|  |
| --- |
| Board Meeting Protocol: Rules, Procedure, Etiquette and More - iBabs |
| Communication Protocols |
| |  |  |  | | --- | --- | --- | | Balija Pavan Kumar | 10/14/23 | Protocols | |

Table of Contents

[**SOME/IP** 2](#_Toc148538655)

[What is SOME/IP 2](#_Toc148538656)

[SOME/IP Short Overview 2](#_Toc148538657)

[Here are the key details about SOME/IP protocol 3](#_Toc148538658)

[Understanding how Communication through Ethernet and SOME/IP Works 6](#_Toc148538659)

[Advantages of SOME/IP 8](#_Toc148538660)

[Signal-Based Communication vs Service-Oriented Architecture 9](#_Toc148538661)

[Examples 10](#_Toc148538662)

[Request / Response 10](#_Toc148538663)

[Subscribe / Notify 14](#_Toc148538664)

[SomeIp\_Method 17](#_Toc148538665)

[**TCP/IP** 19](#_Toc148538666)

[**UDP** 19](#_Toc148538667)

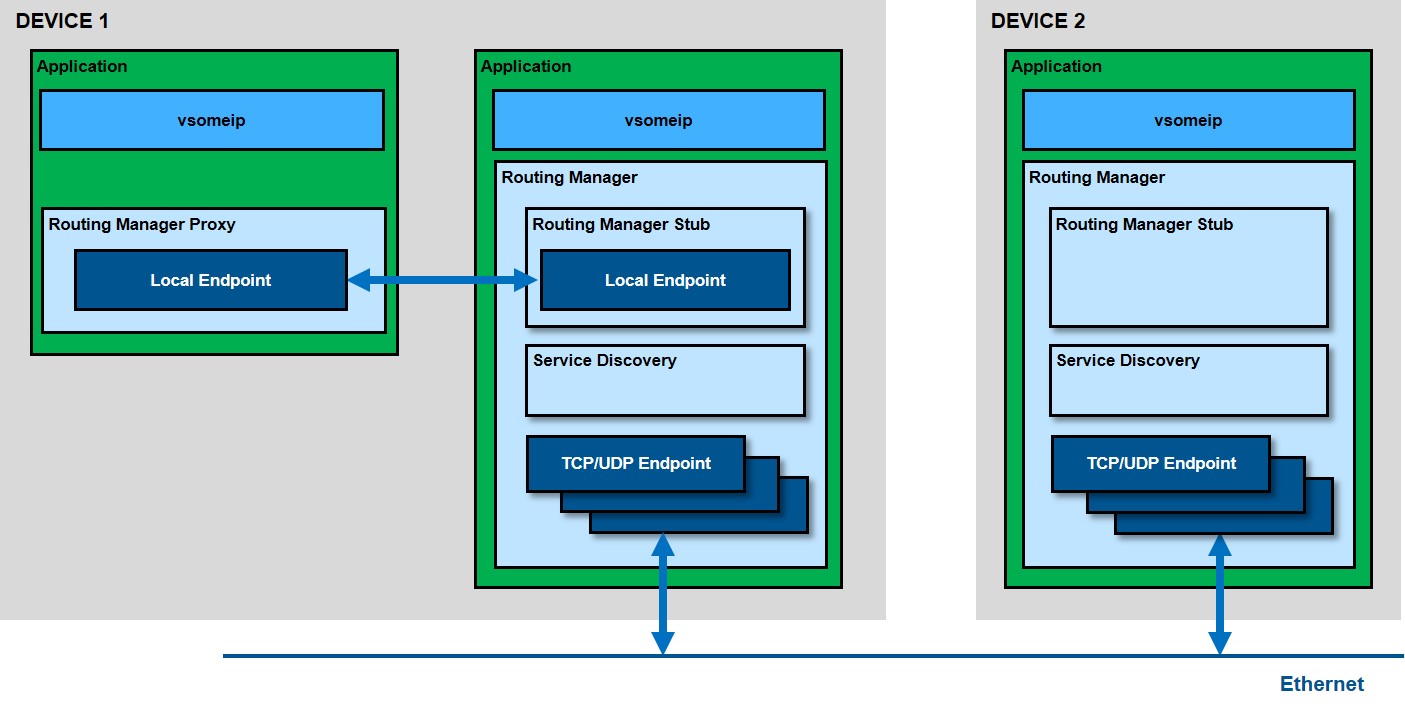
[**CAN** 19](#_Toc148538668)

# **SOME/IP**

## What is SOME/IP

* SOME/IP stands for Scalable Service-Oriented Middleware over IP and was developed by BMW group in the year 2011.
* SOME/IP is a communication protocol used in automotive and other industries for exchanging data and services between electronic control units (ECUs) and other networked devices within a vehicle or a larger system.
* SOME/IP offers a wide range of middleware features like serialization and Remote Procedure Call (RPC) to enable the ECU software to communicate with each other.
* SOME/IP can be implemented on both OS (Genivi, AUTOSAR, Linux and OSEK) and Non-OS embedded system. In the recent past, it has emerged as the middleware of choice for Adaptive AUTOSAR implementation.
* As already mentioned, the service-oriented architecture makes it easier for the software components over diverse networks to communicate with each other. Therefore, in order for these applications on different networks to understand each other, there has to be some kind of a middleware. Its primary role is to resolve the message format and make it comprehendible to the intended recipient of the message. SOME/IP has been specifically designed for this purpose.
* SOME/IP protocol is designed to support the increasing complexity and connectivity demands of modern automotive systems.

## SOME/IP Short Overview

Before we start to implement the introductory example, let's have a short look to the basic structure of the GENIVI implementation of SOME/IP.

As shown in the picture SOME/IP covers not only the SOME/IP communication between devices (external communication) but also the internal inter process communication. Two devices communicate via so-called communication endpoints which determine the used transport protocol (TCP or UDP) and its parameters as the port number or other parameters. All these parameters are configuration parameters which can be set in a SOME/IP configuration file (json file). The internal communication is done via local endpoints which are implemented by unix domain sockets using the Boost.Asio library. Since this internal communication is not routed via a central component (e.g. like the D-Bus daemon), it is very fast.

The central SOME/IP routing manager gets messages only if they have to be sent to external devices and he distributes the messages coming from outside. There is only one routing manager per device; if nothing is configured the first running SOME/IP application also starts the routing manager.

❗ SOME/IP does not implement the serialization of data structures! This is covered by the SOME/IP binding of CommonAPI. SOME/IP just covers the SOME/IP protocol and the Service Discovery.

## Here are the key details about SOME/IP protocol

**Origin and Purpose:**

SOME/IP was developed by the Automotive Special Interest Group (ASIG), a part of the AUTOSAR (AUTomotive Open System ARchitecture) consortium. It was created to provide a standardized and efficient means of communication between ECUs in modern vehicles. The protocol is designed to support the increasing complexity and connectivity demands of modern automotive systems.

**Communication Model:**

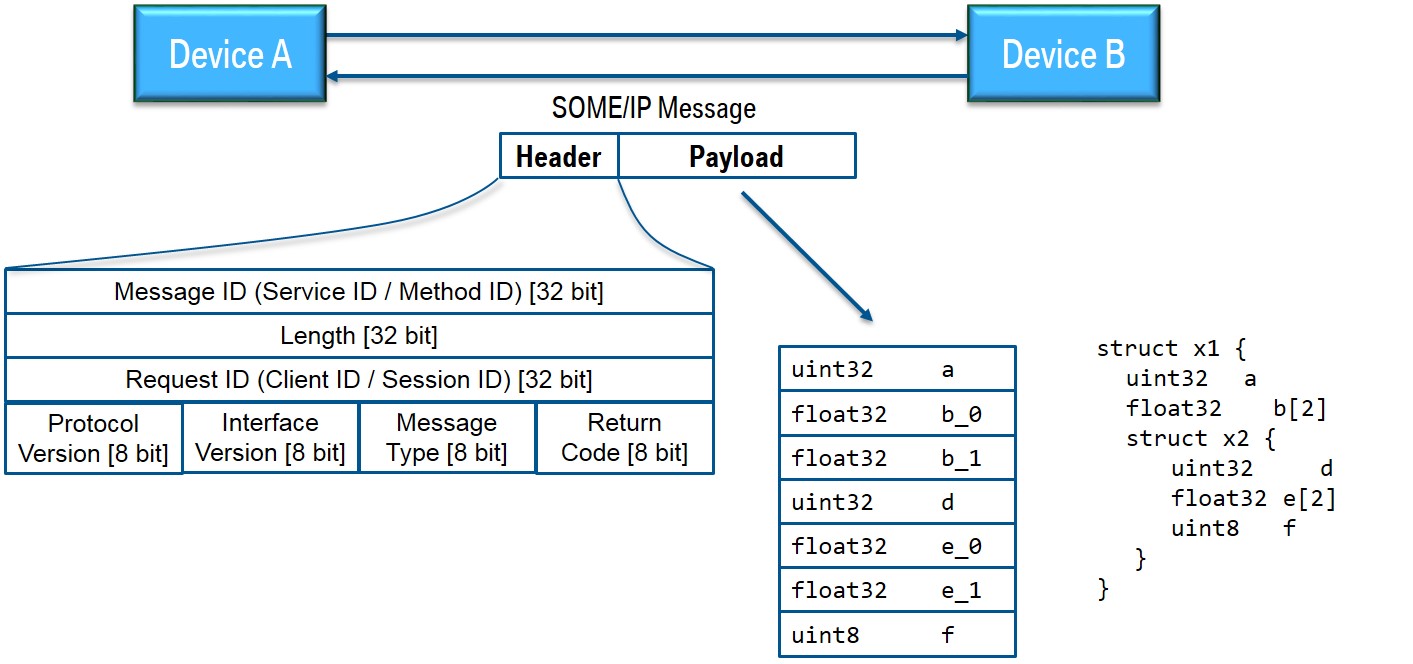
SOME/IP uses a client-server communication model. ECUs can act as clients that request services or data from server ECUs. The protocol defines how these clients and servers communicate, making it easier to implement a service-oriented architecture (SOA) in automotive systems.

**Message Format:**

SOME/IP messages are structured using a message header and payload. The message header includes information like message type, service ID, instance ID, method ID, and message length. The payload carries the actual data or parameters being exchanged between clients and servers.

In principle, SOME/IP communication consists of messages sent between devices or subscribers over IP.

**Consider the following picture:**



There you see two devices (A and B); Device A sends a SOME/IP message to B and gets one message back. The underlying transport protocol can be TCP or UDP; for the message itself this makes no difference. Now we assume that on device B is running a service which offers a function that is called from device A by this message and the message back is the answer.

**SOME/IP messages have two parts: header and payload. In the picture you see that the header consists mainly of identifiers:**

**Service ID:** unique identifier for each service

**Method ID:** 0-32767 for methods, 32768-65535 for events

**Length:** length of payload in byte (covers also the next IDs, that means 8 additional bytes)

**Client ID:** unique identifier for the calling client inside the ECU; has to be unique in the overall vehicle

**Session ID:** identifier for session handling; has to be incremented for each call

**Protocol Version:** 0x01

**Interface Version:** major version of the service interface

**Message Type:** -- REQUEST (0x00) A request expecting a response (even void) -- REQUEST\_NO\_RETURN (0x01) A fire & forget request -- NOTIFICATION (0x02) A request of a notification/event call back expecting no response -- RESPONSE (0x80) The response message

**Return Code:** -- E\_OK (0x00) No error occurred -- E\_NOT\_OK (0x01) An unspecified error occurred -- E\_WRONG\_INTERFACE\_VERSION (0x08) Interface version mismatch -- E\_MALFORMED\_MESSAGE (0x09) Deserialization error, so that payload cannot be deseria-lized -- E\_WRONG\_MESSAGE\_TYPE (0x0A) An unexpected message type was received (e.g. RE-QUEST\_NO\_RETURN for a method defined as RE-QUEST)

We see that there are "REQUESTs" and "RESPONSEs" for normal function calls and notification messages for events to which the client has been subscribed. Errors are reported as normal responses or notifications but with an appropriate return code.

The payload contains the serialized data. The picture shows the serialization in the simple case that the transmitted data structure is a nested structure with only base data types. In this case it is easy: the struct elements are just flattened, that means that they are simply written one after the other into the payload.

**Transport Layer:**

SOME/IP typically operates over the Internet Protocol (IP), either over Ethernet or other network protocols like UDP or TCP/IP. This allows for flexible and scalable communication between ECUs within the same vehicle or even across different vehicles or systems.

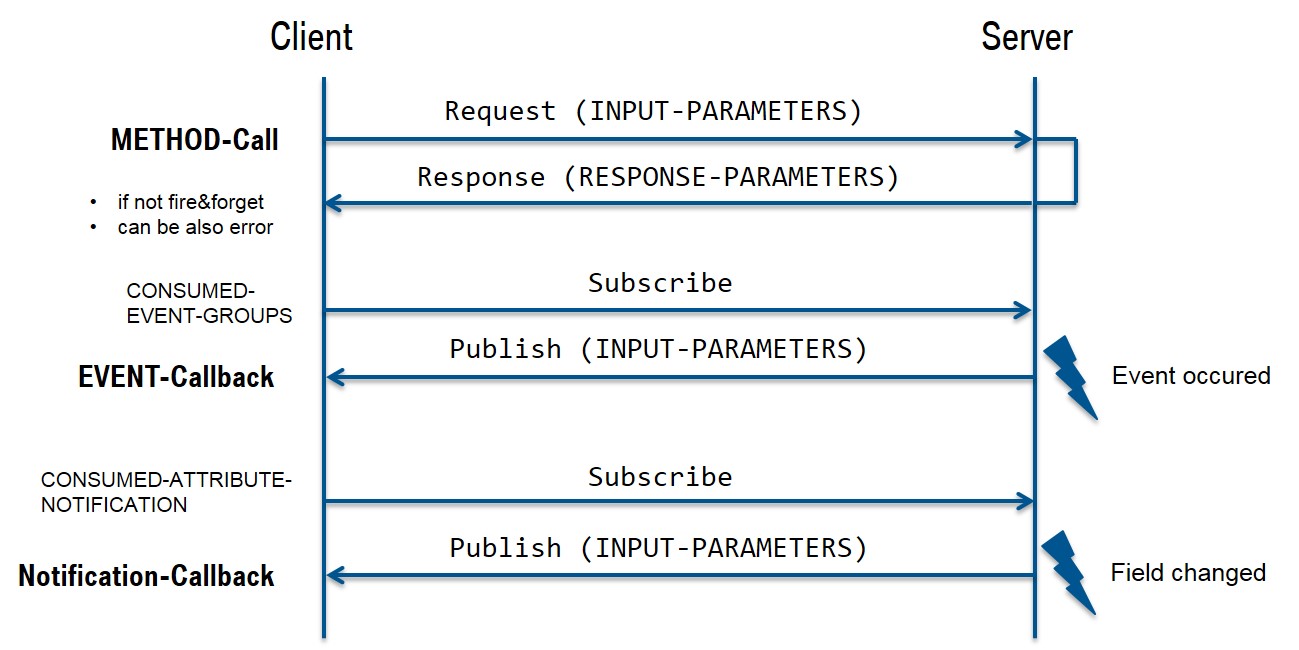
**In this section mainly 2 points are important and shall be described now:**

1. The so-called transport bindings (UDP and TCP)
2. The basic communication patterns publish/subscribe and request/response.

As mentioned above the underlying transport protocol can be UDP or TCP. In the UDP case the SOME/IP messages are not fragmented; it can be that more than one message is in one UDP packet, but one message cannot be longer than a UDP package can be (up to 1400 Bytes). Bigger messages must be transported via TCP. In this case all the robustness features of TCP are used. If a synchronization error in the TCP stream occurs, the SOME/IP specification allows so-call magic cookies in order to find again the beginning of the next message.

Please note that service interfaces must be instantiated and because there might be several instances of the same interface there must be an additional identifier for the instance defined (instance ID). However, the instance ID is not part of the header of the SOME/IP message. The instance is identified via the port number of the transport protocol; that means that it is not possible that several instances of the same interface are offered on the same port.

**Please take now a look at the following picture which shows the basic SOME/IP communication patterns:**



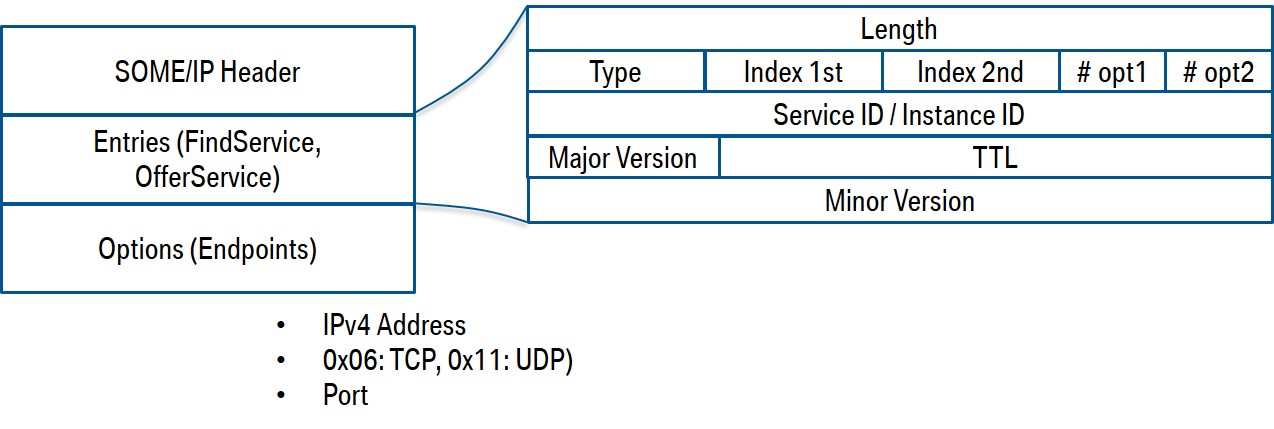
In addition to the standard REQUEST/RESPONSE mechanism for remote procedure calls there is also the PUBLISH/SUBSCRIBE pattern for events. Note that events in the SOME/IP protocol are always grouped in an event group; therefore, it is only possible to subscribe to event groups and not to the event itself. The SOME/IP specification also knows "fields"; in this case the setter/getter methods are following the REQUEST/RESPONSE pattern and notification messages of changes are events. The subscription itself is done via the SOME/IP service discovery.

**Service Discovery and Description:**

SOME/IP uses service discovery and description mechanisms to make services and their parameters known to other ECUs on the network. This allows clients to discover available services and interact with them. Service discovery can be achieved using a central directory or through multicast announcements.

The SOME/IP Service Discovery is used to locate service instances and to detect if service instances are running as well as implementing the Publish/Subscribe handling. This is mainly done via so-called offer messages; that means that each device broadcasts (multicasts) messages which contain all the services which are offered by this device. SOME/IP SD messages are sent via UDP. If services are required by client applications but at the moment not offered, then also "find messages" can be sent. Other SOME/IP SD messages can be used for publishing or subscribing an event group.

**The following picture shows the general structure of a SOME/IP SD message.**

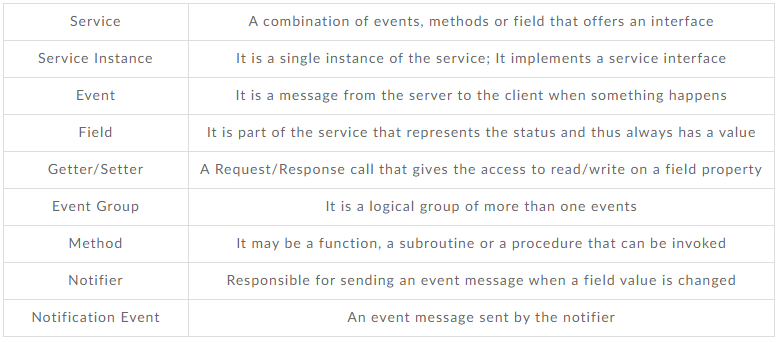


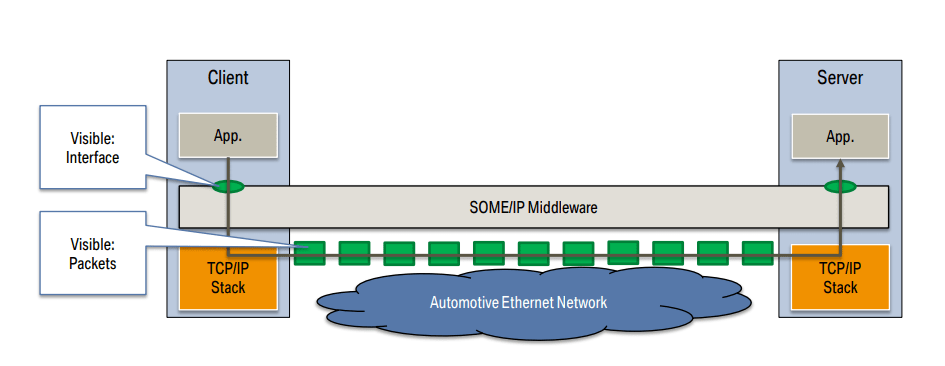
**Remote Procedure Call (RPC):**

This is a method for remote invocation of functions as requested by the Client ECU. It is a data exchange method employed by the client ECU when it requires some data from a server. An RPC may or may not have a return value, i.e., the client can ask for data as a response or simply call a function to perform some task at the server-end.

## Understanding how Communication through Ethernet and SOME/IP Works

Before we start with our exploration, we must understand some terms associated with SOME/IP





The Server ECU provides a service instance which implements a service interface. The client ECU can use this service instance using SOME/IP to request the required data from the server. The Service Discovery protocol has two mechanisms in place by which a client knows about the available services.

The first mechanism is the ‘Offer Service’ using which the server is able to provide the available services to the network. The other one is ‘Find Service’, which enables the clients to request for the available services.

However, in order for the client to use the service, it has to subscribe to the content on server first.

Using the SOME/IP Service Discovery protocol, a client can send a Subscribe Event group to the server. If the subscription request is valid, the server will respond with a positive acknowledgement and vice-versa.

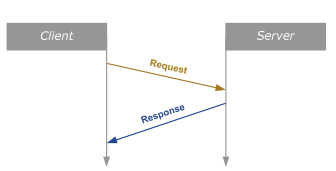
As the applications inside the ECUs are not tightly coupled in a Service-oriented architecture, multiple clients can subscribe to a service on the server simultaneously. The data can be made available either over UDP or TCP. In case of UDP, data is sent to all the clients who are active subscribers. The data transfer is usually sent via unicast, multicast or broadcast. However, with TCP, the requesting client must establish a connection with the server for data transfer.

While the underlying principle of SOME/IP is based on the client-server architecture where a request by a client is followed by a response from the server, there are several methods/communication patterns for the communication. Let’s have a look at them.

1. **Request/Response Method**

A request is a message sent from the client to the server for calling a function.

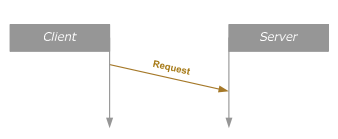
Response is the message sent from the server to the client depicting the result of the function invoked by the client



1. **Fire and Forget Method**

A message is sent to the server from the client to call a function

No response is returned from the server

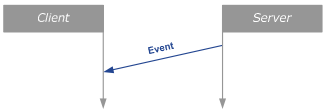


1. **Services: Event**

An event is a callback sent from the server to the client either cyclically or when a change in the server attributes occur

The server notifies about the change only to those clients who have previously subscribed

A notification of event is sent to the client every time the event occurs



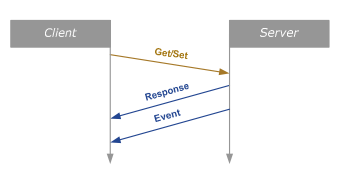
1. **Services: Field**

Field is a property of service that can be remotely accessed using Getters/setters.

Getter is the method to read field value

Setter is the method to set the field value

When a field’s value changes, a notification event is sent out by the notifier.



## Advantages of SOME/IP

**Error Handling:** The protocol defines error codes and mechanisms for handling errors during communication. This includes error handling for issues like timeouts, unavailable services, or invalid requests.

**Security:** Security in SOME/IP can be implemented at various levels. While the protocol itself does not provide built-in security, it can run over secure IP networks using protocols like IPsec or TLS to ensure data integrity and confidentiality.

**Scalability:** SOME/IP is designed to be scalable, allowing for the addition of new services and ECUs without major changes to the existing architecture. This makes it well-suited for the evolving nature of automotive systems.

**Diagnostic and Maintenance:** The protocol includes mechanisms for diagnostic and maintenance services. This allows for monitoring the health of ECUs, troubleshooting, and performing software updates or maintenance tasks remotely.

**Standardization:** SOME/IP is an industry-standard protocol developed under the AUTOSAR consortium. This standardization ensures that all participating devices and ECUs within the system can communicate effectively and reliably, promoting interoperability.

**Service-Oriented Architecture:** It facilitates a service-oriented architecture (SOA) where services and data are exposed by server ECUs and can be easily discovered and consumed by client ECUs. This modular approach simplifies system design and maintenance.

**Efficiency:** SOME/IP is designed for efficient communication, which is crucial in the automotive industry where real-time performance is essential. It uses lightweight message formats and supports both unicast and multicast communication, reducing the overhead associated with data exchange.

**Flexibility:** SOME/IP can run over various network protocols, including Ethernet, UDP, and TCP/IP, allowing it to adapt to different network infrastructures. This flexibility enables its use in diverse environments and use cases.

**Real-Time Communication:** SOME/IP can support real-time requirements in automotive systems. It allows for setting priorities for messages and defining communication cycles, ensuring that critical data is transmitted on time.

**Cost-Effective:** By leveraging IP-based networks, SOME/IP can help reduce the cost of wiring and hardware in automotive systems. It eliminates the need for custom communication networks, such as CAN (Controller Area Network), in favour of standard Ethernet and IP technologies.

## Signal-Based Communication vs Service-Oriented Architecture

**Signal-Based Communication:**

* Signal-based communication has long been used in communication protocols such as CAN, LIN, FlexRay, MOST and more. As the software and hardware in such automotive solutions are closely coupled, the communication between the ECUs are defined statically. It is assumed that the software will not be modified during its lifetime. Signal based communication was best suited for such applications.
* In the realm of Signal-based communication, the data is sent over the network whenever the data values are updated or modified. The sender is not concerned about whether the data is required by a node in the network. Such an arrangement may burden the nodes with unwanted data that they might never require.

**Service-Oriented Architecture:**

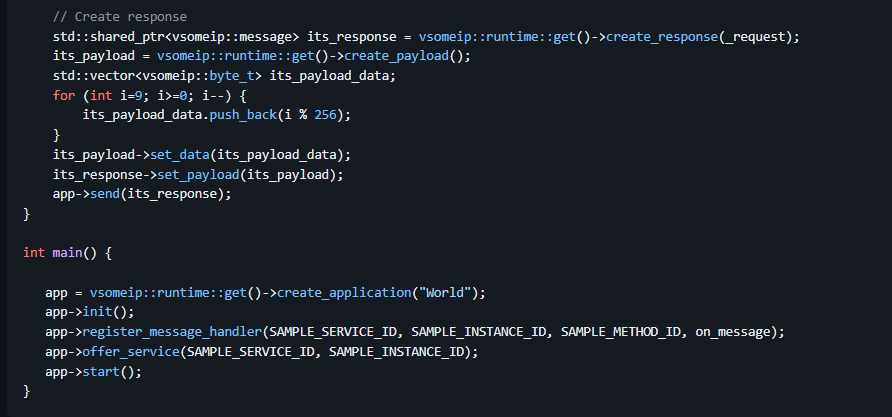
* Fast forward to Service-oriented architecture, the sender sends the data only if a receiver needs it. Therefore, in such an arrangement, the server has to be notified about the receivers that are waiting for the data. This is merely one aspect of the service-based communication.
* When we talk of highly automated driving, ADAS, connected cars, etc., service-oriented architecture (SOA) is a must-have. Powered by Ethernet and SOME/IP, SOA models the entire system as service interfaces. New software can be easily added to the system without worrying about the compatibility with others.
* While Ethernet provided the backbone and TCP & UDP the transport layer, a middleware was required for data serialization, remote call procedure, etc.

## Examples

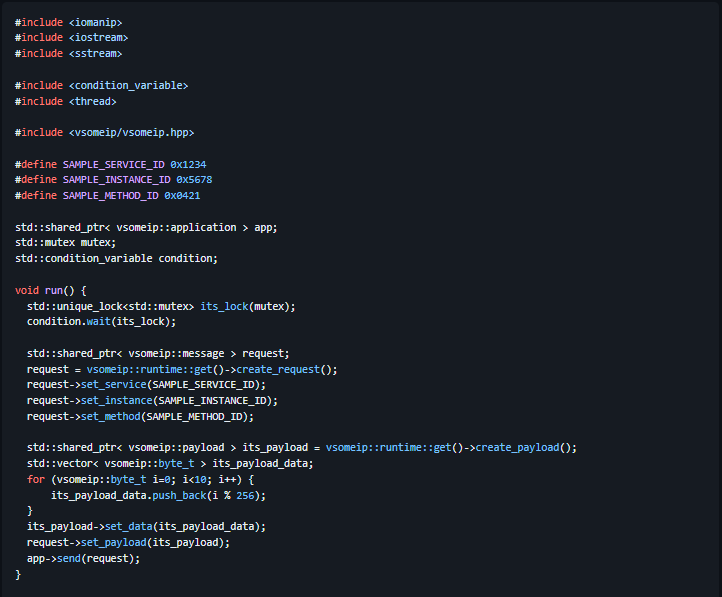
### [Request / Response](https://github.com/COVESA/vsomeip/wiki/vsomeip-in-10-minutes#request--response)

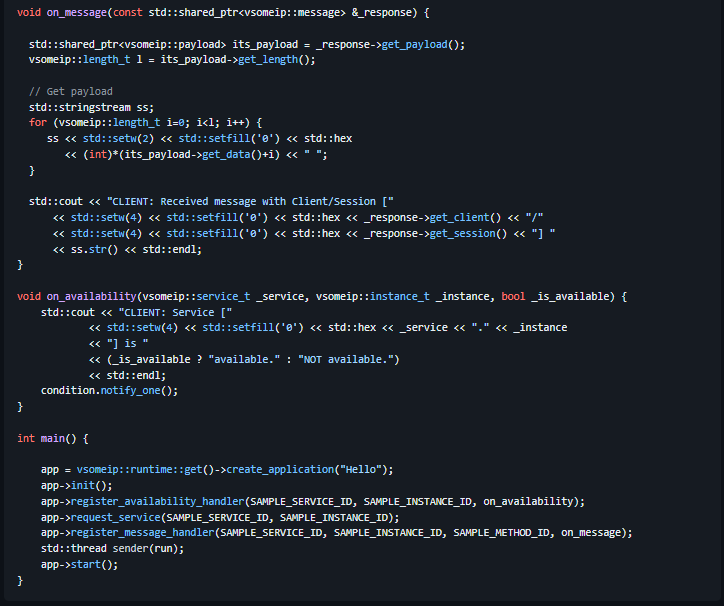
**service-example.cpp with offer and message handler**



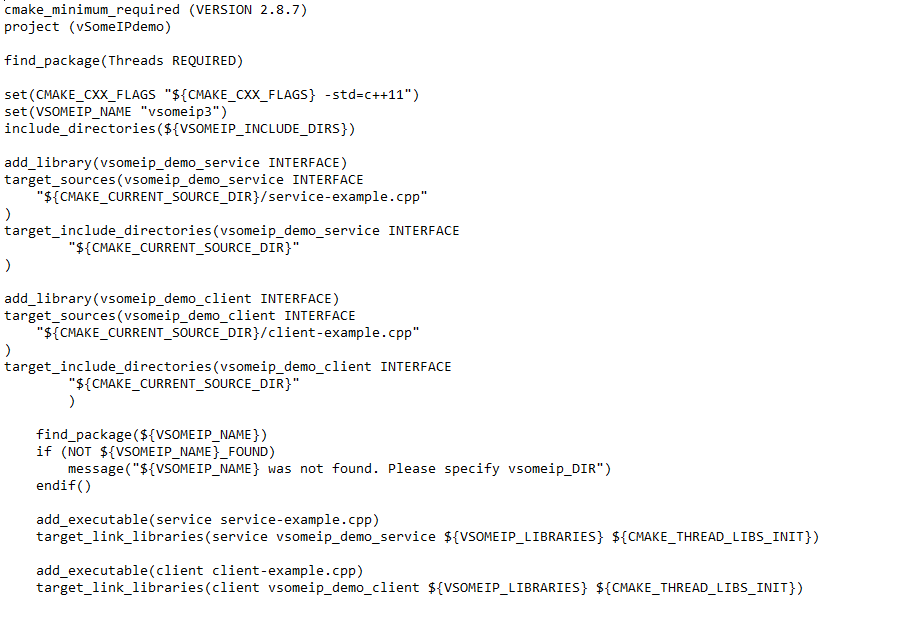


**client-example.** **the message handler and send function**

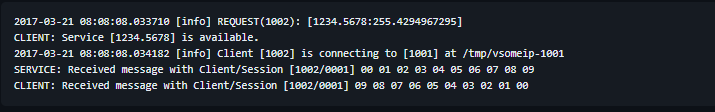




**CMakeLists:**



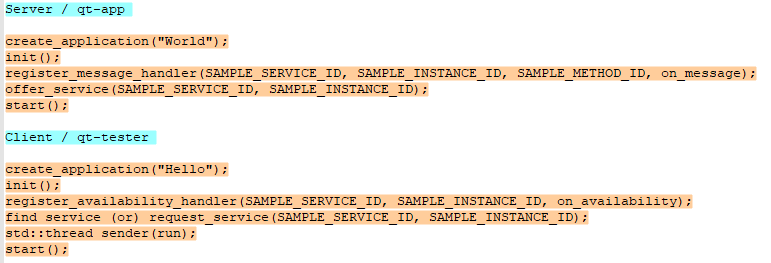
**Result:**



**Explanation:**

Like on service side we need to register a message handler for receiving the response of our call. In principle it is very easy to create the send message (request). Just get the request object by calling create\_request(), set service ID, instance ID and method ID and at the end write the payload with your serialized data. In the example here we write the values from 0 to 9 in the payload (std::vector< vsomeip::byte\_t >).

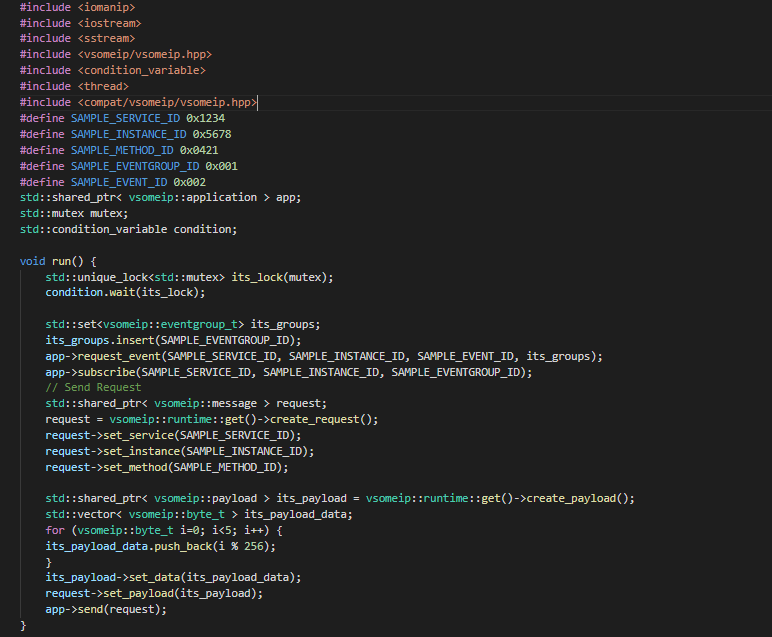
When we try to send our request from the client to the service then we run into a little problem. The application has to be started (app->start()) before we can send the message, because we need a running event-loop in order to process the message. But the method app->start() does not return because it has the running event-loop inside. Therefore we start a thread (run) and wait in this thread for the return of the availability callback before we call app->send(request).



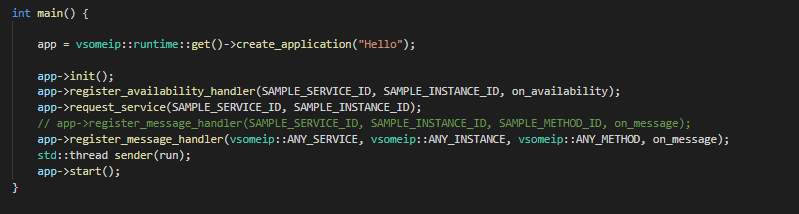
### Subscribe / Notify

So far, we have created a service that implements a method and a client that calls this method. But that is not all that is possible. The SOME/IP specification also describes an event handling. That means that application can send events to which subscribers can subscribe if they are interested. Together with the definition of setter and getter methods it is possible to implement services which provide attributes.

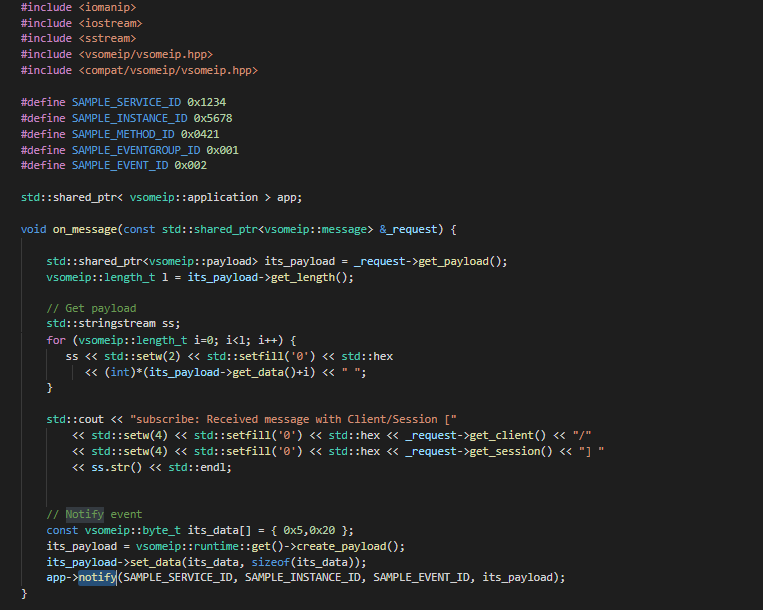
**Subscribe**

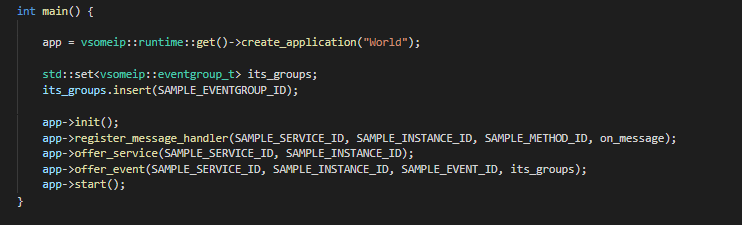




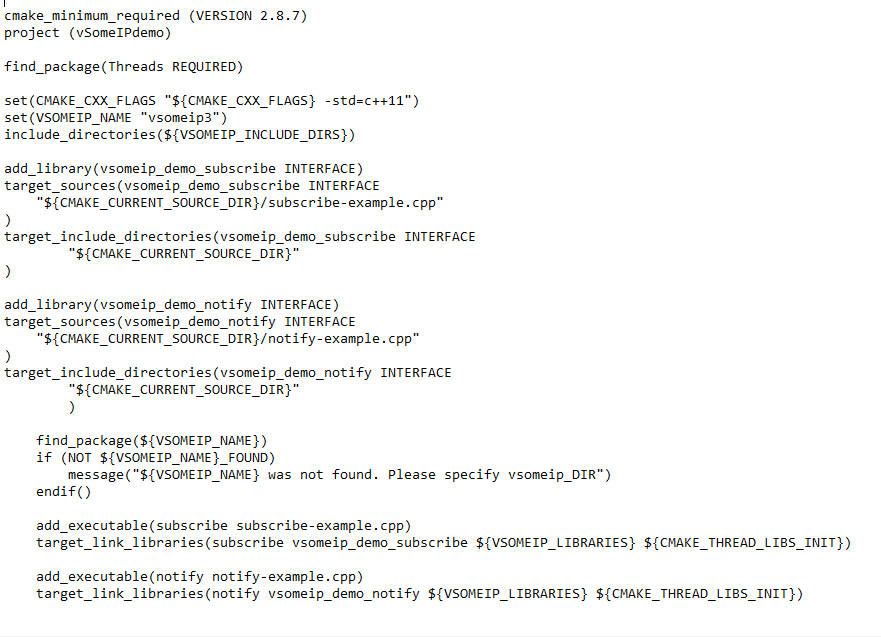


**Notify**



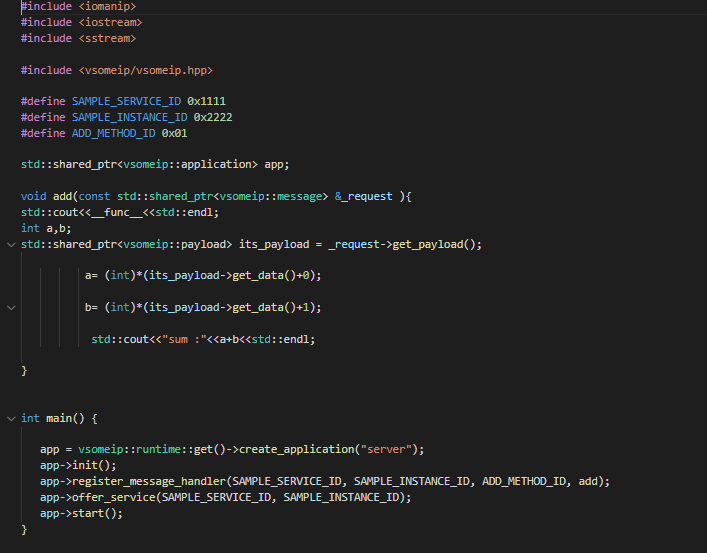


**CMakeLists:**

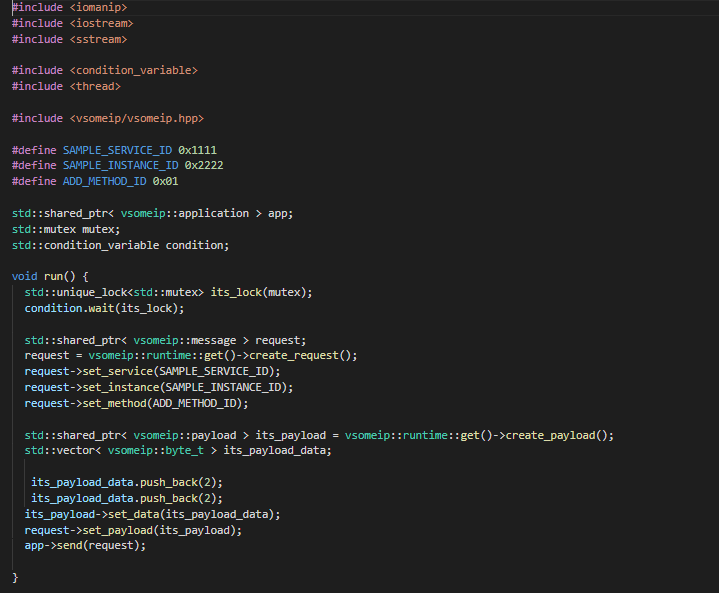


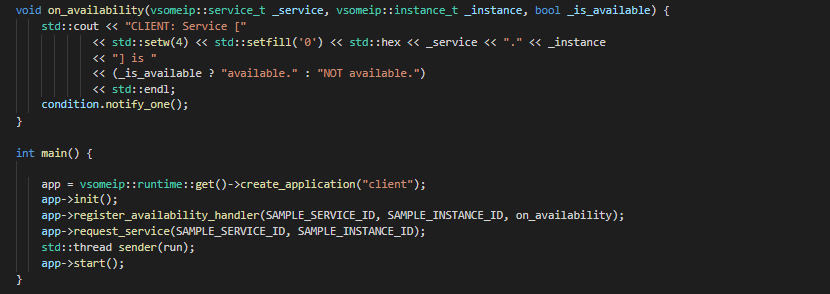
### SomeIp\_Method

**Server:**



**Client:**





**CMakeLists:**



### Reference Links:

<https://github.com/COVESA/vsomeip/wiki/vsomeip-in-10-minutes>

<https://www.embitel.com/blog/embedded-blog/how-some-ip-enables-service-oriented-architecture-in-ecu-network>

# **TCP/IP**

# **UDP**

# **CAN**