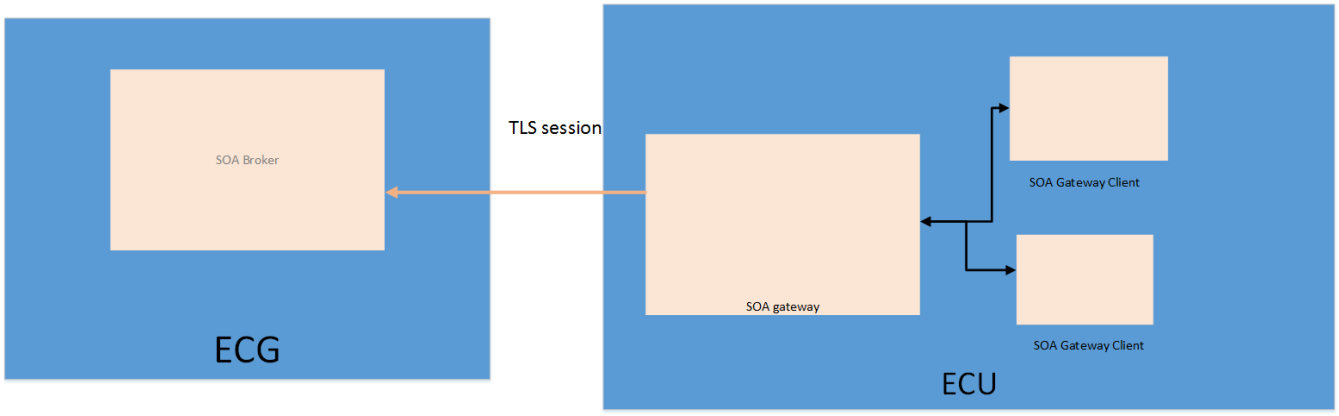
fnv::soa: (g) Gateway (from ECG)

Introduction

The SOA gateway is an extension of the SOA framework to address secure MQTT connection between the ECG and other ECUs.

Any component on an ECU willing to provide SOA services or access SOA services will be connected to a single Gateway per ECU. The Gateway Client developer (the component connecting to the Gateway) does not need to know the details of the Gateway as it is abstracted out by the SOA API. Only few settings are exposed to either connect a SOA component to the SOA broker or the SOA Gateway.

Whatever connection is used, the same SOA API is provided.

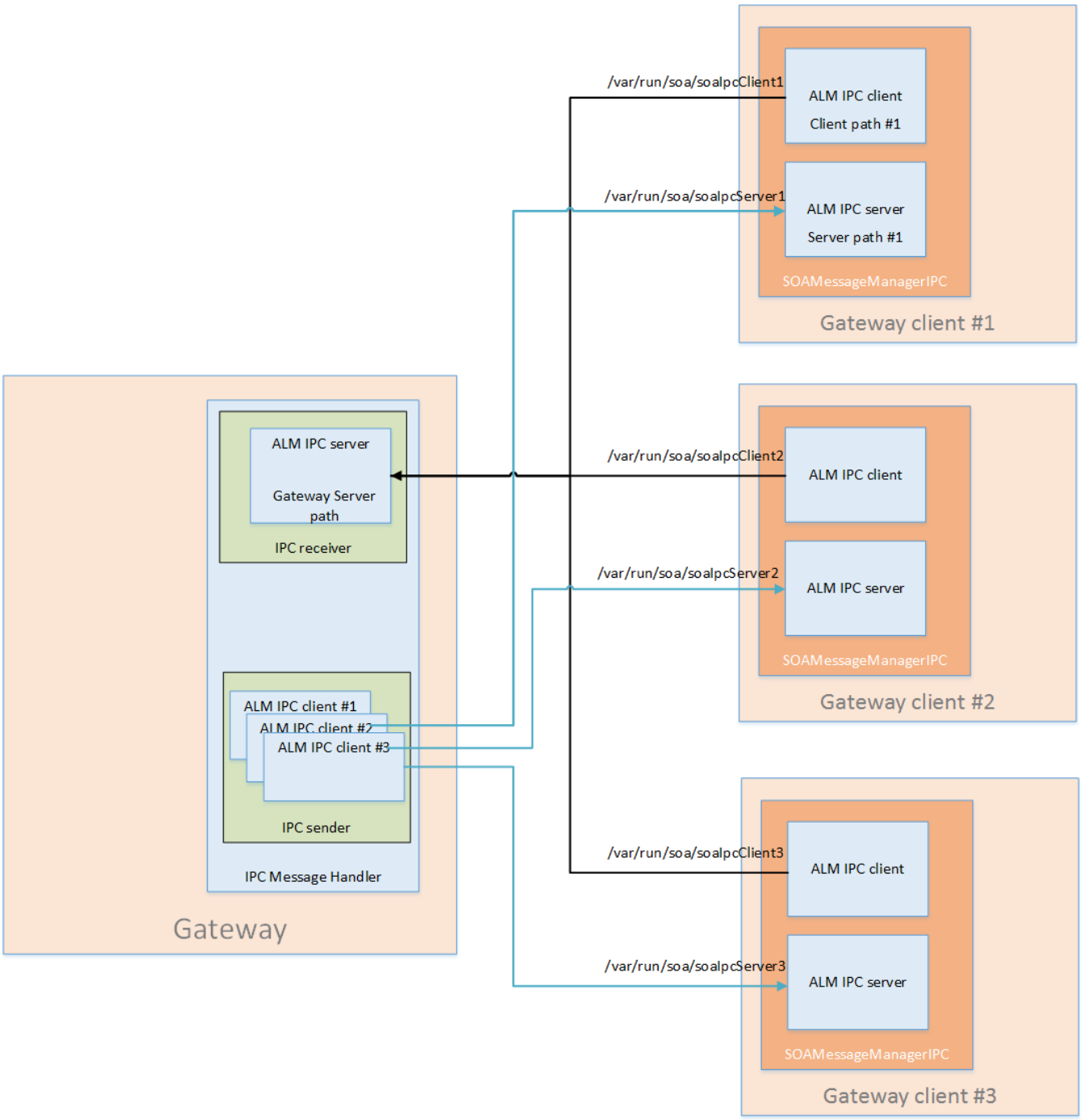


Internal Gateway connection

In order to understand the meaning of the various connection options to the Gateway, the developer should know how the clients are connected to the Gateway internally.

The diagram below shows a configuration with three clients connected to the Gateway. Each client is composed of an IPC interface to allow communication from the client to the Gateway and from the Gateway back to the client. This interface includes two unidirectional links providing a bi-directional communication path.

The underlying method is based off unix domain socket so each socket endpoint is identified by a file which is located in /var/run/soa. This file has to be unique and will by use to map traffic in the Gateway to the appropriate clients.



Connection Options

The difference for a client connecting to the SOA broker or the SOA Gateway is in its connection options. This difference is related first to the list of settings available and second how they are used. For MQTT connection, the options are only used when the component connect to the broker. In the IPC case, the connection options are part of each transactions with the Gateway as CONNECT, but also PUBLISH, SUBSCRIBE, UNSUBSCRIBE and DISCONNECT. Some settings as the retained flag can be changed on a connection basis and not only at connection time.

The sample code below show a typical connection option settings for a SOA broker connection

Broker connection

```
        soa::SoaErrorCode connectionStatus = soa::SoaErrorCode::ERROR_TIMEOUT;
        soa::SoaConsumer::SharedPtr consumer = nullptr;
        //Set connection options
        soa::SoaConnectionOptions::SharedPtr connOpts = soa::SoaConnectionOptions::createConnectionOptions();
        connOpts->setClientId(consumerClientId);
        connOpts->setDebugBrokerUrl(brokerURL);
        connOpts->setConnectionTimeOut(10);
```

For a SOA Gateway connection, the SoaConnectionOptions is replaced by SoaConnectionOptionsIpc with some specific settings:

Gateway connection

```
    fnv::soa::framework::SoaMessageManager::SharedPtr m_messageManager;
    fnv::soa::framework::SoaConnectionOptionsIpc::SharedPtr m_connectionOptions;
    m_connectionOptions = fnv::soa::framework::SoaConnectionOptionsIpc::createConnectionOptionsIpc();

// extra API for SOA release 0.66 and above
    m_connectionOptions->setConnectionListener(SoaConnectionListener connListener);
    m_connectionOptions->setConnLostListener(SoaConnectionLostListener connLostListener);
```

The connection between the clients and the Gateway is done through an IPC interface. The client is required to provide some identifiers so the binding between Gateway and client can happen. This binding relies on file paths. One is for the Gateway to connect to the client and the other one is used by the client to connect to the Gateway.

The following idl file is used to define the connections properties of client to Gateway:

SoaIpcConnectionOptions

```
message SoaIpcConnectionOptions {
    required string serverFilePath = 1;
    required string clientFilePath = 2;
    optional bool cleanSession = 3;
    optional int32keepAlive = 4;
    optional bool lastWillAndTestament = 5;
    optional bool retained = 6;
}
```

Some of the differences in the connection options are highlighted below:

`setClientPath(std::string clientPath)- deprecated in release 0.65`

This setter is specific to the connection between the client and the Gateway. This is used to set the client file path identifying the Unix domain socket. If there are multiple clients, the developer has to make sure each file path is unique. The client name can be used for this purpose and the full path is not needed as by default the file is created in /var/run/soa.

This API will be deprecated as the SOA API will automatically determined the file name

`setServerPath() - deprecated in release 0.65`

Method to set the file path of the client server. This is needed for the Gateway to be able to send data back to the client.

This API will be deprecated as the SOA API will automatically determined the file name

`SetCleanSession()`

This setting is the same as for MQTT/SOA client. If clean session is set to false, all the client subscriptions are going to be saved and restored if the client ungracefully disconnect (e.g. crash).

`setConnectionTimeout()`

This setting as the same meaning as on ECG for the MQTT connection but in that case this is for the IPC link between client and Gateway. If no message is exchanged between client and Gateway within this timeout, the Gateway will ping the client and the client will ping the Gateway. If no response is received after 1.5xConnectionTimeout, the connection is declared as broken meaning that either the Gateway or the client is unresponsive.

`setWill()`

Set the Will and Testament of the MQTT connection between the Gateway and the client. If the client ungracefully disconnect without sending the disconnect command, the Gateway will publish SERVICES/LWT topic including the client ID to the broker so any client subscribing for the LWT will be notified.

The Gateway also set automatically its LWT so if the Gateway unexpectedly disconnects from the SOA broker, the broker will publish the Gateway LWT.

`setRetained()`

Each message published can be set with the retained flag. A retained message is a message stored by the broker so a client subscribing to the same topic as the retained message will receive it even if the subscription happens after the publish has been done.

SetConnectionLostListener()

Set a listener called when the connection between the Gateway and the SOA broker is lost

SetConnectionListener()

Set a listener called when the connection between the Gateway and the SOA broker is established

The Gateway will automatically configure its reconnnection options so the client does not have any settings to configure. The auto reconnection will only be disabled when all the connected clients will disconnect from the Gateway.

Running the SOA Gateway

The gateway should be started as a daemon. Some command line parameters are mandatory. The usage is as follow:

gateway --config /etc/config/config.json <--help>

--config: load the gateway configuration file. For a dev or prod secure device without a debug token, the config file is by default on a non writable partition and as such cannot be edited. The gateway will substitute "/data/" to "/etc/" in the path provided and will load this config file from that path if present. If not present, the file in the /etc/ will be loaded.

--help: show previous parameters

Gateway permission: All the clients connecting to the gateway will need to be part of the soagateway group (SYNC) or soa_gateway group (tcu)

Creation of a Gateway Client

The development of a Gateway Client is not diferent than developing a regular SOA component using the SOA API. The only difference as mentioned above is in the call of

createConnectionOptionsIpc() instead of createConnectionOptions.

Master/Slave paradigm

Starting with SOA release 0.66, the SOA API implements an API to provide the programmer with some flexibility to manage the connection between the Gateway and the ECG. This flexibility is particularly needed for ECUs which can be running while the ECG is down (e.g. TCU) and want to control the connection with the broker. The supported scenarios are described here: SOA Gateway Client Connectivity Use Cases

A master Gateway client (or client) is a SOA component which can act on the Gateway connection to the ECG broker either triggering the connection or interrupting it.

A slave client is not able to influence in anyway this connection with the ECG broker.

API usage for a master client

The first call the client will do is **SoaErrorCode SoaMessageManagerconnectAsync()**. The first thing which done byt this method is to attempt to connect to the Gateway. If the Gateway is not there, this will loop until the connection is successful. Once the connection with the gateway is done, a request is sent over to the Gateway to connect to the broker. As the call is asynchronous, it will return immediately. The connection with the broker is notified through the ConnectionListener callback.

The consumer and/or provider can then be instantiated as the next step and can call **SoaErrorCode initializeAsync(SoaInitializedProviderListener /SoaInitializedConsumerConsumer)**. The purpose is to initiate the connection to the Gateway only and register the callback. As for a master client the **co nnectAsync()** already established the connection with the gateway, the callback should be triggered right away.

SoaErrorCode SoaMessageManagerdisconnectGlobalAsync() this will disconnect the gateway from the broker. All the clients connected will then be disconnected from the broker and will be notified through their callback.

API usage for a slave client

As slave client does not drive the connection of the Gateway with the broker, the first call is the **SoaErrorCode initializeAsync (SoaInitializedProviderListener/SoaInitializedConsumerConsumer)**. They should not call **connectAsync()**.

To disconnect, the client will call **SoaErrorCode disconnectAsync()**. Contrarily to **disconnectGlobalAsync(),** this call will not disconnect the Gateway to the broker if there are still client connected. The condition to close the connection with the broker is to have all the client calling **disconnectAsync()**.

API for no master/slave implementation

In that case, all the Gateway clients will call **SoaErrorCode SoaMessageManagerconnectAsync().** The first client calling this API will trigger the Gateway to connect to the broker. Any subsequent call will just return that the connection is on going or already done.

The 2nd following call is **initializeAsync()**

To disconnect, the client will call **SoaErrorCode disconnectAsync()**.

TLS or non TLS connection

The following table is a summary of the type of connection depending on the device and presence of a debug token.

Device Variant	Gateway	Keys & CA cert
Insecure	TLS or non-TLS	developer
Dev secure + debug token	TLS or non-TLS	developer
Prod secure + debug token	TLS or non-TLS	production
Dev secure	TLS only	developer
Prod secure	TLS only	production

The selection of the connection (TLS or non TLS) on an insecure device or a secure device with a debug token is done through the gateway configuration file. The keyword "enableTLS" can be set to true or false to enable or disable TLS connection to the SOA broker.

On a secure device with no debug token found, the connection will be done over TLS ignoring any setting in the configuration file.

For test purposes, the SOA gateway will support an alternate configuration file in the /data partition which is writable since the system partition is not re-mountable as r/w on secure devices. The configuration file should be copied from /etc/ directory to the /data/ directory. The SOA gateway when started will first check the type of device and if on a secure device with a debug token, it will try to load the configuration file from the /data/ location.

Example:

By default, the gateway is usually started from a startup script which will include the path of the gateway configuration file to be loaded. On TCU the path is: /**etc**/soa/gateway/config/config.json

On a insecure or secure device with a debug token, the soa gateway will substitute "/data/" to "/etc/". So the alternate path will be: /**data**/soa/gateway/config /config.json

This file has to be explicitly copied as it is not included in the build image.

Limitation on SYNC to create the alternate directory structure

On Sync, you may not have the appropriate permission to create the alternate directory structure from a terminal. You will have to follow the steps described on the following wiki pages:

https://www.eesewiki.ford.com/display/sync/A+magic+dev+script+to+modify+behaviour+of+a+read-only+OS+image

https://www.eesewiki.ford.com/display/sync/Using+a+qnx6fs+nested+OS+image+file

In summary, you need to create a startup script which will create the directory architecture in /data. In the example below, a symlink is created to the original configuration file in /etc/ but you can also copy the file to update it.

```
mkdir /fs/storage/rwdata/devtools
cp /etc/soa/gateway/config/config.json /fs/storage/rwdata/devtools/config.json
echo "#!/bin/sh" > /fs/storage/rwdata/devtools/dev-init.sh
echo "ln -sP /fs/storage/rwdata/devtools/config.json /data/soa/gateway/config/config.json" >> /fs/storage/rwdata
/devtools/dev-init.sh
chmod 755 /fs/storage/rwdata/devtools/dev-init.sh
```

Gateway configuration file

A typical soa gateway configuration file (config.json) is shown below:

```
{
"_comment_":"at 1100 ms/retry, 2356363 would retry for 30 days",
"maxRetryCount": 2356360,
"minRetryInterval": 1100,
"maxRetryInterval": 1100,
"baseRetryInterval": 1100,
"ipcRootPath": "/var/run/soa",
"serviceDescriptorPath":"/etc/soa-sm/descriptors/",
"ipcConnectPermissive" : true,
"ipcRootConnectAllowed" : true,
"keepAliveInterval" : 30,
"descriptorSignedFlag" : true,
"enableTLS" : false,
"brokerURL":"10.1.0.1"
}
```

Most of the parameters do not need to be changed. However, for testing purposes, we may have to change:

- **enableTLS:** used to enable or disable TLS connection with the ECG
- **BrokerURL**: if the URL need to be changed and for instance replaced by an IP address, do not add the transport protocol (tcp:// or ssl://) or the port number. The gateway will add it depending on the connection type (TLS or non TLS). If you are running the gateway on a VM with the broker, do not forget to replace the ECG ip address by the local host address (127.0.0.1)

(Note: the variable named _comment_ and its string value in the example config file above is not part of the soagateway configuration, but is present here merely because JSON does not support comments. The 30 day retry limit may or may not be realistic, but serves to explain the maxRetryCount variable.)