2. System APIs

Common software components like the ***SOA Gateway*** need to interface with system libraries for the ECUs that they are ported to. The following System API descriptions and function prototypes provide details on what is required to make the common software components run functionally equivalent to other ECUs.

2.1. Logging APIs

To understand what is happening in the common software components, it helps to have any logging from these software components log to your target ECU's system logger.

2.1.1. Description

It does not make sense to create a common software specific abstracted logging infrastructure for the sake of porting. However, in order to port software components with logging enabled in a vendor's system logs, the following lists what is required from the vendor:

- Headers - logging headers are needed to provide function prototypes which describe how to make calls to log to the system logger
- logging library - what we link to in order to enable logging for our software components
- logging description - example of log messages at different log levels; some of the things typical logging frameworks contain:
    - **log_tag** - describes which component is /log
    - **log_level** - severity of log message (e.g.
    - **log_message** - character array of log information

2.1.2. Example of ECG Abstraction

In this example, the header <ecu_logger.h> provides the following logging function prototype:

Logging example

```
/* log to the system logger
* param logLevel – char – designates the log level where it can be one of ( I, D, W, E,
or C )
* param message – const char * – message to be logged
*/
log( char logLevel, const char * message);

// function with a message array and logging at the different logging levels
log('I', message); // log at info level
log('D', message); // log at debug level
log('W', message); // log at warning level
```

```
log('E', message); // log at error level
log('C', message); // log at critical/catastrophic level
```

2.2. Keystorage APIs

Security certificates/keys are required to do operations securely. For instance, in order for the SOA Gateway to establish a secure connection via TLS with the SOA Broker, the SOA Gateway needs device certificates to authenticate with the SOA Broker. Other examples for where keys and certificates are required include for Software Updates; information in the update binaries are checked against certificates to ensure the update is trusted. The following subsections provide detail on keys/certificates needed for various services:

2.2.1. SOA Gateway

The SOA Gateway needs access to the paths of the following 3 certificates to enable a secure TLS connection with the SOA Broker:

- CERT_ECG_SOA_TLS_CA - ECG CA Certification location
- CERT_CLIENT_SOA_TLS_PUBLIC - client public tls certificate location
- KEY_CLIENT_SOA_TLS_PRIVATE - client private tls key location

In order to port SOA Gateway so that it can securely connect to the broker, the following is needed from the vendor:

- Headers - header with function prototypes which return paths to the different key types stored on the target:
  - paths to one set of keys & certificates
  - or, paths to production keys & certificates AND paths to development keys & certificates
- key-store library - what we link to to get paths so SOA Gateway can read certificates and keys

2.2.1.1. Function Prototypes

Get Development ECG CA Certificate Path Prototype

```
/**
* getDevEcgCaCertificatePath – Read Development ECG CA Certificate Path
* @return char* – path to Development ECG CA Certificate file
*/
char* getDevEcgCaCertificatePath();
```

Get Production ECG CA Certificate Path Prototype

```
/**
* getProdEcgCaCertificatePath – Read Production ECG CA Certificate Path
* @return char* – path to Production ECG CA Certificate file
*/
char* getProdEcgCaCertificatePath();
```

Read Development Device Unique Certificate Path Prototype

```
/**
* getDevDeviceCertPath– Read Development Device Unique Certificate Path
* @return char* – path to Development Device Unique Certificate file
*/
char* getDevDeviceCertPath();
```

Read Production Device Unique Certificate Path Prototype

```
/**
* getProdDeviceCertPath – Read Production Device Unique Certificate Path
* @return char* – path to Production Device Unique Certificate file
*/
char* getProdDeviceCertPath();
```

Read Development Device Unique Key Path Prototype

```
/**
* getDevDeviceKeyPath– Read Development Device Unique Key Path
* @return char* – path to Development Device Unique Key file
*/
char* getDevDeviceKeyPath();
```

Read Production Device Unique Key Path Prototype

```
/**
* getProdDeviceKeyPath– Read Production Device Unique Key Path
* @return char* – path to Production Device Unique Key file
*/
char* getProdDeviceKeyPath();
```