UWC Software from Intel

User Guide for UWC

April 2021

Revision 1.38

The content of this document is still under validation.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: [http://](http://www.intel.com/design/literature.htm)www.intel.com/design/literature.htm

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at h[ttp://www.intel.com/](http://www.intel.com/) or from the OEM or retailer.

No computer system can be absolutely secure.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. (“products”) in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as “commercial” names for products. Also, they are not intended to function as trademarks.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Intel is under license.

\*Other names and brands may be claimed as the property of others.

Copyright © 2021, Intel Corporation. All rights reserved.

Contents

[1.0 Introduction 5](#_Toc69227952)

[1.1 Purpose 5](#_Toc69227953)

[1.2 Scope 5](#_Toc69227954)

[1.3 System Requirements 5](#_Toc69227955)

[2.0 Before Installation 7](#_Toc69227956)

[2.1 UWC for O&G site 7](#_Toc69227957)

[2.2 Upstream Oil and Gas Facilities 7](#_Toc69227958)

[2.3 Understanding UWC Platform 9](#_Toc69227959)

[2.3.1 Modbus containers 10](#_Toc69227960)

[1. Modbus RTU master container 10](#_Toc69227961)

[2.3.2 MQTT-Bridge container 11](#_Toc69227962)

[2.3.3 Sparkplug-Bridge container 11](#_Toc69227963)

[1. SparkPlug MQTT Topic Namespace 11](#_Toc69227964)

[2. Supported message types 12](#_Toc69227965)

[3. Name of edge node 13](#_Toc69227966)

[2.3.4 KPI Application container 13](#_Toc69227967)

[2.3.5 Configurations 13](#_Toc69227968)

[3.0 Installation Guide 14](#_Toc69227969)

[3.1 How to install UWC software with EII installer 14](#_Toc69227970)

[4.0 Container Configuration settings 18](#_Toc69227971)

[4.1 Containers 18](#_Toc69227972)

[4.1.1 Common Configuration for Modbus Client Containers 18](#_Toc69227973)

[4.1.2 Configuration for Modbus Network 19](#_Toc69227974)

[4.2 Modbus TCP Communication 20](#_Toc69227975)

[4.3 Modbus RTU Communication 20](#_Toc69227976)

[4.4 MQTT Bridge 21](#_Toc69227977)

[4.5 MQTT 21](#_Toc69227978)

[4.5.1 Accessing secured MQTT container from an external MQTT client 21](#_Toc69227979)

[4.5.2 Accessing secured MQTT container from a client inside a container 23](#_Toc69227980)

[4.6 Sparkplug-Bridge 24](#_Toc69227981)

[4.6.1 Pre-requisite for running Sparkplug-Bridge 24](#_Toc69227982)

[4.7 KPI App 25](#_Toc69227983)

[5.0 Site Configurations 26](#_Toc69227984)

[5.1 System Level Global Configuration 26](#_Toc69227985)

[5.1.1 Settings for Polling and On-Demand operation 26](#_Toc69227986)

[5.1.2 Settings for Sparkplug-Bridge – SparkPlug communication operation 28](#_Toc69227987)

[5.2 How to Configure Site and Wellhead 28](#_Toc69227988)

[5.2.1 Configuring TCP device in device-group 29](#_Toc69227989)

[5.2.2 Configuring RTU device in device-group 30](#_Toc69227990)

[5.3 How to Configure Devices 31](#_Toc69227991)

[5.4 How to Configure Device points 31](#_Toc69227992)

[5.5 How to add/edit/delete new wellhead/device/point configurations 37](#_Toc69227993)

[5.6 KPI App Configuration 37](#_Toc69227994)

[6.0 UWC Modbus Operations 40](#_Toc69227995)

[6.1 Data Polling 43](#_Toc69227996)

[6.2 On-Demand Write 44](#_Toc69227997)

[6.3 On-Demand Read 46](#_Toc69227998)

[6.4 KPI Application 48](#_Toc69227999)

[7.0 Sparkplug-Bridge Operations 51](#_Toc69228000)

[7.1 App (virtual device) communication 51](#_Toc69228001)

[7.1.1 App Message Topic Format 51](#_Toc69228002)

[7.1.2 App Message - BIRTH 52](#_Toc69228003)

[7.1.3 App Message - DATA 54](#_Toc69228004)

[7.1.4 App Message - CMD 56](#_Toc69228005)

[7.1.5 App Message - DEATH 57](#_Toc69228006)

[7.1.6 App Message - TemplateDef 58](#_Toc69228007)

[7.1.7 START\_BIRTH\_PROCESS 60](#_Toc69228008)

[7.2 Modbus (real) device communication 60](#_Toc69228009)

[7.2.1 Support for DBIRTH 60](#_Toc69228010)

[7.2.2 Support for DDATA 61](#_Toc69228011)

[7.2.3 Support for DCMD 61](#_Toc69228012)

[7.2.4 Support for DDEATH 61](#_Toc69228013)

[7.3 SparkPlug Messages 61](#_Toc69228014)

[7.3.1 NBIRTH Message 61](#_Toc69228015)

[7.3.2 NDEATH Message 63](#_Toc69228016)

[7.3.3 DBIRTH Message 63](#_Toc69228017)

[7.3.4 DDEATH Message 65](#_Toc69228018)

[7.3.5 DDATA Message 65](#_Toc69228019)

[7.3.6 NCMD Message 66](#_Toc69228020)

[8.0 Debugging steps 67](#_Toc69228021)

[9.0 Error Codes 69](#_Toc69228022)

[10.0 Steps to flash LFM BIOS 71](#_Toc69228023)

[11.0 Steps to Apply RT Patch 73](#_Toc69228024)

[11.1 Install prerequisites 73](#_Toc69228025)

[11.2 Keyboard shortcuts for menuconfig UI 73](#_Toc69228026)

[11.3 Steps to apply PREEMPT\_RT patch 73](#_Toc69228027)

[11.4 References 78](#_Toc69228028)

[12.0 Appendix 79](#_Toc69228029)

# Introduction

The Intel® UWC is a reference design for a secured management platform that gives third party application developers an easy access to data services including data collection from field devices, control data pathways, and connections to centralized data systems (i.e. SCADA) for Upstream Oil and Gas facilities including gas well sites.

The Intel UWC platform will provide a secure, management platform for oil and gas upstream process monitoring and control to support oil and gas installations with various artificial lift methods such as plunger lift, gas lift, gas-assisted plunger lift, rod-beam and electronic submersible pump (ESP).

Intel’s primary objective in this market is to move the Upstream Oil and Gas vendors, service providers, and end-users to adopt Intel-based hardware hosting a rich range of open-architecture software-defined platforms. Solution is targeted to address multiple pain areas O&G industry is facing in day-to-day operations. These pain areas are further restricting O&G industry to get benefitted from technology advancements resulting from cloud-based services and applications for business intelligence (BI), analytics, dashboards, etc. There is a need to provide a uniform mechanism to connect, monitor and control various devices in an O&G well-site adhering to real-time nature of the industry.

While the Intel UWC software solution described in this Users’ Guide contains a data model specific to a Gas Wellpad, the software is flexible and can be configured for use with other soft-RT process control sites and operating assets.

## Purpose

This document provides information on how to install UWC software framework and configure the device models, data model and data flows for data collection (from field devices) and reporting (such as to SCADA) at remote process control sites such as a natural gas well Oil (Wellpad).  The document will help the developer to evaluate the performance of the data collection processes using an applet called the   
‘KPI Application’.  The document will; help the developer to set up the security and manageability services.

## Scope

This document aims to provide steps to build, deploy and check how gateway up and running all containers. This document also provides steps to set UWC containers for communication with Modbus devices.

## System Requirements

* Intel processor family
* 4GB memory
* 25GB hard disk space
* Ubuntu 18.04 server operating system with RT Patch (instructions for patching the Ubuntu OS are provided)
* Docker >=19.03.8
* docker-compose version >=1.24.0

# Before Installation

This section explains various concepts used throughout this document.

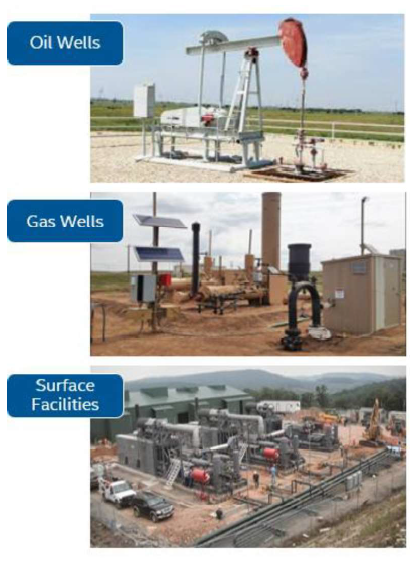
## UWC for O&G site

The UWC is a reference design that provides a secure, management platform for oil and gas upstream process monitoring and control to support oil and gas installations with various artificial lift methods such as plunger lift, gas lift. UWC is a profile on Intel Edge Insights (**EII**) base platform.

UWC provides

* Soft-real time deterministic control (millisecond level control) in the containerized environment
* Configurable user-defined data model describing Oil well site
* Modularized microservices-based extensible open architecture
* Well defined message flow and payload to interact with the vendor application
* Policy-based execution of Soft-RT applications
* Supports multiple well pads and devices configurations
* Polls and controls multiple devices simultaneously
* Data Publish Subscribe ZeroMQ and MQTT
* Device Management – System Telemetry, over-the-network (aka over-the-air, or OTA) updates of firmware (FOTA), system software (SOTA), and applications (AOTA)
* Scalable-down to power-sensitive remote applications
* Scalable-up to edge analytics and vision through the EII base

## Upstream Oil and Gas Facilities



The example provided for UWC in this Guide is for a natural gas wellhead with one or more wells.  The wellhead will have several field devices often using the Modbus protocols (both TCP and RTU). A Modbus device will have number of points to be read and/or written.  The table below shows how wellhead, device, and points are related to each other.

|  |  |  |
| --- | --- | --- |
| **Wellhead** | **Device** | **Point** |
| WellHead1 | flowmeter1 | KeepAlive |
| Flow1 |
| iou | AValve |
|  | | |
| WellHead2 | flowmeter2 | KeepAlive |
|  | | |
| WellHead3 | iou | BValve |

There could be similar devices and similar points in different wellheads. Hence, UWC uses this hierarchy to uniquely name a point. A point is identified like “/device/wellhead/point” e.g. flowmeter1/WellHead1/KeepAlive

UWC defines a data model which can be used to describe the hierarchy of wellhead, device and points.

Site info: List of device groups (i.e. wellheads): Devices\_group\_list.yml

Device group info (e.g. wellhead) - List of devices in a group: Devices\_group.yml

Device info:

iou\_device.yml, flowmeter\_device.yml

Network info - Parameters related to TCP network and RTU networks: E.g. RTU\_network.yml, TCP\_network.yml

Point information: Register map, polling frequency, realtime :

iou\_datapoints.yml, flowmeter\_datapoints.yml

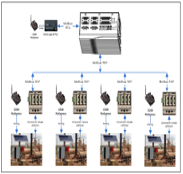
File format: YAML- Human readable

Each device may monitor and control points with different attributes

Site can have more than one device group

A device group can have more than one device

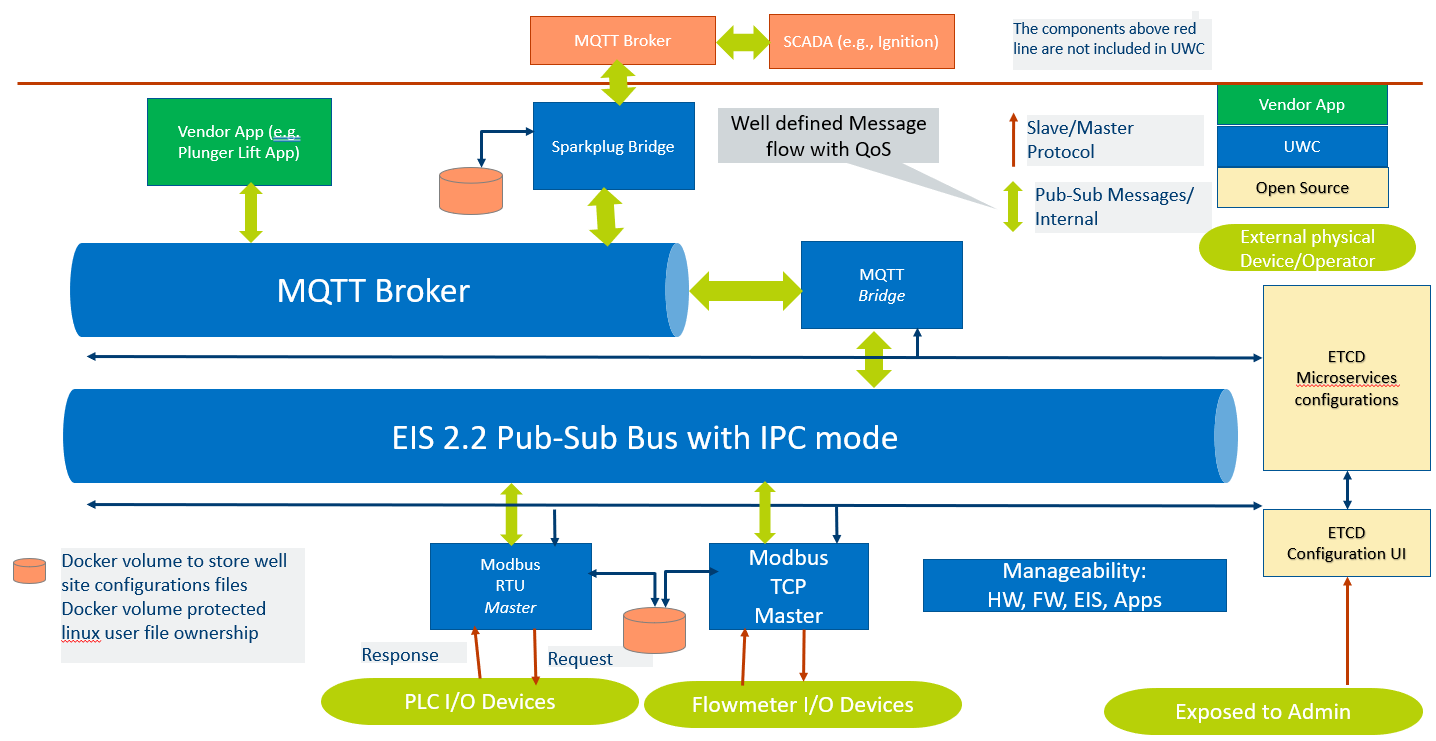
One device group can have multiple networks. Each device in device group will be assigned one network. One network can have multiple devices.



Site Configurations

## Understanding UWC Platform

Following is a high-level block diagram of UWC:



The application can subscribe to MQTT topics to receive polled data. Similarly, the application can publish data to be written on MQTT. The platform will accordingly publish or subscribe to respective topics. The MQTT topics to be used can be configured.

Internally, UWC platform uses a message bus (called ZMQ) for which the topics need to be configured. ZMQ is not shown above for ease of understanding.

### Modbus containers

UWC supports Modbus TCP master and Modbus RTU master for communicating with Modbus slave devices present in a field. These are developed as two separate containers i.e. Modbus TCP container and Modbus RTU container. Please refer diagram in section 2.3.

#### Modbus RTU master container

Modbus RTU devices can be connected using RS485 or RS232. Normally, with RS232, only one device is connected at one time. Hence, to communicate with two Modbus RTU devices over RS232, two different serial ports will be needed.

Modbus RTU protocol with RS485 physical transport uses a twisted pair of wires in a daisy-chain shared media for all devices on a chain. The communications parameters of all devices on a chain should be the same, so if, different devices have different configuration (e.g. different parity, different baud rate, etc.), then, different Modbus RTU chains can be formed. To communicate with two different Modbus RTU networks, two different serial ports will be needed. It is important to verify the analog signal integrity of the RS-485 chains including the use of termination resistors as per well-known RS-485 best-practices.

In UWC, one Modbus RTU master can be configured to communicate over multiple serial ports. Hence a single Modbus RTU master container handles communication with multiple Modbus RTU networks. The configuration for one Modbus RTU network (e.g. port, baud rate, etc.) can be configured in a RTU network configuration file, explained in later section of this document.

### MQTT-Bridge container

Modbus containers communicate over ZMQ. The MQTT-Bridge module enables communication with Modbus containers using MQTT. The MQTT- Bridge module reads data on ZMQ received from Modbus containers and publishes that data on MQTT. Similarly, the MQTT- Bridge module reads data from MQTT and publishes it on ZMQ.

This module was earlier known as MQTT-Export.

### Sparkplug-Bridge container

UWC supports Eclipse Foundation’s SparkPlug standard to expose data to SCADA Master over MQTT. Sparkplug-Bridge implements the standard and enables communication with SCADA Master. This module was earlier known as SCADA-RTU.

This module exposes the data on the platform to an external, centralized, master system for the SCADA:

* Data from base UWC platform i.e., real devices
* Mechanism to expose data from Apps running on UWC i.e., virtual devices

#### SparkPlug MQTT Topic Namespace

Following is the topic format:

spBv1.0/group\_id/message\_type/edge\_node\_id/[device\_id]

##### group\_id

The group\_id element of the Sparkplug™ Topic Namespace provides for a logical grouping of MQTT EoN nodes into the MQTT Server and back out to the consuming MQTT Clients. The value should be descriptive but as small as possible.

The value of the group\_id can be valid UTF-8 alphanumeric string. The string shall not use the reserved characters of ‘+’ (plus), ‘/’ (forward slash), and ‘#’ (number sign).

The value of this field can be configured in a configuration file ([link](#_Settings_for_SCAD-RTU)).

##### message\_type

The message\_type elements are defined for the Sparkplug™ Topic Namespace. The values could be:

* NBIRTH – Birth certificate for MQTT EoN nodes.
* NDEATH – Death certificate for MQTT EoN nodes.
* DBIRTH – Birth certificate for Devices.
* DDEATH – Death certificate for Devices.
* NDATA – Node data message.
* DDATA – Device data message.
* NCMD – Node command message.
* DCMD – Device command message.
* STATE – Critical application state message.

##### edge\_node\_id

The edge\_node\_id element of the Sparkplug™ Topic Namespace uniquely identifies the MQTT EoN node within the infrastructure. The group\_id combined with the edge\_node\_id element must be unique from any other group\_id/edge\_node\_id assigned in the MQTT infrastructure. The topic element edge\_node\_id travels with every message published and should be as short as possible.

The value of the edge\_node\_id can be valid UTF-8 alphanumeric string. The string shall not use the reserved characters of ‘+’ (plus), ‘/’ (forward slash), and ‘#’ (number sign).

The value of this field can be configured in a configuration file ([link](#_Settings_for_SCAD-RTU)).

##### device\_id

The device\_id element of the Sparkplug™ Topic Namespace identifies a device attached (physically or logically) to the MQTT EoN node. The device\_id must be unique from other devices connected to the same EoN node. The device\_id element travels with every message published and should be as short as possible.

The format of the device\_id is a valid UTF-8 alphanumeric String. The string shall not use the reserved characters of ‘+’ (plus), ‘/’ (forward slash), and ‘#’ (number sign).

#### Supported message types

Following message types are supported in current version of UWC:

|  |  |  |
| --- | --- | --- |
| **Message Type** | **Support for real device** | **Support for virtual device (apps)** |
| NBIRTH | Supported. This is an edge level message. | |
| NDEATH | Supported. This is an edge level message. | |
| DBIRTH | Supported. Data is taken from YML file. | Supported. Vendor app should publish data on “BIRTH” topic. |
| DDATA | Supported. Data from Poll-update messages is taken to determine change in data for publishing a DDATA message | Supported using RBE (Report By Exception). Vendor app should publish data on “DATA” topic. |
| DCMD | Supported. A corresponding On-Demand-Write request message is published on internal MQTT for other UWC containers to process a request. | Supported. A corresponding CMD message is published on internal MQTT for vendor app. |
| DDEATH | Supported. Data from Poll-update messages is taken to determine change in data for publishing a DDEATH message in case of error scenarios | Supported. Vendor app should publish data on “DEATH” topic. |
| NDATA | Not Supported | |
| NCMD | Supported “Node Control/Rebirth” control | |
| STATE | Not Supported | |

#### Name of edge node

User should properly configure “group\_id” and “edge\_node\_id” for each edge gateway deployed in a site such that each edge node can be uniquely identified.

### KPI Application container

One sample application called as “KPI Application” is provided to depict how one can develop an application on UWC platform. This is a simple application which demonstrates how “single input, single output” control loop can be implemented.

A control loop is executed continuously to monitor certain parameter and the adjust other parameters. Thus, a control loop consists of one read operation and one write operation. In this sample application, polling mechanism of UWC platform is used to receive values of parameters as per polling interval. The application uses “on-demand-write” operation on receiving data from polling.

This KPI Application can either be executed based on MQTT communication or based on ZMQ communication. Please refer configurations for more details.

The KPI Application also logs all data received as a part of control loop application in a log file. This data can be used for measuring performance of the system.

### Configurations

UWC needs following configuration to function properly:

* Information about device group list (i.e., wellhead), device and points falling under respective Modbus container
* Information about topics for internal message queue, publishers and subscribers

All these configurations are related and depend on the hierarchy of wellhead, device and point.

Following sections detail the UWC installation, configuration process.

# Installation Guide

## How to install UWC software with EII installer

This section provides steps to install and Deploy UWC containers using EII installer

**Pre-requisite:** Internet connection (With proper Proxy Settings, if any) is required for Installation.

**Steps:**

1. Install Ubuntu 18.04 server version on gateway and Apply RT Patch (refer section 11).
2. Git clone in following order will place the directories in proper relative directory structure
   1. $ mkdir new\_directory
   2. $ cd new\_directory
   3. $ git clone https://github.impcloud.net/uwc/eii-core.git IEdgeInsights
   4. $ git clone https://github.impcloud.net/uwc/UWC-Core.git IEdgeInsights/uwc
   5. $ git checkout eii\_2.4
   6. $ git clone https://github.impcloud.net/uwc/eii-zmq-broker.git IEdgeInsights/EISZmqBroker
3. Navigate to $ <working-dir>/IEdgeInsights/build
4. Execute Command $ sudo ./pre\_requisites.sh
5. Navigate to IEdgeInsights/uwc
6. Execute Command $ sudo ./01\_uwc\_pre\_requisites.sh
7. Execute Command $ sudo ./02\_provision\_UWC.sh

It prompts below options –

* Please choose one of the below options based on Dev or Prod mode.

1) Dev

2) Prod

* Please choose one of the below options based on the use case (combination of UWC services) needed.

1) Basic UWC micro-services without KPI-tactic Application & Sparkplug-Bridge - (Modbus-master TCP & RTU, mqtt-bridge, internal mqtt broker, ETCD server, ETCD UI & other base EIS & UWC services)

2) Basic UWC micro-services as in option 1 along with KPI-tactic Application (Without Sparkplug-Bridge)

3) Basic UWC micro-services & KPI-tactic Application along with Sparkplug-Bridge

4) Basic UWC micro-services with Sparkplug-Bridge and no KPI-tactic Application

Following is a sample output for Sparkplug-Bridge related configuration:

* Enter the following parameters required for sparkplug-bridge container..

Is TLS required for sparkplug-bridge (yes/no): yes

Enter the CA certificate full path including file name:/home/ubuntu/new/ca/root-ca.crt

Enter the client certificate full path including file name: /home/ubuntu/new/client/client.crt

Enter the client key certificate full path including file name: /home/ubuntu/new/client/client.key

Enter the external broker address/hostname:192.168.1.11

Enter the external broker port number: 22883

Enter the QOS for scada (between 0 to 2): 1

* Enter the following parameters required for sparkplug-bridge container

Is TLS required for sparkplug-bridge (yes/no): no

Enter the external broker address/hostname:192.168.1.11

Enter the external broker port number: 22883

Enter the QOS for scada (between 0 to 2): 1

1. Execute Command $ sudo ./03\_Build\_Run\_UWC.sh

Above is a process for interactive mode. A non-interactive mode is also supported. Following are the details:

1. To support non-interactive mode, following options are added to 2nd script (02\_provision\_UWC.sh).

|  |  |  |
| --- | --- | --- |
| **Argument** | **Values** | **Description** |
| --deployMode | dev or prod | Deployment mode to be used. |
| --recipe | 1 or 2 or 3 or 4 | Recipe file to be referred for provisioning: 1: Basic UWC micro-services without KPI-tactic app & Sparkplug-Bridge 2: Basic UWC micro-services as in option 1 along with KPI-tactic app 3: Basic UWC micro-services & KPI-tactic app along with Sparkplug-Bridge 4: Basic UWC micro-services with Sparkplug-Bridge and no KPI-tactic |
| --isTLS | yes or no | This option is applicable for Sparkplug-Bridge (i.e., value of "--recipe" is 3 or 4). It tells whether communication with external broker is secured or not. |
| --cafile | File path | This option is applicable for Sparkplug-Bridge (i.e., value of "--recipe" is 3 or 4) and when "--isTLS" is "yes". Root CA file path |
| --crtfile | File path | This option is applicable for Sparkplug-Bridge (i.e., value of "--recipe" is 3 or 4) and when "--isTLS" is "yes". Client certificate file path |
| --keyFile | File path | This option is applicable for Sparkplug-Bridge (i.e., value of "-- recipe" is 3 or 4) and when "--isTLS" is "yes". Client key file path |
| --qos | 0 or 1 or 2 | This option is applicable for Sparkplug-Bridge (i.e., value of "-- recipe" is 3 or 4). It tells QOS value to be used for MQTT communication. |
| --brokerAddr | String | This option is applicable for Sparkplug-Bridge (i.e., value of "-- recipe" is 3 or 4). It tells address to be used for external broker communication. |
| --brokerPort | Number | This option is applicable for Sparkplug-Bridge (i.e., value of "-- recipe" is 3 or 4). It tells port number to be used for external broker communication. |

If required parameters are missing, then those will be requested from user in an interactive mode.

1. Following are sample commands for non-interactive mode execution.

All UWC basic modules (no KPI, no Sparkplug-Bridge)

sudo ./02\_provision\_UWC.sh --deployMode=dev --recipe=1

All UWC modules (with KPI and with Sparkplug-Bridge).

sudo ./02\_provision\_UWC.sh --deployMode=dev --recipe=3 --isTLS=yes --caFile="scada\_ext\_certs/ca/root-ca.crt" --crtFile="scada\_ext\_certs/client/client.crt" --keyFile="scada\_ext\_certs/client/client.key" --brokerAddr="192.168.1.11" --brokerPort=22883 --qos=1

# Container Configuration settings

This section provides details about configuring UWC containers.

## Containers

Following containers are developed under UWC:

* Modbus TCP Master
* Modbus RTU Master
* MQTT-Bridge
* MQTT
* Sparkplug-Bridge
* KPI Application

For configuring these containers, docker-compose.yml file is used. The docker-compose.yml is auto generated based on inputs provided while executing script 02.

For more details, please refer README provided in EII documentation.

UWC containers (i.e., Modbus Clients and MQTT-Bridge) use ZeroMQ to communicate with each other.

At present, recommendation is to have one Modbus-TCP and one Modbus-RTU container. Hence, changes to configuration present in docker-compose.yml file shall not be needed. However, this information is provided in section 4.1.1 and 4.1.2 for completeness. To add support for multiple Modbus RT chains, please refer section 4.1.3.

### Common Configuration for Modbus Client Containers

Following are configurations, applicable for Modbus Client (TCP and RTU both) containers:

|  |  |
| --- | --- |
| **Configuration Parameter** | **Description** |
| MY\_APP\_ID | Unique ID assigned to a container. It can have values from 0 to 15. |
| CUTOFF\_INTERVAL\_PERCENTAGE | Modbus container will send a Bad response if a response is not received from end device for a point in a polling cycle during this cutoff time. (See section 6 for Message formats including Bad messages and see section 8 for error codes associated with Bad responses.)  The cutoff time is defined in terms of percentage of a polling interval. Default value is 90%. So, if polling interval is 1000 ms then cutoff time is 900 ms. This configuration allows to change default setting. |
| DEVICES\_GROUP\_LIST\_FILE\_NAME | Name of devices group list file to be used to obtain information of wellheads, device, datapoints handled by this container. |

Example for Modbus-TCP-Master container from docker-compose.yml file:

AppName: "TCP"

ETCD\_HOST: ${ETCD\_HOST}

ETCD\_CLIENT\_PORT: ${ETCD\_CLIENT\_PORT}

ETCD\_PREFIX: ${ETCD\_PREFIX}

DEV\_MODE: ${DEV\_MODE}

no\_proxy: ${eii\_no\_proxy}

Log4cppPropsFile: "/opt/intel/config/log4cpp.properties"

MY\_APP\_ID: 1

CUTOFF\_INTERVAL\_PERCENTAGE: 90

CertType: "zmq"

PROFILING\_MODE: ${PROFILING\_MODE}

NETWORK\_TYPE: TCP

DEVICES\_GROUP\_LIST\_FILE\_NAME: "Devices\_group\_list.yml"

### Configuration for Modbus Network

A separate network configuration YML file is maintained for each network. E.g., If there are 2 RTU and 1 TCP networks, then there will be 3 network configuration files. This file contains following configuration for both TCP and RTU:

# inter-frame delay and response timeout values are in Millisecond

interframe\_delay: 1

response\_timeout: 80

For Modbus RTU master, following additional configurations are needed apart from above mentioned parameters:

baudrate: 9600

parity: "N"

com\_port\_name: "/dev/ttyS0"

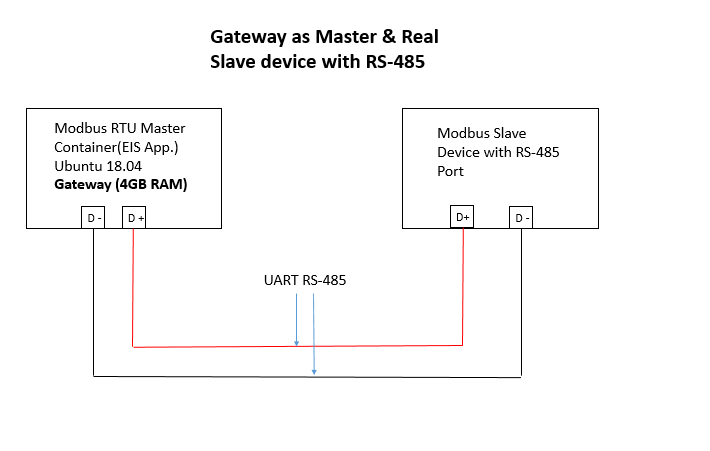
|  |  |
| --- | --- |
| **Configuration Parameter** | **Description** |
| BAUD\_RATE | Baud rate used for this Modbus RTU network and port. |
| PARITY | Parity to be used to communicate for this Modbus RTU network.  Values are: “N” for none, “O” for odd, “E” for even |
| PORT\_NAME | Serial port to be used to communicate for this Modbus RTU network |
| response\_timeout | Maximum time (in milliseconds) to wait to receive a response from end device for a request |
| interframe\_delay | Additional delay (in milliseconds) to wait for before sending a next request to the device.  For Modbus RTU, next request is sent only after previous request has timed out or a response for previous request is received. This additional delay is added when sending a next request.  For Modbus TCP, a next request to the device is sent irrespective of response for previous request. This additional delay is added when sending a next request. |

## Modbus TCP Communication

When used in TCP mode, the publisher of a stream will bind to a TCP socket and the subscribers connect to it to receive data. In Request-Response pattern, the responder acts as the server and binds the socket and the requester connects to it.

## Modbus RTU Communication

Modbus RTU is an open serial protocol derived from the Master/Slave architecture. UWC Modbus-rtu-container is used as master and slave can be configured.



Please note: To communicate with slave device, Modbus RTU mater uses configuration parameters (i.e., baud rate, parity, stop bits) from docker-compose.yml file.

## MQTT Bridge

This container is used to send messages from ZeroMQ to MQTT and vice-versa.

## MQTT

The MQTT container is a mosquitto broker required for MQTT to publish/subscribe data. MQTT broker use port “11883”.

MQTT clients should use above mentioned port for communication.

### Accessing secured MQTT container from an external MQTT client

**Pre-requisites:**

All UWC containers must be deployed-on gateway with DEV\_MODE=false (i.e., secured mode).

**Steps to follow:**

1. Open a terminal and execute following command to create local directory to keep certificates of MQTT broker,

mkdir ~/mqtt\_certs && cd ~/mqtt\_certs

1. Copy ca/ and /mymqttcerts directories in local directory i.e., created in script 2 from   
   /EdgeInsightsSoftware-v2.4-PV/IEdgeInsights/build/provision/Certificates/ directory.

Command to copy ca/ and /mymqttcerts/ dir in local dir (i.e., mqtt\_certs)

sudo cp -r /EdgeInsightsSoftware-v2.4-PV/IEdgeInsights/build/provision/Certificates/ca ~/mqtt\_certs/

1. Assign read permission to local certs using following command,

sudo chown -R $USER:$USER \* && sudo chmod +r ca/\* mymqttcerts/\*

**Please Note:** Read permissions are only required for ca/ and /mymqttcerts directories present inside mqtt\_certs directory copied in step2.

1. Open MQTT client e.g., MQTT.fx
2. Open the connection setting and click on SSL/TLS tab.

>> then click on Self Signed certificate option >> select CA file from mqtt\_certs/ca directory (file name : ca\_certificate.pem) , Client Certificate file from mqtt\_certs/mymqttcerts directory (File name : mymqttcerts\_client\_certificate.pem) and Client key File from mqtt\_certs/mymqttcerts directory (File name : mymqttcerts\_client\_key.pem) copied in step 2

1. Click on PEM Formatted check box and then save the setting and then connect. Refer below screenshot for more details

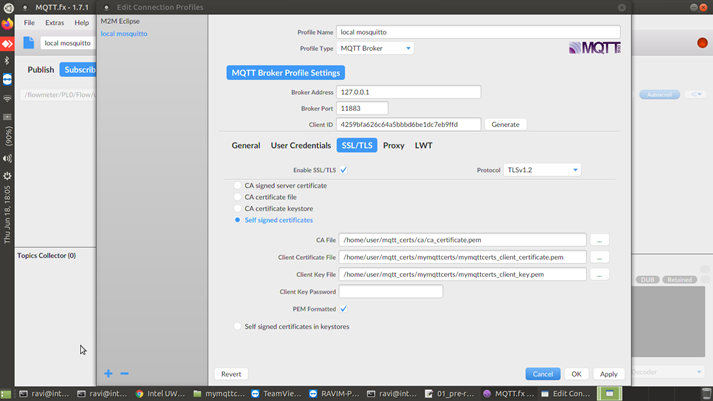


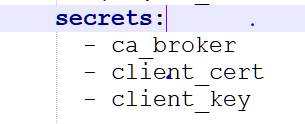
Figure 1 : Screen capture for mqtt.fx client connection

### Accessing secured MQTT container from a client inside a container

1. Mention following secrets for a new container in docker-compose.yml file.

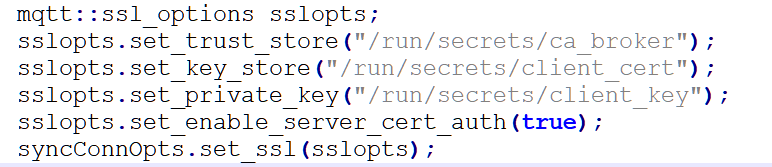
* ca\_broker – CA certificate
* client\_cert – Client certificate
* client\_key – Client Key

Following sample snippet for docker-compose.yml file



1. Use certificates mentioned in step 1 inside application to connect with secured MQTT broker which is running as a part of UWC.

Following is the sample code snippet in C++ to use certificates in a program,



1. Deploy containers using usual deployment process.

## Sparkplug-Bridge

This container implements Eclipse Foundation’s SparkPlug standard to expose data to compliant SCADA Master over MQTT.

### Pre-requisite for running Sparkplug-Bridge

1. SCADA Master (e.g., Ignition System) shall be installed and configured.
2. MQTT broker shall be installed and configured in SCADA Master. At present secured connectivity for MQTT is not supported.
3. Following parameters should be configured for Sparkplug-Bridge in docker-compose.yml file:

|  |  |
| --- | --- |
| DEVICES\_GROUP\_LIST\_FILE\_NAME | Name of devices group list file to be used to obtain information of wellheads, device, datapoints reported by this container as real devices to SCADA Master. |

## KPI App

This is a sample application which implements control loops and logs data in a log file named “AnalysisKPIApp.log”. Normally 3 log files are created on rolling basis i.e., once tie set file size limit is exceeded, a new file is created and likewise max 3 files are created. After this, the log files are overwritten.

The log file size can be updated, if required.

File: log4cpp.properties

Path in release package: kpi-tactic\KPIApp\Config

Path after deployment inside container: /opt/intel/config/log4cpp.properties

Default log file size is around 34mb.



To change the file size, “log4cpp.properties” needs to be changed. Please change the limit highlighted above. The max file size mentioned here is in bytes. Please identify number of bytes as per file size needed and set the value here.

Please run script 03 after changing “log4cpp.properties”.

# Site Configurations

This section provides configurations required to configure site, wellhead, device and points for UWC containers.

## System Level Global Configuration

This file contains configuration to be used for operations across UWC containers for (Modbus-TCP, Modbus-RTU, MQTT-Bridge.)

Global\_Config.yml file location - /opt/intel/eii/uwc\_data/common\_config

Based on realtime requirement, operations are classified into following sub-operations

1. Polling realtime

2. Polling non-realtime

3. On-demand read realtime

4. On-demand read non-realtime

5. On-demand write realtime

6. On-demand write non-realtime

7. SparkPlug communication for Sparkplug-Bridge

### Settings for Polling and On-Demand operation

Following is a sample for Polling operation. Similar is applicable for On-demand operations:

Global:

Operations:

- Polling:

default\_realtime: false

realtime:

operation\_priority: 4

retries: 1

qos: 1

non-realtime:

operation\_priority: 1

retries: 0

qos: 0

Following is a description of each field.

|  |  |  |
| --- | --- | --- |
| Field | Values | Description |
| default\_realtime | true or false | It defines how an operation should be performed in absence of corresponding realtime indicator.  E.g. In case of on-demand-write or read request, the json payload contains a field called "realtime". This field defines realtime nature in absence of such a field in on-demand request or polling configuration |
| realtime |  | This is a group field. Defines configuration for realtime operation. |
| non-realtime |  | This is a group field. Defines configuration for realtime operation. |
| Sub fields for “realtime” and “non-realtime” group | | |
| operation\_priority | 1 to 6 | The field defines priority to be assigned to 6 sub-operations (related to Polling and On-Demand), listed above.  1 is lowest priority and 6 is highest priority.  Hence, if there are 2 requests ready for execution, a request with higher priority will be executed first. |
| retries | 0 to 4 | This field specifies if a request of particular type should be resent if a response is not received within specified limit. Value 0 means no retry is needed. value 1 means, retry once. |
| qos | 0 or 1 or 2 | MQTT-Bridge will use this QoS value while publishing a message on MQTT. |

If incorrect value is specified for any of above fields, a default value (listed below) will be used:

default\_realtime: false

operation\_priority: 1

retries: 0

qos: 0

If configuration parameter or section is missing for any of the sub-operation (related to Polling and On-Demand), then default values mentioned above will be used.

### Settings for Sparkplug-Bridge – SparkPlug communication operation

Following is a sample:

Global:

Operations:

- SparkPlug\_Operation:

group\_id: "UWC nodes"

edge\_node\_id: "RBOX510"

Above are also the default values for mentioned parameters. If configuration parameter or section is missing for SparkPlug communication, then default values mentioned above will be used.

The parameters here are used to form SparkPlug formatted topic name.

These values should properly be configured to ensure correct representation of data under SCADA Master.

Following is a description of each field.

|  |  |  |
| --- | --- | --- |
| Field | Values | Description |
| group\_id | String | It defines value to be used for “group\_id” element in the Sparkplug™ Topic Namespace.  Example value “Huston Site”. |
| edge\_node\_id | String | It defines value to be used for “edge\_node\_id” element in the Sparkplug™ Topic Namespace.  Example value “Node 1”. |

## How to Configure Site and Wellhead

There is one file which lists down reference to device-groups (i.e., wellheads) controlled by one UWC gateway. Ideally, in one UWC gateway there is one TCP and one RTU container. Please note one RTU container can manage communication with multiple RTU networks.

---

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "Common device group file for TCP and RTU devices"

devicegrouplist:

- "Device\_group1.yml"

- "Device\_group2.yml"

Above example shows “Device\_group1” and “Device\_group2” as a reference to group of devices. “Device\_group1.yml” is a separate file listing down all TCP and RTU devices falling under one device-group (e.g. wellhead PL0)

Each device-group file will have information about devices in that group.

---

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "Device group 1"

id: "PL0"

description: "Device group 1"

devicelist:

- deviceinfo: "flowmeter\_device.yml"

id: "flowmeter"

protocol:

protocol: "PROTOCOL\_TCP"

ipaddress: "192.168.0.222"

port: 502

unitid: 1

tcp\_master\_info: "tcp\_master\_info.yml"

- deviceinfo: "iou\_device.yml"

id: "iou1"

protocol:

protocol: "PROTOCOL\_RTU"

slaveid: '10'

rtu\_master\_network\_info: "rtu\_network1.yml"

Following sections provide details about TCP and RTU device configuration in device-group file.

### Configuring TCP device in device-group

E.g.

devicelist:

- deviceinfo: "flowmeter\_device.yml"

id: "flowmeter"

protocol:

protocol: "PROTOCOL\_TCP"

ipaddress: "192.168.0.222"

port: 502

unitid: 1

tcp\_master\_info: "tcp\_master\_info.yml"

Following parameters are needed for each TCP device:

* ipaddress – for TCP communication IP address for slave device required
* port – can be configured as per slave device configuration
* unitid – id can used to distinguish multiple slaves on same ipaddress
* tcp\_master\_info - tcp\_master\_info.yml – In this file interframe delay and response timeout can be configured for TCP network

Following is a sample file for tcp\_master\_info.yml

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "TCP master config parameter file"

# inter-frame delay and response timeout values are in Millisecond

interframe\_delay: 1

response\_timeout: 80

Note: This reference shall be unique across TCP devices and needs to be given for each TCP device.

### Configuring RTU device in device-group

E.g.

devicelist:

- deviceinfo: "iou\_device.yml"

id: "iou1"

protocol:

protocol: "PROTOCOL\_RTU"

slaveid: '10'

rtu\_master\_network\_info: "rtu\_network1.yml"

Following parameters are needed for each RTU device:

* slaveid – This is end device id in case of RTU communication
* rtu\_master\_network\_info: "rtu\_network1.yml" – This file is used to configure RTU configuration for a specific RTU network.

Following is a sample file for rtu\_network1.yml

---

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "RTU Network information for network 1"

baudrate: 9600

parity: "N"

com\_port\_name: "/dev/ttyS0"

# inter-frame delay and response timeout values are in Millisecond

interframe\_delay: 1

response\_timeout: 80

Note: This file needs to be specified for each RTU device. If multiple RTU networks are present (RS485/RS232) then those many files should be created. For each RTU device, an appropriate RTU network reference shall be provided.

## How to Configure Devices

Device contains information of a device. Below is a sample file –

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "Information for Demo IOUnit"

device\_info:

name: "IO Unit"

description: "Power Scout Meter"

manufacturer: "Dent Instruments"

model: "PS3037"

pointlist: "iou\_datapoints.yml"

## How to Configure Device points

Device Point contains end point information. Below is a sample file.

Below parameters can be changed in this file –

* addr - can be of range 0 to 65534
* pollinterval – value in milliseconds
* type – Function Code
* width – Number of bytes to be read
* realtime – To be used for real time, as of date it is false.

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2019"

description: "Data for Demo IOUnit data points"

datapoints:

- id: "Arrival"

attributes:

type: "DISCRETE\_INPUT"

addr: 2048

width: 1

polling:

pollinterval: 250

realtime: true

- id: "AValve"

attributes:

type: "HOLDING\_REGISTER"

addr: 640

width: 1

polling:

pollinterval: 1000

realtime: true

- id: "DValve"

attributes:

type: "COIL"

addr: 2048

width: 1

polling:

pollinterval: 1000

realtime: true

- id: "TubingPressure"

attributes:

type: "INPUT\_REGISTER"

addr: 1025

width: 1

polling:

pollinterval: 250

realtime: true

- id: "CasingPressure"

attributes:

type: "INPUT\_REGISTER"

addr: 1024

width: 1

polling:

pollinterval: 250

realtime: true

- id: "KeepAlive"

attributes:

type: "COIL"

addr: 3073

width: 1

polling:

pollinterval: 2000

realtime: true

**Note** – For coil type width should be 1.

**YML file Configuration table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File Name** | **Key Name** | **Value** | **Optional** | **Description** |
| Devices\_group\_list.yml | file: |  |  | This file contains list of all device groups |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description |  |  |  |
| version |  |  |  |
| devicegrouplist : |  |  |  |
| - Device\_group1.yml | Device\_group1.yml |  |  |
| Device\_group1.yml | description | Device group 1 | Yes | This file contains information about all TCP and RTU devices present under one device group |
| devicelist: |  |  |  |
| -deviceinfo: | flowmeter\_device.yml | No |  |
| Id | flowmeter | No |  |
| protocol: |  |  |  |
| ipaddress | 192.xx.xx.xx | No | IP Address of Modbus TCP Slave |
| port | 502 | No | Port number of Modbus TCP Slave (Default: 502) |
| protocol | PROTOCOL\_TCP | No | Hard Coded String |
| unitid | 1 | No | Range 1- 255 (Default: 1) |
| tcp\_master\_info | tcp\_master\_info.yml | No | This is used to set interframe delay and response timeout for TCP master |
| -deviceinfo: | iou\_device.yml | No | File Name as a string |
| Id | iou | No | String |
| protocol: |  |  |  |
| protocol | PROTOCOL\_RTU | No | Hard Coded String |
| slaveid | ‘10’ | No | Range 1- 255. |
| rtu\_master\_network\_info | rtu\_network1.yml | No | This is network configuration for RTU network in which the RTU slave-device resides. |
| file: |  |  |  |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description | Data for wellhead 1 | Yes |  |
| version | 1.0.0 | Yes |  |
| Id | PL0 | No |  |
| tcp\_master\_info.yml | file |  |  | This file is used to configure TCP master container |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description | TCP master config parameter file | Yes |  |
| version | 1.0.0 | Yes |  |
| interframe\_delay | 1 | Yes | This can be used to add delay (in milliseconds) between consecutive requests |
| response\_timeout | 80 | Yes | This is the response timeout (in milliseconds) to be considered for TCP network |
| rtu\_network1.yml | file |  |  | This file is used to set RTU master configuration for Specific network |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description | RTU Network information for network 1 | Yes |  |
| version | 1.0.0 | Yes |  |
| baudrate | 9600 | No |  |
| parity | N |  |  |
| com\_port\_name | /dev/ttyS0 | No |  |
| interframe\_delay | 1 | Yes | This can be used to add delay (in milliseconds) between consecutive requests |
| response\_timeout | 80 | Yes | This is the response timeout (in milliseconds) to be considered for RTU network |
|  |  |  |  |
| flowmeter\_device.yml | device\_info: |  |  | This file contains reference to datapoints.yml |
| description | Power Scout Meter | Yes |  |
| manufacturer | Dent Instruments | Yes |  |
| model | PS3037 | Yes |  |
| name | Flowmeter Unit | Yes |  |
| file: |  |  |  |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description |  |  |  |
| version |  |  |  |
| pointlist | flowmeter\_datapoints.yml | No |  |
| flowmeter\_datapoints.yml | File |  |  | This file contains all information of datapoint. One datapoint YML file corresponds to one SparkPlug template definition. The YML file name (without extension) corresponds to SparkPlug template name. |
| version |  | No | This is a mandatory field. It is mapped to version of SparkPlug template. |
| -attributes: |  |  |  |
| addr | 1 | No | Address of Slave device (Range 0- 65535) |
| type | COIL | No | Function code String.  F.C. 0x01= COIL  F.C. 0x02=DISCRETE\_INPUT  F.C. 0x03=HOLDING\_REGISTER  F.C. 0x04=INPUT\_REGISTER |
| width | 1 | No | Number of Registers (Range 1 to 125) |
| Id | Flow | No | Endpoint name of the Slave device. |
| wordswap | true | No | Swapping of Registers (2 Byte) |
| byteswap | false | No | Swapping of Byte in a Register. |
| polling: |  |  |  |
| pollinginterval | 250 | No | Polling interval in milliseconds. |
| realtime | true | Yes | Value of this field defines whether a point to be polled in realtime or non-realtime mode. |
| file: |  |  |  |
| author | Intel | Yes |  |
| date | Sun Sep 1 | Yes |  |
| description |  |  |  |
| version |  |  |  |

## How to add/edit/delete new wellhead/device/point configurations

* + 1. User can add/update/edit/delete Oil well configurations files (YML files) from /opt/intel/eii/uwc\_data directory
    2. Open a terminal and go to EdgeInsightsSoftware-v2.1-PV/IEdgeInsights directory
    3. Run below command to apply new Oil well site configurations

sudo ./05\_applyConfigChanges.sh

Note: this script will restart all UWC docker containers

## KPI App Configuration

Following is a sample configuration file for KPI App.

---

file:

version: "1.0.0"

author: "Intel"

date: "Sun Sep 1 13:34:13 PDT 2020"

description: "KPI App Config File"

isMQTTModeApp: false

timeToRun\_Minutes: 10

isRTModeForPolledPoints: true

isRTModeForWriteOp: true

# This section lists down number of control loops.

# For each control loop, following information is presented:

# 1. Point being polled

# 2. Point and value to be used for writing

# 3. Delay to be used before sending a write operation.

controlLoopDataPointMapping:

- polled\_point: "/flowmeter/PL0/P1"

delay\_msec: 5

write\_operation:

datapoint: "/iou/PL0/D1"

dataval: "0x01"

- polled\_point: "/flowmeter/PL0/P2"

delay\_msec: 15

write\_operation:

datapoint: "/flowmeter/PL0/D2"

dataval: "0x1234"

Following is a description of each field.

|  |  |  |
| --- | --- | --- |
| Field | Values | Description |
| isMQTTModeApp | boolean | This field tells whether application runs based on ZMQ or MQTT  Default value is false i.e. ZMQ based app.  Value can be true or false. Set to true for using MQTT based model. |
| timeToRun\_Minutes | Number | This field tells the duration (in minutes) for which this app will run.  Set to 0 for no limit.  Default value is 0 if this field is missing or has incorrect value. |
| isRTModeForPolledPoints | boolean | This field tells which mode (RT or Non-RT) is used to monitor "polling" in control loop.  For ZMQ-based app, RT polling topics will be scanned if set to true.  This field has no effect in case of MQTT-based app. |
| isRTModeForWriteOp | boolean | This field tells which mode (RT or Non-RT) needs to be used for "write operation" in control loop.  When initiatiating a write request, "realtime" field in JSON payload will be set if RT is needed.  For ZMQ-based app, RT write-reponse topics will be scanned if set to true. |
| Sub fields for “controlLoopDataPointMapping” group | | |
| polled\_point | String | It defines point to be monitored for “polling” in the control loop. |
| delay\_msec | Number | It defines delay (in milliseconds) to be used before sending a write operation once polling data is available in a control loop.  Default value of delay is 5 msec if this field is missing or has incorrect value. |
| write\_operation |  | This is a group field. It defines all the parameters required to initiate a write request for the control loop. |
| datapoint | String | It defines point to be used for sending “write request” in the control loop. |
| dataval | String | It defines value to be used in “write request” in the control loop. |

Please note following: This configuration file should be created manually with following considerations:

1. The points in “polled\_point” and “datapoint” fields in this file should be configured as per actual configuration in wellhead, device and datapoints config files.

E.g. If a point to be polled is not present in datapoints config file, then data for that control loop will not be collected.

1. If the points being polled are configured as “realtime” in datapoints config file, then “isRTModeForPolledPoints” should be set to “true”. It should be set to “false” otherwise.
2. ZMQ-based KPI App can monitor either RT or Non-RT points at a time.
3. KPI App container can run either in ZMQ mode or in MQTT mode at a time.

# UWC Modbus Operations

This section provides configurations required to read and write data from sensors and actuators connected over Modbus TCP or RTU on UWC gateway.

An application can perform following operations using UWC containers:

* Data Polling
* On-Demand Write
* On-Demand Read

Following section explains how to use MQTT topics to perform above operations. Further these operations can be performed in realtime (RT) and non-realtime (Non-RT) mode.

Multiple modules are involved in processing the operation. To capture the time taken by each module (i.e. a step), epoch timestamps in microseconds are added at various levels. These timestamps are present in JSON message.

The table of terms here is useful for interpreting the JSON payloads of the messages.

| **Key in JSON message** | **Definition** |
| --- | --- |
| version | MQTT message format. Value “2.0” |
| data\_topic | This is the MQTT bus topic name. MQTT bus topic is a combination of number of parameters. Topic is explained in respective operation sections – 6.1, 6.2, 6.3 |
| wellhead | Well identifier e.g., PL0 |
| metric | Modbus point e.g., TubingPressure |
| realtime | It indicates whether the operation to be handled in real-time manner. Values are:  1: Yes  0: No |
| status | Response status: Good or Bad  **Good**: Success response  **Bad**: Error response. In case of error response, keys “error\_code”, “lastGoodUsec” will be present in json message. Also “value” key will have last known good value, if any. |
| value | Hex string response for a read operation.  E.g., 0x00  This field is not applicable for on-demand-write operation.  In case of bad response, this field will represent last know good value, if any. |
| usec | Epoch timestamp (in microseconds) at which Modbus application container prepares a message to publish on ZMQ. |
| error\_code | This field is present when status is Bad. Refer section 8 for details about error codes. |
| lastGoodUsec | This field is present when status is Bad. This field represents “usec” timestamp of last known good value, if any. |
| timestamp | Local system time in human readable format when a message is created |
| driver\_seq | This is applicable for polling operation. The number is assigned automatically by the Modbus Application when a message with polling data is initiated.  For each polling message, this field will have different value. |
| app\_seq | This is applicable for on-demand read / write operations.  The number is assigned by the application initiating the on-demand operation request.  The Modbus Application uses data from on-demand request while sending back the response.  This sequence number can be used by application to match response for on-demand request. |
| Timestamp fields listed below in Time-sequential order | |
| tsMsgRcvdFromMQTT | This is applicable for on-demand read/ write operation.  Epoch timestamp (in microseconds) at which message from MQTT is read in MQTT-Bridge. |
| tsMsgPublishOnEIS | This is applicable for on-demand read/ write operation.  Epoch timestamp (in microseconds) at which MQTT-Bridge prepares a message to publish on ZMQ. |
| reqRcvdByApp | This is applicable for on-demand read/ write operation.  Epoch timestamp (in microseconds) at which message from ZMQ is read in Modbus application container. MQTT-Bridge sends on-demand request messages to Modbus application over ZMQ. |
| reqRcvdInStack | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which protocol stack received the request message to send on network. This is step inside Modbus application container. |
| reqSentByStack | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which protocol stack sends the request message on network. This is step inside Modbus application container. |
| respRcvdByStack | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which protocol stack reads the response message received from a network. This is step inside Modbus application container. |
| respPostedByStack | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which protocol stack sends the received response message to Modbus application container. This is step inside Modbus application container. |
| usec | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which Modbus application container prepares a message to publish on ZMQ.  This field is a part of data-model. It is repeated here for completeness. |
| tsMsgRcvdForProcessing | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which message from ZMQ is read in MQTT-Bridge. Modbus application container sends messages like polling data and response for on-demand requests on ZMQ to MQTT-Bridge. |
| tsMsgReadyForPublish | This is applicable for all on-demand read/ write and polling operations.  Epoch timestamp (in microseconds) at which MQTT-Bridge prepares a message to publish on MQTT. |

## Data Polling

In datapoint YML configuration file, a polling frequency is configured. As per polling frequency, data is fetched from end point and published on MQTT by UWC container. This section describes how to read data for polled points using MQTT.

The data actions which are “Polling” actions are initiated by the Protocol container (in this case the Modbus protocol application (i.e. the driver) within the Modbus container.

To receive polled data: Application should use a topic in following format to receive (i.e., subscribe) polling data from MQTT:

**MQTT topic to receive (i.e., subscribe) write response: /device/wellhead/point/update**

Please refer table in section 6 for details of fields.

**Example:**

**Polling Topic**: /flowmeter/PL0/D3/update

**Polling Message**: Success Response

{

"driver\_seq": "1153204567290051305",

"timestamp": "2020-05-01T06:40:25",

"version": "2.0",

"realtime": "1",

"data\_topic": /flowmeter/PL0/D3/update",

"wellhead": "PL0",

"metric": "D3",

"tsPollingTime": "1588315225331550",

"reqRcvdInStack": "1588315225331890",

"reqSentByStack": "1588315225333180",

"respRcvdByStack": "1588315225333900",

"respPostedByStack": "1588315225333950",

"status": "Good",

"value": "0x01",

"usec": "1588315225334040",

"tsMsgRcvdForProcessing": "1588315225335060",

"tsMsgReadyForPublish": "1588315225335420"

}

**Polling Message**: Error Response

{

"driver\_seq": "1153204567290051305",

"timestamp": "2020-05-01T06:40:25",

"version": "2.0",

"realtime": "1",

"data\_topic": /flowmeter/PL0/D3/update",

"wellhead": "PL0",

"metric": "D3",

"tsPollingTime": "1588315225331550",

"reqRcvdInStack": "1588315225331890",

"reqSentByStack": "1588315225333180",

"respRcvdByStack": "0",

"respPostedByStack": "1588315225333950",

"status" : "Bad",

"error\_code" : "2003",

"lastGoodUsec" : "1588315225333897",

"value" : "0x01",

"usec" : "1588315225334040",

"tsMsgRcvdForProcessing" : "1588315225335060",

"tsMsgReadyForPublish" : "1588315225335420"

}

## On-Demand Write

This section describes how to write data to some specific Modbus point using MQTT.

To send request: Application should use a topic in following format to send (i.e. publish) write request on MQTT:

**MQTT topic to send (i.e., publish) write request: /device/wellhead/point/write**

To receive response: Application should use a topic in following format to receive (i.e. subscribe) response of write request from MQTT:

**MQTT topic to receive (i.e., subscribe) write response: /device/wellhead/point/writeResponse**

Please refer table in section 6 for details of fields.

**Example:**

**Request Topic**: /flowmeter/PL0/Flow/write

**Request Message**:

{"wellhead":"PL0","command":"Flow","value":"0x00","timestamp":"2019-09-20 12:34:56","usec":"1571887474111145","version":"2.0","app\_seq":"1234"}

A message without “realtime” field is treated as a non-realtime message. To execute a message in realtime way, a field called “realtime” should be added as shown below:

{"wellhead":"PL0","command":"Flow","value":"0x00","timestamp":"2019-09-20 12:34:56","usec":"1571887474111145","version":"2.0","app\_seq":"1234",”realtime”:”1”}

The “value” field represents value to be written to end device as a part of on-demand write operation.

**Response Topic**: /flowmeter/PL0/Flow/writeResponse

**Response Message**: Success Response

{

"app\_seq": "1234",

"version": "2.0",

"realtime": "0",

"data\_topic": "/flowmeter/PL0/Time0/writeResponse",

"wellhead": "PL0",

"metric": "Time0",

"tsMsgRcvdFromMQTT": "1585660044014345",

"tsMsgPublishOnEIS": "1585660044017877",

"reqRcvdByApp": "1585660044021380",

"reqRcvdInStack": "1585660044023582",

"reqSentByStack": "1585660044025671",

"respRcvdByStack": "1585660044025636",

"respPostedByStack": "1585660044025712",

"status": "Good",

"usec": "1585660044026131",

"tsMsgRcvdForProcessing": "1585660044027184",

"tsMsgReadyForPublish": "1585660045024371"

}

**Response Message**: Error Response

{

"app\_seq" : "1234",

"timestamp" : "2020-04-24 06:10:30",

"version" : "2.0",

"realtime" : "1",

"data\_topic" : "/flowmeter/PL0/Flow/writeResponse",

"wellhead" : "PL0",

"metric" : "Flow",

"tsMsgRcvdFromMQTT" : "1587708630464551",

"tsMsgPublishOnEIS" : "1587708630465375",

"reqRcvdByApp" : "1587708630465988",

"reqRcvdInStack" : "1587708630466345",

"reqSentByStack" : "0",

"respRcvdByStack" : "0",

"respPostedByStack" : "1587708630466627",

"status" : "Bad",

"error\_code" : "2003",

"usec" : "1587708630466935",

"tsMsgRcvdForProcessing" : "1587708630467797",

"tsMsgReadyForPublish" : "1587708630468239"

}

**Response Message**: Error Response for Invalid request JSON

{

"app\_seq":"1234",

"timestamp":"2020-05-13 06:43:50",

"version":"2.0",

"realtime":"0",

"topic":"/iou/PL0/D1/writeResponse",

"wellhead":"PL0",

"metric":"D1",

"tsMsgRcvdFromMQTT":"1589352230212884",

"tsMsgPublishOnEIS":"1589352230214516",

"reqRcvdByApp":"1589352230215485",

"reqRcvdInStack":"0",

"reqSentByStack":"0",

"respRcvdByStack":"0",

"respPostedByStack":"0",

"status":"Bad",

"usec":"1589352230217118",

"error\_code":"102",

"tsMsgRcvdForProcessing":"1589352230220450",

"tsMsgReadyForPublish":"1589352230220503"

}

## On-Demand Read

This section describes how to read data from some specific Modbus point using MQTT.

To send request: Application should use a topic in following format to send (i.e., publish) read request on MQTT:

**MQTT topic to send (i.e. publish) read request: /device/wellhead/point/read**

To receive response: Application should use a topic in following format to receive (i.e., subscribe) response of read request from MQTT:

**MQTT topic to receive (i.e., subscribe) write response: /device/wellhead/point/readResponse**

Please refer table in section 6 for details of fields.

**Example:**

**Request Topic**: /flowmeter/PL0/Flow/read

**Request Message:**

{"wellhead":"PL0","command":"Flow","timestamp":"2019-09-20 12:34:56","usec":"1571887474111145","version":"2.0","app\_seq":"1234"}

A message without “realtime” field is treated as a non-realtime message. To execute a message in realtime way, a field called “realtime” should eb added as shown below:

{"wellhead":"PL0","command":"Flow","timestamp":"2019-09-20 12:34:56","usec":"1571887474111145","version":"2.0","app\_seq":"1234",”realtime”:”1”}

**Response Topic**: /flowmeter/PL0/Flow/readResponse

**Response Message**: Success Response

{

"app\_seq":"1234",

"timestamp":"2020-04-24 05:24:02",

"version":"2.0",

"realtime":"0",

"data\_topic":"/flowmeter/PL0/Flow/readResponse",

"wellhead":"PL0",

"metric":"Flow",

"tsMsgRcvdFromMQTT":"1587705842296135",

"tsMsgPublishOnEIS":"1587705842296550",

"reqRcvdInStack":"1587705842296921",

"reqSentByStack":"1587705842298063",

"reqRcvdByApp":"1587705842296836",

"respRcvdByStack":"1587705842298666",

"respPostedByStack":"1587705842298686",

"status":"Good",

"value":"0x01",

"usec":"1587705842298811",

"tsMsgRcvdForProcessing":"1587705842299038",

"tsMsgReadyForPublish":"1587705842299115"

}

**Response Message**: Error Response

{

"app\_seq" : "1234",

"timestamp" : "2020-04-24 06:17:33",

"version" : "2.0",

"realtime" : "0",

"data\_topic" : "/flowmeter/PL0/Flow/readResponse",

"wellhead" : "PL0",

"metric" : "Flow",

"tsMsgRcvdFromMQTT" : "1587709053533410",

"tsMsgPublishOnEIS" : "1587709053534618",

"reqRcvdInStack" : "1587709053535694",

"reqRcvdByApp" : "1587709053535467",

"reqSentByStack" : "0",

"respRcvdByStack" : "0",

"respPostedByStack" : "1587709053536172",

"status" : "Bad",

"value" : "",

"error\_code" : "2003",

"usec" : "1587709053536590",

"tsMsgRcvdForProcessing" : "1587709053537377",

"tsMsgReadyForPublish" : "1587709053537647"

}

**Response Message**: Error Response for Invalid Input JSON

{

"app\_seq" : "1234",

"timestamp" : "2020-04-24 06:22:42",

"version" : "2.0",

"realtime" : "0",

"data\_topic" : "/flowmeter/PL0/Flow/readResponse",

"wellhead" : "PL0",

"metric" : "Flow1",

"tsMsgRcvdFromMQTT" : "1587709362173590",

"tsMsgPublishOnEIS" : "1587709362173872",

"reqRcvdInStack" : "0",

"reqSentByStack" : "0",

"reqRcvdByApp" : "1587709362174221",

"respRcvdByStack" : "0",

"respPostedByStack" : "0",

"status" : "Bad",

"value" : "",

"error\_code" : "2014",

"usec" : "1587709362174333",

"tsMsgRcvdForProcessing" : "1587709362174590",

"tsMsgReadyForPublish" : "1587709362174647"

}

## KPI Application

Following data (if available) is logged in a log-file by KPI Application for control loops.

| **Field** | **Description** |
| --- | --- |
| pollSeq | Driver sequence number received in polling |
| pollTopic | Topic received for polling |
| pollStatus | Status received for polling |
| pollValue | Value received in polling |
| pollError | Error code received for polling |
| tsPollingTime | Timestamp when polling time is triggered |
| pollReqRcvdInStack | Timestamp when polling read request is received in stack |
| pollReqSentByStack | Timestamp when polling read request is sent by stack |
| pollRespRcvdByStack | Timestamp when polling read response is received in stack from device |
| pollRespPostedByStack | Timestamp when polling read response is posted to Modbus container by stack |
| pollRespPostedToEIS | Timestamp when polling is posted to ZMQ by Modbus container |
| pollDataRcvdInExport | Timestamp when polling data is received in MQTT-Bridge (applicable only when KPI App is executed in MQTT mode) |
| pollDataPostedToMQTT | Timestamp when polling data is posted to MQTT by MQTT-Bridge (applicable only when KPI App is executed in MQTT mode) |
| pollDataRcvdInApp | Timestamp when polling data is received in KPI Application |
| appSeq | App sequence number used for the write operation |
| wrRspTopic | Topic received for write response |
| wrRspStatus | Status received for write response |
| wrRspError | Error code received for write response. It represents the value received from Modbus container.  Additionally, following 2 dummy error codes are set to handle error conditions within KPI App:   * WrReqInitFailed/writeResponse: It means the write request could not be initiated. * WrRespNotRcvd/writeResponse: It means the write response of last sent request is not received. |
| wrReqCreation | Timestamp when KPI App creates a write request |
| wrReqRcvdInExport | Timestamp when write request is received in MQTT-Bridge (applicable only when KPI App is executed in MQTT mode) |
| wrReqPublishOnEIS | If KPI App is running in MQTT mode, it represents a timestamp when the write request is published to EIS by MQTT-Bridge  If KPI App is running in ZMQ mode, it represents a timestamp when the write request is published to EIS by KPI App |
| wrReqRcvdByModbus | Timestamp when write request is received in Modbus container |
| wrReqRcvdInStack | Timestamp when write request is received in stack |
| WrReqSentByStack | Timestamp when write request is sent by stack |
| WrRespRcvdByStack | Timestamp when write response is received in stack from device |
| wrRespPostedByStack | Timestamp when write response is posted to Modbus container by stack |
| wrRespPostedToEIS | Timestamp when write response is posted to ZMQ by Modbus container |
| wrRespRcvdInExport | Timestamp when the write response is received in MQTT-Bridge (applicable only when KPI App is executed in MQTT mode) |
| wrRespPostedToMQTT | Timestamp when the write response is posted to MQTT by MQTT-Bridge (applicable only when KPI App is executed in MQTT mode) |
| wrRespRcvInApp | Timestamp when write response is received in application |

# Sparkplug-Bridge Operations

Sparkplug-Bridge implements Eclipse Foundation’s SparkPlug standard.

(Ref: <https://www.eclipse.org/tahu/spec/Sparkplug%20Topic%20Namespace%20and%20State%20ManagementV2.2-with%20appendix%20B%20format%20-%20Eclipse.pdf>)

This section explains the features in detail. UWC gateway acts as a “node” as per SparkPlug standard. Please note that Sparkplug-Bridge is an under-development feature and hence not all message types are supported from SparkPlug.

This section also explains how information from real device and virtual device is mapped to SparkPlug formatted data.

## App (virtual device) communication

Apps running on UWC platform can be represented as a SparkPlug device to SCADA Master. SCADA Master can monitor, control these apps using SparkPlug mechanism. Sparkplug-Bridge defines following to enable this communication between apps and SCADA Master:

TemplateDef message: This allows providing a definition for a Sparkplug Template i.e., UDT

BIRTH message: This corresponds to a SparkPlug DBIRTH message.

DEATH message: This corresponds to a SparkPlug DDEATH message.

DATA message: This corresponds to a SparkPlug DDATA message.

CMD message: This corresponds to a SparkPlug DCMD message.

Apps and Sparkplug-Bridge communicate over internal MQTT using above defined messages.

### App Message Topic Format

MQTT Topic: MESSAGETYPE/APPID/SUBCLASS

Where,

* MESSAGETYPE: Any of “BIRTH”, “DEATH”, “DATA”, “CMD”
* APPID: Any string e.g., “UWCP”
* SUBCLASS: Any string like wellhead-id e.g., “PL0”. This is not needed in case of DEATH message.

Sparkplug-Bridge uses following format to represent name of virtual device in SparkPlug Topic namespace:

[value of “APPID” from app message topic] + “-“ + [value of “SUBCLASS” from app message topic]

### App Message - BIRTH

MQTT Topic: BIRTH/APPID/SUBCLASS

Message format:

It is a JSON format message which contains a list of metrics having following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| name | String | Metric name |
| dataType | String | Datatype of the metric. Allowed values:   * Boolean * UInt8 * UInt16 * UInt32 * UInt64 * Int8 * Int16 * Int32 * Int64 * Float * Double * String * UDT\*   Note: “UDT” type is explained in subsequent section. |
| value | As per dataType | Value of a metric as per dataType defined above |
| timestamp | UInt64 | Time the message was transmitted.  (UNIX, UTC, Milliseconds) |

Example:

{

"metrics": [

{

"name": "Properties/Version",

"dataType": "String",

"value": "2.0.0.1",

"timestamp": 1486144502122

},

{

"name": "Properties/RTU\_Time",

"dataType": "String",

"value": "1234",

"timestamp": 1486144502122

},

{

"name": "UDT/Prop1",

"dataType": "UDT",

"value": {

"udt\_ref": {

"name": "custom\_udt",

"version": "1.0"

},

"metrics": [

{

"name": "M1",

"dataType": "String",

"value": "2.0.0.1",

"timestamp": 1486144502122

},

{

"name": "RTU\_Time",

"dataType": "Int32",

"value": 1234,

"timestamp": 1486144502122

}

],

"parameters": [

{

"name": "P1",

"dataType": "String",

"value": "P1Val"

},

{

"name": "P2",

"dataType": "Int32",

"value": 100

}

]

}

}

]

}

Data Flow:

This message is published by App over MQTT broker and subscribed by Sparkplug-Bridge. This message provides information about all metrics related to a SUBCLASS which App wants to expose to a SCADA Master.

Sparkplug-Bridge publishes a DBIRTH message to SCADA Master if metrics contain a new metric or if datatype of any of metrics is changed.

Notes:

* If the App publishes multiple BIRTH messages for a SUBCLASS, then Sparkplug-Bridge remembers all metrics reported in all BIRTH messages. Sparkplug-Bridge reports all these metrics to SCADA Master in DBIRTH message. This data with Sparkplug-Bridge is cleared on restart of gateway or Sparkplug-Bridge container.
* A DBIRTH message can result in refreshing of data in Sparkplug-Bridge and in SCADA Master. Hence, it is recommended for an App to provide information about all metrics in one BIRTH message. App should avoid using multiple BIRTH messages for same SUBCLASS.
* If App wants to publish a metric of type “UDT”, the definition of “UDT” should be provided prior to publishing the BIRTH message. UDT definition can be provided using “TemplateDef” message, explained in subsequent section.

Following information is required as a part of “value” key when UDT type is used:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| udt\_ref | Collection of fields | It consists of following fields:   * name: String type * version: String type   Above fields indicate the UDT definition used (i.e., adhered) by this UDT metric. |
| metrics | Array of metrics | Each metric consists of following fields:   * name: String type * dataType: String type * value: As per dataType * timestamp: UInt64   For details, refer UDT definition message. |
| parameters | Array of metrics | Each parameter consists of following fields:   * name: String type * dataType: String type * value: As per dataType   For details, refer UDT definition message. |

### App Message - DATA

MQTT Topic: DATA/APPID/SUBCLASS

Message format:

It is a JSON format message which contains a list of metrics having following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| name | String | Metric name |
| dataType | String | Datatype of the metric. Allowed values:   * Boolean * UInt8 * UInt16 * UInt32 * UInt64 * Int8 * Int16 * Int32 * Int64 * Float * Double * String * UDT\*   Note: “UDT” type is explained in subsequent section. |
| value | As per dataType | Value of a metric as per dataType defined above |
| timestamp | UInt64 | Time the message was transmitted.  (UNIX, UTC, Milliseconds).  This is an optional field. If not provided, then system will assign a timestamp while publishing a corresponding SparkPlug message to SCADA Master. If this value is present, then it will be used in SparkPlug message. |

Example:

{

"metrics": [

{

"name": "Properties/Version",

"dataType": "String",

"value": "5.0.0.1",

"timestamp": 1486144502122

},

{

"name": "UDT/Prop1",

"dataType": "UDT",

"value": {

"metrics": [

{

"name": "M1",

"dataType": "String",

"value": "a.b",

"timestamp": 1486144502122

}

]

}

}

]

}

Data Flow:

This message is published by App over MQTT broker and subscribed by Sparkplug-Bridge. This message provides information about all changed metrics related to a SUBCLASS.

Sparkplug-Bridge publishes a DDATA message to SCADA Master if value of any of “known metrics” is changed compared to last known value from a BIRTH or DATA message.

Note: A “known metric” is one which was reported in BIRTH message. The name and datatype for a “known metric” in DATA message and BIRTH message shall match.

### App Message - CMD

MQTT Topic: CMD/APPID/SUBCLASS

Message format:

It is a JSON format message which contains a list of metrics having following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| name | String | Metric name |
| dataType | String | Datatype of the metric. Allowed values:   * Boolean * UInt8 * UInt16 * UInt32 * UInt64 * Int8 * Int16 * Int32 * Int64 * Float * Double * String * UDT\*   Note: “UDT” type is explained in subsequent section. |
| value | As per dataType | Value of a metric as per dataType defined above |
| timestamp | UInt64 | Time the message was transmitted.  (UNIX, UTC, Milliseconds).  This is an optional field. If not provided, then system will assign a timestamp while publishing a corresponding SparkPlug message to SCADA Master. If this value is present, then it will be used in SparkPlug message. |

Example:

{

"metrics": [

{

"name": "Properties/Version",

"dataType": "String",

"value": "7.0.0.1",

"timestamp": 1486144502122

},

{

"name": "UDT/Prop1",

"dataType": "UDT",

"metrics": [

{

"dataType": "Int32",

"value": 4,

"name": "RTU\_Time",

"timestamp": 1614512107195

}

],

"timestamp": 1614512107195

}

]

}

Data Flow:

This message is published by Sparkplug-Bridge over MQTT broker and subscribed by App. This message provides information about control command i.e., DCMD received from SCADA Master.

Sparkplug-Bridge publishes a CMD message to the App if DCMD message is received for a known metric.

Note: A “known metric” is one which was reported in BIRTH message. The name and datatype for a “known metric” in DCMD message and BIRTH message shall match.

### App Message - DEATH

MQTT Topic: DEATH/APPID

Message format:

It is a JSON format message which contains following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| timestamp | UInt64 | Time the message was transmitted.  (UNIX, UTC, Milliseconds).  This is an optional field. If not provided, then system will assign a timestamp while publishing a corresponding SparkPlug message to SCADA Master. If this value is present, then it will be used in SparkPlug message. |

Example:

{

"timestamp": 1486144502122

}

Data Flow:

When App’s connection with MQTT broker breaks then this message is published.

Sparkplug-Bridge publishes a DDEATH message to SCADA Master for all known SUBCLASS associated with the App.

### App Message - TemplateDef

MQTT Topic: TemplateDef

Message format:

It is a JSON format message which contains a list of metrics having following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |
| udt\_name | String | Name of Sparkplug Template i.e., UDT |
| version | String | UDT definition version |
| metrics | List of metrics | A UDT consists of nested metrics. This field defines the metrics which are a part of UDT.  Each metric contains following fields:   * name * dataType * value   These fields are explained in following rows. |
| Keys within each “metric” | | |
| name | String | Name of metric inside UDT definition |
| dataType | String | Datatype of the metric. Allowed values:   * Boolean * UInt8 * UInt16 * UInt32 * UInt64 * Int8 * Int16 * Int32 * Int64 * Float * Double * String * UDT |
| value | As per dataType | Value of a metric as per dataType defined above |
|  | | |
| parameters | List of parameters | Each parameter contains following fields:   * name * dataType * value   These fields are like the metric fields, explained earlier.  Parameters are provided for an App in DBIRTH message. |

Example:

{

"udt\_name": "custom\_udt",

"version": "1.0",

"metrics": [

{

"name": "M1",

"dataType": "String",

"value": ""

},

{

"name": "RTU\_Time",

"dataType": "Int32",

"value": 0

}

],

"parameters": [

{

"name": "P1",

"dataType": "String",

"value": ""

},

{

"name": "P2",

"dataType": "Int32",

"value": 0

}

]

}

Data Flow:

App should use this message to provide definition of a Sparkplug Template i.e., UDT. UDT definitions are published as a part of NBIRTH message. Hence, after receiving a UDT definition, Sparkplug-Bridge publishes NDEATH and then NBIRTH to SCADA-Master.

### START\_BIRTH\_PROCESS

MQTT Topic: START\_BIRTH\_PROCESS

Message format:

It is an empty JSON format message:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Datatype** | **Description** |

Example:

{

}

Data Flow:

This message is published by Sparkplug-Bridge over MQTT broker and subscribed by App. This message tells the App to publish following:

* Definition of Sparkplug Templates i.e., UDT which are used by App in BIRTH message
* BIRTH messages for all SUBCLASS the App is having. The App shall publish BIRTH messages on receiving START\_BIRTH\_PROCESS message.

START\_BIRTH\_PROCESS message will be sent on restart of Sparkplug-Bridge container or whenever Sparkplug-Bridge container needs to refresh the data that it maintains for virtual devices.

## Modbus (real) device communication

Modbus devices present in network are reported to SCADA Master using SparkPlug mechanism.

Apps and Sparkplug-Bridge communicate over internal MQTT using above defined messages.

### Support for DBIRTH

Data from device YML configuration files is used to form a DBIRTH message for real devices at the start of Sparkplug-Bridge container. One datapoint YML file corresponds to one SparkPlug template definition. One real Modbus device contains one metric of type SparkPlug template. The SparkPlug template in turn contains all other metrics which correspond to datapoints mentioned in datapoints-YML file.

### Support for DDATA

Data from polling operation published by MQTT-Bridge over internal MQTT is used to determine a change in value of any of metrics associated with a real device. If a change is detected, a DDATA message is published by Sparkplug-Bridge.

### Support for DCMD

When a DCMD message is received from a SCADA Master for a real device for a “known metric”, then an on-demand write operation is initiated by SCAD-RTU and sent to MQTT-Bridge over internal MQTT.

Notes:

* A “known metric” is one which is present in device YML configuration file. The name and datatype for a “known metric” in DCMD message and YML file shall match.
* A DCMD message can result in multiple on-demand write operations.

### Support for DDEATH

Data from polling operation published by MQTT-Bridge over internal MQTT is used to determine whether a device is reachable or not, based on error\_code. If device unreachable error-code is found, a DDEATH message is published by Sparkplug-Bridge. When correct values are found, a DBIRTH message is published.

## SparkPlug Messages

Refer SparkPlug standard for more information.

### NBIRTH Message

NBIRTH is Node-Birth.

On start-up, Sparkplug-Bridge module publishes this message over MQTT broker. The message is published in SparkPlug encoded format.

For Modbus real device, one datapoint YML file corresponds to one SparkPlug template. These template definitions are sent in NBIRTH message. DBIRTH message for Modbus device specifies a particular SparkPlug template.

Following are sample contents in simplified JSON format:

Topic: spBv1.0/UWC nodes/NBIRTH/RBOX510-00

Message:

{

"timestamp": 1608243262157,

"metrics": [

{

"name": "Name",

"timestamp": 1608243262157,

"dataType": "String",

"value": "SCADA RTU"

},

{

"name": "bdSeq",

"timestamp": 1608243262157,

"dataType": "UInt64",

"value": 0

},

{

"name": "Node Control/Rebirth",

"timestamp": 1608243262157,

"dataType": "Boolean",

"value": false

},

{

"name": "iou\_datapoints",

"timestamp": 1608243262157,

"dataType": "Template",

"value": {

"version": "1.0.0",

"reference": "",

"isDefinition": true,

"metrics": [

{

"name": "D1",

"timestamp": 1608243262157,

"dataType": "String",

"properties": {

"Pollinterval": {

"type": "UInt32",

"value": 0

},

"Realtime": {

"type": "Boolean",

"value": false

}

},

"value": ""

},

{

"name": "D2",

"timestamp": 1608243262157,

"dataType": "String",

"properties": {

"Pollinterval": {

"type": "UInt32",

"value": 0

},

"Realtime": {

"type": "Boolean",

"value": false

}

},

"value": ""

}

],

"parameters": [

{

"name": "Protocol",

"type": "String",

"value": ""

}

]

}

}

],

"seq": 0,

"uuid": "SCADA-RTU"

}

### NDEATH Message

NDEATH is Node-Death.

Whenever Sparkplug-Bridge module’s connection with MQTT broker breaks, the MQTT broker publishes this message. The message is published in text format.

Following are sample contents in simplified JSON format:

Topic: spBv1.0/UWC nodes/NDEATH/RBOX510-00

Message: {

"timestamp": 1592306298537,

"metrics": [

{

"name": "bdSeq",

"alias": 10,

"timestamp": 1592306298537,

"dataType": "UInt64",

"value": 0

}

],

"seq": 0

}

### DBIRTH Message

DBIRTH is Device-Birth.

On start-up, Sparkplug-Bridge module publishes this message over MQTT broker. The message is published in SparkPlug encoded format.

Following are sample contents in simplified JSON format for a Modbus device:

{

"timestamp": 1608242600219,

"metrics": [

{

"name": "iou",

"timestamp": 1608242600219,

"dataType": "Template",

"value": {

"version": "1.0.0",

"reference": "iou\_datapoints",

"isDefinition": false,

"metrics": [

{

"name": "D1",

"timestamp": 1608242599889,

"dataType": "String",

"properties": {

"Pollinterval": {

"type": "UInt32",

"value": 1000

},

"Realtime": {

"type": "Boolean",

"value": false

}

},

"value": ""

},

{

"name": "D2",

"timestamp": 1608242599889,

"dataType": "String",

"properties": {

"Pollinterval": {

"type": "UInt32",

"value": 1000

},

"Realtime": {

"type": "Boolean",

"value": false

}

},

"value": ""

}

],

"parameters": [

{

"name": "Protocol",

"type": "String",

"value": "Modbus RTU"

}

]

}

}

],

"seq": 1

}

### DDEATH Message

DDEATH is Device-Death.

Sparkplug-Bridge module publishes this message over MQTT broker whenever it detects that device is not reachable. The message is published in SparkPlug encoded format.

Following are sample contents in simplified JSON format:

{

"timestamp":1599467927490,

"metrics":[],

"seq":7

}

### DDATA Message

DDATA is Device-Data.

Sparkplug-Bridge module publishes this message over MQTT broker whenever it detects a change in value of any of metrics of devices. The message is published in SparkPlug encoded format.

Following are sample contents in simplified JSON format for a Modbus device:

{

"timestamp": 1608242631070,

"metrics": [

{

"name": "iou",

"timestamp": 1608242631070,

"dataType": "Template",

"value": {

"version": "1.0.0",

"reference": "iou\_datapoints",

"isDefinition": false,

"metrics": [

{

"name": "D1",

"timestamp": 1571887474111145,

"dataType": "String",

"value": "0x00"

}

]

}

}

],

"seq": 2

}

### NCMD Message

NCMD is Node-Command.

SCADA Master can tell edge node to reinitiate the node birth process. The node starts publishing NBIRTH, DBIRTH messages after receiving NCMD.

Following are sample contents in simplified JSON format:

Topic: spBv1.0/UWC nodes/NCMD/RBOX510-00

Message: {

"timestamp": 1615619351980,

"metrics": [

{

"name": "Node Control/Rebirth",

"timestamp": 1615619351980,

"dataType": "Boolean",

"value": true

}

],

"seq": -1

}

# Debugging steps

**Checking logs**

**1**. Syntax - sudo docker logs <container\_name>

e.g.:- To check modbus-tcp-container logs execute "sudo docker logs modbus-tcp-container" command.

**2**. Command to check logs inside the container "sudo docker exec -it <container\_name> bash" and go to the “logs” directory using “cd logs”.

**3**. Use "cat <log\_file\_name>" to see log file inside the container

**4**. Copying logs from container to host machine

Syntax - docker cp <container\_name>:<file to copy from container> <file to be copied i.e. host directory>

**5**. To check the IP address of machine, use "ifconfig" command.

**6**. For Modbus RTU, to check attached COM port for serial communication, use "dmesg | grep tty" command.

Redirect docker logs to file including errors

docker logs modbus-tcp-container > docker.log 2>&1

Accessing container logs through docker volumes:

Go to docker volume directory using

cd /opt/intel/eii/container\_logs/<container\_name>

Where <container name> is directory name for each container (i.e. “modbus-tcp-master”, “modbus-rtu-master”, “mqtt-bridge”, etc).

Example to access container logs for modbus-tcp-master container,

cd /opt/intel/eii/container\_logs/modbus-tcp-master

cat Modbus\_App.log

Please note: These logs are persisted across container/gateway restart.

Steps to apply new configuration (i.e. YML files)

Once YML files/docker-compose.yml are changed/Modified in /opt/intel/eii/uwc\_data directory then, execute following command to apply new configurations,

sudo ./05\_applyConfigChanges.sh

How to bring up/down UWC containers

docker-compose down - bring down all containers

docker-compose up - bring up all containers

# Error Codes

**MODBUS\_EXCEPTION Codes**

Response JSON message from Modbus container contains a code in case of error scenario.

**Format:** “error\_code”: “number”

There are 3 classes of error codes. This section lists down the error codes:

| Error Code | Description |
| --- | --- |
| **Modbus Application Error Codes** | |
| 0 | APP\_SUCCESS |
| 100 | APP\_ERROR\_DUMMY\_RESPONSE |
| 101 | APP\_ERROR\_REQUEST\_SEND\_FAILED |
| 102 | APP\_ERROR\_INVALID\_INPUT\_JSON |
| 103 | APP\_ERROR\_CUTOFF\_TIME\_INTERVAL |
| 106 | APP\_JSON\_PARSING\_EXCEPTION |
| 107 | APP\_ERROR\_UNKNOWN\_SERVICE\_REQUEST |
| 108 | APP\_ERROR\_POINT\_IS\_NOT\_WRITABLE |
|  |  |
| **Modbus Protocol Standard Error Codes (Returned by slave devices)** | |
| 1001 | ILLEGAL FUNCTION |
| 1002 | ILLEGAL DATA ADDRESS |
| 1003 | ILLEGAL DATA VALUE |
| 1010 | GATEWAY PATH UNAVAILABLE |
|  |  |
| **Modbus Protocol Stack Error Codes** | |
| 2001 | STACK\_TXNID\_OR\_UNITID\_MISSMATCH |
| 2002 | STACK\_ERROR\_SOCKET\_FAILED |
| 2003 | STACK\_ERROR\_CONNECT\_FAILED |
| 2004 | STACK\_ERROR\_SEND\_FAILED |
| 2005 | STACK\_ERROR\_RECV\_FAILED |
| 2006 | STACK\_ERROR\_RECV\_TIMEOUT |
| 2007 | STACK\_ERROR\_MALLOC\_FAILED |
| 2008 | STACK\_ERROR\_QUEUE\_SEND |
| 2009 | STACK\_ERROR\_QUEUE\_RECIVE |
| 2010 | STACK\_ERROR\_THREAD\_CREATE |
| 2011 | STACK\_ERROR\_INVALID\_INPUT\_PARAMETER |
| 2012 | STACK\_ERROR\_PACKET\_LENGTH\_EXCEEDED |
| 2013 | STACK\_ERROR\_SOCKET\_LISTEN\_FAILED |
| 2014 | STACK\_ERROR\_MAX\_REQ\_SENT |
| 2015 | STACK\_ERROR\_FAILED\_Q\_SENT\_REQ |
| 2016 | STACK\_INIT\_FAILED |
| 2017 | STACK\_ERROR\_QUEUE\_CREATE |
| 2018 | STACK\_ERROR\_MUTEX\_CREATE |
| 2019 | STACK\_ERROR\_STACK\_IS\_NOT\_INITIALIZED |
| 2020 | STACK\_ERROR\_STACK\_IS\_ALREADY\_INITIALIZED |
| 2021 | STACK\_ERROR\_SERIAL\_PORT\_ERROR |

# Steps to flash LFM BIOS

LFM is Low Frequency Mode. This mode is enabled on gateway for efficient power usage.

Following steps and BIOS are applicable for below model of Device:

RBOX510 ATEX & C1D2 ANTI-EXPLOSIVE CERTIFIED ROBOST DIN-RAIL FANLESS SYS.W/ATOM E3827(1.75GHz)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **LFM (500)** | **LFM\_1500** | **Original** |
|  | **Axiomtek ICO-300** | 87842XV.102 | None | 87842V.105 |
|  | **Axiomtek RBOX510** | XN.001 | 0502\_1500M.ROM | A.103 |

**Steps for converting to LFM**

Use files from **0502\_1500m.zip** folder

1. Create a FAT32 file-system on a USB stick, then dump each BIOS file structure to the root as needed.
2. Prepare the LFM BIOS first on USB, then boot to the USB’s EFI device (boot priorities).
3. After setting the Boot priorities save & exit from boot menu. You will be switched to Command window. Press any key to enter to command line, type flash.nsh to update the BIOS.
4. Restart the HW and you will now be locked at 1500MHz
5. Check frequency of device using command $ cat /proc/cpuinfo | grep “MHz”

**Steps for reverting to normal Frequency**

Use files from **A.103** folder

1. Create a FAT32 file-system on a USB stick, then dump each BIOS file structure to the root as needed.
2. Prepare the LFM BIOS first on USB, then boot to the USB’s EFI device (boot priorities).
3. After setting the Boot priorities save & exit from boot menu. You will be switched to Command window. Press any key to enter to command line, type flash.nsh to update the BIOS.
4. Restart the HW and you will now be locked at 1500MHz
5. Check frequency of device using command $ cat /proc/cpuinfo | grep “MHz”

# Steps to Apply RT Patch

## Install prerequisites

Install all the prerequisites using following command

sudo apt-get install -y libncurses-dev libssl-dev bison flex build-essential wget

**Note:** It will prompt to update package runtime, click on yes button.

## Keyboard shortcuts for menuconfig UI

|  |  |  |
| --- | --- | --- |
| # | **Task/Use** | **Keyboard Key** |
| 1 | To select specific kernel feature | Space bar |
| 2 | To come out of current window | Esc Esc |
| 3 | To save current setting | Click on <save> button |
| 4 | Exit | Click on <exit> button |

## Steps to apply PREEMPT\_RT patch

This section provides steps to apply PREEMPT\_RT patch

**Steps:**

1. Make a working directory on system

$ mkdir ~/kernel && cd ~/kernel

1. Download kernel in ~kernel directory created in step 1

* **Download kernel manually**

Link for download - <https://www.kernel.org/pub/linux/kernel/>

This will download kernel manually or use following command to download it from command line inside current directory.

$ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.19.72.tar.gz

**Recommendation:** Please get Linux kernel version 4.19.72

* **Download preempt rt patch**

Link for download - <https://www.kernel.org/pub/linux/kernel/projects/rt/> this will download patch manually or use following command to download it from command line inside current directory.

wget https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/4.19/older/patch-4.19.72-rt26.patch.gz

**Recommendation:** Please get PREEMPT\_RT version 4.19.72-rt26

1. Unzip the kernel using following command

$ tar -xzvf linux-4.19.72.tar.gz

1. Patch the kernel

$ cd linux-4.19.72

$ gzip -cd ../patch-4.19.72-rt26.patch.gz | patch -p1 --verbose

1. Launch the graphical UI for setting configurations

The next command launches a graphical menu in the terminal to generate the .config file

$ make menuconfig

**Graphical UI is shown below:**

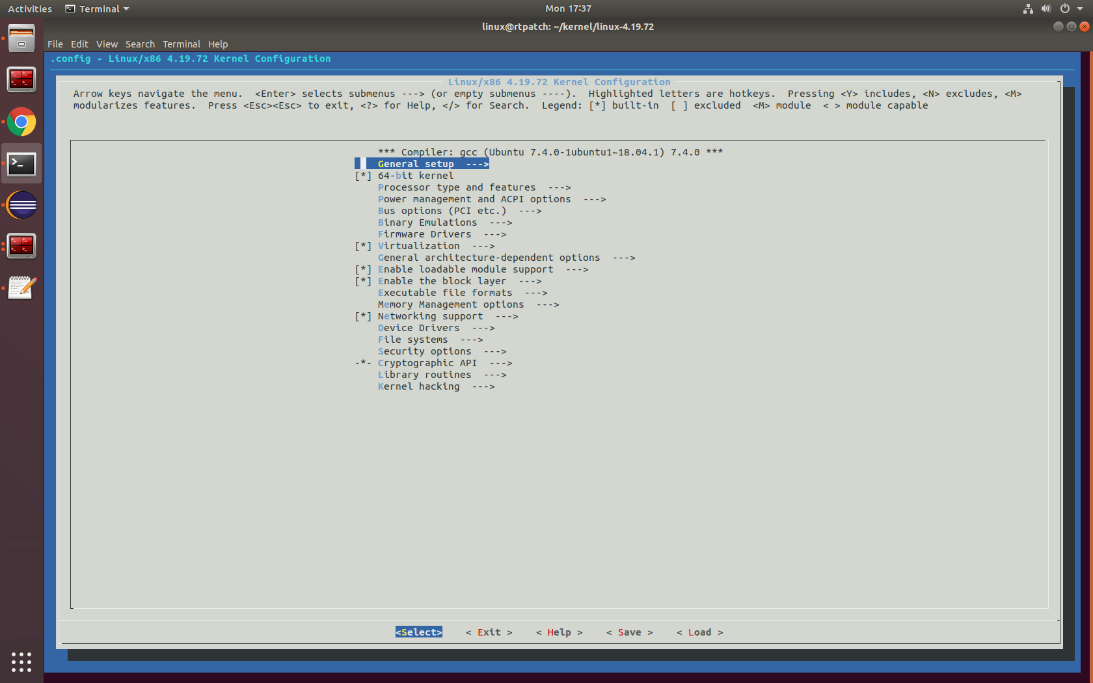


Figure 2: Main launching screen

1. Select the preemption model as Basic RT using tab key on keyboard
2. Select and enter on “General setup” option.
3. Select and Enter on Preemption Model (Voluntary Kernel Preemption (Desktop))
4. Select and Enter on Preemption Model (Fully Preemptible Kernel (RT))
5. After successful selection click on save button and then come back to main page using Esc button on keyboard.

Refer the following screen capture for more details

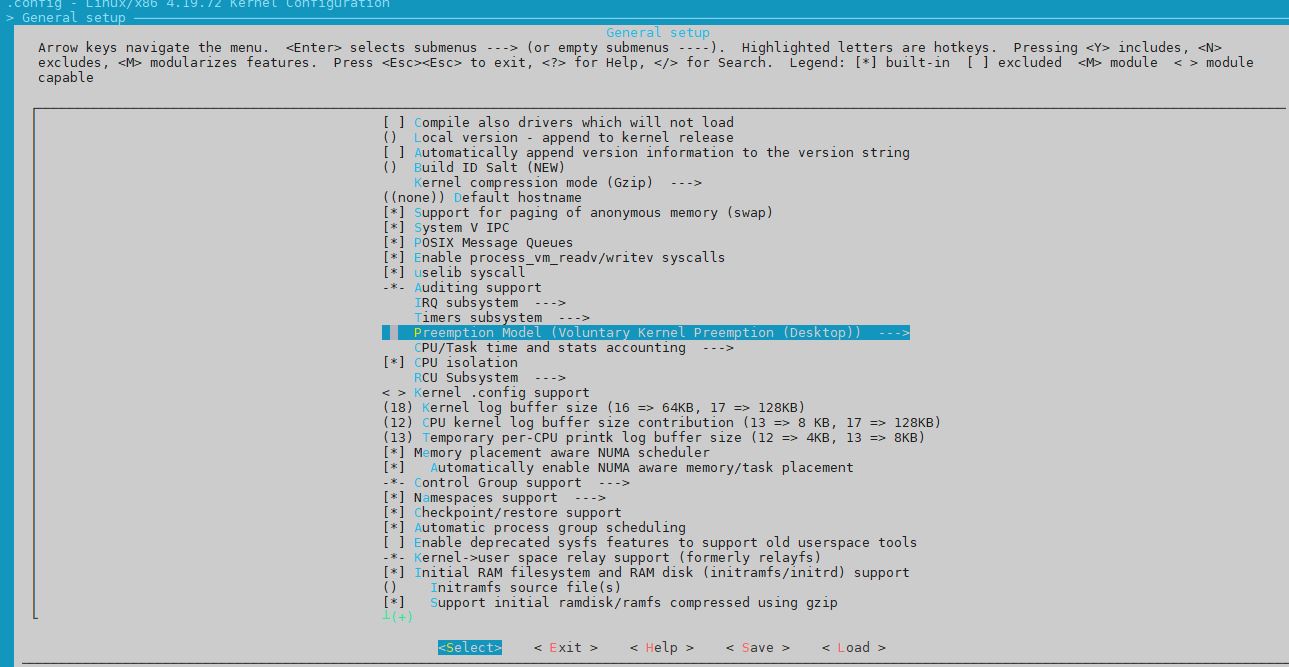


Figure 3 : Preemption Model (Voluntary Kernel Preemption (Desktop))

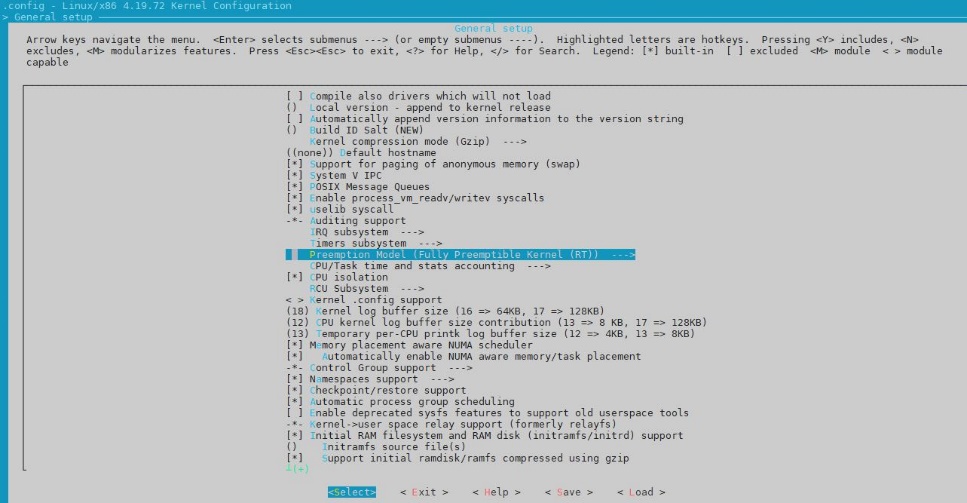
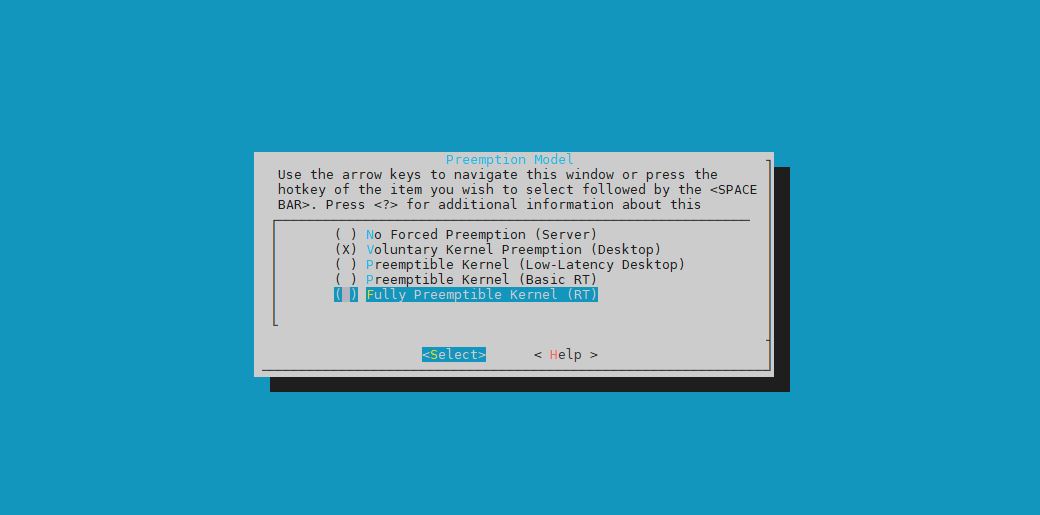


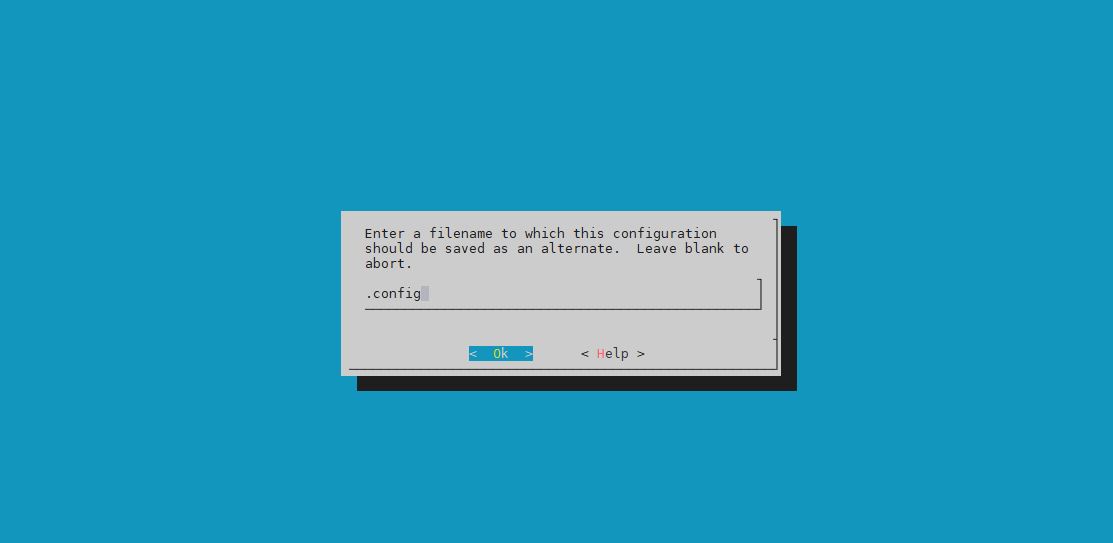
Figure 4: Preemption Model (Fully Preemptible Kernel (RT))



**Figure 5: Fully Preemption Kernel (RT)**

Save and exit

To save the current setting click on <save> button and then exit the UI using <exit> button.



**Figure 6: Click on ‘OK’**

1. Compile the kernel (Execute the following commands)

$ make –j20

$ sudo make INSTALL\_MOD\_STRIP=1 modules\_install -j20

$ sudo make install -j20

1. Verify and update Verify that initrd.img-4.19.72-rt26, vmlinuz-4.19.72-rt26, and config-4.19.72-rt26 are generated in /boot directory and update the grub.

$ cd /boot

$ ls

$ sudo update-grub

Verify that there is a menuentry containing the text "menuentry 'Ubuntu, with Linux 4.9.72-rt26'" in /boot/grub/grub.cfg file

To change default kernel in grub, edit the GRUB\_DEFAULT value in /etc/default/grub to your desired kernel.

NOTE: 0 is the 1st menuentry

1. Reboot and verify using command

$ sudo reboot

Once the system reboots, open the terminal and use uname -a to check the kernel version

Command will show below output for successfully applied RT patch –

Linux ubuntu 4.19.72-rt26 #1 SMP PREEMPT RT Tue Mar 24 17:15:47 IST 2020 x86\_64 x86\_64 x86\_64 GNU/Linux

## References

[*https://stackoverflow.com/questions/56189710/how-to-enable-config-rt-group-sched-in-ubuntu-to-make-it-rt*](https://stackoverflow.com/questions/56189710/how-to-enable-config-rt-group-sched-in-ubuntu-to-make-it-rt)

[*https://stackoverflow.com/questions/51669724/install-rt-linux-patch-for-ubuntu*](https://stackoverflow.com/questions/51669724/install-rt-linux-patch-for-ubuntu)

[*https://unix.stackexchange.com/questions/270390/how-to-reduce-the-size-of-the-initrd-when-compiling-your-kernel/270418*](https://unix.stackexchange.com/questions/270390/how-to-reduce-the-size-of-the-initrd-when-compiling-your-kernel/270418)

# Appendix

UWC contains below folders/files:

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Folder/File** | **Destination** | **Comment** |
| 1 | Modbus-master | IEdgeInsights/uwc | Modbus application folder used to install TCP/RTU container |
| 2 | MQTT | IEdgeInsights/uwc | It is used to install mosquito (mqtt) container |
| 3 | mqtt-bridge | IEdgeInsights/uwc | Mqtt-bridge application folder used to install mqtt-bridge container |
| 4 | sparkplug-bridge | IEdgeInsights/uwc | Sparkplug-Bridge application folder used to install Sparkplug-Bridge container. |
| 5 | kpi-tactic | IEdgeInsights/uwc | kpi-tactic application folder used to install kpi-app container. |
| 6 | Others | /opt/intel/eii/uwc\_data/ | All yml files containing device, datapoints etc. configurations. It also contains Global\_config.yml |
| 7 | uwc\_common | IEdgeInsights/uwc | Common libraries installation docker file and source code. |