4. To change the default message directory on the device, select the checkbox next to `rsuMessagesPath` and enter a new directory.

5. Click on the Save & Apply button for the changes to take effect on the device.

Alternatively, the `muci` tool can be used as follows:

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. To set a new UDP listening port number, run the following command:

```
muci set ifm.listenPort <new port number>
```

3. To set a new message directory on the device, run the following command:

```
muci set ifm.rsuMessagesPath </new directory>
```

4. Restart the stack using the command

```
unplugged-rt-restart.sh
```

5.3.2. Formatting IFMs

Please note that all IFMs need to be in RSU 4.1 format. The message format needs to match the regional standard.

The following example shows a US-standard IFM:

```
Version=0.7
Type=BSM
PSID=20
Priority=0
TxMode=CONT
TxChannel=SCH
TxInterval=0
DeliveryStart=
DeliveryStop=
Signature=False
Encryption=False
Payload=0012...
```

The following example shows an IFM adapted for ETSI standard messages:

```
Version=0.7
Proto=GNP
GnMethod=GBC
DestinationAreaType=circle
DestinationAreaLatitude=456612674
DestinationAreaLongitude=81757023
DestinationAreaDistanceA=1000
DestinationAreaDistanceB=0
DestinationAreaAngle=0
Type=DENM
PSID=25
Priority=1
TxMode=CONT
TxChannel=180
TxInterval=0
DeliveryStart=
DeliveryStop=
Signature=False
Encryption=False
Payload=0201...
```

As can be seen in both examples, the transmission interval (`TxInterval`) needs to be set to 0, while the `DeliveryStart` and `DeliveryStop` fields needs to be left empty.

Please ensure that the signing of the messages are compatible with the security settings of the device:

- If the system is enrolled and the security is turned on, use `Signature=True` to transfer signed IFMs. Setting the signature to `False` is typically not recommended on an enrolled device.
- If the system is not enrolled or security is turned off, use `Signature=False` to trasmit unsigned IFMs. **Do not** set the signature to `True` on such devices, as the messages cannot be signed, and fail to be transmitted.

5.4. Deploying and transmitting SRMs/IFMs

To deploy a message on the RSU and start the transmission of the messages, proceed as follows:

1. Copy an SRM or IFM in the appropriate format described above to the messages folder on the RSU using SCP as follows:

   ```
   scp <msg_file> root@<IP address of>:/rwdata/etc/rsu_msgs/<msg_file>
   ```

   Alternatively, WinSCP can be used on Windows computers.

   If the `scp` command returns with the error message `ash: /usr/libexec/sftp-server: not found` and fails, the -O switch needs to be used after the command.

   To use the ASN1.X tool to generate the message payload, refer to the section "Converting data formats" [23]. Messages can be modified over SNMP as well.

2. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

3. Use the following command to restart the SRM-IFM tool:

```
/etc/init.d/srm-ifm-tool restart
```

   Upon restarting the tool, the software stack on the device reads the message files in the `rsu_msgs` folder and transmit them. If the message was modified over SNMP, then the software stack does not need to be restarted.

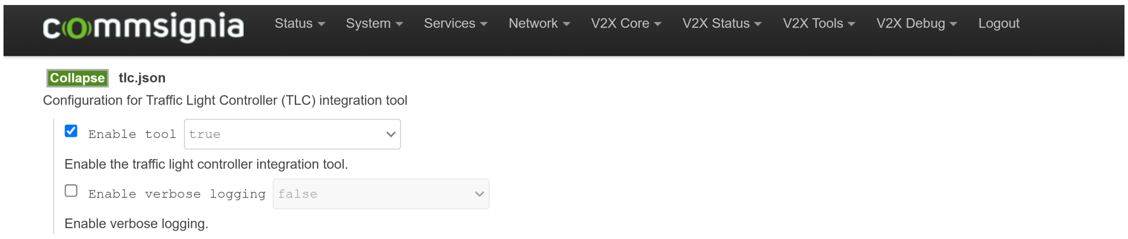
The transmission of messages can be verified under the **V2X Status** → **Status** menu by expanding the **srmStatictics** option. If the values of the counters are not increasing, please refer to the section "Troubleshooting V2X communication" [59].

5.5. Transmitting Signal Phase and Timing (SPaT) messages on RSUs

Signal Phase and Timing messages are used to relay traffic light information in an intersection. The message contains identifiers of the traffic lights, traffic light groups, intersection lanes, and other information related to traffic control. Before enabling SPaT message transmission, please make sure that the RSU is integrated to the TLC.

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. To enable the transmission of SPaT content from the connected TLC, open the **V2X Tools** → **TLC** menu, check the box near `Enable tool`, and set it to `true`, as shown in Figure 32.



tlc.json

Configuration for Traffic Light Controller (TLC) integration tool

☑ Enable tool: true

Enable the traffic light controller integration tool.

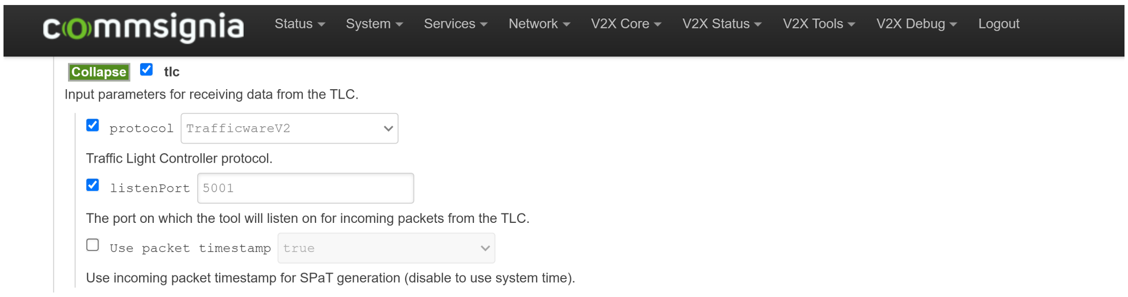
☐ Enable verbose logging: false

Enable verbose logging.

Figure 32. TLC configuration page

3. Configure the appropriate protocol by expanding the **tlc** option.

   a. For the TrafficWare proprietary UDP bit stream, check the box near `protocol` and select `TrafficwareV2` or `TrafficwareV3` as applicable, as shown in Figure 33.



☑ tlc

Input parameters for receiving data from the TLC.

☑ protocol: TrafficwareV2

Traffic Light Controller protocol.

☑ listenPort: 5001

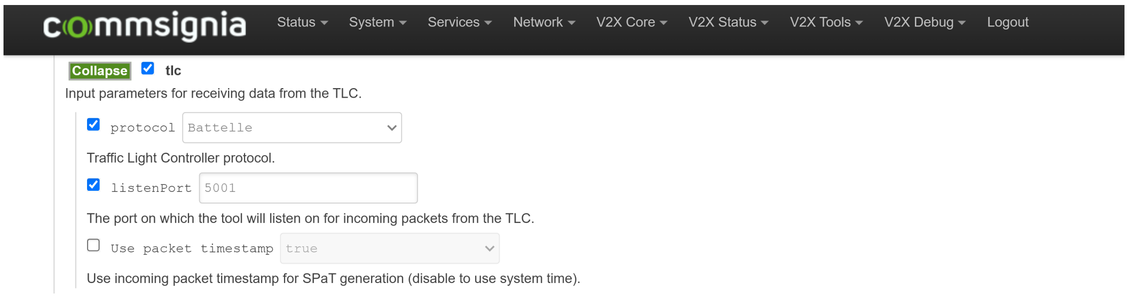
The port on which the tool will listen on for incoming packets from the TLC.

☐ Use packet timestamp: true

Use incoming packet timestamp for SPaT generation (disable to use system time).

Figure 33. TrafficWare protocol settings

   b. For the Battelle format, check the box near `protocol` and select `Battelle`, as shown in Figure 34.



☑ tlc

Input parameters for receiving data from the TLC.

☑ `protocol` Battelle

Traffic Light Controller protocol.

☑ `listenPort` 5001

The port on which the tool will listen on for incoming packets from the TLC.

☐ `Use packet timestamp` true

Use incoming packet timestamp for SPaT generation (disable to use system time).

Figure 34. Battelle protocol settings

The port for both protocols can be specified as well. Please make sure that the port is not blocked in the Firewall settings (see Firewall settings for further information).

4. An intersection ID for the TLC can be defined and signal groups can be managed by expanding the **TLC to V2X mapping** option.

5. Click on the Save & Apply button for the changes to take effect on the device.

The transmission of SPaT messages can be verified under the **V2X Status** → **Status** menu by expanding the **tlcStatictics** option and expanding counters **receivedPackets** and **sentPackets**. If the values of the counters are not increasing, please refer to the section "Troubleshooting V2X communication" [59].

6. Additional features

6.1. Datalogger tool

The Datalogger is a tool provided by Commsignia as part of the software stack running on the device. It collects and processes data, from the V2X device, such as V2X messages or navigation data, based on a pre-defined set of filters for data analysis, such as troubleshooting, central data collection, real-time data processing.

6.1.1. Description of the Datalogger tool

The Datalogger tool processes all JSON configuration files in the `/rwdata/v2x_configs/data_logger_ftw` directory, where the required data sources, filters, and destination options can be defined by the user. The tool gathers information from all defined data sources, automatically organize these inputs in a queue, and then applies the user-defined filters and outputs the results to the chosen destination type; the process is shown in Figure 35.

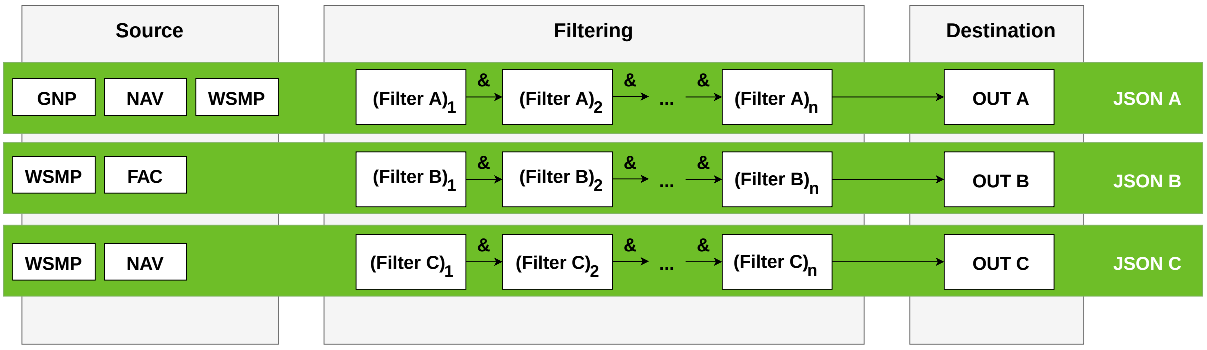


Figure 35. Schematic of the Datalogger operation

6.1.1.1. Source

The following type of sources can be defined in the configuration file as strings: "Gnp" (GeoNetworking), "Nav" (Navigation), "Wsmp" (Wave Short Message Protocol), or "Fac" (Facility). Multiple sources can be defined in the configuration file.

6.1.1.2. Filtering

Several filtering options can be set in the configuration file; filters are applied sequentially and every filter can obtain a different data packet from the data sources available in the queue. Invalid filter configurations do not cause the Datalogger to halt but may result in warning messages in the data output. Table 1 shows the configurable filtering options available for different inputs.

Table 1. Filtering options for the Datalogger tool

Filter name	Type	Description	Note
`"btpPost"`	Integer	Geonet BTP destination port	Only Geonet packets can be filtered out
`"direction"`	String enum	Packet direction (NAV packets are forwarded)	Values: `"In,"` `"Out,"` and `"Both"`
`"interval"`	Integer	Save only every *N*th packet	Value must be greater or equal to 1
`"psid"`	Integer	WSMP PSID	Only WSMP packets are filtered out

Filter name	Type	Description	Note
`"radioInterfaceName"`	String	Interface name	"Non-radio packets are forwarded (preferred over "radioInterface')
`"radioInterface"`	Integer	Interface ID	Non-radio packets are forwarded
`"facMessageType"`	String enum	Facility message types to forward	Non facility messages are forwarded
`"rssi"`	Integer	Save only packets where RSSI exceeds this number in [dBm]	Only incoming radio packets can be filtered out. In case of C-V2X communication, the value of RSSI is always 0.
`"start"`	Integer	Save only after this Unix timestamp (in [ms])	Value must be greater or equal to 1072915200000
`"stop"`	Integer	Save only after this Unix timestamp (in [ms])	Value must be greater or equal to 1072915200000

6.1.1.3. Destination

6.1.1.3.1. Output types

The user definable output types are "socket" and "file." Table 2 summarizes the available options for different output types.

Table 2. Options for different output types

Output type	Option	Values (Type)	Note
`socket`	`protocol`	`udp` (string) `tcp` (string)	Communication protocol can be TCP or UDP
	`destHost`	(string)	IPv6 or IPv4 address of the destination
	`detPort`	(integer)	Value must be greater than or equal to 0 and less than or equal to 65535
`file`	`fileName`	(string)	File name with full path; the path must exists. If name contains %d, it will be replaced with the current date
	`maxAge`	(integer)	Maximum age of the file in [s]. If fileName contains %d, then the file will be rotated, otherwise the capture stops. Value must be greater than or equal to 1
	`maxSIze`	(integer)	Maximum file size in kB. Value must be greater than or equal to 1

6.1.1.3.2. Output formats

The "format" object in the JSON schema describes the output format of the gathered data, which can be either "raw" or "prefixed."

Raw data is the unmodified filtered data collected from the specified data sources. The message content will be the same as the original content, for example NAV data will result in a raw NAV data output.

Prefixed or encapsulated format will result in a header and a metadata footer attached to the data output. The files saved in the prefixed output format consist of a header, content, and metadata. The header contains the size of the content payload and everything after that is the metadata of the output. Metadata contains the Packet ID which is equal to the PSID or BTP port of the original V2X message content. This can be used to differentiate messages sent to the same recipient because raw data will not contain this information. This can also be used to differentiate between different types of V2X messages as well.

The Datalogger tool uses the following UPER encoded data formats:

- MSG_FRAME for WAVE radio data
- CAM/DENM/etc for ETSI radio data

Navigation data

Navigation data has the following directly serialized C++ struct format, as it can be found in the `include/cms_v2x/nav.h` header file in the Remote C SDK:

```
typedef struct cms_nav_fix_t {

    /** Flag to indicate whether the NAV fix is valid.
    @note If this flag is true, all other fields will be valid, recent and
    consistent. If the flag is false, either all fields will be N/A, or they
    will contain the most recent consistent data, but it can be very old. */
    bool is_valid;

    cms_utc_timestamp_ms_t timestamp;              /
**< UTC milliseconds since Unix epoch */
    uint64_t leap_seconds;                         /
**< Leap seconds, i.e the difference between UTC and TAI in [s] */
    cms_latitude_t latitude;                       /**< Latitude angle */
    cms_longitude_t longitude;                     /**< Longitude angle */
    cms_altitude_t altitude;                       /**< Altitude */
    cms_altitude_t altitude_confidence;            /**< Confidence of altitude */
    cms_length_t pce_semi_major;                   /
**< Position Confidence Ellipse: half of major axis length */
    cms_length_t pce_semi_minor;                   /
**< Position Confidence Ellipse: half of minor axis length */
    cms_heading_t pce_orientation;                 /
**< Position Confidence Ellipse: direction of the major axis */
    cms_heading_t heading;                        /**< Heading of the motion */
    cms_heading_confidence_t heading_confidence;  /**< Confidence of heading */
    cms_speed_t speed;                            /**< Speed of the motion */
    cms_speed_t speed_confidence;                 /**< Confidence of speed */
    cms_nav_drive_direction_t drive_direction;    /**< Drive direction */
    uint8_t number_of_used_satellites;            /
**< Satellites used to determine position */
} CMS_PACKED cms_nav_fix_t;
```

The types are defined in the `include/cms_v2x/common_types.h` header in Unplugged_RT.

6.1.2. Configuring the Datalogger

If the Datalogger tool finds at least valid JSON configuration file in the `/rwdata/v2x_configs/data_logger_ftw` directory of the device, then it starts and runs according to the configuration.

If there is no JSON file in the directory, the Datalogger prints the following message into the system-log and stops:

```
data-logger-ftw: No configuration file at /etc/data_logger_ftw/
```

In case of an invalid JSON file, such as syntax or sequence error, the Datalogger prints the following message into the systemlog:

```
data-logger-ftw[13439]: INFO: Skipping invalid/
disabled configuration file: dsrcfwd_02.json
```

If all JSON files are invalid, , the Datalogger prints the following message into the systemlog and quits:

```
data-logger-ftw[13439]: ERROR:   No valid & enabled configuration files found!
```

In the following example, WSMP packets are filtered and sent to an UDP port.

```
{
  "enabled": true,
  "source": [ "Wsmp" ],
  "filters": [
    { "start": 1686838229000 },
    { "stop": 1786838229000  },
    { "psid": 32  },
    { "rssi": -90 },
    { "interval": 3 }
  ],
  "out": {
    "format": "Prefixed",
    "socket":{
      "protocol":"udp",
      "destHost": "192.168.9.192",
      "destPort": 42000
    }
  },
  "version": 6
}
```

To create a configuration file proceed as follows:

1. Enable the tool as:

   ```
   "enabled": true,
   ```

2. Set the source according to the description given in section “Source” [30]. Several sources can be given in the schema such as:

   ```
   "source": [
       "Wsmp",
       "Gnp",
       "Nav"
     ],
   ```

3. Configure filters according to the description given in section “Filtering” [30]. In the example above only the necessary options are shown.

   Please note that the sequence of filters is important. For example, always set the PSID first and than the interval.

   a. The "start" and "stop" fields give the starting and stopping times of the logging in Unix time in [ms]. For example, to convert Sunday, July 2, 2023, 11:45:00 local time to a Unix timestamp, the following command can be used:

```
date -d "2023-07-02 11:45:00" +%s
```

This results in a timestamp 1688291100, which needs to be extended with three digits representing milliseconds.

b. To filter the required message type, specify the **decimal value** of the required PSID in the "psid" field. The most commonly used PSID are summarized in Tables 3 and 4.

Table 3. PSIDs and message IDs of various messages (US)

Message	PSID (dec)	PSID (hex)	PSID (p-encoded)	Message ID
BSM	32	0x20	0p20	20
MISBEHAV	38	0x26	0p26	
PSM	39	0x27	0p27	32
RTCM	128	0x80	0p80-00	28
SPaT	130	0x82	0p80-02	19
TIM	131	0x83	0p80-03	31
PVD	132	0x84	0p80-04	26
WSA	135	0x87	0p80-07	
SDSM	144	0x90	0p80-10	
SSM	2113685	0x20-40-95	0pE0-00-00-15	30
SRM	2113686	0x20-40-96	0pE0-00-00-16	29
MAP	2113687	0x20-40-97	0pE0-00-00-17	18
RWA	2113689	0x20-40-99	0pE0-00-00-19	

Table 4. ITS-AID and message IDs of various messages (EU)

Message	ITS-AID (dec)	ITS-AID (hex)	ITS-AID (p-encoded)	Message ID	BTP dest port
CAM	36	0x24	0p24	2	2001
DENM	37	0x25	0p25	1	2002
TLM	137	0x89	0p80-09		
SPATEM	137	0x89	0p80-09	4	2004
RLT	138	0x8A	0p80-0A		
MAPEM	138	0x8A	0p80-0a	5	2003
IVIM	139	0x8B	0p80-0b	6	2006
SREM	140	0x8C	0p80-0c	9	2007
GNMGMT	141	0x8D	0p80-0D		
SSEM	637	0x02-7D	0p81-FD	10	2008
CPM	639	0x02-7F	0p81-FF		

c. Set the "rssi" field to log only signals with a given strength. The value needs to be in [dBm], and signals with strength greater than or equal to this value are logged.

d. To log every *N*th packet, set the "interval" field to *N*.

4. The destination parameters can be set under "out" according to section "Destination" [31]. In the example above a prefixed data format and an UDP sockets are used. To save the data in a file in raw format, the following example can be used:

```
  "out": {
    "format": "Raw",
    "file":{
      "fileName": "/var/log/iflog/if_0_in_%d.pcap",
      "maxAge": 40,
      "maxSize": 50
    }
```

Further configuration options are described in section "Description of the Datalogger tool" [30].

6.2. Integrating object detections from smart sensors into the Cooperative Filtering and Fusion framework

6.2.1. Overview

To insert object detections from an external smart sensor, such as a camera, into the Cooperative Filtering and Fusion (CFF) framework, a convenient interface based on a User Datagram Protocol (UDP) adapter is provided. Through this adapter, objects encoded in an appropriate form can be fed into the CFF, as shown in Figure 36. The encoded messages are expected on a preconfigured port encapsulated in UDP messages. The adaptation of a new smart sensor requires the development of an application that transforms the sensor measurements into a format compatible with the UDP adapter, the specification of the appropriate configuration of the message transmit in the configuration of the safety applications (saf.json), and the provision of certain sensor parameters in the CFF configuration (cff.json) if the detections are intended to be broadcast in such message types (e.g. as Collective Perception Messages (CPMs)). If required, the appropriate message transmission modules also need to be enabled in the stack, for example when the user intends to broadcast CPMs.

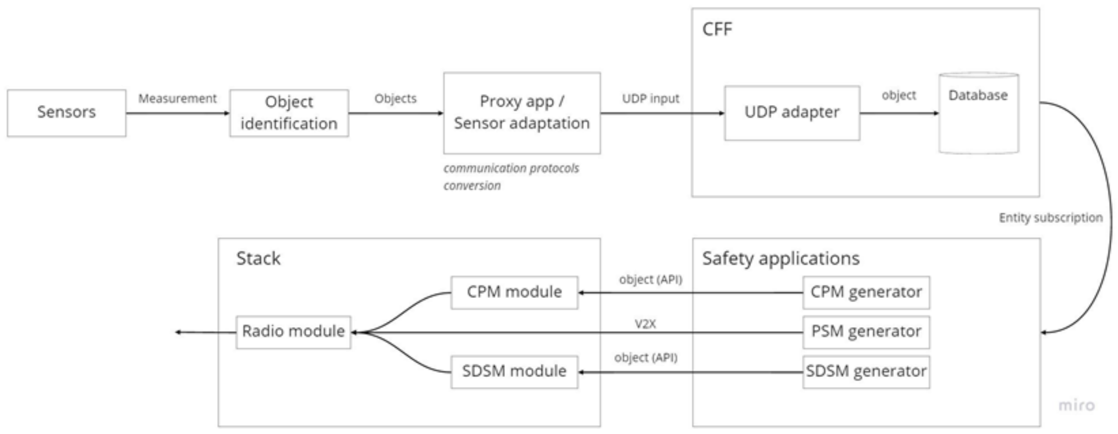


Figure 36. Schematics of smart sensor integration into the CFF

6.2.2. Interfacing with the CFF UDP adapter

The UDP adapter is designed to provide a convenient Python/C++ interface for injecting different entities into the CFF.

The message structure is defined in flatbuffers and communication is carried out via a configurable UDP port (12321 by default).

Currently, the adapter has limited functionality, only objects and a subset of their properties are supported. Similar to other CFF application programming interfaces (APIs), the convention is that angle related values need to be in degrees, timestamps in ms (Coordinated Universal Time (UTC)), while the rest of the quantities need to be in SI units.

6.2.3. Data requirements

In order to inject valid objects into the CFF, the object information has to contain valid position, heading, and speed values and the object needs to have a unique identifier. The adapter provides an interface to add various other optional parameters as well. If these values are available, the user may include them in the input data.

The CFF allows practically any values, provided that they are consistent (e.g., if vehicle sizes are known, then both the length/width parameters need to be added, not only one of them). It is the responsibility of the user to specify them correctly. However, V2X messages allow values satisfying certain range constraints. Therefore, if the detections need to be transmitted via CPM, sensor data sharing message (SDSM), or personal safety message (PSM), the supported value ranges need to be checked (e.g., in CPM messages, the object length and width need to be in the range of 0.1–102.2 m and its height in the range of 0.1–6.1 m).

6.2.4. Message composition using Python language

For message composition, helper classes and functions are available from Commsignia on request. The pattern for creating and sending input to a CFF instance is as follows.

- A builder object has to be created. This will serve as the buffer for the message contents and will encode the message once its construction is finished.
- The root object of the message to be sent over UDP is of type *Input*.
- Each object has the following creator function(s), depending on whether they are defined as structs or tables in flatbuffers:
  - structs have a creator function *Create<StructName>(builder, value1,..., valueN)*.
  - tables have a *<TableName>Start(builder)*, a *<TableName>End(builder)*, and specific setter functions for each field and follow the *<TableName>Add<FieldName>(builder, value)* naming pattern.
- Due to the nature of the builder class, the creation of tables cannot be embedded into each other, that is, an object needs to be created as a whole before creating another.
- Once the message fields are set, the byte stream can be created by calling the *Output()* function of the builder.

In the following, a simple example is provided, which injects an object into the CFF instance running on the local host.

To feed an object into the UDP adapter, a flatbuffers stream needs to be assembled first. For this, the flatbuffers package, together with the UDP adapter related packages need to be imported as shown below:

```
import flatbuffers
import socket
import time

from Cff.Api.UdpInput.AbsolutePosition import *
from Cff.Api.UdpInput.DataElement import *
from Cff.Api.UdpInput.Dimensions import *
from Cff.Api.UdpInput.Electronics import *
from Cff.Api.UdpInput.Input import *
from Cff.Api.UdpInput.Motion import *
from Cff.Api.UdpInput.Object import *
from Cff.Api.UdpInput.ObjectType import *
from Cff.Api.UdpInput.Position import *
from Cff.Api.UdpInput.RelativePosition import *
from Cff.Api.UdpInput.ValueWithConf import *
from Cff.Api.UdpInput.VehicleType import *
from Cff.Api.UdpInput.Version import *
```

Then, a flatbuffers builder of suitable size needs to be created, which will later be used to encode the UDP adapter input:

```
builder = flatbuffers.Builder(1024)
```

This builder can then be used to assemble the detected object step-by-step, as follows.

Assemble the dimensions as:

```
DimensionsStart(builder)
DimensionsAddLength(builder, 5.3)
DimensionsAddWidth(builder, 2.1)
DimensionsAddHeight(builder, 1.5)
dim = DimensionsEnd(builder)
```

Assemble the motion as:

```
RelativePositionStart(builder)       / AbsolutePositionStart(builder)
RelativePositionAddX(builder, 20.0)  / AbsolutePositionAddLatitude(builder, \
47.475555)
RelativePositionAddY(builder, 10.0)  / AbsolutePositionAddLongitude(builder, \
19.057777)
RelativePositionAddZ(builder, 5.0)   / AbsolutePositionAddAltitude(builder, 120.0)
position = RelativePositionEnd(builder) / position = \
AbsolutePositionEnd(builder)

MotionStart(builder)
MotionAddPosition(builder, position)

MotionAddPositionType(builder, Position().RelativePosition) /    \
MotionAddPositionType(builder, Position().AbsolutePosition)

MotionAddSpeed(builder, CreateValueWithConf(builder, 2.7, 1.0))
MotionAddAcceleration(builder, CreateValueWithConf(builder, -0.3, 1.0))
MotionAddHeading(builder, CreateValueWithConf(builder, 0.17, 1.0))
MotionAddYawRate(builder, CreateValueWithConf(builder, 2.1, 1.0))
motion = MotionEnd(builder)
```

Assemble the electronics as:

```
addElectronics = 0
if addElectronics:
    ElectronicsStart(builder)
    ElectronicsAddLeftTurnSignal(builder, 1)
    ElectronicsAddRightTurnSignal(builder, 0)
    electr = ElectronicsEnd(builder)
```

Assemble the object as:

```
ObjectStart(builder)
ObjectAddId(builder, 22)
ObjectAddObjectType(builder, ObjectType().Vehicle)
ObjectAddVehicleType(builder, VehicleType().Truck)
ObjectAddDimensions(builder, dim)
ObjectAddMotion(builder, motion)
if addElectronics:
    ObjectAddElectronics(builder, electr)
obj = ObjectEnd(builder)
```

Assemble the UDP input contents as:

```
InputStart(builder)
InputAddVersion(builder, Version.CurrentVersion)
InputAddSourceId(builder, 111)
InputAddTimestamp(builder, int(round(time.time() * 1000)))
InputAddDataType(builder, DataElement().Object)
InputAddData(builder, obj)
inputData = InputEnd(builder)
```

The root type is *Input*. Once it is created, the buffer needs to be finalized and the encoded data can be accessed as shown below:

```
builder.Finish(inputData)
data = builder.Output()
```

Finally, the encoded data can simply be sent via UDP to the appropriate IP address and port as

```
UDP_IP = "127.0.0.1"
UDP_PORT = 12321
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (UDP_IP, UDP_PORT))
```

6.2.5. Message composition using C++ language

Apart from certain differences, the procedure in C++ is similar to that of the Python case presented in the previous section; here, the procedure of assembling the message and retrieving the encoded result is shown. It is assumed in the subsequent code parts that the necessary information for creating a flatbuffers object is included in the variables *obj* and *timestamp*.

The main difference is that the input elements do not directly use the builder as a workspace; it is required only when the data to be encoded is already constructed. To improve the readability of the code, the following lines can be added:

```
using namespace Cff::Api::UdpInput;
```

Then, the object can be assembled as follows.

Assemble an object as:

```
auto dimension = CreateDimensions(builder, 4.5, 2.5, 1.5);  /   Length, width, \
height

auto position = CreateAbsolutePosition(builder,
                                       47.475555,           /   Latitude
                                       19.057777,           /   Longitude
                                       1.0,                 /   Position
                                       1.0,                 /   ...
                                       0.0,                 /   confidence
                                       120.0,               /   Altitude
                                       1.0);                /   Altitude \
confidence

auto motion = CreateMotion(builder,
                           Position_AbsolutePosition,       /   or \
Position_RelativePosition
                           position.o,
                           ValueWithConf(12.7, 1.0),        /   Speed
                           ValueWithConf(-0.3, 0.0),        /   Acceleration
                           ValueWithConf(180.0, 0.0),       /   Heading
                           ValueWithConf(4.1, 0.0));        /   Yawrate

auto object = CreateObject(builder,
                           123456,                          /   Id
                           ObjectType_Vehicle,
                           VehicleType_Truck,
                           dimension,
                           motion);
```

The creation of the root object is intentionally not added to the previous code block. For the correct operation of the flatbuffers builder, it needs to be allocated dynamically as follows:

```
auto input = CreateInput(builder, Version_CurrentVersion, timestamp, sourceId, \
DataElement_Object, object);
```

From this point, the builder will manage the life cycle of the root object. After the root object is fully created, it can be passed to a builder which can be used to create the encoded input as:

```
flatbuffers::FlatBufferBuilder builder;

FinishInputBuffer(builder, input->Finish());
```

Finally, the encoded byte array can be retrieved as follows:

```
auto encodedData = std::vector<uint8_t> {builder.GetBufferPointer(), \
builder.GetBufferPointer() + builder.GetSize()};
```

This array can be sent to the UDP adapter.

6.2.6. Configuring the UDP adapter and message generation on the RSU

Sensor specific configuration parameters are collected in the "sensorAdapters" section in the file `cff.json` and in the "sensorSharing" section in the file `saf.json`. The CFF configuration (`cff.json`) contains all sensor specific settings, while the message types that encapsulate detected object information can be specified in the safety application configuration (`saf.json`).

6.2.6.1. Configuring the UDP adapter

To configure the UDP adapter on the RSU, proceed as follows:

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. Locate the `cff.json` CFF configuration file on the device; on Commsignia devices, the file can be found in the `/rwdata/v2x_configs/` directory. If the file is not available or on other devices use the following command to locate the file:

   ```
   find / | grep cff.json
   ```

3. Copy and edit the following "sensorAdapters" section example into the `cff.json` file as follows:

```
{
...
"sensorAdapters": {
  "udpAdapter": {
    "enable": true,
    "port": 12321
  },
  "externalSensors": [
    {
      "id": 111,
      "type": "Lidar",
      "referenceType": "Fixed",
      "position": {
        "latitude": 47.47555,
        "longitude": 19.057777,
        "altitude": 120
      },
      "direction": 90
    },
    {
      "id": 112,
      "type": "Ultrasonic",
      "referenceType": "CurrentPosition",
      "positionOffset": {
        "x": 3,
        "y": 6
      },
      "direction": 135,
      "fieldOfView": {
        "vertices": [
          {
            "x": 1.1,
            "y": 2.2
          },
          {
            "x": 4.4,
            "y": 5.5
          }
        ]
      },
      "detectionAreas": [
        {
          "vertices": [
            {
              "x": 1.1,
              "y": 2.2
            },
            {
              "x": 3.3,
              "y": 4.4
            }
          ]
        }
      ]
    }
  ]
}
...
}
```

Here, parameters and parameter groups are defined as follows:

- **udpAdapter**: the settings of the UDP adapater are collected in this group. The adapter can be enabled/disabled and the UDP port it listens to can be specified.
- **externalSensors**:
  - **list** of sensors that are connected to the RSU, for each sensor you can specify the following parameters:
    - **id**: The ID of the sensor. The corresponding measurements fed via the UDP adapter need to be the same as those set here.
    - **type**: Type of the sensor [camera, lidar. radar, ultrasonic, other].
    - **referenceType**: If "**Fixed**" is selected, the position and direction of the sensor need to be specified. Otherwise (**CurrentPosition**), the ego position and heading is used as a reference and the direction is interpreted in the ego frame.
    - **position**: Position of the sensor. Required only when the reference is set to "**Fixed**."
    - **positionOffset**: Sensor position offset relative to the reference. Required only when the reference is **not** set to "**Fixed**."
    - **direction**: The direction in [deg] to which the sensor points. When the reference type is "**Fixed**," it is interpreted in the WGS84 frame. Otherwise, in the frame of reference of the ego object.
    - **fieldOfView**: The field-of-view (FOV) of the sensor can be specified in this parameter. A valid FOV is a polygon with at least three vertices specified by their x/y relative coordinates.
    - **detectionAreas**: This optional parameter is a vector containing the areas in which valid detections can be expected from the smart sensor. These areas are polygons similar to the FOV and need to fulfill the same requirements. The platform is able to restrict objects to be broadcast in V2X messages only to the specified regions.

6.2.6.2. Configuring message generation

To configure the message generation on the RSU, proceed as follows:

1. Log into the device using SSH. For more information, refer to section “Connecting to the RSU over wireless or wired connection” [2].
2. Locate the `saf.json` safety applications configuration file on the device; on Commsignia devices the file can be found in the `/rwdata/v2x_configs/` directory. If the file is not available or on other devices use the following command to locate the file:

   ```
   find / | grep saf.json
   ```

3. Copy and edit the following "sensorSharing" section example into the `saf.json` file as follows:

```
{
...
"sensorSharing": {
  "enable": true,
  "mode": "Cpm"
}
...
}
```

Here, parameters and parameter groups are defined as follows:

- **enable**: The message generation can be enabled/disabled using "true" of "false," respectively.
- **mode**: The message types that encapsulate detected object information can be specified here "Cpm" (CPM-Eu), "Psm" (PSM-US), or "Sdsm" (SDSM-US).

6.2.7. Quick guide for configuration

- Applications (saf.json)
  - **enable** sensorSharing [**true**] under native section
  - select message generation **mode** [**Cpm/Psm/Sdsm**]
- Fusion-filtering (cff.json)
  - **enable** udpAdapter [**true**] under sensorAdapters section
  - select **port** number [**12321**]
  - set sensor (camera) **position** (**lat**, **lon**) [**47.4755**, **19.0584**]
    - in case of CPM: injected objects and the sensor must be closer than **1000** m (CpmBuilder::isInRange())
- Core stack (its.json)
  - **enable** transmission module for chosen messagetype [**true**]
    - Cpm → CPM transmission configuration
    - Psm → WSMP transmission configuration
    - Sdsm → SDSM transmission configuration
- **set** navigation properties
  - in case of CPM: sensor position and device position must be closer than **1000** m
  - for example, use manual navigation **latitude** and **longitude** [**47.4758**, **19.0582**]

6.2.8. Restrictions

Please note the following restrictions for the injected objects and message types:

- Incoming objects cannot be EGO.
- For `Psm` object, the type must be pedestrian, cyclist, or animal.
- In case of `Cpm`:
  - The injected objects and the sensor must be closer than 1000 m (CpmBuilder::isInRange()).

    - The sensor position and device position must be closer than 1000 m.

- If detection areas are set, then detections must be inside the poligons (areaRestrictor.isRelevant()).

7. Advanced configuration of the software stack

7.1. Factory reset

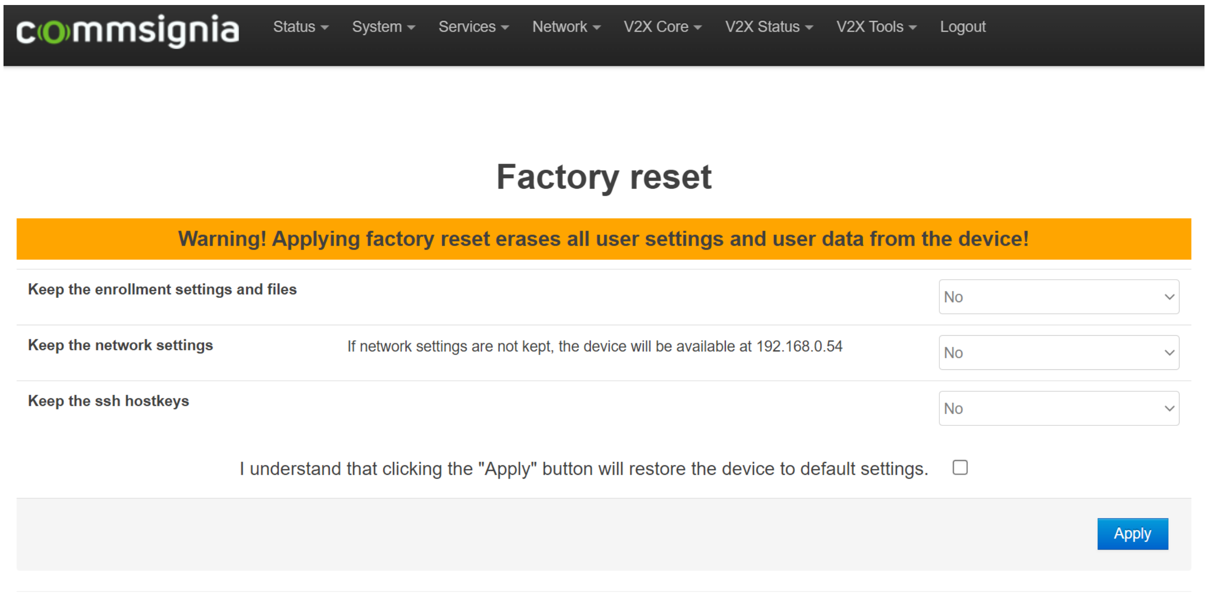
If, for any reason, the device needs to be restored to its factory default, a reset option is available both on the GUI or the CLI of the device.

7.1.1. Performing factory reset on the GUI

To perform a factory reset on the GUI, proceed as follows:

1. Log into the GUI. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. Open the **System** → **Factory Reset** menu as shown in Figure 37.



Factory reset

Warning! Applying factory reset erases all user settings and user data from the device!

Keep the enrollment settings and files		No
Keep the network settings	If network settings are not kept, the device will be available at 192.168.0.54	No
Keep the ssh hostkeys		No

I understand that clicking the "Apply" button will restore the device to default settings. ☐

Apply

Figure 37. Factory reset page

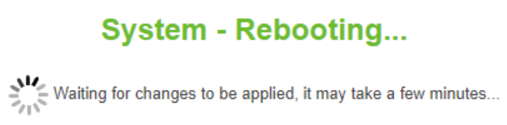
3. Select the entities in the drop-down menu that need to be kept on the device after the reset: all other data will be erased. The available options are as follows:

   - Keeping all enrollment data
   - Keeping all previously configured network settings
   - Keeping all previously defined SSH hostkeys

4. Select the checkbox after the confirmation note.

⚠ **Warning! There is no further confirmation of the procedure: clicking on the Apply button starts the factory reset, which ERASES ALL USER DATA!**

5. Click on the Apply button to perform the reset; Figure 38 shows the screen indicating the reboot process.



System - Rebooting...

Waiting for changes to be applied, it may take a few minutes...

Figure 38. Rebooting screen

Following the operation, the login screen of the device is returned. If the network setting have been reset, the device can be accessed at `192.168.0.54`. To log into the device use the original password given in section "Connecting to the RSU over wireless or wired connection" [2].

7.1.2. Performing factory reset on the CLI

To perform a factory reset on the GUI, proceed as follows:

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. Use the command

```
platform-factory-reset
```

with the following options:

`-d`	Perform the factory reset
`-e`	Keep the enrollment settings and files
`-n`	Keep the network settings
`-s`	Keep the SSH hostkeys
`-h`	Print out a help screen

Please note that using the command without switches returns the help screen. Pressing [ENTER] in this screen list all files to be deleted in a factory reset.

For example, to perform a full factory reset that deletes all previously defined user data, use the command

```
platform-factory-reset -d
```

To perform a factory reset that deletes all previously defined user data except the enrollment settings and files, use the command

```
platform-factory-reset -de
```

3. After using the command with the appropriate switch the following screen is shown:

```
FACTORY RESET STARTED

Your last chance to abort the process with Ctrl+C

To perform the FACTORY RESET, please enter the 'YES, I know, what I am \
doing' string and press [ENTER]
```

To abort the process, press Ctrl+C.

4. To proceed with the factory reset, enter the confirmation string `YES, I know, what I am doing` without quotation marks.

Warning! There is no further confirmation of the procedure: entering the confirmation string and pressing [ENTER] starts the factory reset which ERASES ALL USER DATA!

5. Press [ENTER] to start the procedure; the factory reset starts as follows:

```
PERFORMING FACTORY RESET!!!!
/etc/init.d/unplugged-rt-control stop
Lock Unplugged-RT Restart
Restarting Unplugged-RT
Stop all processes
<< List of all deleted files
...
>>
Generate Factory ITS config.
FACTORY RESET FINISHED

REBOOT

You can access the device on <<IP_ADDRESS>> address.
```

Following the operation, the device can be reached at its original IP address, or, if the network setting have been reset, at `192.168.0.54`. To log into the device use the original password given in section "Connecting to the RSU over wireless or wired connection" [2].

7.2. Upgrading the firmware

It is generally recommended to use the latest firmware version on the V2X device. The latest firmware releases are regularly announced by Commsignia.

7.2.1. Prerequisites

To update the firmware of the device, the following items are required:

- An operational RSU.
- The latest firmware obtained directly from Commsignia.
- Computer with internet connection.
- On Windows computers, SCP and SSH clients.

7.2.1.1. Checking the firmware variant

According to the filesystem and booting of the RSU, there are three firmware variants. These variants can be identified by the firmware filenames, as shown in Table 5.

Table 5. Firmware variants

Filesystem	Secureboot	Firmware filename
Read-only	Enabled	rs4-<...>-**ro-secureboot**-y<...>.tar.sig
Read-write	Enable	rs4-<...>-**rw-secureboot**-y<...>.tar.sig
Read-write	Disabled	rs4-<...>-**rw**-y<...>.tar.sig

To check the filesystem and booting of the RSU, proceed as follows:

1. Log into the RSU using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. To print out the device information, use the following command:

   ```
   commsignia-device-info
   ```

   The command lists all device information; the filesystem and boot type can be found in the first section. This can be

   ```
   Secure boot: secureboot enabled
   Root filesystem: read-only
   ```

   or

   ```
   Secure boot: secureboot enabled
   Root filesystem: read-write
   ```

   or

   ```
   Secure boot: secureboot disabled
   Root filesystem: read-write
   ```

   In any other case or if these information cannot be determined, please contact Commsignia Support before commencing the firmware upgrade.

7.2.2. Upgrade process

If all prerequisites have been satisfied, upgrade the firmware as follows:

1. Obtain a firmware file from Commsignia Support.

2. Copy the firmware file to the `/tmp` directory of the RSU as follows:

   a. On a Linux computer, open a terminal and use the following command:

      ```
      scp rs4-<...>-{variant}-y<...>.tar.sig root@<IP_address_of_the_RSU>:/tmp/
      ```

   b. On a Windows computer, use an SCP client, such as WinSCP to copy the file.

   If the `scp` command returns with the error message `ash: /usr/libexec/sftp-server: not found` and fails, the -O switch needs to be used after the command.

3. Log into the RSU using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].
   Run the `signedUpgrade.sh` command as follows:

   ```
   signedUpgrade.sh /tmp/rs4-<...>-{variant}-y<...>.tar.sig
   ```

The script provides the following information about its progress:

```
Starting signedUpgrade with /tmp/rs4-generic-rw-y20.34.5-
b190705.tar.sig firmware
Verified OK
/dev/updateaux
bin/imx6-glibc/openwrt-v2.2.3-imx6-development-initramfs-signed
CP437: Success
fsck.fat 3.0.28 (2015-05-16)
/dev/bootfs: 6 files, 34498/516216 clusters
VALIDATING RESULT
/dev/bootdir/zImage-signed-b: OK
/dev/bootdir/dtb-signed-b: OK
/dev/bootdir/initramfs-signed-b: OK
bin/imx6-glibc/uboot-imx6-apalis_imx6_it-development//u-boot-SRK_fuse.bin: OK
bin/imx6-glibc/uboot-imx6-apalis_imx6_it-development//u-boot-
SRK_table.bin: OK
bin/imx6-glibc/uboot-imx6-apalis_imx6_it-development//u-boot-signed-
mmc.imx: OK
bin/imx6-glibc/uboot-imx6-apalis_imx6_it-development//u-boot-signed-
usb.imx: OK
bin/imx6-glibc/uboot-imx6-apalis_imx6_it-development//u-boot.imx: OK
Current U-Boot version: 1
New U-Boot version: 1
U-Boot is up to date.
```

After running, the program confirms the successful firmware upgrade as follows:

```
ALL OK
Firmware upgrade from "ob4-generic-rw-y20.23.3-b168981" to "rs4-generic-rw-
y20.34.5-b190705": OK
```

4. Reboot the device with the `reboot` command as:

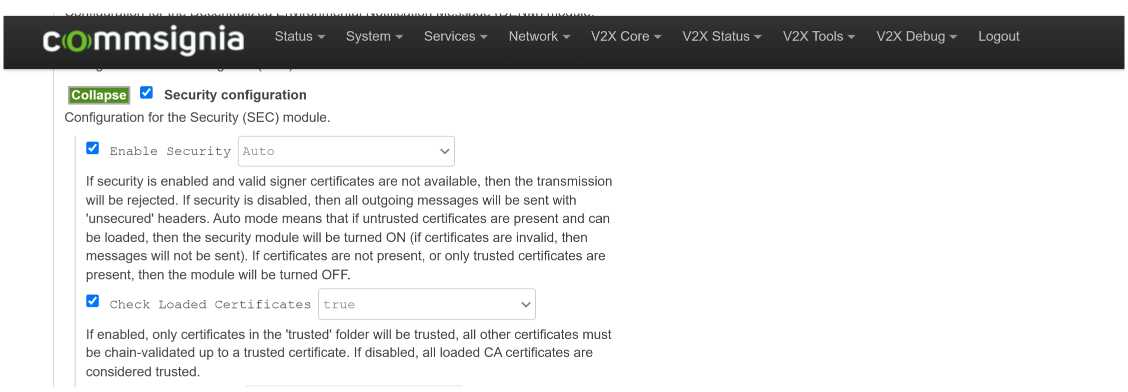
```
reboot
```

7.3. Enabling security for V2X messages using the GUI

Automatic signing of all outgoing facility messages (such as BSMs and CAMs) using a trusted certificate pack can be enabled using the GUI. Please note that a verified certification pack, including a root certificate authority (CA) certificate, is required. Commsignia provides a certification pack for testing purposes; however, in a live enrollment scenario a certificate pack is required that is acquired from a verified source. Subscribing to a certificate provider service is not in the scope of this guide. Test certificate packs provided by Commsignia need to be copied to `/rwdata/etc/security_us` or `/rwdata/etc/security_eu`.

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. Open the **V2X Core** → **Core stack** menu and expand and check the box next to the **Security configuration** item, as shown in Figure 39.



Security configuration

Configuration for the Security (SEC) module.

Enable Security: Auto

If security is enabled and valid signer certificates are not available, then the transmission will be rejected. If security is disabled, then all outgoing messages will be sent with 'unsecured' headers. Auto mode means that if untrusted certificates are present and can be loaded, then the security module will be turned ON (if certificates are invalid, then messages will not be sent). If certificates are not present, or only trusted certificates are present, then the module will be turned OFF.

Check Loaded Certificates: true

If enabled, only certificates in the 'trusted' folder will be trusted, all other certificates must be chain-validated up to a trusted certificate. If disabled, all loaded CA certificates are considered trusted.

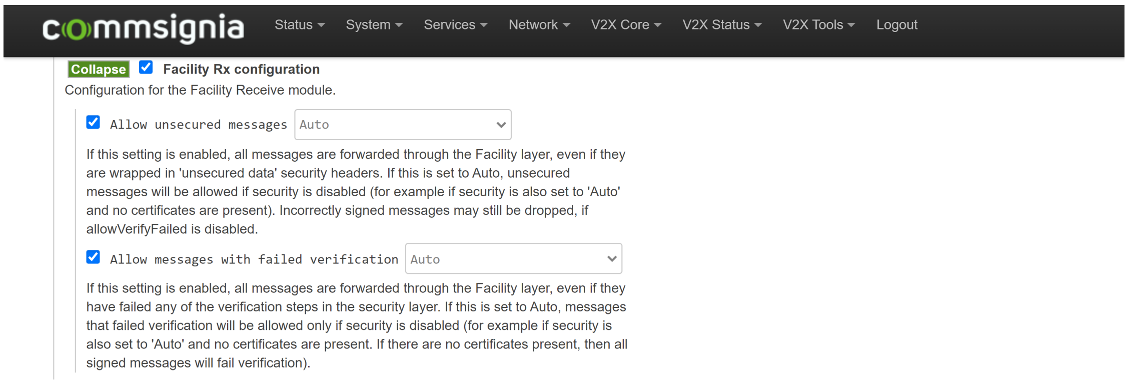
Figure 39. Security configuration

3. Check the box next to `Enable security` and set its values as follows:

   - if the value is `Auto` or `Yes` and valid certificates are present on the device, then the messages are sent signed and received messages are verified. Messages with invalid signature or without security header are dropped according to the settings of the Facility receive module.
   - if the value is `Auto` and no certificates are present on the device, then the messages are sent with unsecured headers and received messages are processed without security verification.
   - if the value is `Yes` and no certificates are present on the device, then the messages are not sent and received messages are dropped due to verification failure.
   - if the value is `Auto` or `Yes` and invalid certificates are present on the device, then the messages are not sent.
   - if the value is `No` then the messages are sent with unsecured header in any case and received messages are processed without security verification.

   If the box next to `Check Loaded Certificates` is checked and its value is set to `true`, then the stack validates the certificates against the trusted root certificates located either at `/etc/security_us` or `/etc/security_eu`. If this value is set to `true` and a test certificate pack is used, then the root certificate from the test certificate pack needs to be copied to `/etc/security_us` or `/etc/security_eu`. If this value if set to `false` than all CA certificates are considered trusted.

4. Optional security settings for transmitting messages are available by expanding the **Facility Rx configuration** item and selecting the checkbox next to it as shown in Figure 40.



Facility Rx configuration

Configuration for the Facility Receive module.

Allow unsecured messages: Auto

If this setting is enabled, all messages are forwarded through the Facility layer, even if they are wrapped in 'unsecured data' security headers. If this is set to Auto, unsecured messages will be allowed if security is disabled (for example if security is also set to 'Auto' and no certificates are present). Incorrectly signed messages may still be dropped, if allowVerifyFailed is disabled.

Allow messages with failed verification: Auto

If this setting is enabled, all messages are forwarded through the Facility layer, even if they have failed any of the verification steps in the security layer. If this is set to Auto, messages that failed verification will be allowed only if security is disabled (for example if security is also set to 'Auto' and no certificates are present. If there are no certificates present, then all signed messages will fail verification).

Figure 40. Facility receive module configuration

Select the checkbox next to `Allow unsecured messages` and set its value as follows:

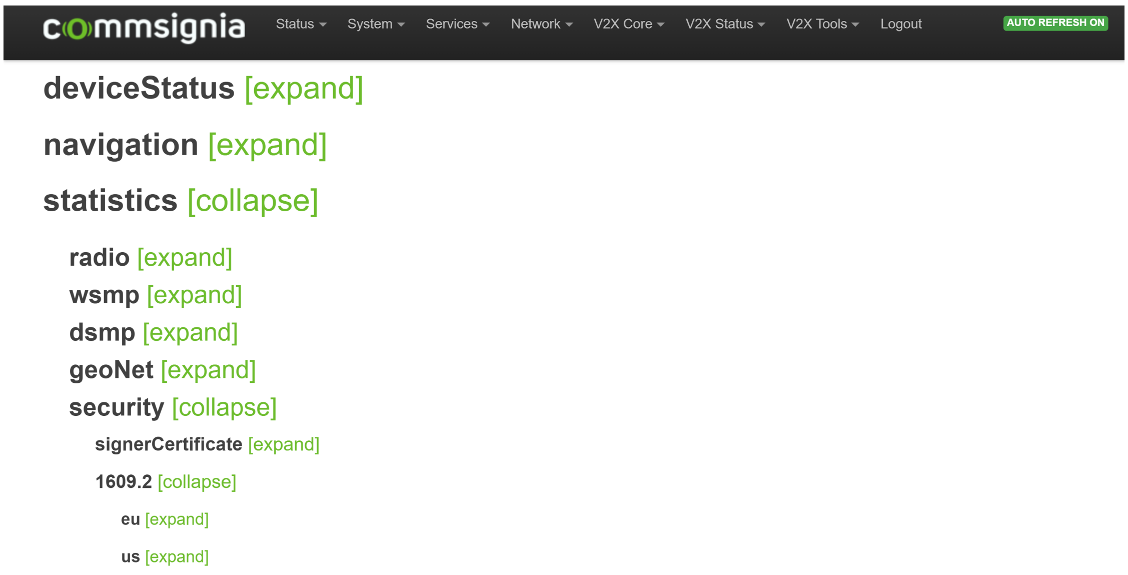
- If the value is set to `Auto`, then unsecured messages are allowed through the Facility layer only if security is disabled (for example if security is also set to `Auto` and no certificates are present).
- If the value is set to `Yes`, then all unsecured messages are allowed through the Facility layer.
- If the value is set to `No`, then no unsecured messages are allowed through the Facility layer.

Select the checkbox next to `Allow messages with failed verification` and set its value as follows:

- If the value is set to `Auto`, then messages that failed verification in the Security layer are allowed through the Facility layer only if security is disabled (for example if security is also set to `Auto` and no certificates are present).
- If the value is set to `Yes`, then all messages that failed verification in the Security layer are allowed through the Facility layer.
- If the value is set to `No`, then no messages that failed verification in the Security layer are allowed through the Facility layer.

5. Click on the **Save & Apply** button for the changes to take effect on the device.

6. To verify that the outgoing V2X messages are automatically signed before they are sent, open the **V2X Core** → **Core status** menu item, as shown in Figure 41, expand **statistics**, expand **security**, and expand **1609.2**. To access the counters, expand the appropriate region (**eu/us**) under this item. The transmission of secured messages can be verified by the increase of the value of the txSignedPacket counter.



deviceStatus [expand]

navigation [expand]

statistics [collapse]

radio [expand]
wsmp [expand]
dsmp [expand]
geoNet [expand]
security [collapse]
signerCertificate [expand]
1609.2 [collapse]
eu [expand]
us [expand]

Figure 41. Statistics for secured messages

7.4. Relicensing the device

In certain cases, such as problems due to a new firmware verson or to enable new features on an older device, the RSU might require to be relicensed. For these cases Commsignia provides new license packs.

7.4.1. Prerequisites

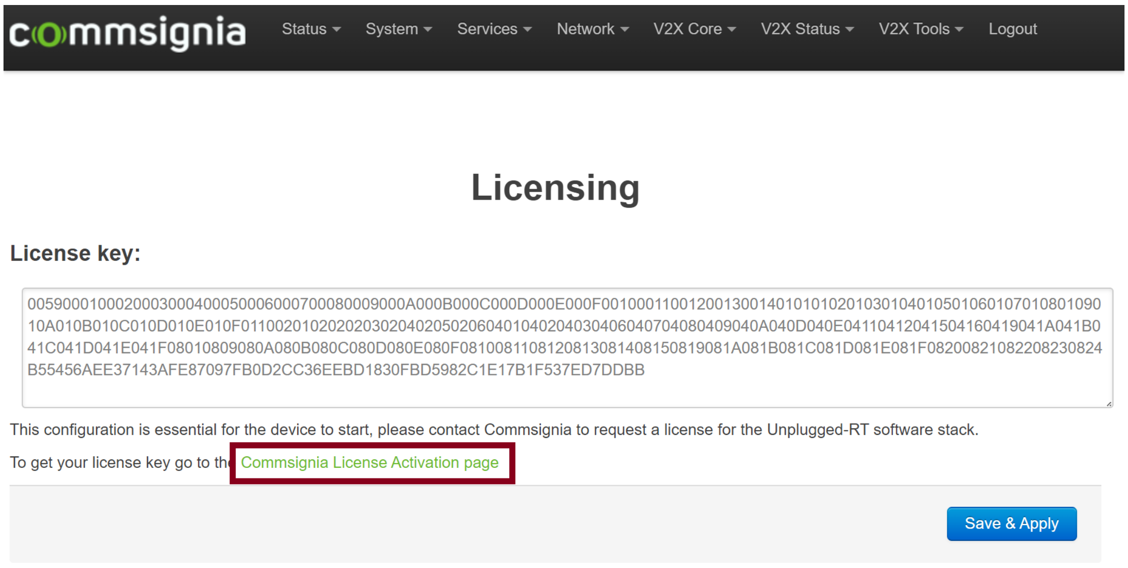
To relicense the device, the following items are required:

- An operational RSU with GUI.
- A license pack (*.pack extension) obtained directly from Commsignia.
- Computer with web browser and internet connection.

7.4.2. Relicensing process

If all prerequisites have been satisfied, relicense the device as follows:

1. Obtain a license pack file from Commsignia Support.
2. Log into the device using SSH. For more information, refer to section “Connecting to the RSU over wireless or wired connection” [2].
3. Open the **V2X Core** → **License** menu item and click on the link "Commsignia License Activation page" as shown in Figure 42. This opens the License Activation page.



Licensing

License key:

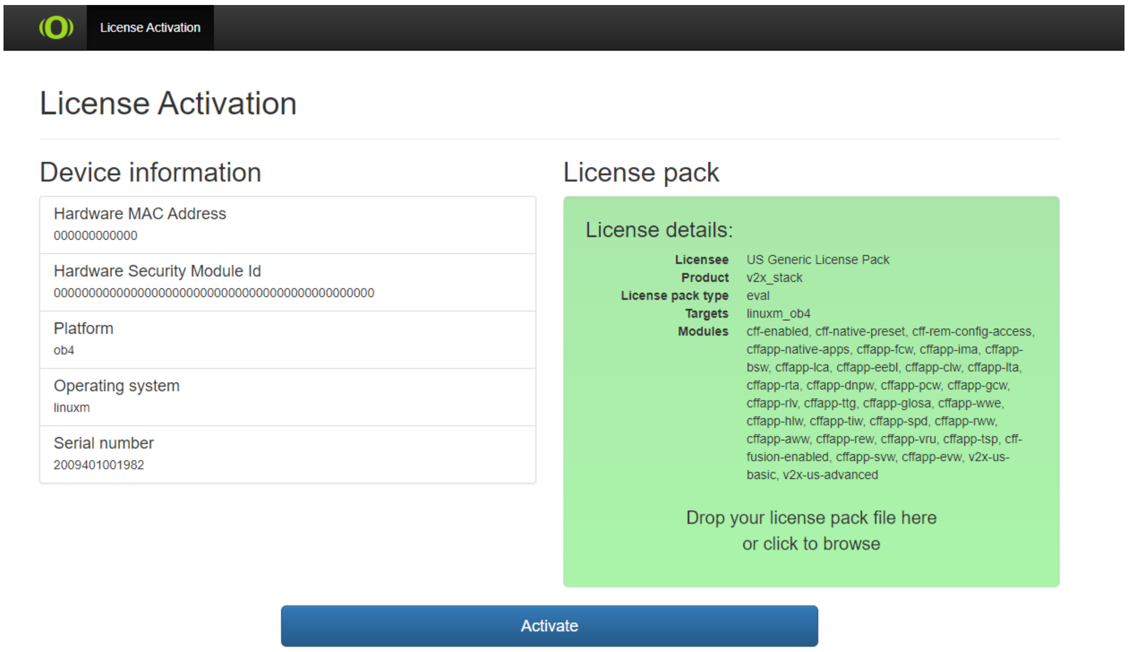
```
0059000100020003000400050006000700080009000A000B000C000D000E000F00100011001200130014010101020103010401050106010701080109O
10A010B010C010D010E010F01100201020202030204020502060401040204030406040704080409040A040D040E0411041204150416041904 1A041B0
41C041D041E041F08010809080A080B080C080D080E080F0810081108120813081408150819081A081B081C081D081E081F0820082108220823082 4
B55456AEE37143AFE87097FB0D2CC36EEBD1830FBD5982C1E17B1F537ED7DDBB
```

This configuration is essential for the device to start, please contact Commsignia to request a license for the Unplugged-RT software stack.

To get your license key go to the Commsignia License Activation page

Figure 42. Licensing page on the GUI

4. On the License Activation page, drag and drop the license file (*.pack) onto the gray area or click on the area and select the file in the browser as shown in Figure 43.



License Activation

Device information

Hardware MAC Address
000000000000

Hardware Security Module Id
0000000000000000000000000000000000000000000000000

Platform
ob4

Operating system
linuxm

Serial number
2009401001982

License pack

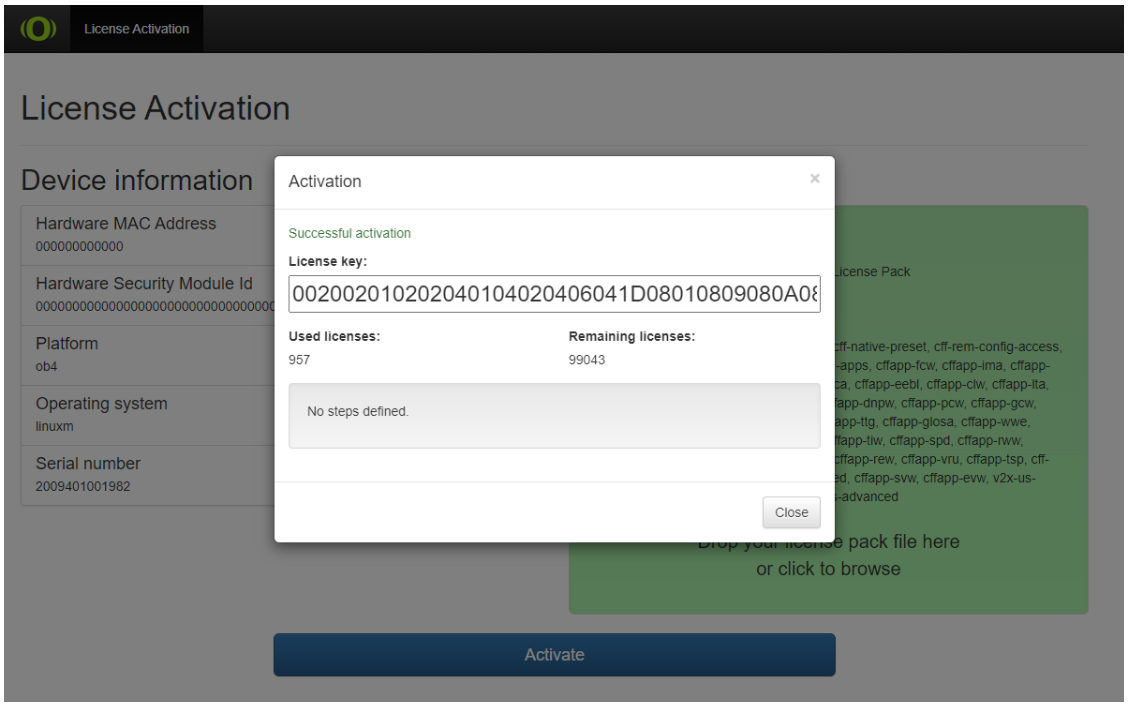
License details:

Licensee	US Generic License Pack
Product	v2x_stack
License pack type	eval
Targets	linuxm_ob4
Modules	cff-enabled, cff-native-preset, cff-rem-config-access, cffapp-native-apps, cffapp-fcw, cffapp-ima, cffapp-bsw, cffapp-lca, cffapp-eebl, cffapp-clw, cffapp-lta, cffapp-rta, cffapp-dnpw, cffapp-pcw, cffapp-gcw, cffapp-rlv, cffapp-ttg, cffapp-glosa, cffapp-wwe, cffapp-hlw, cffapp-tiw, cffapp-spd, cffapp-rww, cffapp-aww, cffapp-rew, cffapp-vru, cffapp-tsp, cff-fusion-enabled, cffapp-svw, cffapp-evw, v2x-us-basic, v2x-us-advanced

Figure 43. License Activation page

Click on the Activate button.

5. After the validation of the license pack, a new license string will be displayed, as shown in Figure 44.



Activation

Successful activation

License key:
00200201020204010402040604 1D08010809080A0

Used licenses: 957

Remaining licenses: 99043

No steps defined.

Figure 44. Generation of a new license key

Copy this license key, open the **V2X Core** → **License** menu and paste this key into the License key field, overwriting the old key.

6. Click on the Save & Apply button for the changes to take effect on the device.

7.5. Enabling IPv6 tunneling on RSUs

For testing purposes, the RSU can be configured such that it can be accessed over the internet using the IPv6 protocol and it can provide IPv6 tunneling to nearby OBUs. This feature is only compliant with the US regional V2X standards and supported for DSRC only.

The RSU needs to be connected to a central router and an IPv6 prefix needs to be pre-assigned to the it. Every RSU connected to a central router needs to have its own prefix and routing set up statically. All RSUs need to have their own /64 IPv6 subnet.

IPv6 tunneling enables the communication of compatible V2X devices with each other using the IPv6 protocol. Using this feature, properly configured OBU devices that can successfully communicate with a compatible RSU using standard V2X messages can also access the Internet.

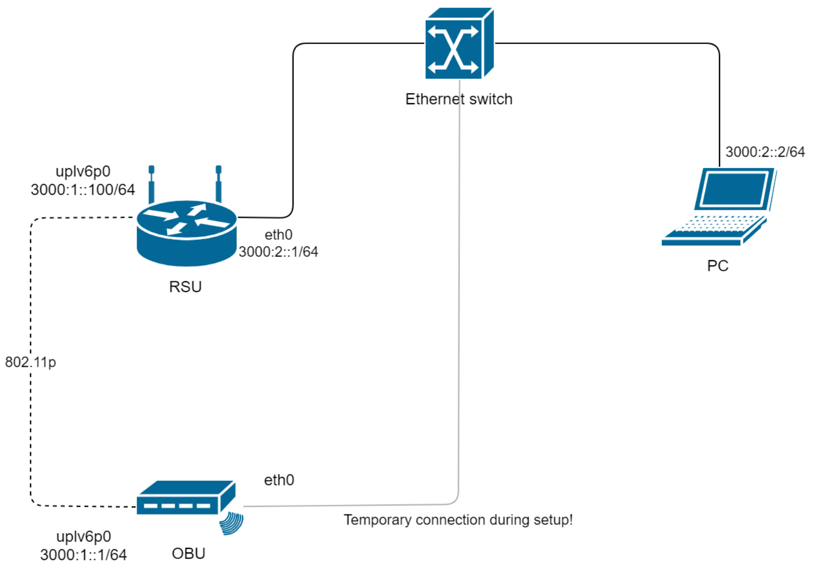
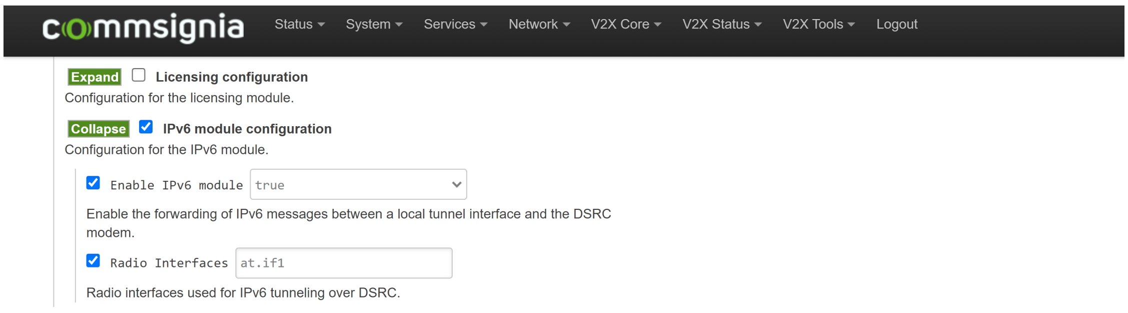


Figure 45. IPv6 tunneling setup

1. Log into the device using SSH. For more information, refer to section "Connecting to the RSU over wireless or wired connection" [2].

2. Under the `V2X Core` → `Core stack` menu item expand and check the box next to **IPv6 module configuration**, as shown in Figure 46.



☐ Licensing configuration
Configuration for the licensing module.

☑ IPv6 module configuration
Configuration for the IPv6 module.

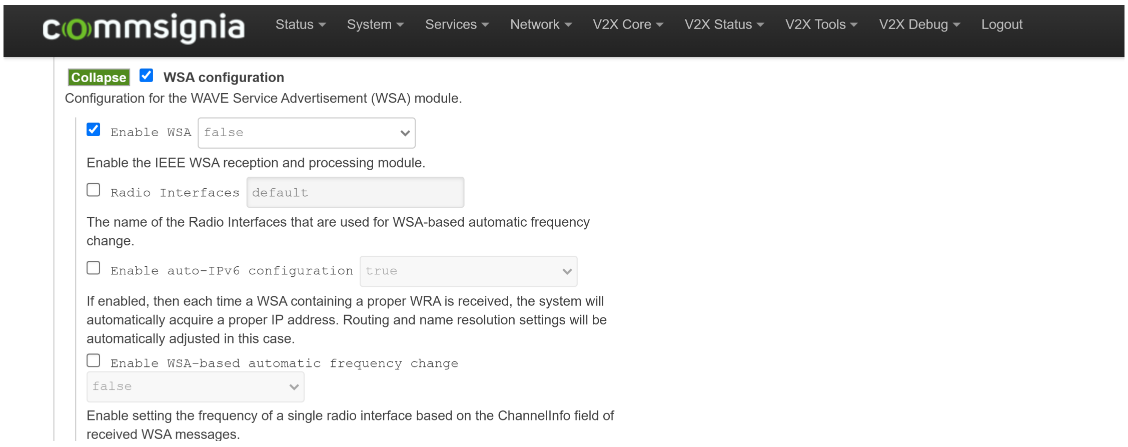
☑ Enable IPv6 module: true
Enable the forwarding of IPv6 messages between a local tunnel interface and the DSRC modem.

☑ Radio Interfaces: at.if1
Radio interfaces used for IPv6 tunneling over DSRC.

Figure 46. IPv6 module configuration

3. Check the box next to `Enable IPv6 module` and set its value to `true`.

4. WAVE Service Advertisement (WSA) messages contain an IPv6 prefix range assigned to the RSU and the OBUs allocate an IPv6 address from that range upon receiving a WSA. Minimum /64 prefix is required for the OBUs. This address is a global IPv6 address that can be routed to and can be accessed from the Internet. WSA messages also contain the DNS address and the default gateway (this is typically the address of the RSU). The WSA Rx module must be turned off on the RSU, to avoid receiving another WSA from a nearby RSU. Under the **V2X Core** → **Core stack** menu item expand and check the box next to **WSA configuration**, as shown in Figure 47.



☑ WSA configuration

Configuration for the WAVE Service Advertisement (WSA) module.

☑ `Enable WSA` `false`

Enable the IEEE WSA reception and processing module.

☐ `Radio Interfaces` `default`

The name of the Radio Interfaces that are used for WSA-based automatic frequency change.

☐ `Enable auto-IPv6 configuration` `true`

If enabled, then each time a WSA containing a proper WRA is received, the system will automatically acquire a proper IP address. Routing and name resolution settings will be automatically adjusted in this case.

☐ `Enable WSA-based automatic frequency change`
`false`

Enable setting the frequency of a single radio interface based on the ChannelInfo field of received WSA messages.

Figure 47. WSA configuration

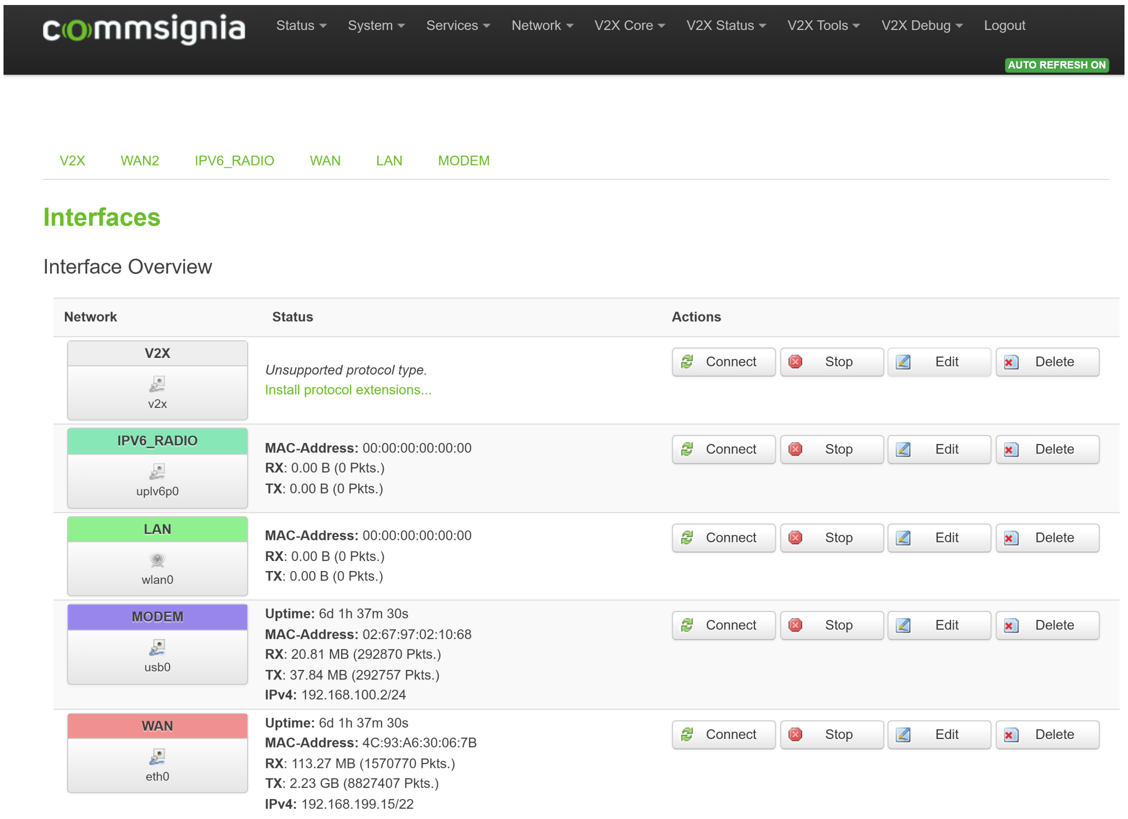
5. Check the box next to `Enable WSA` and set its value to `false`.

6. Click on the Save & Apply button for the changes to take effect on the device.

7. Set an IPv6 address for the tunnel manually:

   a. The `uplv6p0` stack interface needs to have an IPv6 address assigned from the prefix range. Assign **[PREFIX]::1**

   b. You can set the address on the **Network** → **Interfaces** tab.



Interfaces

Interface Overview

Network	Status	Actions
V2X v2x	*Unsupported protocol type.* Install protocol extensions...	Connect Stop Edit Delete
IPV6_RADIO uplv6p0	**MAC-Address:** 00:00:00:00:00:00 **RX:** 0.00 B (0 Pkts.) **TX:** 0.00 B (0 Pkts.)	Connect Stop Edit Delete
LAN wlan0	**MAC-Address:** 00:00:00:00:00:00 **RX:** 0.00 B (0 Pkts.) **TX:** 0.00 B (0 Pkts.)	Connect Stop Edit Delete
MODEM usb0	**Uptime:** 6d 1h 37m 30s **MAC-Address:** 02:67:97:02:10:68 **RX:** 20.81 MB (292870 Pkts.) **TX:** 37.84 MB (292757 Pkts.) **IPv4:** 192.168.100.2/24	Connect Stop Edit Delete
WAN eth0	**Uptime:** 6d 1h 37m 30s **MAC-Address:** 4C:93:A6:30:06:7B **RX:** 113.27 MB (1570770 Pkts.) **TX:** 2.23 GB (8827407 Pkts.) **IPv4:** 192.168.199.15/22	Connect Stop Edit Delete

Figure 48. Setting the static IPv6 address for the device

8. Set up the WRA message content, which contains the IPv6 prefix of the RSU as well as the routing address, subnet, and, optionally a DNS address.

   a. Use the following template for creating your own WRA message:

```
<SrvAdvMsg>
  <version>1</version>
  <body>
    <changeCount>
      <saID>0</saID>
      <contentCount>1</contentCount>
    </changeCount>
    <routingAdvertisement>
      <lifetime>10</lifetime>
      <ipPrefix>2001047022AAAAAA000000000000DDDD</ipPrefix>
      <ipPrefixLength>60</ipPrefixLength>
      <defaultGateway>2001047022aaaaaa72b3d5fffef2CCCC</defaultGateway>
      <primaryDns>200148604860000000000000000AAAA</primaryDns>
      <extensions/>
    </routingAdvertisement>
  </body>
</SrvAdvMsg>
```

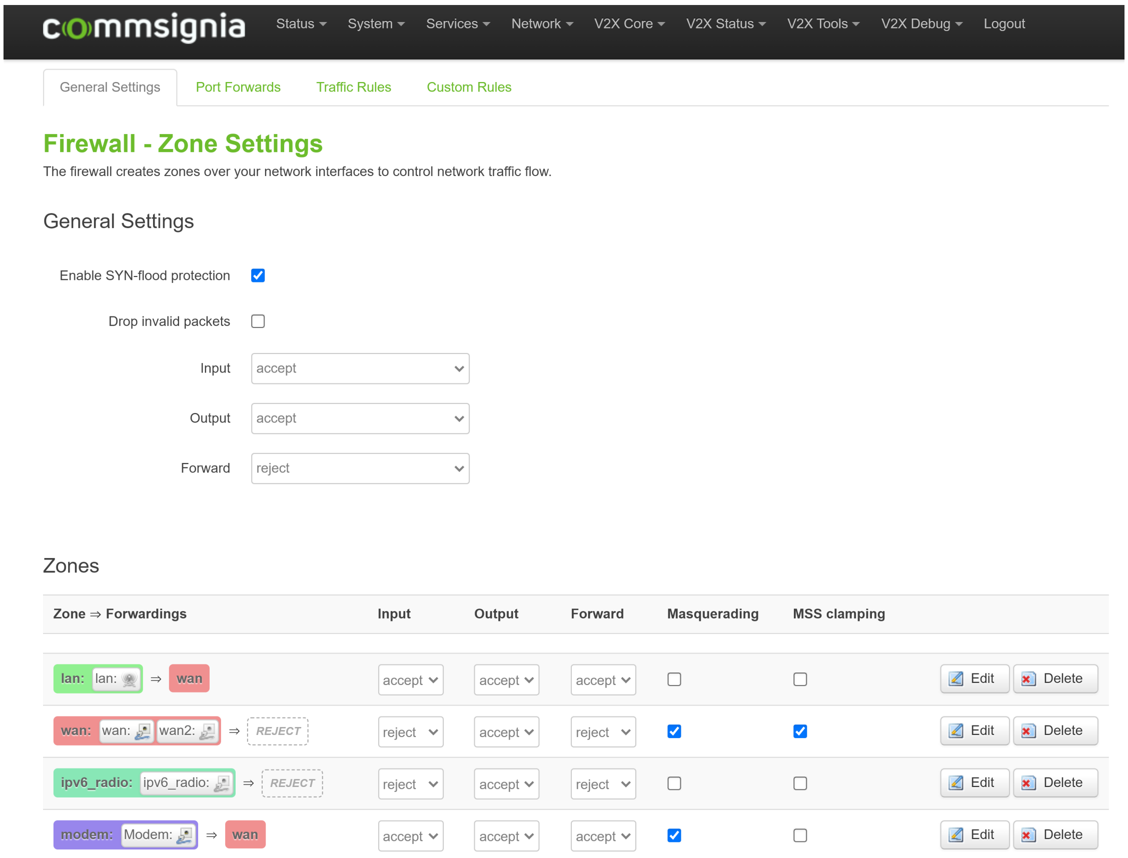
   b. Convert the XML template using the ASN1.X tool on the device. For more information, see the section "Converting data formats" [23] .

   c. Copy the created RSU message file to the `/rwdata/etc/rsu_msg/` folder of the RSU.

9. Turn on IPv6 forwarding on the RSU.

10. Enable forwarded packets on the RSU firewall and the central routers firewall.

a. To edit the IPv6 zone, open the **Network** → **Firewall** menu item, as shown in Figure 49, and under the Zones section, the line of **ipv6_radio:** click on Edit.



Firewall - Zone Settings

The firewall creates zones over your network interfaces to control network traffic flow.

General Settings

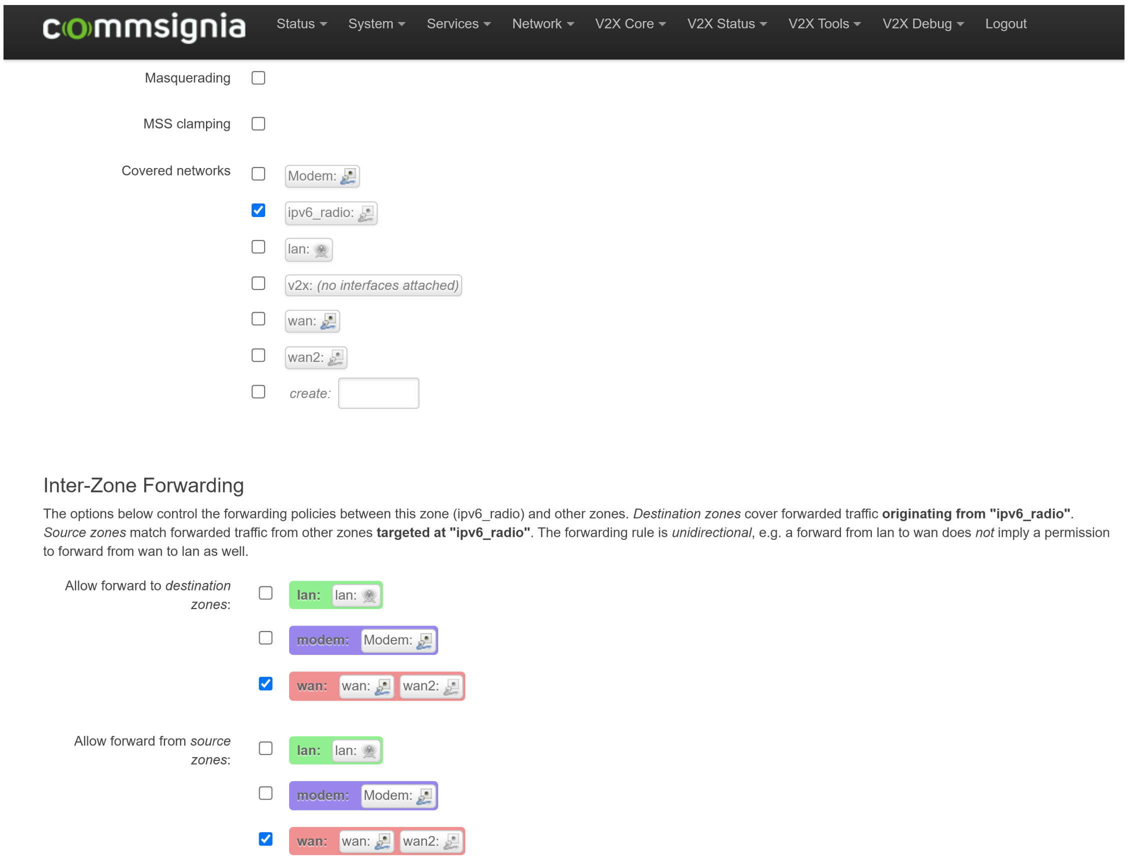
Enable SYN-flood protection	☑
Drop invalid packets	☐
Input	accept
Output	accept
Forward	reject

Zones

Zone ⇒ Forwardings	Input	Output	Forward	Masquerading	MSS clamping	
lan: lan: ⇒ wan	accept	accept	accept	☐	☐	Edit Delete
wan: wan: wan2: ⇒ REJECT	reject	accept	reject	☑	☑	Edit Delete
ipv6_radio: ipv6_radio: ⇒ REJECT	reject	accept	reject	☐	☐	Edit Delete
modem: Modem: ⇒ wan	accept	accept	accept	☑	☐	Edit Delete

Figure 49. Firewall settings

b. Under Inter-Zone Forwarding, check the boxes next to **wan:** at "Allow forward to *destination zones*:" and "Allow forward from *source zones*:," as shown in Figure .50



Masquerading ☐

MSS clamping ☐

Covered networks
☐ Modem:
☑ ipv6_radio:
☐ lan:
☐ v2x: *(no interfaces attached)*
☐ wan:
☐ wan2:
☐ *create:*

Inter-Zone Forwarding

The options below control the forwarding policies between this zone (ipv6_radio) and other zones. *Destination zones* cover forwarded traffic **originating from "ipv6_radio"**. *Source zones* match forwarded traffic from other zones **targeted at "ipv6_radio"**. The forwarding rule is *unidirectional*, e.g. a forward from lan to wan does *not* imply a permission to forward from wan to lan as well.

Allow forward to *destination zones*:
☐ lan: lan:
☐ modem: Modem:
☑ wan: wan: wan2:

Allow forward from *source zones*:
☐ lan: lan:
☐ modem: Modem:
☑ wan: wan: wan2:

Figure 50. Enabling IPv6 forwarding in the Firewall settings.

c. Click on the Save & Apply button for the changes to take effect on the device.

The RSU is configured to communicate over IPv6 either with the central router or with nearby OBUs.

8. Troubleshooting V2X communication

This chapter details the most common problematic conditions of V2X communications using the Commsignia software stack and the possible solutions for them.

8.1. General validation steps

Go through these steps first if you experience any problems with your V2X device:

- Make sure that all antennas are properly connected according to the device's hardware description.
- Make sure the device is powered on.
- Make sure that the device and the software stack is licensed with Commsignia.
- Make sure you are connected to the same network as the device.
- Check if your navigation settings are set to "Manual" or if the device has a proper GNSS fix.

8.2. V2X messages are not secured or not transmitted

Go through the general validation steps detailed in this chapter, then check the following:

- SEC module is enabled
- A valid certificate pack (including a CA root certificate) is used on the device
- The same CA root certificate is used on both the transmitting and receiving test devices

8.3. The HMI is not displaying SPaT on the map

Go through the general validation steps detailed in this chapter, then check the following:

- Make sure that the HMI is connected to the Internet or it has offline maps downloaded for the location you are testing for.
- Make sure that the Intersection ID contains the same location for both the MAP and the SPaT.

8.4. The HMI displays a SPaT that is different from the actual traffic signal

Go through the general validation steps detailed in this chapter, then check the following:

- Make sure that the MAP message contains the correct lane-to-lane connection signal group ID
- Make sure that a compatible data format is used, for example Batelle or UDP
- Make sure that the signal group ID, phase, and overlap values are all correct

8.5. The HMI is not displaying the local vehicle

Go through the general validation steps detailed in this chapter, then check the following:

- Make sure that the vehicle has valid navigation data; either set in "Manual mode" or it has a proper GNSS fix
- Make sure that the SEC module is enabled in the configuration and there is a valid certificate pack (including a CA root certificate) on the device
- Make sure that you are connected to the correct station in the HMI.

Appendix A. Glossary of terms

ASN.1	Abstract Syntax Notation One
BSM	Basic Safety Message
C2P	Commsignia Capture Protocol
CA	Certificate authority
CAM	Cooperative Awareness Message
CLI	Command line interface
DNS	Domain Name System
DSRC	Dedicated Short-Range Communication
GUI	Graphical user interface
HMI	Human–machine interface
IFM	Immediate Forward Message
IP	Internet Protocol
JDK	Java Development Kit
OBU	Onboard unit
PoE	Power over Ethernet
RSU	Roadside unit
SCP	Secure Copy Protocol
SNMP	Simple Network Management Protocol
SPaT	Signal Phase and Timing
SRM	Store-and-Repeat Message
SSH	Secure Shell Protocol
SSID	Service set identifier
TLC	Traffic Light Controller
UDP	User Datagram Protocol
UPER	Unaligned Packed Encoding Rules
WAVE	Wireless Access for Vehicular Environment
WSA	WAVE Service Advertisement
WSMP	WAVE Short Message Protocol