<IID>	1	The response from the data storage container after a get request by Application Instance <IID>.	Data Storage Application	The <IID> Application which requested the table	{ "Status": <Integer>, "Response": [{ "TableName": <IID>, "TableRows": [{ "ID":<Integer>, "Timestamp": <Int64>, "SubTopic": <string>, "Data": <JSON Object>, }, {more rows}] }] }	The response Payload keys are documented in Table 17.

Table 19 – Data Storage Container

Because of the automatic storing of Algorithm output described in Section 8.4, it will be unusual to need to explicitly store data using the "storage/data/" topic. Publishing on the "storage/data/<IID>" topic has exactly the same results as publishing on the "algorithm/data/<IID>" topic with the ToStore flag true.

The fields of the message on the request topic are documented in Table 16, and those of the response message on the response topic in Table 17. These objects are deliberately the same as those used by the Data Upload API. The same rules for InstanceID and SubTopic apply. Similarly, "storage/response/newdata/<IID>" and "storage/response/<IID>" use the same response format, although in this API there will only ever be data from one table and so only one element in the Response array. Note that the Instance ID <IID> in the topic names refers to the Application making the requests and receiving the data, not the table being accessed (except in the first topic documented, where they are the same).

9. JSON Object Structures

All messages passed through the Data Marketplace, and all Application Configuration data, is serialised as JSON objects. Whilst for some purposes bespoke JSON object structures are necessary, wherever possible use should be made of the standard JSON object structures defined in this section.

Using standard object structures ensures that data can be passed from any Application to any other Application without the need for bespoke software development. It minimises the need for Applications to cope with data from different sources in different formats. Applications which output in standard formats will be best placed to take advantage of facilities provided by LV-CAP and other Applications.

In applying these Object formats consideration should also be given to the general principals set out in Section 0.

9.1 Scalar Object Format

The default choice of JSON Object for almost all sensor readings and many algorithm outputs will be the Scalar Object. It represents a single value at a single point in time, for instance a temperature or a power flow. To provide more metadata about the value and how it was arrived at, a separate Data Series Metadata Object should be used (see Section 9.4).

```
{
    "Timestamp": <Int64>,
    "Value": < >,
    "Valid": <Boolean>,
    "ToStore": <Boolean>,
    "Priority": <Int>,
    "Signature": {}
}
```

Figure 8 - Scalar Object Format Structure

Key	Status	Description
Timestamp	Required	Standard LV-CAP timestamp (see Section 4.5) when the reading was made.
Value	Required	The reading, converted to base engineering units. The reading can be of any scalar type (Boolean, Integer or Floating Point).
Valid	Required	A logical value, showing if the Value is within the expected range (configured).
ToStore	Optional	Flag used historically to indicate whether the data should be stored by the Data Storage Application or not. May be over-ridden by the Data Storage Application configuration.
Priority	Optional	A priority indicator as in section 4.9, which allows the upload of certain messages to be prioritised by Data Upload Applications.
Signature	Reserved	See Section 8.1.2

Table 20 - Scalar Object Format Keys

The Value is always given in the base SI unit for the quantity being measured or calculated, as in Section 4.5. The units can be given explicitly in the optional Data Series Metadata Object (Section 9.4).

9.2 Series Object Format

Where a series of closely related values are to be sent as a set then a Series Object provides a way to package the complete set of values in a single JSON Object. It can represent a time series of values (anything from a fault waveform recorder (sampling many times per mains cycle) to a load profile (hourly load values), or a frequency spectrum. The object contains fields to record what range of source data was used to produce series. To provide more metadata about the value and how it was arrived at, a separate Data Series Metadata Object should be used (see Section 9.4).

```
{
    "Timestamp": <Int64>,
    "StartPoint": <Int64 or float>,
    "Interval" : <float>,
    "Value": [< >],
    "Confidence": [< >],
    "Valid": <Boolean>,
    "TimestampStart": <Int64>,
    "TimestampEnd": <Int64>,
    "ToStore": <Boolean>,
    "Priority": <Int>,
    "Signature": {}
}
```

Figure 9 - Scalar Object Format Structure

Key	Status	Description
Timestamp	Required	Standard LV-CAP timestamp (see Section 4.5) when the series was produced.
Interval	Required	Interval between values in the series. For time series, this is the time between samples in seconds (or decimals of them), for frequency spectrums Hertz and so on.
StartPoint	Required	The x-axis co-ordinate of the first value in the series. For time series this is the timestamp of the first value, for frequency spectrums the frequency of the first bin and so on.
Value	Required	An array of series values in base engineering units. The values can be of any scalar type (Boolean, Integer or Floating Point).
Confidence	Optional	An array, the same size as the Value array, of values giving the confidence in the values. This may be used to represent the uncertainties caused by missing input data or inadequate system history.
Valid	Required	A logical value, showing if the Series as a whole is thought to be valid for further use.
TimestampStart	Optional	The standard LV-CAP timestamp of the earliest source data used to produce this series.
TimestampEnd	Optional	The standard LV-CAP timestamp of the latest source data used to produce this series.
ToStore	Optional	Flag used historically to indicate whether the data should be stored by the Data Storage Container or not. May be over-ridden by the Data Storage Container configuration.
Priority	Optional	A priority indicator as in section 4.9, which allows the upload of certain messages to be prioritised by Data Upload Applications.
Signature	Reserved	See Section 8.1.2

Table 21 - Scalar Object Format Keys

This is deliberately an abstract data format designed to be capable of accommodating a wide range of different data types. The Value are always given in the base SI unit for the quantity being measured or calculated, as in Section 4.5. The units can be given explicitly in the optional Data Series Metadata Object (Section 9.4). Some practical examples of its use are given in Figure 10 and Figure 11.

Figure 10 shows the Series Object Format used for a predicted load profile.

- It was calculated and published at 14:30:05 UTC on 8th March 2017.
- The first predicted load segment in the prediction starts at 14:30:00 UTC on 8th March 2017
- The prediction is composed of half-hourly (1800 seconds) load current values.
- The prediction is for 4 hours, so has 8 values of load current in amps.
- The predictor is confident in the prediction for the first two hours and the last one, but is aware of limitations in the data for the third hour (e.g. because there are problems with missing data in that hour). These reduce the confidence in those predictions.
- Overall the predictor thinks that this data is valid for use.
- The prediction is built on the previous 4 weeks of data, so the oldest data used was from 14:30:00 UTC on 8th March 2017.
- The most recent data used was from 15:00:00 UTC on 1st March 2017, the end of the half hour period one week ago.
- The prediction is not signed.
- The predictor does not stipulate whether this data is to be stored or not.

```
{
    "Timestamp": 1488983405,
    "StartPoint": 1488983405,
    "Interval" : 1800.0,
    "Value": [120.0, 122.5, 125.7, 130.9, 124.8, 121.4, 118.6, 115.3],
    "Confidence": [1.0, 1.0, 1.0, 1.0, 0.85, 0.75, 1.0, 1.0],
    "Valid": true,
    "TimestampStart": 1486564200,
    "TimestampEnd": 1488380400
}
```

Figure 10 – Example of a Scalar Object used for a load prediction

Figure 11 shows the Series Object Format used for a harmonic spectrum.

- It was calculated and published at 11:00:00 UTC on 5th March 2017.
- The first harmonic in the spectrum is 50Hz
- The spectrum is composed of values for each harmonic, so every 50 Hz.
- The spectrum is for the first 5 harmonics only.
- The spectrum is calculated, so no confidence values are given.
- Overall the calculation succeeded, so the data is valid for use.
- The spectrum was calculated from the previous 10 minutes of data, so the oldest data used was from 10:50:00 UTC on 5th March 2017.
- The most recent data used was from 10:59:59 UTC on 5th March 2017, the end of the 10-minute period.
- The publishing container thinks that this data should be stored in the Data Storage Container.
- The publishing container has assigned this data the lowest available priority.
- The spectrum is not signed.

```
{
    "Timestamp": 1488711600,
    "StartPoint": 50,
    "Interval" : 50,
    "Value": [76423.0, 122.5, 86.7, 57.9, 12.4],
    "Valid": true,
    "TimestampStart": 1488711000,
    "TimestampEnd": 1488711599,
    "ToStore": true,
    "Priority": 5
}
```

Figure 11 – Example of a Scalar Object used for a harmonic spectrum

9.3 Co-ordinate Object Format

Where a group of co-ordinates are produced then a Co-ordinate Object provides a way to package them in a single JSON Object. The co-ordinates may be in a real space (e.g. Latitude and Longitude for geographic position) or a conceptual one (real and imaginary power in a power flow vector diagram). To provide more metadata about the value and how it was arrived at, a separate Data Series Metadata Object should be used (see Section 9.4).

```
{
    "Timestamp": <Int64>,
    "Coordinates": [< >],
    "System": <String>,
    "Valid": <Boolean>,
    "ToStore": <Boolean>,
    "Priority": <Int>,
    "Signature": {}
}
```

Figure 12 - Co-ordinate Object Format Structure

Key	Status	Description
Timestamp	Required	Standard LV-CAP timestamp (see Section 4.5) when the reading was made.
Coordinates	Required	Array of co-ordinate values, in base engineering units. The co-ordinate values will be Integer or Floating-Point values.
System	Optional	The co-ordinate system being used, e.g. Cartesian (for x-y plots) or WGS84 latitude and longitude.
Valid	Required	A logical value, showing if the Value is within the expected range (configured).
ToStore	Optional	Flag used historically to indicate whether the data should be stored by the Data Storage Container or not. May be over-ridden by the Data Storage Container configuration.
Priority	Optional	A priority indicator as in section 4.9, which allows the upload of certain messages to be prioritised by Data Upload Applications.
Signature	Reserved	See Section 8.1.2

Table 22 - Co-ordinate Object Format Keys

No implementation of this Object Format yet exists.

9.4 Data Series Metadata Object Format

There will be various pieces of metadata (that is, information about the data) associated with the data published on a given topic. It may be desirable to transmit these in a machine-readable format, so that consuming Applications can make use of them. However, this metadata does not change from reading to reading, so it would be inefficient to transmit (and especially store) them alongside the readings themselves. Instead a separate /meta/ sub-topic is used for metadata, which is transmitted as Data Series Metadata Objects.

The metadata objects are sent only when an Application starts, or there is a change to the metadata. In order that subscribing Applications always receive this information, the messages are published with the Retained flag set to true. This means that the Data Marketplace will automatically send a copy of the latest metadata to any new client which subscribes the /meta/ sub-topic, without any effort from the publishing Application.

```
{
    "Name": <String>,
    "Units": <String>,
    "DisplayUnits": <String>,
    "SigFigs": <Integer>,
}
```

Figure 13 - Data Series Metadata Object Format Structure

Key	Status	Description
Name	Required	String to be used as the name for this data, e.g. labels on graphs
Units	Optional	String giving the units of the data sent on the topic. As the data will be in base engineering units (Section 4.5), this just gives the correct SI unit, for instance amps or metres per second.
DisplayUnits	Optional	String giving the units for display of the data. This may include the use of SI prefixes for convenient display, e.g. kVAr for reactive power flow.
SigFigs	Optional	Integer indicating how many significant figures the values should be displayed to, to avoid the spurious display of excess decimal places caused by binary floating-point representation.

Table 23 - Data Series Metadata Object Format Keys

An example of the Data Series Metadata for a topic carrying a voltage measurement is shown in Figure 14.

```
{
    "Name": "Line1 to Neutral",
    "Units": "Vrms",
    "DisplayUnits": "Vrms",
    "SigFigs": 3
}
```

Figure 14 - Scalar Object Format Structure

9.5 Application Configuration Format

All configuration files must be valid JSON objects. Application configuration data will differ significantly from Application to Application depending on their structure, and only be of use to the Application it is intended for. To accommodate this, the structure of the configuration file for each container is largely up to the Application author, but a standard top-level structure is required in order to deliver updated configuration information to the correct container. Application authors are free to structure the ContainerConfig object within their configuration in whatever way suits their application, provided that it is a valid JSON object.

```
{
      "ContainerName": "<IID>",
      "ContainerConfig":
      {
            "<examplekey1>": <configvalue1>,
            "<examplekey2>": <configvalue2>,
            "<examplekeyN>": <configvalueN>
      }
}
```

Figure 15 - Third Party Container Configuration File Example

Key	Status	Description
ContainerName	Required	IID of the Application the configuration is for.
ContainerConfig	Required	JSON object containing the Application Instance configuration. This object's contents and structure will change from Application to Application.

Table 24 - Third Party Configuration File Keys

10. References

The following external resources provide more information to support this specification:

1. The MQTT Standard, version 3.1.1: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
2. ECMA Standard 404, "The JSON Data Interchange Format" http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
3. Blog Post "MQTT Topics & Best Practices" http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices
4. Docker Documentation for docker tag command: https://docs.docker.com/v1.12/engine/reference/commandline/tag/ and https://docs.docker.com/v17.03/engine/reference/commandline/tag/

Global Footprint

We provide products, services and support for customers in 90 countries, through our offices in Australia, China, Europe, Singapore, UAE and USA, together with more than 40 distribution partners.

Our Expertise

We provide world-leading asset management solutions for power plant and networks.

Our customers include electricity generation, transmission and distribution companies, together with major power plant operators in the private and public sectors.

- Our products, services, management systems and knowledge enable customers to:
- Prevent outages
- Assess the condition of assets
- Understand why assets fail
- Optimise network operations
- Make smarter investment decisions
- Build smarter grids
- Achieve the latest standards
- Develop their power skills

Safer, Stronger, Smarter Networks

www.eatechnology.com
Australia | China | Singapore | UAE | Europe | USA

Main reception: +44(0) 151 339 4181
EA Technology, Capenhurst Technology Park,
Capenhurst, Chester, CH1 6ES, United Kingdom

5 Appendix B – Nortech Application Container

Requirement Specification

Drawing Number	**2626-RQSPC-SHT04-V00.01.01 Nortech Comms Container Requirements.docx**
Project Title	**OpenLV: Nortech Comms Application Requirements**
Electronics Systems Project No.	**E717**
Charge Code	**EX267**
Supplier	**Nortech**
Request Date	**2017/07/01**

1 Background

EA Technology undertook an InnovateUK Energy Catalyst project with University of Manchester and Nortech Management Ltd to develop a Common Application Framework for LV Network Management. The core software platform developed by EA Technology under the project is called LV-CAP and consists of a number of Docker containers communicating with each other using MQTT.

WPD has now been awarded a Network Innovation Competition (NIC) project to trial the LV-CAP platform on real networks. This project is known as OpenLV. The OpenLV project will use a Nortech iHost server to store data from the OpenLV hardware, and as the management platform for the LV-CAP software running on the OpenLV hardware. To send data to and from the iHost server, a communications Application written by Nortech will be used. This Application was created as part of the Innovate UK LV-CAP project, but some enhancements are necessary to meet the requirements of the OpenLV project. This specification details the enhancements to the Nortech Comms Application required to meet the needs of the OpenLV Trial Programme.

2 Requirements

2.1 Hardware Environment

The OpenLV project hardware consists of an industrial PC based around a dual-core Intel Core i3 processor with 8GB of RAM and a 512GB SSD. This PC provides the processing power and storage for the whole LV-CAP solution. It has two Ethernet ports for network communications:

1. Local Ethernet link to the GridKey MCU520, fitted with an additional Ethernet module.
2. Ethernet link to a stand-alone 4G router which provides wide area network communications.

The Nortech Comms Application will communicate with the iHost sever via the wide area network.

2.1.1 Networking

See EATL drawing 2626-IMSPC-SHT03-V00.01.00 for the expected network architecture.

The volume of mobile data transferred must be managed to reduce the operating costs of the OpenLV system.

2.2 Base Operating System

The OpenLV project PC will be running 64-bit Ubuntu Server 16.04 LTS with current updates applied.

2.3 LV-CAP Environment

The Nortech Comms Application is a "core" Application on the LV-CAP Platform which fulfils the Management Comms and Data Upload roles. This is described in the Public API document 2383-MANUL-V04.02.07 and the Internal API document 2362-MANUL-V04.01.05, hereafter referred to as "the LV-CAP API".

2.3.1 API Implementation Status

The following features of the LV-CAP API are not expected to be implemented in time for the OpenLV project:

1. Individual message signing (section 8.1.2) will not be implemented.
2. Signing of Docker Image files will not be implemented.
3. Only one instance of each Application will be run (section 4.2) on LV-CAP.
4. As a result, Applications may continue to use legacy GUID identifiers.
5. To simplify TLS implementation, TLS keys and certificates will be built into Docker Image files. The end date of TLS certificates should be set beyond the end of the OpenLV project trials in September 2019. TLS implementation is mandatory.
6. The Priority feature of the data storage APIs will not be implemented, with all queries returning messages of all priorities. Applications are free to output Priority data, but it will not be parsed yet. Similarly requests may be made with Priority key values, but the key will be ignored.

2.3.2 Application Identification

The Vendor string for Nortech is "nortech". The Application Name for this Application is "commscontainer". Each release of the Nortech Comms Application should be tagged as described in section 4.2 of the LV-CAP API. For example the Docker tags would be:

nortech/commscontainer:0.1.0
nortech/commscontainer:0.1.2
nortech/commscontainer:0.1.3
nortech/commscontainer:1.0.0
nortech/commscontainer:1.0.3
nortech/commscontainer:1.2.0

These tags are important as they are used by Docker to load and run the Application on the LV-CAP. Incorrect tags may result in the incorrect version of the image being deployed and run. The tag of each released image must be documented along with the released Docker Image file.

2.4 Download and Management

No major changes are required to the Application download or configuration download parts of the Application.

The maximum size of a Docker Image which can be deployed via the iHost management system will be increased to 300MB.

2.5 Data Upload

2.5.1 Data Input

The Nortech Comms Application will obtain the data to be uploaded from the Data Storage Application via the Data Upload API (see section 8.5 of the LV-CAP API). Once data has been successfully uploaded to the iHost server it must be marked as uploaded in the database so that it is not re-transmitted in future.

2.5.1.1 Data Format

No Change

2.5.2 Data Destination

The data from each LV-CAP system running the Nortech Comms Application must be uploaded as a separate RTU (or multiple virtual RTUs) within the iHost server.

2.6 Configuration

The Nortech Comms Application must be configured via the standard LV-CAP configuration mechanism (see sections 8.2.1 and 9.5 of the LV-CAP API). The configuration is likely to be altered in the course of the OpenLV Trials, so the configuration settings available must be documented alongside the Application.

The configuration is expected to cover the following areas:

- iHost server settings (included where to send the data, and authentication settings).
- Data Selection settings, i.e. which topics are to be uploaded to the iHost server.
- (Optionally) Where data is to be placed in the iHost structure.

2.7 Security

2.7.1 Authentication

The Nortech Comms Application and the iHost server must mutually authenticate each other so that only authorised data uploads occur, and Man-in-the-Middle attacks are prevented.

2.7.2 Confidentiality

Measures must be taken to ensure that the data uploaded remains confidential in transit, to comply with the OpenLV Project Data Protection Strategy.

2.7.3 Audit

As part of the OpenLV Project, a Cyber-Security review of the LV-CAP™ platform and Applications deployed within the project is to be undertaken. The Cyber-Security supplier will be undertaking an audit of the LV-CAP™ platform and it should be expected that this will include an audit of the software Application and associated documentation created by Nortech as part of the project.

3 Timescales

Delivery Date	**2017/07/22**
Prepared by	**Richard Ash**
Date	**2017/07/01**

6 Appendix C – Lucy Electric Application Container

Requirement Specification

Drawing Number	**2626-RQSPC-SHT02-V00.02.01 Lucy Gridkey Sensor Container Requirements.docx**
Project Title	**OpenLV: Lucy GridKey Sensor Container Requirements**
Electronics Systems Project No.	**E717**
Charge Code	**EX267**
Supplier	**Lucy Gridkey**
Request Date	**2017/06/05**

1 Background

EA Technology undertook an InnovateUK Energy Catalyst project with University of Manchester and Nortech Management Ltd to develop a Common Application Framework for LV Network Management. The core software platform developed by EA Technology under the project is called LV-CAP and consists of a number of Docker containers communicating with each other using MQTT.

WPD has now been awarded a Network Innovation Competition (NIC) project to trial the LV-CAP platform on real networks. This project is known as OpenLV. The OpenLV project will use the proven Lucy GridKey MCU520 measurement unit to make electrical measurements of the substation load. To integrate the GridKey MCU520 into the LV-CAP platform Lucy GridKey will create a sensor container for the LV-CAP platform to receive data from the GridKey MCU. This specification details the LV-CAP GridKey Sensor Container required to meet the needs of the OpenLV Trial Programme.

2 Requirements

2.1 Hardware Environment

The OpenLV project hardware consists of an industrial PC based around a dual-core Intel Core i3 processor with 8GB of RAM and a 512GB SSD. This PC provides the processing power and storage for the whole LV-CAP solution. It has two Ethernet ports for network communications:

1. Local Ethernet link to the GridKey MCU520, fitted with an additional Ethernet module.
2. Ethernet link to a stand-alone 4G router which provides wide area network communications.

The GridKey Sensor Container will communicate directly with the GridKey MCU520 via the local Ethernet port. It will not have access to the wide area communications network.

2.2 Base Operating System

The OpenLV project PC will be running 64-bit Ubuntu Server 16.04 LTS with current updates applied.

2.3 LV-CAP Environment

The GridKey Sensor Container must run as a "third party" container on the LV-CAP Platform. This is described in the Public API document 2383-MANUL-V04.02.00, hereafter referred to as "the LV-CAP API".

In order to be deployed via the iHost management system, the maximum size of the GridKey Sensor Container as an uncompressed TAR file is 100MB.

The GUID assigned to this container is "96d6f19b-7022-45f2-b753-cb5012626b4d"

The Docker "repository" string assigned to this container is "lucy/gridkeysensor". Each release of the GridKey Sensor Container should be tagged with this repository string and its version number, separated by a colon. The version number must monotonically increase with each release. For example:

lucy/gridkeysensor:0.1
lucy/gridkeysensor:0.2
lucy/gridkeysensor:1.0
lucy/gridkeysensor:1.1
lucy/gridkeysensor:1.3
lucy/gridkeysensor:2.0

These tags are important as they are used by Docker to load and run the Containers on the LV-CAP. Incorrect tags may result in the incorrect version of the container being deployed and run. The tag of each released container must be documented along with the released Container file.

2.4 Electrical Measurements

The following electrical measurements must be made available to the LV-CAP platform. Each measurement point should be updated at intervals of 10 seconds or less.

The three phases shall be designated "L1", "L2" and "L3" as on the MCU520 hardware. The current measurement channels shall be designated "Feeder1" through "Feeder5" as on the MCU520 hardware.

Other measurement data may be included if available.

2.4.1 Voltage Measurements

At the substation busbars:

- RMS Voltage phase to phase (x3)
- RMS Voltage phase to neutral (x3)

2.4.2 Current Measurements

For each circuit measured:

- RMS current in each phase
- Power factor for each phase
- Real and Reactive power flow each phase (including direction, so reverse power is read as negative current)

2.5 Output Messages

Each of the above measurements must be output as JSON messages on a separate MQTT topic, as described in the Sensor Data API (Section 8.3 of the LV-CAP API). The format of each JSON message shall be the LV-CAP Scalar Object

Format as described in Section 9.1 of the LV-CAP API. The output topic names should be chosen as described in Section 4.4 of the LV-CAP API. A suggested set of topic names is given in Section 4. of this document.

Output messages must be produced at all times as described in Section 4.8 of the LV-CAP API, with the valid flag set appropriately if there is a problem receiving data from the GridKey MCU.

The container may optionally provide measurement metadata as set out in Section 8.3.2 of the LV-CAP API.

2.6 Time

The LV-CAP system clock will be kept set accurately by LV-CAP using a combination of Network Time Protocol and GPS time reference information. The messages output from the GridKey Sensor Container must contain time stamps which are synchronised to this clock to avoid confusion.

2.7 Configuration

The GridKey Sensor Container must be configured via the standard LV-CAP configuration mechanism and a JSON configuration file as described in Section 9.5 of the LV-CAP API. This file can be used to store any relevant configuration parameters for the container. The following parameters must be configurable:

- The IP address of the GridKey MCU.

2.8 GridKey Firmware Update

The implementation of functionality to enable the firmware of the GridKey MCU to be updated in the field would be desirable, but is not required at this stage of the project. It would be acceptable to implement this feature in a future update to the Gridkey Sensor Container. The remote updating of containers on the LV-CAP platform is a core feature which will be proven before deployment of the system, so such an update can readily be deployed to operational LV-CAP systems.

In order to update the firmware on the MCU, two things need to happen:

1. The new firmware file is downloaded over the wide area network to the LV-CAP platform
2. The firmware is transmitted over the local Ethernet connection from the LV-CAP computer to the MCU.

The GridKey Sensor Container can access local file storage on the LV-CAP computer and the Ethernet link to the MCU. The second step (updating the MCU from the local file) should be carried out by the Gridkey Sensor Container.

The GridKey Sensor Container will not have access to the wide area network and so should not perform the first step (downloading firmware). There are several options for delivering the firmware update to the LV-CAP platform:

- The firmware file is packed within the GridKey Sensor Container image. When new firmware needs to be deployed, a new release of the GridKey Sensor Container is made and deployed via the normal LV-CAP mechanism. This will increase the size of the Container image however.
- The firmware file is separately downloaded via the normal LV-CAP mechanism and made available to the GridKey Sensor Container for installation. This would require changes to the LV-CAP APIs and so will be difficult to achieve for this project.
- The firmware file is downloaded by the Data Centre Communications Container, and transferred from that container to the GridKey Sensor Container for installation. This requires a mechanism for the two containers to transfer files between them to be designed and implemented, but does not affect the LV-CAP core.

3 Timescales

Delivery Date	**2017/06/23**
Prepared by	**Richard Ash**
Date	**2017/06/08**

4 Proposed Output Topics

The following output topic names are proposed to comply with the requirements in Section 2.5 of this document. They are based on the assumption that RMS values will be output once per second, and will need to be adjusted if this is not the case.

For the measurements in Section 2.4.1 of this document:

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L1-N_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L2-N_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L3-N_RMS

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L1-L2_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L2-L3_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Busbar/voltage/1s/L3-L1_RMS

For the measurements in Section 2.4.2 of this document:

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/current/1s/L1_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/current/1s/L2_RMS
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/current/1s/L3_RMS

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/power_factor/1s/L1
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/power_factor/1s/L2
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/power_factor/1s/L3

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/real_power/1s/L1
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/real_power/1s/L2
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/real_power/1s/L3

sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/reactive_power/1s/L1
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/reactive_power/1s/L2
sensor/data/96d6f19b-7022-45f2-b753-cb5012626b4d/Feeder1/reactive_power/1s/L3

and similarly for each of the 5 feeders supported by the hardware.