Product Development

FNV2-Ford Cloud Interface FCI API (libfciclient)

Version 1.28

Version Date: June 19th, 2019

UNCONTROLLED COPY IF PRINTED

Contents

Description

libfciclient (FCI client) is a shared library to facilitate communication with SDN for apps and services.

The API abstracts the fci daemon, the fci IDL, SOA messaging and provides easy to use methods for common use cases.

The API does not abstract the FTCP IDL/protobuf. End user remains responsible for handling of the FTCP message payload and providing a valid FTCP response if applicable.

Please see the library public header here: FCI API

Set up

You will need to supply the ID of your app/service (serviceID) and fully initialized instances of fnv::soa::framework::SoaConsumer and fnv::soa::framework::SoaProvider to obtain an instance of the client library.

The service ID must be unique across all ECUs in FNV2.

You must retain references to the SoaConsumer and SoaProvider. FCI client itself does not keep a strong reference to these objects and, consequently, would become unusable if you would dispose of your references.

Each call of the static factory method createInstance() will instantiate a new library object of type FciClient.

It is assumed, but not enforced, that the caller will not re-use the same consumer/provider references for subsequent instantiations of FCI client. Should the caller re-use the same consumer/provider to obtain another instance of the fci client, the functionality of both (all) instances would become undefined.

As mentioned above, the serviceID passed with each createInstance() call must be unique.

Service ID

The only constraint on this parameter is that it must be unique among all FCI users. Please see System Service Descriptor Programmer's Guide

Enabling the library in your code

To use the library in your code:

1) add *LIBS += -lfciclient -lsoaframework* to your makefile. Note that you will most likely need additional libraries to handle FTCP payload in your app (e.g. lftcpidl, lprotobuf);

2) include *libfciclient/FciClient.hpp*

Usage

You can update the default timeout values for the synchronous function calls by calling the corresponding setters on your consumer/provider.

Set listeners

If you are interested in receiving callback notifications for either of:

- FTCP messages received
- Sent messages' delivery confirmation
- SDN (cloud) connection status updates

You must supply definitions for the corresponding classes and call the set*Listener() methods.

You must keep the references to the listeners that you register for as long as you expect the listeners to provide callbacks.

Register for FTCP Commands

FTCP messages are defined in the FTCP specification and corresponding protobuf (/fnv/idl/ftcp).

You have to register to receive one or more defined FTCP Commands to Service Manager using the descriptor. See the System Service Descriptor Programmer's Guide for details.

FTCP messages are defined in the FTCP specification and corresponding protobuf (/fnv/idl/ftcp) and FTCP message name is used in the descriptor to register the FTCP command.

Send FTCP messages

You can send FTCP command responses, Alerts and Queries.

To send a command response or a correlated Alert, the FciData parameter must have transaction ID set to be identical to that in the command received.

To send an non-correlated Alert or a Query, the FciData parameter transaction ID should be set by your application/service such that it makes sense for you. For Tx messages, your transaction ID will allow you to track delivery confirmations and received response messages.

Create FTCP message vehicle status

static function FciClient::createFtcpVehicleStatus: caller need to get the VIM info itself
as vimsoainterface::VimPrimitiveDynamicInfo format, and allocate a empty VehicleStatue message. This function will input the fields in the VehicleStatue.

Clean up

You must call the cleanup() method before deleting the fci client instance or letting the reference go out of scope.

It is possible to continue using the library instance after you call cleanup(), however, you'd need to register for callbacks and messages again.