fnv::soa: (a) General Information (from ECG)

SOA API overview

Overall architecture

The main purpose of the Service Oriented Architecture (SOA) is to provide an efficient communication framework for the various in-vehicle components to exchange messages and data. The SOA API is the library used by a component to send and receive messages over the SOA framework. This shared library provides a variety of additional features as Remote Procedure Call (request response communication), synchronous and asynchronous methods and service registration. Multiple SOA clients integrating each the SOA API library are all connected through the SOA framework.

The SOA framework is the core of the system. There is a single instance running on the FNV2-ECG. The two main systems are the MQTT broker and the Service Manager.

The SOA client connects to the SOA framework over either MQTT transport layer or IPC (not defined/implemented yet). This client can be a service consumer and listening to messages or a service provider, sending information to consumers. There is no restriction for a client to be both consumer of provider.

The service Manager has a number of purposes. The first is to register services which are downloaded and dynamically created. Any SOA client through the SOA API can get access to the Service Directory and query if a service (provider) is present and available. in addition to the service status, the Service Manage also provides a reference used to access it. This reference is the endpoint of the service.

The last critical part of the SOA framework is the Access Control List. This security mechanism is in charge of authorizing transactions to be pushed to the SOA clients. For each consumer communicating over the framework, the services which are not accessible are listed in the ACL database. The broker checks each transactions to make sure it is authorized.

Forking not supported in QNX

WARNING: 2018/03/09 - The use of the fork() function to create a new process is NOT SUPPORTED for SOA Framework applications when running in QNX. This is because a QNX forked child process will apparently only have one thread, while the SOA Framework uses multiple background threads. This needs more investigation.

Listener Lifecycles Caveat - Avoid Crashes

IMPORTANT info to avoid crashes

You MUST read the following section!

Listener classes written by application developers using the SOA Framework API implement callback methods called by the SOA framework when messages arrive from the network. Listener classes are defined by the application developers as derived from SOA listener classes (ex. SoaDataListener), and objects are instantiated by the application code. Therefore, the application code controls the lifecycle of the listener objects. The base listener classes implemented by the SOA Framework include a resource tracking feature which blocks the framework from calling the callback if the application's instantiated listener object was destroyed.

However, there exists a race condition scenario during the destruction of the listener object where the framework can call the callback while the listener's destructor is executing if the explicit use of a framework-provided API is not made. This can cause a crash! Your requirement as the application developer is to add a call to

```
this->removeResource();    // declared as virtual void SoaResourceManager::Resource::removeResource() final;
```

to the **beginning** of your listener's destructor. (The first line!) Adding this call ensures that your destructor will not execute until after a race-condition callback call has finished executing. This will serialize the execution of a callback for that listener and the destruction of your listener.

If your listener class uses multiple inheritance to derive from multiple listeners to implement multiple callbacks, you need to call removeResource() once for each base listener class where the "this" pointer is cast to each base listener class.

Example:

```
YourDerivedListener::~YourDerivedListener() {

    static_cast<SoaDataListener*>(this)->removeResource();

    static_cast<SoaActionResultListener<SoaSubscribeToDataContext>*>(this)->removeResource();

    // the rest of the code in your destructor below here ...

}
```

SoaResourceManager::Resource::removeResource**()** will be available in releases SOA-R0.67.5 and later.

You could use SoaResourceManager::instance().deleteResource(this) before that time, but long term use is discouraged because the removeResource() method is more efficient.

N.B. You'll see that all of the Soa*???*Listener base class definitions include a call to *removeResource()* in their destructors. This DOES NOT MEAN you do not need to put it in your derived listener destructor. Destructors are executed in order from most derived to base class, and you need to protect your derived class data from being accessed by another thread concurrent with being destroyed/deallocated.

Failure to comply

Failing to do this results in a warning being logged whenever your objects are destroyed...

Example:

Warning 2018-11-20T16:01:40.023876-05:00 tcumaind 21840 21840 soa [GWSV.21840] 0679 - SoaResourceManager: Class most derived from Resource did not call removeResource() in destructor, derived from SoaServiceRequestListener

If failing to do this leads to a detected race condition of your destroyed derived listener callback being called during object destruction, this critical error will get logged.

$$
Race condition suspected! Parent virtual callback method called.
Likely cause: User-implemented derived listener class destructor does not call removeResource()

SoaListenerLifecycleHelper

Use the SoaListenerLifecycleHelper wrapper instead! We've made things a little easier for you to automate calling removeResource() on your behalf.

Instead of adding one or more calls to removeResource() in your destructor, you are recommended to wrap your derived listener class with the SoaListenerLifecycleHelper class template. This is automatically available to all listener class implementations.

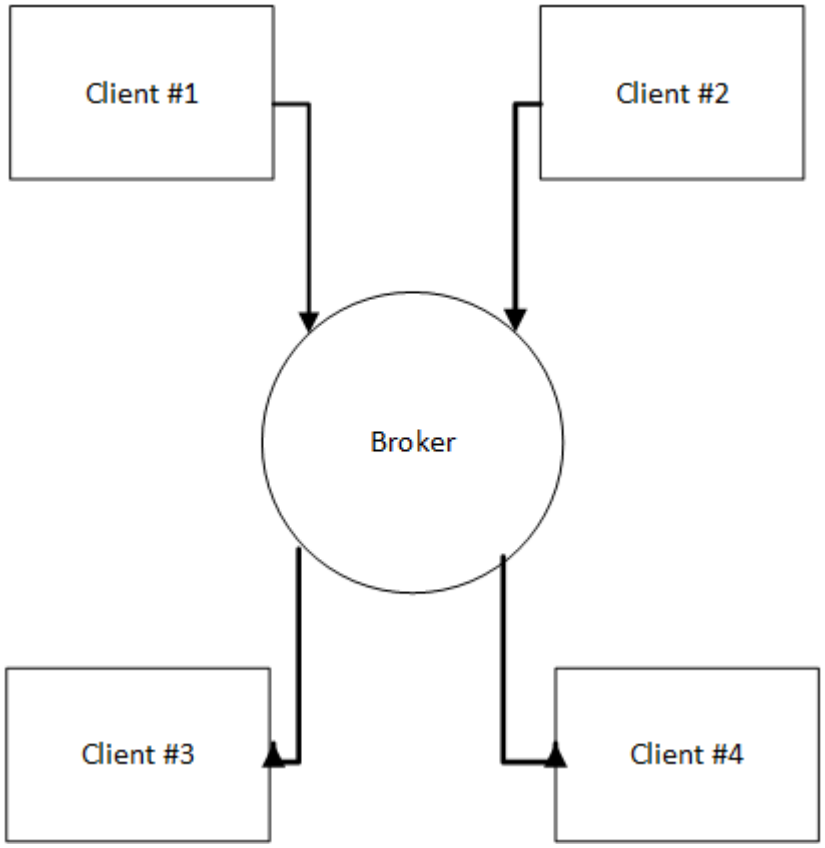
This class template supports derived listener classes which inherit from one SOA listener class or from multiple SOA listener classes. To wrap your derived listener class with this template, your listener class becomes a template instantiation argument for a new class which can be used exactly as your listener class was used. Provide your listener class followed by all of the SOA listener classes your listener class derives from as template arguments.

Example of wrapping a listener class with SoaListenerLifecycleHelper

```
// Example of your existing listener class
class MyDataListener :
            public SoaDataListener,
            public SoaActionResultListener<SoaSubscribeToDataContext>
{
   // ... user class definition, including callbacks declarations

   virtual void onDataReceived(SoaMessage::SharedPtr message);

   virtual void onActionCompleted(SoaErrorCode error, std::shared_ptr<SoaSubscribeToDataContext> actionContext);
};


// A wrapped class is created as follows:


using MySafeDataListener = SoaListenerLifecycleHelper < MyDataListener, SoaDataListener,
SoaActionResultListener<SoaSubscribeToDataContext> >;
// or
typedef SoaListenerLifecycleHelper < MyDataListener, SoaDataListener,
SoaActionResultListener<SoaSubscribeToDataContext> > MySafeDataListener;
```

Basic MQTT concepts

MQTT is the publish/subscribe architecture on the ECG supporting Service Oriented Architecture (SOA). Messages are pushed to the clients instead of using a request/response typical of HTTP. Each client is connected to a central broker. This section will cover the main concepts as the SOA framework is built on top of MQTT.



Clients

A MQTT client is the component connected to the broker. It can publish messages to the broker and/or receive messages from the broker. Clients are not aware of the network topology and do not need to know each other to send and receive messages. The client pushes message to the broker to publish and the broker pushes messages to the client subscribing for the message.

Broker

The MQTT broker can be seen as a central hub receiving and dispatching messages between the connected clients. The connection between the broker and the clients rely on TCP. If for any circumstances the connection is interrupted, the broker has some buffering capability and can send them back to the client once the connection is resumed.

Topics

Topic is a simple string with some hierarchy levels used by the broker to route messages. A client willing to receive messages subscribes to certain topics. The broker will only push messages the client subscribed to. A sample topic to send vehicle speed from the ECG Vehicle Information Manager could be:

SERVICES/DATA/ECG/VIM/DASHBOARD/VEHICLESPEED

Any client interested in this information will subscribe to this exact topic. When a message is pushed to the broker, the broker will check if any client is subscribed to the topic and forward the message to the registered client if so.

There is also an option for the client to not necessarily subscribe to the exact topic but to a broader topic. A client can be interested in receiving all the messages from the VIM/DASHBOARD. For this purpose, the wildcard subscription is possible.

SERVICES/DATA/ECG/VIM/DASHBOARD/+

Now if the client wants to receive messages from more than one level, the multilevel wildcard can be used:

SERVICES/DATA/ECG/VIM/#

Publish

Publishing is the action to push a message with a given topic to the broker. Once received by the broker, the topic is checked and if any client is interested in the topic, the message is pushed to the client(s).

Subscribe

Subscribing is the action for a client to tell the broker in which messages with a particular topic it is interested in. The client can subscribe to multiple topics and can unsubscribe as well.

Payload

Each message will include a payload including the actual data. MQTT is data agnostic and does not specify the format. It is up to the client to use whatever it required. It means that if 2 clients communicate together, they must agree on the data format to be able to read properly the payload.

SOA API library

A component willing to use the SOA framework for communication purpose is required to include the SOA shared library which includes all the features providing various communication mechanisms. This library can be found on GitHub .

SOA concepts

The SOA framework has been developed on top of MQTT publish/subscribe so some of the concepts are related to MQTT.

Service consumer

The service consumer is a SOA client mainly listening to data. The API providing this capability is mainly part of the SoaConsumer class with the subscribeToData() call. The consumer has to provide the endpoints it is interested in. There is no limitation of the number of subscriptions. Once the consumer disconnect and reconnect, the subscriptions are lost except if the persistent session is set.

Both blocking and non blocking calls are provided. For the former, the developer provides a callback only called when a message is received from the registered service. The function block until the request returns its status. (publish succeeded or failed). The non blocking call required one additional callback to provide a way to receive the transaction status as The call does not wait for the subscribe status and returns immediately.

For blocking calls, a timeout can be specified by the developer if a status is not received within a given duration.

Service provider

The service provider (or provider in short) is a SOA client as well but providing data for other clients (the consumers). The SoaProvider class provides all the communication APIs to support services. PublishMessage() is one of them pushing data to be broker.

endpoints

Remote procedure Call

One type of communication mechanism not supported by the typical publish/subscribe is request-response also know as Remote Procedure Call (RPC). The SOA framework supports this RPC providing the remoteCall API on the consumer side and the remoteCallResponse() on the provider side.

As for SOA publish and subscribe, blocking and non blocking calls are supported. The blocking call will wait until the response is received. The non blocking implementation relies on a callback to receive the response.

On-demand broadcast

On a regular MQTT, services or data can be pushed to the broker even if there is no consumer listening to it. This mechanism is called broadcast. This is every inefficient as bandwidth is consumed with nobody consuming data. To make sure the resources are better utilized, this communication type is not permitted. If a comsumer is interested in receiving data, it need to send a request to the service provider first. The provider keeps a reference of the request for a particular ervice type. If there is no consumer listening to, the publication is stopped. This feature is provided by the SOA API (Phase 2) so the service developer does not need to implement it.

Service Manager

SOA production repository

The SOA production repository is hosted on Github: SOA code.

We also have a wiki page describing the release dates and content: SOA Middleware Delivery Schedule

Installing the SOA library

The SOA client comes as a shared library

Initialization

Create a SoaConnectionOptions object and set desired connection options.

Most of the options have a default value so there is no need to set every option.

```
    SoaConnectionOptions::SharedPtr connOpts = SoaConnectionOptions::createConnectionOptions();
    connOpts->setClientId(UNIQUE_CLIENT_ID); // deprecated as client id is now determined by the soa framework
relying on ECU name and user Id
    connOpts->setDebugBrokerUrl("tcp://10.1.10.1:1883"); // deprecated starting with release 0.67.5
    connOpts->setConnectionTimeOut(CONNECTION_TIMEOUT_IN_SECONDS);
```

Create a SoaMessageManager using the SoaConnectionOptions object created.

Then SoaConsumer::initialize() and SoaProvider must be called to establish connection to SOA Broker with connection options specified in connOpts. There is one connection per SoaMessageManager which is shared by all of the SoaProvider and/or SoaConsumer client objects, but all client objects must be initialized. Subsequent client objects for which initialize is called will silently bypass the connection operation.

Note: that SoaConsumer::initialize() and SoaProvider::initialize() call connect() under the hood, and the public SoaMessageManager::connect() blocking interface is meant only for TCU applications, and then only works after a connection that was first established by calling initialize() was later disconnected by calling disconnect(). It can be used to connect already initialized clients.

However, there is a non-blocking version of connect called connectAsync() that may be called by all applications. After a successful connection status is returned to the provided listener callback, client objects can be successfully initialized using the blocking initialize() call. Calling initialize() will fail if its call to connect() fails, so calling connectAsync() can be used to establish the connection before calling initialize(). Caveat: initialize() cannot be called from the callback thread, on which the connection status callback will be called.

See the section below on SOA Asynchronous Connection.

Create a SoaServiceDirectoryManager using the object as parameter for the constructor. Also a unique SoaClientEndpoint must be specified.

```
    SoaServiceDirectoryManager::SharedPtr svcDirMan = SoaServiceDirectoryManager::create(msgMan,
SERV_DIR_ENDPOINT);
```

Create a SoaConsumer if needed, passing the SoaMessageManager and SoaServiceDirectoryManager objects as parameter for the constructor.

A unique SoaClientEndpoint must be specified for receiving response messages during remote procedure calls.
SoaConsumer::initialize() should be called on the consumer object to establish connection to SOA Broker if not done already, and subscribe to CONSUMER_ENDPOINT for all response message sent to this endpoint.

Call SoaConsumer::setTimeout to specify the maximum waiting time for blocking calls that does not take a timeout parameter, also for the maximum time before SoaActionResultCallback is called with a timeout error for non-blocking calls.
The default timeout is 1000ms, all timeout should be specified in ms as a int32_t value.

```
    SoaConsumer::SharedPtr consumer = SoaConsumer::createSoaConsumer(msgMan, svcDirMan, CONSUMER_ENDPOINT);
        consumer->setTimeout(2000);
        consumer->initialize();
```

Create a SoaProvider if needed, passing the SoaMessageManager object as parameter for the constructor.

```
    provider = SoaProvider::createSoaProvider(msgMan);
        provider->setTimeout(1500);
```

Writing a service provider

Using the blocking API

Using the asynchronous API

Writing a service consumer

Setup

SOAServiceDirectoryManager

One of the main methods of SoaConsumer is "remoteCall" to a service. But before invoking "remoteCall" a consumer needs to know the service status. Developers can request the service status by invoking SoaConsumer::requestServiceStatus function with the endpoint of service.

```
SoaAciontResult<ServiceStatus>::SharedPtr serviceStatus = consumer->requestServiceStatus(SERVICE_ENDPOINT,
timeout);
```

For dynamic services the API will send a service status request to Service Manager service. However to improve performance and reduce SOA traffic it was suggested not to involve Service Manager service in tracking system always-running services. Therefore SoaConsumer developers should manually populate SOAServiceDirectoryManager with system services contracts.

`SOAServiceDirectoryManager::registerSystemService API.`

```
/**
* Use this API to statically register system services you are going to access in you code.
* System service assumption:
*       It is a well knows service with the contract defined in a header file
*       The service has global lifecycle
* @param contract The contract
* @return The error code
*/
SOAErrorCode SOAServiceDirectoryManager::registerSystemService(SOAServiceContract::SharedPtr contract,
SOAServiceStatus status)
```

`SOAServiceDirectoryManager::registerSystemService API usage.`

```
// This should be defined in an external header file
const SOAServiceContract::SharedPtr testServiceContract(new SOAServiceContract(SOAClientEndpoint::
createClientEndpointMqtt("TEST_TOPIC")));


// Creating a service directory manager
SOAMessageManager::SharedPtr messageManager;
SOAClientEndpoint::SharedPtr sdEndpoint;
SOAServiceDirectoryManager::SharedPtr sdManager = SOAServiceDirectoryManager::create(sdEndpoint,
messageManager);


// Registering system test service.
sdManager->registerSystemService(testServiceContract, SOAServiceStatus::RUNNING);


// The sdManager instance can now be shared between different SoaConsumers.
SoaConsumer::SharedPtr consumer = SoaConsumer::createSoaConsumer(messageManager, sdManager, consumerEndpoint);


// Requesting the test service status
SoaAciontResult<ServiceStatus>::SharedPtr serviceStatus = consumer->requestServiceStatus(SERVICE_ENDPOINT,
timeout); // the status should be SOAServiceStatus::RUNNING
```

Using the blocking API

Using the asynchronous API

How to implement Remote Procedure Call

i. Setup

Refer to Initialization section for setting up conusmer and provider for remote call.
Note that the request response channel option have to be set to true to enable remote call functionalities.

```
        SoaConnectionOptions connOpts;
    connOpts.setRequestResponseChannel(true);
```

ii. Using the blocking API

The blocking API intend to be used to write sequential SOA programs, all API blocks until action is complete, when error occurs or when timeout is reached.

Consumer side:

Building request message:

Create a SoaMessage::SharedPtr, set a response endpoint , and set command id for specific command to the service.

```
        SoaMessage::SharedPtr request = make_shared<SoaMessage>();
        request->setConsumerEndpoint(CONSUMER_ENDPOINT);
        request->setCommandID(CMD_DO_SOMETHING);
```

Getting service status:
Before performing a remote procedure call, it is necessary to know if the service handling the call is available.
This can be achieved using SoaConsumer::requestServiceStatus function with the endpoint of service interested in and a timeout in ms.

```
SoaAciontResult<ServiceStatus>::SharedPtr serviceStatus = consumer->requestServiceStatus
(SERVICE_ENDPOINT, timeout);
```

This function returns a pointer to SoaActionResult which can be used to retrieve the error occurred, if any, and the service status returned.

If service is not currently available, a SoaServiceStatusListener can be implemented to listen for updates in service status, and wait until service become available.

```
//Declaration
class ServiceStatusListener : public SoaServiceStatusListener{
        public:
            StatusListener(SoaServiceStatus & status);
            virtual void onStatusChanged(const SoaClientEndpoint::SharedPtr endpoint,
SoaServiceStatus status);
        private:
                SoaServiceStatus & m_status;
};
//Implementation
ServiceStatusListener::StatusListener(SoaServiceStatus & status) : m_status(status){}
ServiceStatusListener::onStatusChanged(const SoaClientEndpoint::SharedPtr endpoint,
SoaServiceStatus status){
        m_status = status;
}


//in main
SoaServiceStatus status = SoaServiceStatus::NOT_AVAILABLE;
ServiceStatusListener* statListener = new ServiceStatusListener(status);
consumer->subscribeToServiceStatusUpdate(PROVIDER_ENDPOINT, *statListener);
while(status == SoaServiceStatus::NOT_AVAILABLE){
        //Sleep and wait
}
consumer->unsubscribeToServiceStatusUpdate(PROVIDER_ENDPINT);
```

After service become available, remote procedure call can be performed.

```
        SoaMessage::SharedPtr responseMsg;
        SoaActionResult<SoaMessage>::SharedPtr response = consumer->remoteCall
(PROVIDER_ENDPOINT, request, RPC_TIMEOUT);
        if(response->getError() == SoaErrorCode::NO_ERROR){
                responseMsg = response->getResult();
        }
```

The returned SoaAcitonResult<SoaMessage> would contain the response message

Provider Side:

Provider need to implement a listener to listen to incoming request messages.

```
//Declaration
SoaMessage::SharedPtr requestMsg;
class RequestListener : public SoaServiceRequestListener{
public:
    RequestListener();
    virtual void onRequestReceived(SoaMessage::SharedPtr request);
};

//Implementation
void RequestListener::onRequestReceived(SoaMessage::SharedPtr request){
    requestMsg = request;
}


requestMsg = nullptr;
SoaMessage::SharedPtr response = nullptr;
while(true){
        while(requestMsg == nullptr){
                //Sleep and wait
        }
        response = std::make_shared<SoaMessage>(requestMsg);
        if(requestMsg->getCommandID() == "Command 1"){
                response->setPayload(response1protobuff);
        }
        else if(requestMsg->getCommandID() == "Command 2"){
                response->setPayload(response2protobuff);
        }
        provider->remoteCallResponse(requestMsg->getConsumerEndpoint(), response, timeout);
}
```

Create response message from request message, the response message need to have the same commandID, transactionID, and serviceVersion as the request message.

Choose the appropriate response payload to put in the response message, then send the response using SoaProvider:: remoteCallResponse.

iii. Using the asynchronous API

The asynchronous API is intended to provide a way to perform SOA acitons without blocking the executing thread, suitable for situations where handling multiple servies is desired.

Consumer side:

Building request message:

Create a SoaMessage::SharedPtr, set a response endpoint , and set command id for specific command to the service.

```
        SoaMessage::SharedPtr request = make_shared<SoaMessage>();
        request->setConsumerEndpoint(CONSUMER_ENDPOINT);
        request->setCommandID(CMD_DO_SOMETHING);
```

Getting service status:
Before performing a remote procedure call, it is necessary to know if the service handling the call is available.

Create a implementation of SoaServiceStatusListener similar to the using synchronous API section, and instantiate multiple listeners for listening to multiple service status.

```
//Declaration
class ServiceStatusListener : public SoaServiceStatusListener{
        public:
            StatusListener(SoaServiceStatus & status);
            virtual void onStatusChanged(const SoaClientEndpoint::SharedPtr endpoint,
SoaServiceStatus status);
        private:
                SoaServiceStatus & m_status;
};
//Implementation
ServiceStatusListener::StatusListener(SoaServiceStatus & status) : m_status(status){}
ServiceStatusListener::onStatusChanged(const SoaClientEndpoint::SharedPtr endpoint,
SoaServiceStatus status){
        m_status = status;
}


//in main
SoaServiceStatus statusA = SoaServiceStatus::NOT_AVAILABLE;
SoaServiceStatus statusB = SoaServiceStatus::NOT_AVAILABLE;
ServiceStatusListener* statListenerA = new ServiceStatusListener(statusA);
ServiceStatusListener* statListenerB = new ServiceStatusListener(statusB)
consumer->subscribeToServiceStatusUpdate(PROVIDER_ENDPOINT_A, *statListenerA);
consumer->subscribeToServiceStatusUpdate(PROVIDER_ENDPOINT_B, *statListenerB);
while(statusA == SoaServiceStatus::NOT_AVAILABLE || statusB == SoaServiceStatus::
NOT_AVAILABLE){
        //Sleep and wait
}
consumer->unsubscribeToServiceStatusUpdate(PROVIDER_ENDPOINT_A);
consumer->unsubscribeToServiceStatusUpdate(PROVIDER_ENDPOINT_B);
```

Another two listeners should be implemented before the asynchronous remote procedure call can happen.

Listener to response and action result:

A SoaActionResultListener have to be implemented to listen to the remote call action result, and a SoaRemoteCallresponseListener have to be implemented to listen to response to the remote call.

```
//Declaration
class RemoteCallResponseListener: public SoaRemoteCallResponseListener{
        public:
            RemoteCallResponseListener(SoaMessage::SharedPtr& response);
            virtual void onResponseReceived(SoaMessage::SharedPtr response);
        private:
                SoaMessage::SharedPtr &  m_response;
};


class ActionResultListener: public SoaAcitonResultListener<SoaRemoteCallContext>{
        public:
                ActionResultListener(SoaErrorCode & error, SoaConsumer::SharedPtr consumer);
                virtual void onActionCompleted(SoaErrorCode error,
shared_ptr<SoaRemoteCallContext> actionContext);
        private:
                SoaErrorCode & error;
                SoaConsumer::SharedPtr m_consumer;
                int32_t m_retryCounter;
}
//Implementation
RemoteCallResponseListener::RemoteCallResponseListener(SoaMessage::SharedPtr& response) :
m_response(response){}
RemoteCallResponseListener::onResponseReceived(SoaMessage::SharedPtr response){
        m_response = response;
}


ActionResultListener::ActionResultListener(SoaErrorCode & error, SoaConsumer) : m_error
(error), m_consumer(consumer), m_retryCounter(0) {};
ActionResultListener::onActionComplete(SoaErrorCode error, SoaRemoteCallContext
actionContext){
        m_error = error;
        //Auto retry can be implemented using the information in SoaRemoteCallContext object
        if(error != SoaErrorCode::NO_ERROR){
                if(retryCounter < 3){
        m_consumer->remoteCallAsync(actionContext->getProviderEndpoint(), actionContext-
>getRequestMessage(), actionContext->getRemoteCallListener(),

*reinterpret_cast<SoaActionResultListener<SoaRemoteCallContext>*>(this));
                m_retryCounter++;
         }
 else{
 m_error = error;

 }
        }
}
```

Now remote procudure call can be performed

```
        SoaMessage::SharedPtr responseMsgA;
        SoaMessage::SharedPtr responseMsgB;
        SoaErrorCode errorCodeA;
        SoaErrorCode errorCodeB;
        ActionResultListener actionListenerA;
        ActionResultListener actionListenerB;
        RemoteCallListener responseListenerA;
        RemoteCallListener responseListenerB;


        consumer->remoteCallAsync(PROVIDER_ENDPOINT_A, requestA, responseListenerA,
actionListenerA);
        consumer->remoteCallAsync(PROVIDER_ENDPOINT_B, requestB, responseListenerB,
actionListenerB);
```

The returned SoaAcitonResult<SoaMessage> would contain the response message

ProviderSide:

Provider need to implement a listener to listen to incoming request messages.

Since we are using the asynchronous API, a SoaActionResultListener have to be implemented to listen for the result of remote call response action.

```
//Declaration
mutex requestMut, actionMut;
queue<SoaMessage::SharedPtr> requestQueue;
condition_variable requestCv, actionCv;
class RequestListener : public SoaServiceRequestListener{
public:
    RequestListener();
    virtual void onRequestReceived(SoaMessage::SharedPtr request);
};


class RespondActionResultListener : public SoaActionResultListener<SoaPublishMessageContext>
{
public:
        ResponseActionResultListener(SoaErrorCode & error);
        onActionCompleted(SoaErrorCode error, SoaPublishMessageContext actionContext);
private:
        SoaErrorCode & m_error;
}

//Implementation
void RequestListener::onRequestReceived(SoaMessage::SharedPtr request){
    {
                unique_lock<mutex> lock(requestMut);
                requestQueue.push(request);
        }
        requestCv.notify_one();


}
ResponseActionResultListener::ResponseActionResultListener(SoaErrorCode & error) : m_error
(error){}
ResponseActionResultListener::onActionCompleted(SoaErrorCode error,
SoaPublishMessageContext actionContext){
        m_error = error;
}



SoaMessage::SharedPtr request= nullptr;
SoaMessage::SharedPtr response = nullptr;
while(true){
        requestCv.wait();
        {
                unique_lock<mutex> lock(requestMut);
                request = requestQueue.front();
                requestQueue.pop();
        }
        response = std::make_shared<SoaMessage>(request);
        ResponseActionRequestListener actionListener;
        if(request->getCommandID() == "Command 1"){
                response->setPayload(response1protobuff);
        }
        else if(request->getCommandID() == "Command 2"){
                response->setPayload(response2protobuff);
        }
        provider->remoteCallResponseAsync(requestMsg->getConsumerEndpoint(), response,
actionListener);
}
```

Create response message from request message, the response message need to have the same commandID, transactionID, and serviceVersion as the request message.

Choose the appropriate response payload to put in the response message, then send the response using SoaProvider:: remoteCallResponse.

SOA Reconnection Options

The SOA "reconnection" options are used to tweek how reconnecting and non-blocking connecting is handled.

Connection Lost: When the client's connection to the broker is lost a callback is called. This callback uses the user set connectionLostListener, if one has been set, to notify the user. It then checks if the auto-reconnect is enabled or not. If it is enabled, the system starts attempting to reconnect automatically. The first attempt is immediate. The system continually tries to connect over constant or increasing intervals until it either successfully connects, times out or fails for some reason. In either case the reconnectionListener, if on has been set, is used to notify the user that the system has timed out or succesfully reconnected (ERROR_TIMEOUT or NO_ERROR respectively) or another error resulted. If auto-reconnect is not enabled the user can explicitly call reconnect, but only if a reconnectionListener has been set.

Non-blocking Connection: THis is used for initial connection or connection after willful disconnect. This uses the same algorithm as described above for reconnection after the connection is lost, except the eventual connection status is reported through a different connectionListener.

The soa reconnection options are located within the connection options.
There are 8 settings related to the re/connection system:

Setting	Meaning	Default Value
bool autoReconnect	If true, the reconnection system will automatically attempt to reconnect to the broker if connection is lost. If false, the reconnection system must be called manually to reconnect. The connection lost and reconnection listeners are called in both cases if set.	false
int reconnectTimeout	The maximum amount of time (in milliseconds) the reconnection/connection system will attempt to connect - exclusive of the final millisecond. Must be greater than or equal to 0 (zero). If set to 0, connection will be attempted once only before it times out. If set > 0, connection will be attempted again repeatedly until successful or this timeout value is reached. NOTE: If a retry would be scheduled by the reconnection/connection system to be run on or after the millisecond of the reconnectTimeout, the system will not schedule another retry but will stop with an error of ERROR_TIMEOUT. For example, if you want the system to retry every minute for 1 hour, pad the reconnectTimeout value by a few seconds or milliseconds to get the full 61 attempts before it times out. (Ex. 3605000 for 1 hour and 5 seconds)	3600000 ms (1 hour)
int minRetryInterval	The minimum amount of time (in milliseconds) between each connection attempt. Must be less than or equal to maxRetryInterval and reconnectTimeout. Must be greater than 0 (zero) except if the reconnectTimeout is 0	500ms (0.5 seconds)
int maxRetryInterval	The maximum amount of time (in milliseconds) between each reconnection attempt.	60000ms (1 minute)
float retryIntervalMultiplier	The value by which each successive retry interval is multiplied. Retry intervals begin with the min retry interval and after each connection attempt is multiplied by this value. If the next retry interval would be greater than the max retry interval, the next retry interval is set to the max retry interval. Must be greater than or equal to 1.0F	2
SoaReconnectionListener* reconnectionListener	Pointer to an instance of a user created class that inherits from SoaReconnectionListener and implements void onReconnect (SoaErrorCode errorCode). This pointer is used to call onReconnect(...) on a successful reconnect() call or an auto-reconnect (errorCode = NO_ERROR), or on a timeout (errorCode = ERROR_TIMEOUT), among other possible errors.	nullptr
SoaConnectionLostListener* connectionLostListener	Pointer to an instance of a user created class that inherits from SoaConnectionLostListener and implements void onConnectionLost(const std::string& cause). This pointer is used to call onConnectionLost(..) when the connection to the broker is lost. This callback can be used to manually reconnect if the auto reconnect option is off.	nullptr
SoaConnectionListener* connectionListener	Pointer to an instance of a user created class that inherits from SoaConnectionListener and implements void onConnect (SoaErrorCode errorCode). This pointer is used to call onConnect(...) on a successful connectAsync() (errorCode = NO_ERROR), or on a timeout (errorCode = ERROR_TIMEOUT), among other possible errors. This listener MUST be set as a connection option or else SoaMessageManager::connectAsync() will fail with an immediate error of ERROR_INVALID_CALLBACK_PARAM	nullptr

You may implement a single common class to realize all three of the pure-virtual listener interface classes.

Setters:
void setConnectionLostListener(SoaConnectionLostListener& listener);
void setReconnectionListener(SoaReconnectionListener& listener);
void setConnectionListener(SoaConnectionListener& listener);
void setAutoReconnect(bool autoReconnect);
SoaErrorCode setReconnectionTimeOptions(int32_t reconnectTimeout, int32_t minRetryInterval, int32_t maxRetryInterval, float retryIntervalMultiplier); —> values set only if NO_ERROR.

Getters:
bool isAutoReconnect();
int32_t getReconnectTimeout();
int32_t getMinRetryInterval();
int32_t getMaxRetryInterval();
float getRetryIntervalMultiplier();

SOA Asynchronous Connection

Methods

SoaMessageManager::reconnect()

SoaMessageManager::connectAsync()

Whereas SoaMessageManager::reconnect() initiates a non-blocking action to attempt to reconnect to the broker if an existing connection was lost, SoaMessageManager::connectAsync() initiates a non-blocking action to attempt to connect to the broker if the application has not yet connected or has disconnected.

Both of these methods run as a background task to try to connect and keep trying according to the parameters configured as the SOA Reconnection Options. Only one will run at a time. Only one set of options can be defined to specifiy how connection and/or reconnection are handled.

A call to any blocking or non-blocking connection API (including initialize()) will fail if the same or another is active (having not yet established a connection).

These methods are non-blocking with respect , but they do perform pre-check

Connection States

When willfully disconnected or not yet connected, the state is SoaConnectionState::DISCONNECTED

Once connected, the state is SoaConnectionState::CONNECTED.

While trying to connect, the connection state is SoaConnectionState::CONNECTING.

If ungracefully disconnected, the connection state is SoaConnectionState::

Non-blocking connection APIs, return values and status values

When the connection state is CONNECTION_LOST, call SoaErrorCode SoaMessageManager::reconnect()

Immediate return values ..

Return Value	Meaning for reconnect()
NO_ERROR	All pre-checks passed and the reconnection/connection retry system has been started. Note: if already connected, NO_ERROR is returned instead of "wrong state" and the callback will be called with a status of NO_ERROR although the re/connection system is not started.
ERROR_WRONG_STATE	Pre-check failure: the connection state is not one of CONNECTION_LOST or CONNECTED
ERROR_INVALID_CALL BACK_PARAM	Pre-check failure: a reconnection listener has not been set in the connection options (this condition does not inhibit auto-reconnect)

Error codes provided to the SoaConnectionListener::onReconnect(...) callback ...

Argument Value	Meaning for reconnect() SoaConnectionListener::onReconnect(SoaErrorCode)
NO_ERROR	Connected successfully

ERROR_TIMEOUT	The MQTT server was found to be unavailable on each connection attempt up to the configured reconnect timeout
ERROR_WRONG_STATE	If the application explicitly calls SoaMessageManager::disconnect() during connection attempts, a "wrong state" error code results This condition aborts the connection retries when detected. Detection is of a finer "paranoid" granularity than minRetryInterval. Initial release granularity of this check is sub-second, and may change if deemed to be too frequent.
ERROR_CONNECT_FAIL	The MQTT library reported an error other than server unavailable This condition aborts the connection retries immediately when found

When the connection state is DISCONNECTED, call SoaErrorCode SoaMessageManager::connectAsync()

Immediate return values ...

Return Value	Meaning for connectAsync()
NO_ERROR	All pre-checks passed and the reconnection/connection retry system has been started. Note: if already connected, NO_ERROR is returned instead of "wrong state" and the callback will be called with a status of NO_ERROR although the re/connection system is not started.
ERROR_INVALID_CALLBACK_PARAM	Pre-check failure: a connection listener has not been set in the connection options
ERROR_USERNAME_OR_PASSWORD_OR_CLIENTID_TOO_LONG	Pre-check failure: one or more of username, password or client id are too long
ERROR_CLIENT_ID_NOT_SET	Pre-check failure: MQTT broker client id has not been set
ERROR_CANNOT_CREATE_CLIENT	Pre-check failure: the MQTT library failed to allocate resources required
ERROR_WRONG_STATE	Pre-check failure: the connection state is not one of DISCONNECTED, CONNECTION_LOST or CONNECTED
ERROR_FAILED_SETTING_CALLBACKS	Pre-check failure: the MQTT library failed to bind SOA library resources

Error codes provided to the SoaConnectionListener::onConnect(...) callback ...

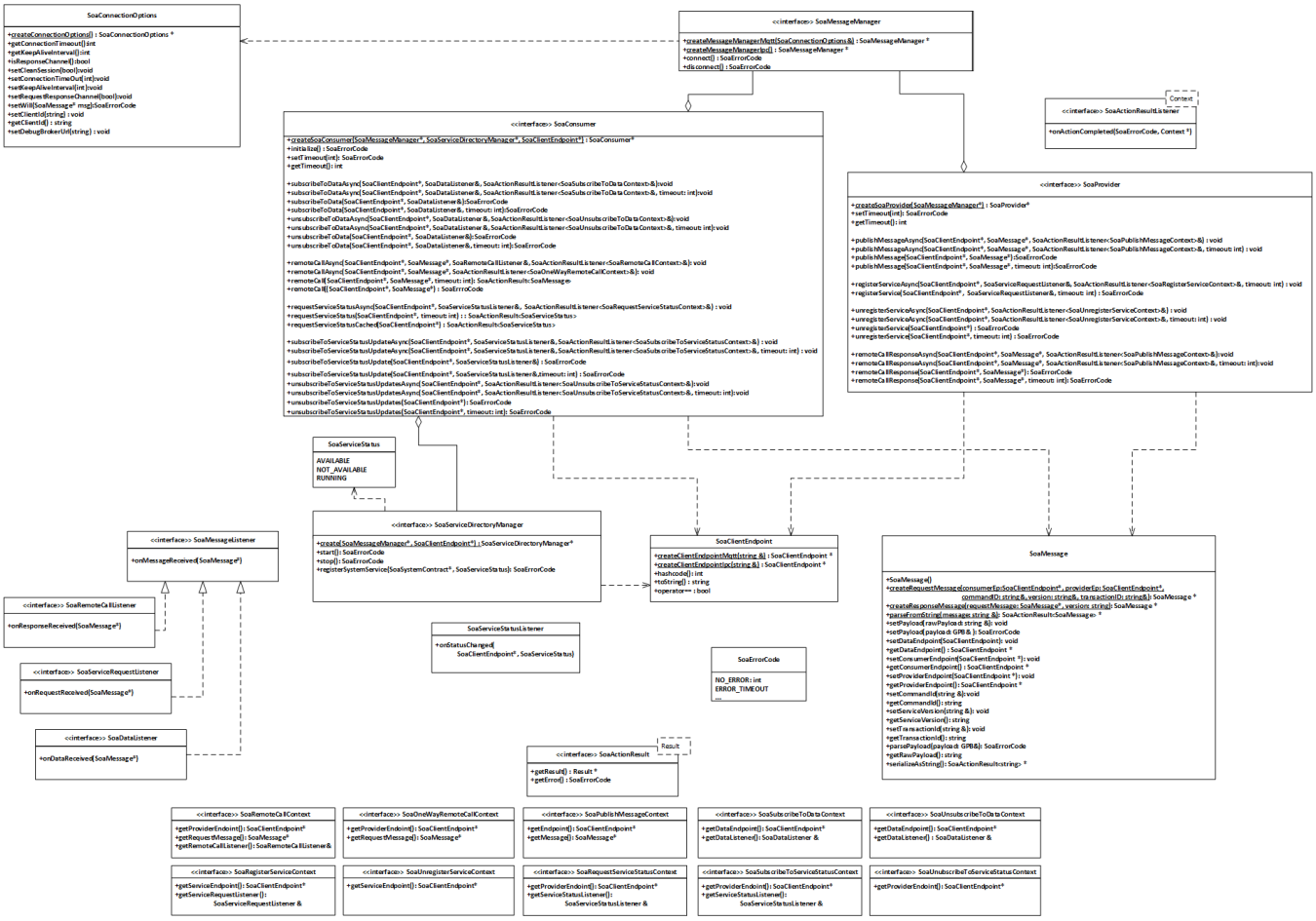
Argument Value	Meaning for connectAsync() SoaConnectionListener::onConnect(SoaErrorCode)
NO_ERROR	Connected successfully
ERROR_TIMEOUT	The MQTT server was found to be unavailable on each connection attempt up to the configured reconnect timeout
ERROR_WRONG_STATE	If the application explicitly calls SoaMessageManager::disconnect() during connection attempts, a "wrong state" error code results This condition aborts the connection retries when detected. Detection is of a finer "paranoid" granularity than minRetryInterval. Initial release granularity of this check is sub-second, and may change if deemed to be too frequent.
ERROR_CONNECT_FAIL	The MQTT library reported an error other than server unavailable This condition aborts the connection retries immediately when found

1. Examples
    a. Publishing to a topic
    b. Subscribing to a topic
    c. Remote Procedure call

Appendix

Class Diagram

SOA_C++PublicApiClassDiagram_20170602.vsdx



SOA message protobuf

```
// Contains SOA message header fields
message SOAProtobufMessage
{

        // Service ID.
        required string serviceID = 1;


        // The Service Command ID.
        optional string serviceCommandID = 2;


        // The Service version
        optional string serviceVersion = 3;


        // Response/Request mapping
        required int32 transactionID = 4;


        // The payload
        optional bytes payload = 5;

}
```

Class diagram

Class documentation

fnv::soa::framework::SoaConnectionManager Class Reference

#include <soa_connection_manager.hpp>

Public Types

- typedef shared_ptr< **SoaConnectionManager** > **SharedPtr**
- **SoaConnectionManager** ()
- SoaErrorCode **registerConnectionCallbacks** (**SoaConnectionCallbacks** callbacks)
- SoaErrorCode **connect** (**SoaConnectionOptions** opts)
- SoaErrorCode **disconnect** (int32_t timeout)
- **~SoaConnectionManager** ()

Public Member Functions

Detailed Description

This class is in charge of managing the connection with ECG-SOA framework

Definition at line 37 of file soa_connection_manager.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaConnectionManager> fnv::soa::framework::SoaConnectionManager::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 42 of file soa_connection_manager.hpp.

Constructor & Destructor Documentation

fnv::soa::framework::SoaConnectionManager::SoaConnectionManager ()

Default constructor

fnv::soa::framework::SoaConnectionManager::~SoaConnectionManager ()

Default destructor

Member Function Documentation

SoaErrorCode fnv::soa::framework::SoaConnectionManager::connect (SoaConnectionOptions *opts*)

Connects to the MQTT broker using the provided connection options.

Parameters:

opts	a **SoaConnectionOptions** specifying options used for this connection

Returns:

an integer connection status of SUCCESS/REFUSED

SoaErrorCode fnv::soa::framework::SoaConnectionManager::disconnect (int32_t *timeout*)

Closes the connection after a given timeout with the ECG-SOA framework and free up any resources.

Parameters:

timeout	a integer specifying timeout before disconnecting in ms

Returns:

a integer status of SUCCESS/FAILURE

SoaErrorCode fnv::soa::framework::SoaConnectionManager::registerConnectionCallbacks (SoaConnectionCallbacks *callbacks*)

Registers the connection related callbacks.

Parameters:

callbacks	a **SoaConnectionCallbacks** object specifying connection callbacks

Returns:

a integer status of SUCCESS/FAILURE

The documentation for this class was generated from the following file:

- C:/Users/frisache/Documents/GitHub/FNV-SOA/soa/include/cpp/framework/soa_connection_manager.hpp

fnv::soa::framework::SoaConnectionOptions Class Reference

#include <soa_connection_options.hpp>

Public Types

- typedef shared_ptr< **SoaConnectionOptions** > **SharedPtr**
- **SoaConnectionOptions** ()
- int32_t **getConnectionTimeout** ()
- int32_t **getKeepAliveInterval** ()
- std::string **getPassword** ()

- bool **isResponseChannel** ()
- std::string **getUserName** ()
- void **setCleanSession** (bool isCleanSession)
- void **setConnectionTimeOut** (int32_t connTimeout)
- void **setKeepAliveInterval** (int32_t keepalive)
- void **setPassword** (const std::string password)
- void **setUserName** (const std::string username)
- ~~SoaErrorCode **setWill** (**SoaMessage** msg)~~ Deprecated - never implemented, will not be implemented
- void **setRequestResponseChannel** (bool isReqRespChannel)
- **~SoaConnectionOptions** ()

Public Member Functions

Detailed Description

Class used to set connection options.

Definition at line 43 of file soa_connection_options.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaConnectionOptions> fnv::soa::framework::SoaConnectionOptions::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 49 of file soa_connection_options.hpp.

Constructor & Destructor Documentation

fnv::soa::framework::SoaConnectionOptions::SoaConnectionOptions ()

Create container with default connection options.

fnv::soa::framework::SoaConnectionOptions::~SoaConnectionOptions ()

Default destructor

Member Function Documentation

int32_t fnv::soa::framework::SoaConnectionOptions::getConnectionTimeout ()

Returns the connection timeout value.

Returns:

connection timeout value

int32_t fnv::soa::framework::SoaConnectionOptions::getKeepAliveInterval ()

Returns the connection "keep alive" interval.

Returns:

keep alive interval

std::string fnv::soa::framework::SoaConnectionOptions::getPassword ()

Returns password associated with connection username

Returns:

user password

std::string fnv::soa::framework::SoaConnectionOptions::getUserName ()

Returns username associated with this connection

Returns:

username

bool fnv::soa::framework::SoaConnectionOptions::isResponseChannel ()

Returns true if the response channel has been setup (for RPC)

Returns:

is channel setup

void fnv::soa::framework::SoaConnectionOptions::setCleanSession (bool *isCleanSession*)

Configures clean session option

Parameters:

isCleanSession	a boolean for clean session option

void fnv::soa::framework::SoaConnectionOptions::setConnectionTimeOut (int32_t *connTimeout*)

Configures connection timeout

Parameters:

connTimeout	a integer specifying connection timeout in ms

void fnv::soa::framework::SoaConnectionOptions::setKeepAliveInterval (int32_t *keepalive*)

Configures keep alive interval

Parameters:

keepalive	a integer specifying keep alive interval in ms

void fnv::soa::framework::SoaConnectionOptions::setPassword (const std::string *password*)

Sets password to be used for connection

Parameters:

password	a password string

void fnv::soa::framework::SoaConnectionOptions::setRequestResponseChannel (bool *isReqRespChannel*)

Configures request-response channel

Parameters:

isReqRespChannel	a boolean that should be set to true if a request and/or response channel need to be created (for RPC)

void fnv::soa::framework::SoaConnectionOptions::setUserName (const std::string *username*)

Sets username to be used for connection

Parameters:

username	a username string

~~SoaErrorCode fnv::soa::framework::SoaConnectionOptions::setWill (SoaMessage *msg*)~~

Deprecated - was never implemented - WIll not be implemented - Instead, all SOA clients register an LWT (MQTT last will and testament message) at connection time. This includes gateway clients. If a gateway client crashes, the Stability

Monitor generates a crashinfo message which is received by the gateway, which in turn broadcasts the regsitered LWT for that process.

~~Specifies will message~~

~~Parameters:~~

~~*msg*~~	~~a **SoaMessage** for will, with Will topic, QoS and retain options set~~

~~Returns:~~

~~Status of action of setting will~~

The documentation for this class was generated from the following file:

- C:/Users/frisache/Documents/GitHub/FNV-SOA/soa/include/cpp/framework/soa_connection_options.hpp

fnv::soa::framework::SoaConsumer Class Reference

#include <soa_consumer.hpp>

Public Types

- typedef shared_ptr< **SoaConsumer** > **SharedPtr**
- virtual SoaErrorCode **initialize** ()=0
- virtual void **setTimeout** (int32_t timeout)=0
- virtual int32_t **getTimeout** ()=0
- virtual void **subscribeToDataAsync** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener, **SoaActionResultListener** < **SoaSubscribeToDataContext** > &actionResultListener)=0
- virtual SoaErrorCode **subscribeToData** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener)=0
- virtual SoaErrorCode **subscribeToData** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener, int32_t timeout)=0
- virtual void **remoteCallAsync** (**SoaClientEndpoint::SharedPtr** providerEndpoint, **SoaMessage::SharedPtr** request, **SoaRemoteCallListener** & callListener, **SoaActionResultListener**< **SoaRemoteCallContext** > &actionResultListener)=0
- virtual void **remoteCallAsync** (**SoaClientEndpoint::SharedPtr** providerEndpoint, **SoaMessage::SharedPtr** request, **SoaActionResultListener**< **SoaOneWayRemoteCallContext** > &actionResultListener)=0
- virtual **SoaActionResult**< **SoaMessage** >::**SharedPtr remoteCall** (**SoaClientEndpoint::SharedPtr** providerEndpoint, **SoaMessage::SharedPtr** request, int32_t timeout)=0
- virtual SoaErrorCode **remoteCall** (**SoaClientEndpoint::SharedPtr** providerEndpoint, **SoaMessage::SharedPtr** request)=0
- virtual **SoaActionResult**< SoaServiceStatus >::**SharedPtr requestServiceStatusCached** (**SoaClientEndpoint::SharedPtr** serviceEndpoint)=0
- virtual void **requestServiceStatusAsync** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceStatusListener** &listener, **SoaActionResultListener**< **SoaRequestServiceStatusContext** > &actionResultListener)=0
- virtual **SoaActionResult**< SoaServiceStatus >::**SharedPtr requestServiceStatus** (const **SoaClientEndpoint::SharedPtr** serviceEndpoint, const int32_t timeout)=0
- virtual void **subscribeToServiceStatusUpdateAsync** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceStatusListener** &listener, **SoaActionResultListener**< **SoaSubscribeToServiceStatusContext** > &actionResultListener)=0
- virtual SoaErrorCode **subscribeToServiceStatusUpdate** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceStatusListener** &listener)=0
- virtual SoaErrorCode **subscribeToServiceStatusUpdate** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceStatusListener** &listener, int32_t timeout)=0
- virtual void **unsubscribeToDataAsync** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener, **SoaActionResultListener**< **SoaUnsubscribeToDataContext** > &actionResultListener)=0
- virtual SoaErrorCode **unsubscribeToData** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener)=0
- virtual SoaErrorCode **unsubscribeToData** (**SoaClientEndpoint::SharedPtr** dataEndpoint, **SoaDataListener** &dataListener, int32_t timeout)=0
- virtual void **unsubscribeToServiceStatusUpdateAsync** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaActionResultListener**< **SoaUnsubscribeToServiceStatusContext** > &actionResultListener)=0
- virtual SoaErrorCode **unsubscribeToServiceStatusUpdate** (**SoaClientEndpoint::SharedPtr** serviceEndpoint)=0
- virtual SoaErrorCode **unsubscribeToServiceStatusUpdate** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, int32_t timeout)=0
- virtual **~SoaConsumer** ()
- static **SharedPtr createSoaConsumer** (**SoaMessageManager::SharedPtr** msgManager, **SoaServiceDirectoryManager::SharedPtr** svcDirMgr, **SoaClientEndpoint::SharedPtr** endpoint)

Public Member Functions

Static Public Member Functions

Detailed Description

This class is used to define the service consumer

Definition at line 50 of file soa_consumer.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaConsumer> fnv::soa::framework::SoaConsumer::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 55 of file soa_consumer.hpp.

Constructor & Destructor Documentation

virtual fnv::soa::framework::SoaConsumer::~SoaConsumer ()[inline], [virtual]

Destructor

Definition at line 455 of file soa_consumer.hpp.

Member Function Documentation

static SharedPtr fnv::soa::framework::SoaConsumer::createSoaConsumer (SoaMessageManager::SharedPtr *msgManager*, SoaServiceDirectoryManager::SharedPtr *svcDirMgr*, SoaClientEndpoint::SharedPtr *endpoint*)[static]

Factory

Parameters:

msgManager	a **SoaMessageManager** that delivers and receives messages
endpoint	a **SoaClientEndpoint** specifying the endpoint of this consumer

virtual int32_t fnv::soa::framework::SoaConsumer::getTimeout ()[pure virtual]

Gets the common timeout value which applies to the blocking methods, and to the maximum time before an action status callback is called for non-blocking methods.

Returns:

the timeout value in milliseconds

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::initialize ()[pure virtual]

Initialize consumer, subscribe to consumer endpoint and establish connection to SOA broker if it is not established yet

Returns:

a SoaErrorCode to specify the type of error occurred

virtual SoaActionResult<SoaMessage>::SharedPtr fnv::soa::framework::SoaConsumer::remoteCall (SoaClientEndpoint::SharedPtr *providerEndpoint*, SoaMessage::SharedPtr *request*, int32_t *timeout*)[pure virtual]

Requests service and block until the response is received or timeout is reached.

Parameters:

providerEndpoint	The provider's endpoint

request	a **SoaMessage** specifying the request details e.g. CommandID request endpoint, and response endpoint
timeout	a integer specifying the call timeout in ms

Returns:

a **SoaActionResult** providing the status code and conditional response message if successful

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::remoteCall (SoaClientEndpoint::SharedPtr *providerEndpoint*, SoaMessage::SharedPtr *request*)[pure virtual]

Initiates a remote call to a service provider that will not provide a response, and block until local status of attempt is known. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the sending of the remote call has failed

Parameters:

providerEndpoint	The provider's endpoint
request	a **SoaMessage** specifying the request details e.g. CommandID request endpoint, and response endpoint

Returns:

a SoaErrorCode specifying the type of error occurred if any

virtual void fnv::soa::framework::SoaConsumer::remoteCallAsync (SoaClientEndpoint::SharedPtr *providerEndpoint*, SoaMessage::SharedPtr *request*, SoaRemoteCallListener & *callListener*, SoaActionResultListener< SoaRemoteCallContext > & *actionResultListener*)[pure virtual]

Requests a service and expect a response received by the listeners

Parameters:

request	a **SoaMessage** specifying the request details e.g. CommandID request endpoint, and response endpoint
callListener	a **SoaRemoteCallListener** that listens for response message
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

virtual void fnv::soa::framework::SoaConsumer::remoteCallAsync (SoaClientEndpoint::SharedPtr *providerEndpoint*, SoaMessage::SharedPtr *request*, SoaActionResultListener< SoaOneWayRemoteCallContext > & *actionResultListener*) [pure virtual]

Initiates a remote call to a service provider that will not provide a response This is a non-blocking method. The status of the remote call attempt is sent to the callback.

Parameters:

providerEndpoint	The provider's endpoint
request	a **SoaMessage** specifying the request details e.g. CommandID request endpoint, and response endpoint
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

virtual SoaActionResult<SoaServiceStatus>::SharedPtr fnv::soa::framework::SoaConsumer::requestServiceStatus (const SoaClientEndpoint::SharedPtr *serviceEndpoint*, const int32_t *timeout*)[pure virtual]

Gets the service status from the Service Directory and blocks until status is received, or the timeout expires. This method has the side effect of creating an implicit persistent subscription such that status updates will be received by the framework and the cached status will remain valid unless an unsubscribe status method is called. Calling this method after explicitly subscribing to status updates will not cancel the explicit status subscription. The timeout provided is for this subscription request only and will not impact the blanket timeout. The timeout may expire resulting in an error code indicating a timeout occurred, in which case the no implicit subscription is created

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in
timeout	a integer specifying request timeout

Returns:

a **SoaActionResult** containing a error code to specify the type of error occurred if any and a SoaServiceStatus

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual void fnv::soa::framework::SoaConsumer::requestServiceStatusAsync (SoaClientEndpoint::SharedPtr *serviceEndpoint*, SoaServiceStatusListener & *listener*, SoaActionResultListener< SoaRequestServiceStatusContext > & *actionResultListener*)[pure virtual]

Requests the service status from the Service Directory. The status is received by the provided status listener, but only once to provide the current status. This method does not create an explicit persistent subscription such that the listener is called again. It does, however, create an implicit persistent subscription such that status updates will be received by the framework and the cached status will remain valid unless an unsubscribe status method is called. Calling this method after explicitly subscribing to status updates will not cancel the explicit status subscription. The status listener is guaranteed to not be called before the status callback is called and returns.

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in
listener	a **SoaServiceStatusListener** to listen for service status returned from Service Directory
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaActionResult<SoaServiceStatus>::SharedPtr fnv::soa::framework::SoaConsumer::requestServiceStatusCached (SoaClientEndpoint::SharedPtr *serviceEndpoint*)[pure virtual]

Returns the last known status of a given service without blocking, if available. Else an error code is returned without blocking. A cached status is only available if this or one of the related methods has already been called causing an explicit or implicit subscription for service status, and the subscription has provided at least one status notification. Subsequent explicit or implicit status update notifications will ensure the cached value is always valid. Unsubscribing to service status will immediately invalidate the cached status.

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in

Returns:

a **SoaActionResult** containing a error code to specify the type of error occurred if any and a SoaServiceStatus

See also:

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual void fnv::soa::framework::SoaConsumer::setTimeout (int32_t *timeout*)[pure virtual]

Sets the common timeout for the blocking methods, and the maximum time before an action status callback is called for non-blocking methods.

Parameters:

timeout	The timeout in milliseconds

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::subscribeToData (SoaClientEndpoint::SharedPtr *dataEndpoint*, SoaDataListener & *dataListener*)[pure virtual]

Subscribes to a provider specified by dataEndpoint. The call is blocking until an error code is returned The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

dataEndpoint	a pointer to a **SoaClientEndpoint** specifying the data endpoint to subscribe to
dataListener	a **SoaDataListener** that listens to incoming message from provider

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

setTimeout

SoaConsumer::subscribeToDataAsync

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&, int32_t)

SoaConsumer::unsubscribeToDataAsync

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::subscribeToData (SoaClientEndpoint::SharedPtr *dataEndpoint*, SoaDataListener & *dataListener*, int32_t *timeout*)[pure virtual]

Subscribes to a dataEndpoint and wait up to the given timeout in ms to get an error code. The message is received by the subListener. The timeout provided is for this subscription request only and will not impact the blanket timeout. The timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

dataEndpoint	a **SoaClientEndpoint** specifying the data endpoint to subscribe to
dataListener	a **SoaDataListener** that listens to incoming message from provider
timeout	a integer specifying the subscription timeout in ms

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

SoaConsumer::subscribeToDataAsync

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&)

SoaConsumer::unsubscribeToDataAsync

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr, int32_t)

virtual void fnv::soa::framework::SoaConsumer::subscribeToDataAsync (SoaClientEndpoint::SharedPtr *dataEndpoint*, SoaDataListener & *dataListener*, SoaActionResultListener< SoaSubscribeToDataContext > & *actionResultListener*) [pure virtual]

Subscribes to provider specified by dataEndpoint. The message is received by the listener and the subscription status is received by the status callback. The message listener is guaranteed to not be called before the status callback is called and returns. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

dataEndpoint	a **SoaClientEndpoint** specifying the data endpoint to subscribe to
dataListener	a **SoaDataListener** that listens to incoming message from provider
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

See also:

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&)

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&, int32_t)

SoaConsumer::unsubscribeToDataAsync

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::subscribeToServiceStatusUpdate (SoaClientEndpoint::SharedPtr *serviceEndpoint*, SoaServiceStatusListener & *listener*)[pure virtual]

This method creates an explicit service status subscription and blocks until Requests the service status from the Service Directory AND creates an explicit subscription for subsequent status updates such that all status values are provided to the status listener. The explicit subscription will also ensure the cached status is always valid, unless an unsubscribe status method is called. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in
listener	a **SoaServiceStatusListener** to listen for a service status update

See also:

setTimeout

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

Returns:

a SoaErrorCode to specify the type of error occurred

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::subscribeToServiceStatusUpdate (SoaClientEndpoint::SharedPtr *serviceEndpoint*, SoaServiceStatusListener & *listener*, int32_t *timeout*)[pure virtual]

This method creates an explicit service status subscription and blocks until Requests the service status from the Service Directory AND creates an explicit subscription for subsequent status updates such that all status values are provided to the status listener. The explicit subscription will also ensure the cached status is always valid, unless an unsubscribe status method is called. The timeout provided is for this subscription request only and will not impact the blanket timeout. The timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in
listener	a **SoaServiceStatusListener** to listen for a service status update
timeout	a integer specifying a timeout to the subscription action in ms

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual void fnv::soa::framework::SoaConsumer::subscribeToServiceStatusUpdateAsync (SoaClientEndpoint::SharedPtr *serviceEndpoint*, SoaServiceStatusListener & *listener*, SoaActionResultListener< SoaSubscribeToServiceStatusContext > & *actionResultListener*)[pure virtual]

Requests the service status from the Service Directory AND creates an explicit subscription for subsequent status updates such that all status values are provided to the status listener. The explicit subscription will also ensure the cached status is always valid, unless an unsubscribe status method is called. The status listener is guaranteed to not be called before the status callback is called and returns. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the subscription has failed (the framework guarantees no subscription is created if a timeout occurs)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service interested in
listener	a **SoaServiceStatusListener** to listen for a service status update
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::unsubscribeToData (SoaClientEndpoint::SharedPtr *dataEndpoint*, SoaDataListener & *dataListener*)[pure virtual]

Unsubscribes to a data endpoint, returns only when done unsubscribing or an error has occurred The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The service listener may still get called.)

Parameters:

dataEndpoint	a **SoaClientEndpoint** specifying the data endpoint no longer interested in
dataListener	The listener provided in a subscribeToData method.

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

setTimeout

SoaConsumer::subscribeToDataAsync

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&)

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&, int32_t)

SoaConsumer::unsubscribeToDataAsync

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::unsubscribeToData (SoaClientEndpoint::SharedPtr *dataEndpoint*, SoaDataListener & *dataListener*, int32_t *timeout*)[pure virtual]

Unsubscribes to a data endpoint, returns only when done unsubscribing, when error has occurred, or timeout is reached. The timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The service listener may still get called.)

Parameters:

dataEndpoint	a **SoaClientEndpoint** specifying the data endpoint no longer interested in
dataListener	The listener provided in a subscribeToData method.
timeout	a integer specifying a timeout to the unsubscribe action in ms

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

SoaConsumer::subscribeToDataAsync

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&)

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&, int32_t)

SoaConsumer::unsubscribeToDataAsync

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr)

virtual void fnv::soa::framework::SoaConsumer::unsubscribeToDataAsync (SoaClientEndpoint::SharedPtr *dataEndpoin*, SoaDataListener & *dataListener*, SoaActionResultListener< SoaUnsubscribeToDataContext > & *actionResultListener*) [pure virtual]

Unsubscribes to a data endpoint. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The service listener may still get called.)

Parameters:

dataEndpoint	a **SoaClientEndpoint** specifying the data endpoint no longer interested in
dataListener	The listener provided in a subscribeToData method.
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

See also:

SoaConsumer::subscribeToDataAsync

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&)

SoaConsumer::subscribeToData(SoaClientEndpoint::SharedPtr, SoaSubscribeListener&, int32_t)

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToData(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::unsubscribeToServiceStatusUpdate (SoaClientEndpoint::SharedPtr *serviceEndpoint*)[pure virtual]

Unsubscribes to the status update of a specific service, returns only when done unsubscribing or when an error has occurred This method cancels the implicit subscription and immediately makes the cached status invalid (i.e. unavailable) regardless of the success or failure of the unsubscribe of an explicit subscription. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The status update listener may still get called.)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service no longer interested in

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

setTimeout

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

virtual SoaErrorCode fnv::soa::framework::SoaConsumer::unsubscribeToServiceStatusUpdate (SoaClientEndpoint::SharedPtr *serviceEndpoint*, int32_t *timeout*)[pure virtual]

Unsubscribes to the status update of a specific service, returns only when done unsubscribing or when an error has occurred This method cancels the implicit subscription and immediately makes the cached status invalid (i.e. unavailable) regardless of the success or failure of the unsubscribe of an explicit subscription. The timeout provided is for this subscription request only and will not impact the blanket timeout. The timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The status update listener may still get called.)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service no longer interested in
timeout	an integer specifying a timeout to the unsubscribe action in ms

Returns:

a SoaErrorCode to specify the type of error occurred

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdateAsync

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

virtual void fnv::soa::framework::SoaConsumer::unsubscribeToServiceStatusUpdateAsync (SoaClientEndpoint::SharedPtr *serviceEndpoint*, SoaActionResultListener< SoaUnsubscribeToServiceStatusContext > & *actionResultListener*)[pure virtual]

Unsubscribes to the status update of a specific service. This method cancels the implicit subscription and immediately makes the cached status invalid (i.e. unavailable) regardless of the success or failure of the unsubscribe of an explicit subscription. The blanket timeout may expire resulting in an error code indicating a timeout occurred, in which case the unsubscribe can be assumed to have failed. (i.e. The status update listener may still get called.)

Parameters:

serviceEndpoint	a **SoaClientEndpoint** specifying the service no longer interested in
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

See also:

SoaConsumer::requestServiceStatusCached

SoaConsumer::requestServiceStatusAsync

SoaConsumer::requestServiceStatus

SoaConsumer::subscribeToServiceStatusUpdateAsync

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&)

SoaConsumer::subscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, SoaServiceStatusListener&, int32_t)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr)

SoaConsumer::unsubscribeToServiceStatusUpdate(SoaClientEndpoint::SharedPtr, int32_t)

fnv::soa::framework::SoaMessage Class Reference

#include <soa_message.hpp>

Public Types

- enum **TransactionError** { **TransactionError::NO_ERROR** = SoaMessageProto_TransactionError_NO_ERROR, **TransactionError::GENERAL_ERROR** = SoaMessageProto_TransactionError_GENERAL_ERROR, **TransactionError::VERSION_MISMATCH** = SoaMessageProto_TransactionError_VERSION_MISMATCH }
- typedef shared_ptr< **SoaMessage** > **SharedPtr**
- **SoaMessage** ()
- virtual **~SoaMessage** ()
- void **setPayload** (const std::string &rawPayload)
- SoaErrorCode **setPayload** (const ::google::protobuf::Message &payload)
- void **setDataEndpoint** (**SoaClientEndpoint::SharedPtr** endpoint)
- **SoaClientEndpoint::SharedPtrgetDataEndpoint** () const
- void **setConsumerEndpoint** (**SoaClientEndpoint::SharedPtr** endpoint)
- **SoaClientEndpoint::SharedPtrgetConsumerEndpoint** () const
- void **setProviderEndpoint** (**SoaClientEndpoint::SharedPtr** id)
- **SoaClientEndpoint::SharedPtrgetProviderEndpoint** () const
- void **setCommandId** (const string &id)
- string **getCommandId** () const

- void **setServiceVersion** (const string &version)
- string **getServiceVersion** () const
- void **setTransactionId** (const string &id)
- string **getTransactionId** () const
- SoaErrorCode **parsePayload** (::google::protobuf::Message &payloadObject)
- std::string **getRawPayload** ()
- **SoaActionResult**< std::string >::**SharedPtrserializeAsString** ()
- void **setTransactionError** (**TransactionError** error)
- **TransactionErrorgetTransactionError** ()
- void **setTransactionErrorDetails** (const std::string &errorDetails)
- std::string **getTransactionErrorDetails** ()
- static **SoaMessage::SharedPtrcreateRequestMessage** (**SoaClientEndpoint::SharedPtr** consumerEndpoint, **SoaClientEndpoint::SharedPtr** p roviderendpoint, const string &commandID, const string &version, const string &transactionID)
- static **SoaMessage::SharedPtrcreateResponseMessage** (**SoaMessage::SharedPtr** requestMessage, const string &version)
- static **SoaActionResult**< **SoaMessage** >::**SharedPtrparseFromString** (const std::string &message)

- const std::string& **SoaMessage::getSenderEcuId()** const Note: every SoaMessage sent to the broker (including those sent via an IPC/MQTT gateway) include the sender's ECU ID (ex ecg, tcu, synch, lxcluster)
- const std::string& **SoaMessage::getSenderUid()** const Note: every SoaMessage sent to the broker (including those sent via an IPC/MQTT gateway) include the sender's UID (the getter returns the integer value as a std::string)

Public Member Functions

Static Public Member Functions

Detailed Description

SOA message envelop class. Contains SOA message header and a payload.

Definition at line 45 of file soa_message.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaMessage> fnv::soa::framework::SoaMessage::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 68 of file soa_message.hpp.

Member Enumeration Documentation

enum fnv::soa::framework::SoaMessage::TransactionError[strong]

Enum to report transaction error if any. An example can be version mismatch between RPC provider and consumer.

Enumerator:

NO_ERROR	No error
GENERAL_ERROR	General error, allows to report errors not listed in the enum
VERSION_MISMATCH	Version mismatch between RPC provider and consumer.

Definition at line 51 of file soa_message.hpp.

Constructor & Destructor Documentation

fnv::soa::framework::SoaMessage::SoaMessage ()

SOAMessage constructor

virtual fnv::soa::framework::SoaMessage::~SoaMessage ()[virtual]

SOAMessage destructor

Member Function Documentation

static SoaMessage::SharedPtr fnv::soa::framework::SoaMessage::createRequestMessage (SoaClientEndpoint:: SharedPtr *consumerEndpoint*, SoaClientEndpoint::SharedPtr *providerendpoint*, const string & *commandID*, const string & *version*, const string & *transactionID*)[static]

Creates an RPC request message

Parameters:

consumerEndpoint	The service consumer endpoint
providerendpoint	The service provider endpoint
commandID	The command ID
version	The service version
transactionID	The transaction ID

static SoaMessage::SharedPtr fnv::soa::framework::SoaMessage::createResponseMessage (SoaMessage::SharedPtr *requestMessage*, const string & *version*)[static]

Creates a response message using transaction context provided in the request

Parameters:

requestMessage	The request message version The service provider version.

Returns:

The response message

string fnv::soa::framework::SoaMessage::getCommandId () const

Gets service command ID

Returns:

: The ID

SoaClientEndpoint::SharedPtr fnv::soa::framework::SoaMessage::getConsumerEndpoint () const

Gets consumer endpoint

Returns:

a **SoaClientEndpoint** spcifying the consumer endpoint

SoaClientEndpoint::SharedPtr fnv::soa::framework::SoaMessage::getDataEndpoint () const

Gets data endpoint. "Data endpoint" message field should be used to deliver data multicast endpoint to a listener.

Returns:

The endpoint

SoaClientEndpoint::SharedPtr fnv::soa::framework::SoaMessage::getProviderEndpoint () const

Gets service provider endpoint

Returns:

The endpoint

std::string fnv::soa::framework::SoaMessage::getRawPayload ()

Get the raw payload bytes.

Returns:

the raw payload bytes wrapped by a string object for convenience

string fnv::soa::framework::SoaMessage::getServiceVersion () const

Gets the service version

Returns:

: The service version

TransactionError fnv::soa::framework::SoaMessage::getTransactionError ()

Returns transaction error

Returns:

The error

std::string fnv::soa::framework::SoaMessage::getTransactionErrorDetails ()

Returns transaction error details

Returns:

The error description

string fnv::soa::framework::SoaMessage::getTransactionId () const

Gets transaction id.

Returns:

: The message id. Allows to resquest/response mapping.

const std::string& SoaMessage::getSenderEcuId() const

Get the ECU ID of the sender of the message. Ex ecg, or tcu, or sync, or lxcluster, or ID of future supported ECU

Returns:

A const reference to a string indicating the sender's ECU identifier.

One use for this info is when combined with the sender's UID, the client can map the sender to the service. Once subscribed to LWT messages, the client can learn if the service provider has crashed by comparing these values with LWT message fields.

The combination of the ECU ID and UID of the sender uniquely identifies the sender client in the FNV2 vehicle's domain

const std::string& SoaMessage::getSenderUid() const

Get the operating system UID of the sender of the message. The UID is an integer value, but this interface returns a const reference to a string representing the integer value. (Ex. UID 60 is returned as "60") Valid UIDs are greater than or equal to zero (0)

Returns:

A const reference to a string representing the the sender's UID identifier.

One use for this info is when combined with the sender's UID, the client can map the sender to the service. Once subscribed to LWT messages, the client can learn if the service provider has crashed by comparing these values with LWT message fields.

The combination of the ECU ID and UID of the sender uniquely identifies the sender client in the FNV2 vehicle's domain

static SoaActionResult<SoaMessage>::SharedPtr fnv::soa::framework::SoaMessage::parseFromString (const std::string & *message*)[static]

Parses a message from raw protobuf string

Parameters:

message	The raw protobuf string

Returns:

The action result containing action error code and **SoaMessage** object.

SoaErrorCode fnv::soa::framework::SoaMessage::parsePayload (::google::protobuf::Message & *payloadObject*)

Convenience method to parse a Google protocol message payload.

Parameters:

payloadObject	Google protobuf object representing the payload
version	API version

Returns:

Error code

SoaActionResult<std::string>::SharedPtr fnv::soa::framework::SoaMessage::serializeAsString ()

Returns an action result including the action status and serialized message

void fnv::soa::framework::SoaMessage::setCommandId (const string & *id*)

Sets service command id

Parameters:

id	The id

void fnv::soa::framework::SoaMessage::setConsumerEndpoint (SoaClientEndpoint::SharedPtr *endpoint*)

Sets consumer endpoint

Parameters:

endpoint	a **SoaClientEndpoint** spcifying the consumer endpoint

void fnv::soa::framework::SoaMessage::setDataEndpoint (SoaClientEndpoint::SharedPtr *endpoint*)

Sets data endpoint. "Data endpoint" message field should be used to deliver data multicast endpoint to a listener.

Parameters:

endpoint	The endpoint

void fnv::soa::framework::SoaMessage::setPayload (const std::string & *rawPayload*)

Set payload from a raw string of bytes wrapped by the string type

Parameters:

rawPayload	A std::string wrapped binary byte sequence for the message payload

SoaErrorCode fnv::soa::framework::SoaMessage::setPayload (const ::google::protobuf::Message & *payload*)

Set payload from a Google Protocol Buffer object

Parameters:

payload	A protocol buffer object for the message payload

void fnv::soa::framework::SoaMessage::setProviderEndpoint (SoaClientEndpoint::SharedPtr *id*)

Sets service provider endpoint.

Parameters:

The	endpoint

void fnv::soa::framework::SoaMessage::setServiceVersion (const string & *version*)

Sets service version The version

Parameters:

version	

void fnv::soa::framework::SoaMessage::setTransactionError (TransactionError *error*)

Use the method to set an error if the transaction cannot be completed

Parameters:

error	The error

void fnv::soa::framework::SoaMessage::setTransactionErrorDetails (const std::string & *errorDetails*)

Use the methods to set any additional transaction error details

Parameters:

errorDetails	The details

void fnv::soa::framework::SoaMessage::setTransactionId (const string & *id*)

Sets transaction ID

Parameters:

id	The ID

fnv::soa::framework::SoaMessageManager Class Reference

#include <soa_message_manager.hpp>

Public Types

- typedef shared_ptr< **SoaMessageManager** > **SharedPtr**
- virtual SoaErrorCode **connect** ()=0

Public Member Functions

connect - Connect the message manager to the transport layer

- virtual SoaErrorCode **disconnect** ()=0

disconnect - Disconnect the message manager from the transport layer

- virtual **~SoaMessageManager** ()

Destructor.

Static Public Member Functions

- static **SoaMessageManager::SharedPtr createMessageManagerMqtt** (const **SoaConnectionOptions** &options)

createMessageManagerMqtt - MQTT message manager factory method

- static **SoaMessageManager::SharedPtr createMessageManagerIpc** ()

createMessageManagerIpc - IPC message manager factory method

Detailed Description

The SOA client must instantiate an object of this class to communicate with other clients in the system. There are two choices for the message transport mechanism: MQTT and IPC. This class includes factory methods for creating the message manager object needed by the SOA client. The factory methods return a object of this type. This class represents an opaque reference (empty interface) to the message manager object, whose private methods are used by the SOA Middleware layer.

A SOA client does not need to instantiate more than one message manager object per transport type for a set of connection options.

Definition at line 51 of file soa_message_manager.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaMessageManager> fnv::soa::framework::SoaMessageManager::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 56 of file soa_message_manager.hpp.

Member Function Documentation

static SoaMessageManager::SharedPtr fnv::soa::framework::SoaMessageManager::createMessageManagerIpc ()[static]

createMessageManagerIpc - IPC message manager factory method

Returns:

a pointer to a **SoaMessageManager**

static SoaMessageManager::SharedPtr fnv::soa::framework::SoaMessageManager::createMessageManagerMqtt (const SoaConnectionOptions & *options*)[static]

createMessageManagerMqtt - MQTT message manager factory method

Parameters:

options	Reference to a connection option object specifying options to be used for the connection

Returns:

a pointer to a **SoaMessageManager**

The documentation for this class was generated from the following file:

- C:/Users/frisache/Documents/GitHub/FNV-SOA/soa/include/cpp/framework/soa_message_manager.hpp

fnv::soa::framework::SoaMessageOptions Class Reference

#include <soa_message_options.hpp>

Public Types

- typedef shared_ptr< **SoaMessageOptions** > **SharedPtr**
- **SoaMessageOptions** ()
- **SoaMessageOptions** (int32_t qos, bool retained)
- int32_t **getQos** () const

Public Member Functions

Quality of service option getter.

- void **setQos** (int32_t qos)

Quality of service option setter.

- bool **isRetained** () const

Boolean getter for retained setting.

- void **setRetained** (bool retained)

Setter for retained option.

Detailed Description

Container class for message send options

Definition at line 35 of file soa_message_options.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaMessageOptions> fnv::soa::framework::SoaMessageOptions::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 44 of file soa_message_options.hpp.

Constructor & Destructor Documentation

fnv::soa::framework::SoaMessageOptions::SoaMessageOptions ()[inline]

constructor

Definition at line 49 of file soa_message_options.hpp.

fnv::soa::framework::SoaMessageOptions::SoaMessageOptions (int32_t *qos*, bool *retained*)[inline]

that sets qos and retained with given value

Parameters:

qos	a integer specifying the desired message QoS
retained	a boolean specifying if this message should be retained

Definition at line 59 of file soa_message_options.hpp.

Member Function Documentation

int32_t fnv::soa::framework::SoaMessageOptions::getQos () const[inline]

Quality of service option getter.

Returns:

The QOS (Quality of service) value

Definition at line 68 of file soa_message_options.hpp.

bool fnv::soa::framework::SoaMessageOptions::isRetained () const[inline]

Boolean getter for retained setting.

Returns:

true if retained, false otherwise

Definition at line 86 of file soa_message_options.hpp.

void fnv::soa::framework::SoaMessageOptions::setQos (int32_t *qos*)[inline]

Quality of service option setter.

Parameters:

qos	The QOS (Quality of service) value to set

Definition at line 77 of file soa_message_options.hpp.

void fnv::soa::framework::SoaMessageOptions::setRetained (bool *retained*)[inline]

Setter for retained option.

Parameters:

retained	If the message is to be retained or not

Definition at line 95 of file soa_message_options.hpp.

fnv::soa::framework::SoaProvider Class Reference

Public Types

- typedef shared_ptr< **SoaProvider** > **SharedPtr**

Public Member Functions

- virtual void **setTimeout** (int32_t timeout)=0
- virtual int32_t **getTimeout** ()=0
- virtual void **publishMessageAsync** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** msg, **SoaActionResultListener**< **SoaPublishMessageContext** > &actionResultListener)=0
- virtual SoaErrorCode **publishMessage** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** msg)=0
- virtual SoaErrorCode **publishMessage** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** msg, int32_t timeout)=0
- virtual void **registerServiceAsync** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceRequestListener** &requestListener, **SoaActionResultListener**< **SoaRegisterServiceContext** > &actionResultListener)=0
- virtual SoaErrorCode **registerService** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceRequestListener** &requestListener)=0
- virtual SoaErrorCode **registerService** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaServiceRequestListener** &requestListener, int32_t timeout)=0
- virtual void **unregisterServiceAsync** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, **SoaActionResultListener**< **SoaUnregisterServiceContext** > &actionResultListener)=0
- virtual SoaErrorCode **unregisterService** (**SoaClientEndpoint::SharedPtr** serviceEndpoint)=0
- virtual SoaErrorCode **unregisterService** (**SoaClientEndpoint::SharedPtr** serviceEndpoint, int32_t timeout)=0
- virtual void **remoteCallResponseAsync** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** response, **SoaActionResultListener**< **SoaPublishMessageContext** > &actionResultListener)=0
- virtual SoaErrorCode **remoteCallResponse** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** response)=0
- virtual SoaErrorCode **remoteCallResponse** (**SoaClientEndpoint::SharedPtr** endpoint, **SoaMessage::SharedPtr** response, int32_t timeout)=0
- virtual **~SoaProvider** ()

Static Public Member Functions

- static **SharedPtr createSoaProvider** (**SoaMessageManager::SharedPtr** msgMan)

Detailed Description

Definition at line 42 of file soa_provider.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaProvider> fnv::soa::framework::SoaProvider::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 47 of file soa_provider.hpp.

Constructor & Destructor Documentation

virtual fnv::soa::framework::SoaProvider::~SoaProvider ()[inline] , [virtual]

Destructor

Definition at line 185 of file soa_provider.hpp.

Member Function Documentation

static SharedPtr fnv::soa::framework::SoaProvider::createSoaProvider (SoaMessageManager::SharedPtr *msgMan*) [static]

Factory

Parameters:

msgMan	a **SoaMessageManager** that delivers and receives messages

virtual int32_t fnv::soa::framework::SoaProvider::getTimeout ()[pure virtual]

Gets the common timeout value which applies to the blocking methods, and to the maximum time before an action status callback is called for non-blocking methods.

Returns:

the timeout value in milliseconds

virtual SoaErrorCode fnv::soa::framework::SoaProvider::publishMessage (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *msg*) [pure virtual]

Publishes a message and wait until message is delivered or blanket timeout is reached

Parameters:

endpoint	a **SoaClientEndpoint** specifying the consumer or data endpoint to publish to
msg	a **SoaMessage** containing the information to be published to the consumer

Returns:

a SoaErrorCode specifying the type of error occurred during publish if any

virtual SoaErrorCode fnv::soa::framework::SoaProvider::publishMessage (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *msg* , int32_t *timeout*) [pure virtual]

Publishes a message and wait until message is delivered or timeout is reached

Parameters:

endpoint	a **SoaClientEndpoint** specifying the consumer or data endpoint to publish to
msg	a **SoaMessage** containing the information to be published to the consumer
timeout	an integer specifying the timeout for publish action

Returns:

a SoaErrorCode specifying the type of error occurred during publish if any

virtual void fnv::soa::framework::SoaProvider::publishMessageAsync (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *msg* , SoaActionResultListener< SoaPublishMessageContext > & *actionResultListener*) [pure virtual]

Publishes a message and get the publish status in the listener

Parameters:

msg	a **SoaMessage** containing information to be published to the consumer
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

virtual SoaErrorCode fnv::soa::framework::SoaProvider::registerService (SoaClientEndpoint::SharedPtr *serviceEndpoint* , SoaServiceRequestListener & *requestListener*) [pure virtual]

Subscribes to requests from consumers for remote call services. This method will bind a service request listener object to the endpoint of the service. This method returns when registration is complete, when the blanket timeout is reached or when an error occurs

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be registered
requestListener	A listener object that implements a callback method to be called when a request is received

Returns:

a SoaErrorCode specifying the type of error occurred during register if any

virtual SoaErrorCode fnv::soa::framework::SoaProvider::registerService (SoaClientEndpoint::SharedPtr *serviceEndpoint* , SoaServiceRequestListener & *requestListener* , int32_t *timeout*) [pure virtual]

Subscribes to requests from consumers for remote call services. This method will bind a service request listener object to the endpoint of the service. This method returns when registration is complete, when the timeout is reached or when an error occurs

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be registered
requestListener	A listener object that implements a callback method to be called when a request is received
timeout	an integer specifying the timeout for register action

Returns:

a SoaErrorCode specifying the type of error occurred during register if any

virtual void fnv::soa::framework::SoaProvider::registerServiceAsync (SoaClientEndpoint::SharedPtr *serviceEndpoint* , SoaServiceRequestListener & *requestListener* , SoaActionResultListener< SoaRegisterServiceContext > & *actionResultListener*) [pure virtual]

Subscribes to requests from consumers for remote call services. This method will bind a service request listener object to the endpoint of the service. This method will not block, and provides action status via a callback.

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be registered
requestListener	A listener object that implements a callback method to be called when a request is received
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

virtual SoaErrorCode fnv::soa::framework::SoaProvider::remoteCallResponse (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *response*) [pure virtual]

Response to remote call request received, does not return until response sent This method returns when remote call response is complete, when the blanket timeout is reached or when an error occurs

Parameters:

response	a **SoaMessage** as response to request with endpoint specified by consumer in request message

Returns:

a SoaErrorCode specifying the error occurred during response action if any

virtual SoaErrorCode fnv::soa::framework::SoaProvider::remoteCallResponse (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *response* , int32_t *timeout*) [pure virtual]

Response to remote call request received, does not return until response sent or timeout reached This method returns when remote call response is complete, when the timeout is reached or when an error occurs

Parameters:

response	a **SoaMessage** as response to request with endpoint specified by consumer in request message
timeout	a integer specifying the timeout for response action in ms

Returns:

a SoaErrorCode specifying the error occurred during response action if any

virtual void fnv::soa::framework::SoaProvider::remoteCallResponseAsync (SoaClientEndpoint::SharedPtr *endpoint* , SoaMessage::SharedPtr *response* , SoaActionResultListener< SoaPublishMessageContext > & *actionResultListener*) [pure virtual]

Response to remote call request received and monitor response status via callback

Parameters:

response	a **SoaMessage** as response to request with endpoint specified by consumer in request message
actionResultListener	a **SoaActionResultListener** for the status of this response option

virtual void fnv::soa::framework::SoaProvider::setTimeout (int32_t *timeout*) [pure virtual]

Sets the common timeout for the blocking methods, and the maximum time before an action status callback is called for non-blocking methods.

Parameters:

timeout	The timeout in milliseconds

virtual SoaErrorCode fnv::soa::framework::SoaProvider::unregisterService (SoaClientEndpoint::SharedPtr *serviceEndpoint*) [pure virtual]

Unsubscribes to requests from consumers for remote call services This method returns when deregistration is complete, when the blanket timeout is reached or when an error occurs

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be unregistered

Returns:

a SoaErrorCode specifying the type of error occurred during unregister if any

virtual SoaErrorCode fnv::soa::framework::SoaProvider::unregisterService (SoaClientEndpoint::SharedPtr *serviceEndpoint* , int32_t *timeout*) [pure virtual]

Unsubscribes to requests from consumers for remote call services This method returns when unregistration is complete, when the blanket timeout is reached or when an error occurs

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be unregistered
timeout	an integer specifying the timeout for unregister action

Returns:

a SoaErrorCode specifying the type of error occurred during unregister if any

virtual void fnv::soa::framework::SoaProvider::unregisterServiceAsync (SoaClientEndpoint::SharedPtr *serviceEndpoint* , SoaActionResultListener< SoaUnregisterServiceContext > & *actionResultListener*) [pure virtual]

Unregisters service from Service Directory and unsubscribes to requests from consumers for remote call services This method will not block, and provides action status via a callback.

Parameters:

serviceEndpoint	a **SoaClientEndpoint** that specifies the service to be unregistered
actionResultListener	A callback object that implements a callback method which is guaranteed to be called when the status of this action is known

fnv::soa::framework::SoaServiceDirectoryManager Class Reference

#include <soa_service_directory_manager.hpp>

Public Types

- typedef shared_ptr< **SoaServiceDirectoryManager** > **SharedPtr**
- virtual SoaErrorCode **start** ()=0
- virtual SoaErrorCode **stop** ()=0
- virtual SoaErrorCode **registerSystemService** (**SoaServiceContract::SharedPtr** contract, SoaServiceStatus status)=0
- virtual **~SoaServiceDirectoryManager** ()
- static **SharedPtr create** (**SoaMessageManager::SharedPtr** messageManager, **SoaClientEndpoint::SharedPtr** endpoint)

Public Member Functions

Static Public Member Functions

Detailed Description

Local SOA Service Directory manager. The class is a central point for clients to send requests to Service Directory Service. The class will cache the latest service status and contract to avoid extra SOA traffic.

Definition at line 46 of file soa_service_directory_manager.hpp.

Member Typedef Documentation

typedef shared_ptr<SoaServiceDirectoryManager> fnv::soa::framework::SoaServiceDirectoryManager::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 51 of file soa_service_directory_manager.hpp.

Constructor & Destructor Documentation

virtual fnv::soa::framework::SoaServiceDirectoryManager::~SoaServiceDirectoryManager ()[inline], [virtual]

destructor

Definition at line 86 of file soa_service_directory_manager.hpp.

Member Function Documentation

static SharedPtr fnv::soa::framework::SoaServiceDirectoryManager::create (SoaMessageManager::SharedPtr *messageManager*, SoaClientEndpoint::SharedPtr *endpoint*)[static]

Factory method to create an object of this type

Parameters:

messageManager	
endpoint	

Returns:

a shared pointer to a **SoaServiceDirectoryManager** object

virtual SoaErrorCode fnv::soa::framework::SoaServiceDirectoryManager::registerSystemService (SoaServiceContract::SharedPtr *contract*, SoaServiceStatus *status*)[pure virtual]

Use this API to statically register system services you are going to access in you code. System service assumption: It is a well knows service with the contract defined in a header file THe service has global lifecycle

Parameters:

contract	The contract

Returns:

The error code

virtual SoaErrorCode fnv::soa::framework::SoaServiceDirectoryManager::start ()[pure virtual]

Starts the manager: Makes the manager a SOA client listening to the endpoint provided in the constructor

Returns:

virtual SoaErrorCode fnv::soa::framework::SoaServiceDirectoryManager::stop ()[pure virtual]

Stops the manager from listening to the endpoint

fnv::soa::framework::SoaClientEndpoint Class Reference

Public Types

- typedef std::shared_ptr< **SoaClientEndpoint** > **SharedPtr**
- virtual **~SoaClientEndpoint** ()
- virtual size_t **hashcode** () const =0
- virtual const std::string & **toString** () const =0
- virtual bool **operator==** (const **SoaClientEndpoint** &other) const =0
- static **SharedPtr createClientEndpointMqtt** (const std::string &endpoint)
- static **SharedPtr createClientEndpointIpc** (const std::string &endpoint)
- **SoaClientEndpoint** ()

Public Member Functions

Static Public Member Functions

Protected Member Functions

Constructor.

Detailed Description

Definition at line 35 of file soa_client_endpoint.hpp.

Member Typedef Documentation

typedef std::shared_ptr<SoaClientEndpoint> fnv::soa::framework::SoaClientEndpoint::SharedPtr

Typedef of scoped shared pointer type to this class

Definition at line 40 of file soa_client_endpoint.hpp.

Constructor & Destructor Documentation

virtual fnv::soa::framework::SoaClientEndpoint::~SoaClientEndpoint ()[inline], [virtual]

Destructor

Definition at line 57 of file soa_client_endpoint.hpp.

Member Function Documentation

static SharedPtr fnv::soa::framework::SoaClientEndpoint::createClientEndpointIpc (const std::string & *endpoint*)[static]

IPC endpoint factory method

Returns:

a shared pointer to an IPC endpoint object

static SharedPtr fnv::soa::framework::SoaClientEndpoint::createClientEndpointMqtt (const std::string & *endpoint*) [static]

MQTT endpoint factory method

Returns:

a shared pointer to an MQTT endpoint object

virtual size_t fnv::soa::framework::SoaClientEndpoint::hashcode () const[pure virtual]

Returns the endpoint hashcode

Returns:

The hashcode

virtual bool fnv::soa::framework::SoaClientEndpoint::operator== (const SoaClientEndpoint & *other*) const[pure virtual]

Comparison for equality

Parameters:

other	the other endpoint to compare to this endpoint

Returns:

true if this endpoint equals the other endpoint

virtual const std::string& fnv::soa::framework::SoaClientEndpoint::toString () const[pure virtual]

Returns the endpoint as a string

Returns:

a reference to the string representation of the endpoint

SOA Error Code

```
/*
 * SOA action error code
 */
enum class SoaErrorCode {
        NO_ERROR,
        ERROR_NOT_IMPLEMENTED,                          //The feature is not implemented yet. This may be related
to an early drop with partial implementation
        ERROR_TIMEOUT,                                          // The function call timed out for a variety of
reason as disconnection with the broker, service not running, etc.
        ERROR_PARAM_MISSING,                          // one or several parameters are missing
        ERROR_PARAM_BAD,                                  // One or several parameters have a bad value
        ERROR_PARAM_NULL_REFERENCE,
        ERROR_INVALID_TIMEOUT_PARAM,         // the timeout parameter specified is invalid e.g. <0
        ERROR_INVALID_ENDPOINT_PARAM,         // The endpoint is not defined properly e.g. empty string
        ERROR_INVALID_MESSAGE_PARAM,         //if the message pointer is null
        ERROR_INVALID_CALLBACK_PARAM,         //if the listener param is a null reference
        ERROR_MALFORMED_OBJECT,          // constructor set flag that construction failed
        ERROR_UNKNOWN_EXCEPTION,                // unknown error
        ERROR_RESOURCE_UNAVAIL,                         // service status. Service is not running for instance
        ERROR_NOT_CONNECTED,                        // client is not connected to the transport layer (MQTT)
        ERROR_CONNECT_FAIL,                                 //returned if connection with MQTT broker failed
        ERROR_DISCONNECT_FAIL,                          // returned if disconnection with the broker failed
        ERROR_TOPIC_PATTERN_ALREADY_REGISTERED,  // unique subscribe failure
        ERROR_LISTENER_ALREADY_REGISTERED_TO_TOPIC_PATTERN,  // non-unique subscribe failure
        ERROR_SOA_MESSAGE_SERIALIZATION_FAILED, // General SOAMessage serialization error
        ERROR_ILLEGAL_CALL_THREAD, //When Synchronous methods are invoked from callback thread or worker thread
        ERROR_SOA_MESSAGE_PARSING_FAILED, // General SOAMessage parsing error
        ERROR_CANNOT_CREATE_CLIENT,                //returned if the MQTT client object creation fails
        ERROR_WRONG_STATE,                                 // try to connect a client already connected
        ERROR_EMPTY_TOPIC,
        ERROR_EMPTY_TOPIC_PATTERN,
        ERROR_PUBLISH_FAILED,
        ERROR_SUBSCRIBE_FAILED,
        ERROR_UNSUBSCRIBE_FAILED,
        ERROR_TOPIC_NO_SUBSCRIBED,
        ERROR_LISTENER_NOT_BOUND,
        ERROR_CLIENT_ID_NOT_SET,
        ERROR_DUPLICATE_COMMAND_HANDLER,        //
        ERROR_FAILED_SETTING_CALLBACKS,
        ERROR_ALREADY_MAPPED,   // tried to add a unique listener again
        ERROR_UNIQUENESS_MISMATCH,
        ERROR_INTERNAL_ERROR,
        ERROR_WILDCARD_FIXED_PATTERN_OVERLAP, // a fixed pattern overlaps with a wildcard pattern
        ERROR_SOA_MESSAGE_FACTORY_ERROR,
        /**
         * A command is not supported by the selected command handler
         */
        ERROR_HANDLER_COMMAND_NOT_SUPPORTED,
        ERROR_COMMAND_HANDLER_ERROR,
        ERROR_INITIALIZATION_ERROR,
        ERROR_NOT_INITIALIZED,
        ERROR_BAD_STATE,
        ERROR_FATAL,
        ERROR_TRANSACTION_FAILED,
        ERROR_RESPONSE_DATA_BAD,
        ERROR_REQUEST_DATA_BAD,

        // When adding new error codes, please also add them to ECG/soa/src/cpp/soa_error_code_utils.cpp
        ERROR_LAST_ONE__PUT_ALL_BEFORE_THIS
};
```

Optimizations to be aware of ...

SOAClientEndpoint::operator ==

The SOAClientEndpoint is a container class for a portion of the MQTT topic (when used with MQTT). Continually doing a string comparison to check equality on an endpoint is not optimal. Many endpoints being compared by client code may have common prefix text, such as /vim/ or /tcu/. If using chained "if/else if" blocks to discriminate events based on endpoints, mismatches will not be detected until after common prefix text has been iterated over if the string lengths are equal. An attempt to optimize these comparisons has been done by storing a hashcode of the endpoint string in the SOAClientEndpoint object. The hashcode is calculated in the constructor of the SOAClientEndpoint object using an STL hashcode method. The operator == implementation on SOAClientEndpoint uses the integer hashcode in an optimized equality comparison. Inequality of two client endpoint objects is guaranteed if the hashcodes are different. However, equality is not guaranteed if the hashcodes are equal. The implementation therefore also compares length of the strings. If this is equal, it also does a string comparison on the last 3 characters in each endpoint. The possibility that the endpoints are not equal if the check confirms equality is extremely unlikely.

```
bool SOAClientEndpointMqtt::operator==(const SOAClientEndpoint& other) const {
    // NOTE: toString() is a 0-time pseudo-operation, so toString().size() is instantaneous
    // hashcode() returns a previously calculated value
        if(hashcode() == other.hashcode() && toString().size() == other.toString().size()) {
                // very, very low chance that the strings are not equal, so let's compare last three characters
(max)
                // we use the last three, because a lot of different endpoints will have the same first three
characters
                size_t size = toString().size();
                size_t sampleSize = size > 3 ? 3 : size;
                // checking for size == 0 should not be needed but avoids possible exception in string
comparison
                return (size == 0) || (toString().substr(size-sampleSize, sampleSize) == other.toString().substr
(size-sampleSize, sampleSize));
        }
        return false;
}
```

FAQ

What is a Client Endpoint vs Topic?

NOTE: This section is unedited text that needs feedback leading to more definitive wording...

The way the terms are being used may at times be like of the classic confusion of what is a URL vs a URI.

These seem to be the related terms:

Client endpoint, (MQTT) topic, service.

The SOA Middleware layer needs to be much thicker than has been understood until just recently. This is because it is less of a pass through layer to MQTT and is more involved in the routing of messages to app provided callbacks than understood until now. We rely less on more detailed MQTT subscribe messages than some documentation might hint (ex. VIM DD, SOA Topic Structure). This "new" revelation may result in a shift in understanding some of the assumptions and restrictions imposed on app developers and how we describe the use of our API.

Client endpoint:

This seems to be a logical entity. These are identified by short strings mostly independent of MQTT topics: Ex. ecg/vim

Please think of this as a SOA endpoint, NOT an MQTT endpoint. (MQTT is being abstracted by the SOA framework).

For a provider, it is a service. Services may be one blind data broadcaster and/or zero or one remote call responder.

For a consumer, this is the parent of zero or more remote call requesters and a blind data subscriber of zero or more providers data publications.

Service:

There is a one to one relationship of services to provider client endpoints.

Ex. the ecg/vim team may provide a dashboard service

Ex. ecg/vim/dashboard

SOA providers and SOA consumers:

These are types of SOA clients. Subtypes are data consumers, data providers, remote call consumers and remote call providers.

SOA providers and SOA consumers are built upon client endpoints.

MQTT topics:

Topics are routing strings identifying the nature of a published message. Topics are strings which look like paths.

Ex. /services/data/ecg/vim/dashboard

Wildcard strings are topic query strings, but are not topics.

Ex. /services/data/ecg/vim/dashboard/#

Topic strings and topic query strings are built up from SOA subtypes of client endpoints with or without response routing. The client endpoint short string identifier is part of every topic. And every topic prepends the client endpoint short string with an one of "data", "request" or "response" depending on the SOA client type sending or logically subscribing to the topic.

Ex. /services/**request**/ecg/vim/canpduserver

What does it mean to "logically subscribe" to a topic?

SOA clients don't trigger MQTT subscriptions for services unless they are for blind data broadcasts. The SOA framework does do MQTT wildcarded subscriptions for client endpoints based on SOA client type.

The SOA framework implicitly does a real MQTT wildcarded subscription only once on behalf of a SOA remote call consumer. This minimizes network traffic. The apps DO NOT subscribe explicilty and therefore do not provide listener callback methods for these subscriptions.

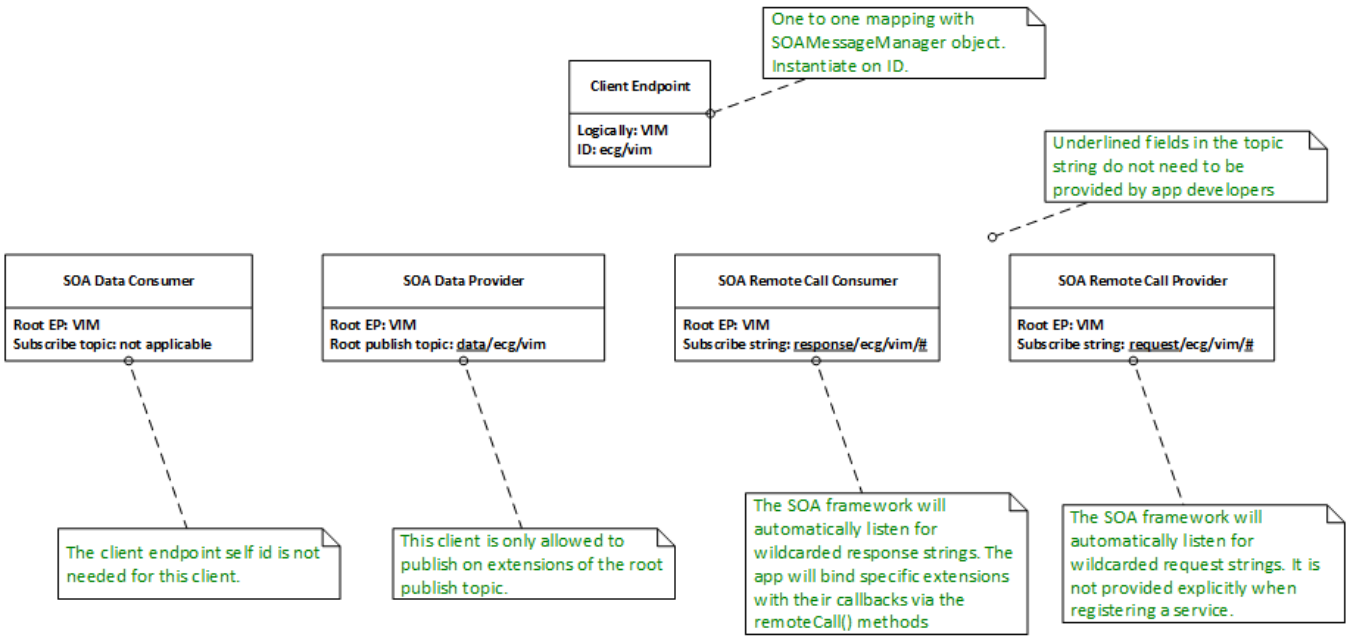
ex. /services/response/ecg/appxyz/#

However, when the SOA consumer explicitly calls the SOA framework remote call method, they are creating a logical subscription for a response on a specific topic. The response topic MUST be a superset string of the client endpoint (short string identifier). The SOA framework routes the response from the internal wildcard subscription listener to the app-provided listener.

Example: Therefore, the SOA framework could accept either "ecg/appxyz/canpduclient" or simply "canpduclient" because it already knows the client endpoint is "ecg/appxyz". If the consumer were to provide an endpoint-response-routing combination which is not a superset of their client endpoint string, the response can never be received.

When a SOA consumer subscribes from a blind data broadcast, the topic the data is published on is that of a different client endpoint, and therefore a real MQTT subscribe for that topic needs to be triggered. The SOA consumer need only provide the client endpoint and service suffix of the topic. The SOA framework can auto prepend this string with /services/data/ to create the full topic for the subscription.

Q: Will we support wildcarded data broadcast subscriptions? Do we auto-append a wildcard?



Once connected, the state is SoaConnectionState::CONNECTED.