MRM VCIL User Manual

5/4/2017
SSID.SRP

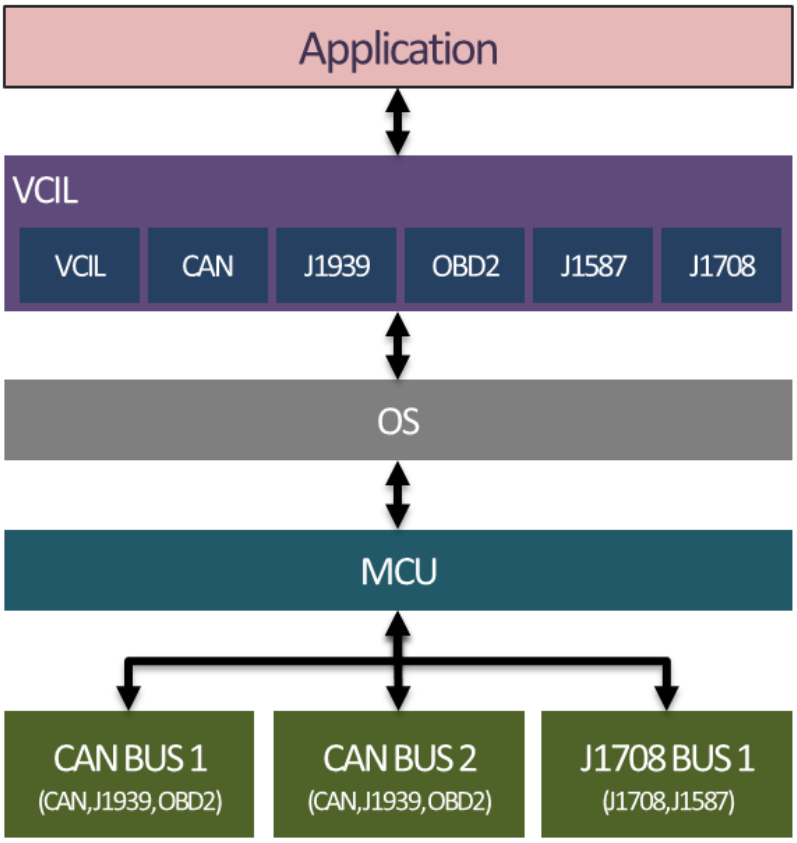
Table of contents

1 Introduction

1.1 VCIL Overview

MRM VCIL SDK is a set of libraries for user to control the vehicle communication protocols on devices which are equipped with VCIM (Vehicle Communication Interface Module) MCU. VCIM supports two CAN ports for CAN, J1939, OBD2 protocols, and one J1708 port for J1708, J1587 protocols and controls the data flow of those protocols on the data buses.

MRM VCIL SDK provides API modules for applications to control each protocol to achieve various purposes. The software stack of SDK can be described as the following figure.



MRM VCIL API supports the following platforms:

For Windows / Linux:

The MRM VCIL API is designed as a library (**vcil.dll** / **vcil.so**) for the customer's APP to load and access.

Prior to use VCIL APIs, you should call **vcil_init()**. Normally the API should return 32-bits error code **MRM_ERR_NO_ERROR(0).** You must confirm the return value to ensure that the command is work. After calling **vcil_init()**, the library will initialize VCIL and start to accept command.

For Android:

The MRM VCIL API is designed as libraries for the customer's APP to load and access

The MRM VCIL API for Android includes three parts:

- MRM Data Classes And Constants Definitions (**MrmDef.jar**)
- MRM Java APIs (**MrmJni.jar**)
- Native libs (**libvcilJni.so, libvcil.so**)

To use the VCIL, you must import the above libraries in your APP project.
In your APP, you must call **vcil_init()** to initialize the MCU before accessing the VCIL APIs and call **vcil_deinit()** to release allocated resources before your APP is destroyed.

1.2 System Requirements

Hardware:

- VCIL Module

Operating system:

- Windows
- Linux
- Android

Recommended Development Tool

- Visual Studio 2008 or above
- GCC 4.6+
- Android Studio 1.3.2+

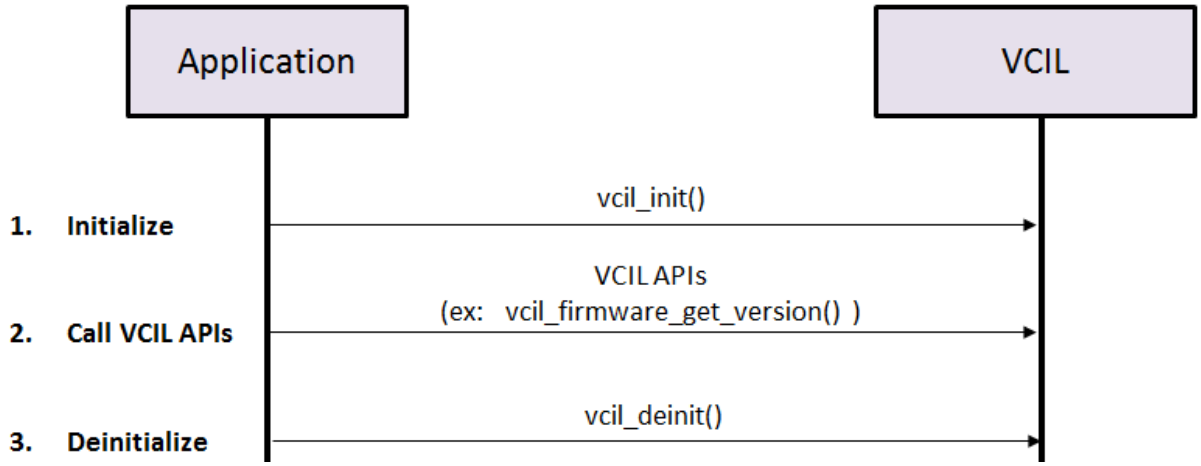
2 Using VCIL

2.1 VCIL Conventions

2.1.1 Windows/Linux

- **Basic API usage**
  - Each VCIL API has a prefix "**vcil**_" immediately followed by the function name and the operation name.
  - To use the VCIL APIs, you must first initialize the library and deinitialize before your APP is closed.

    The flow is described as following:
    1. You must **vcil_init()** before using the other IVCP APIs.
    2. Call VCIL APIs.
    3. You must call **vcil_deinit()** before you APP closed.



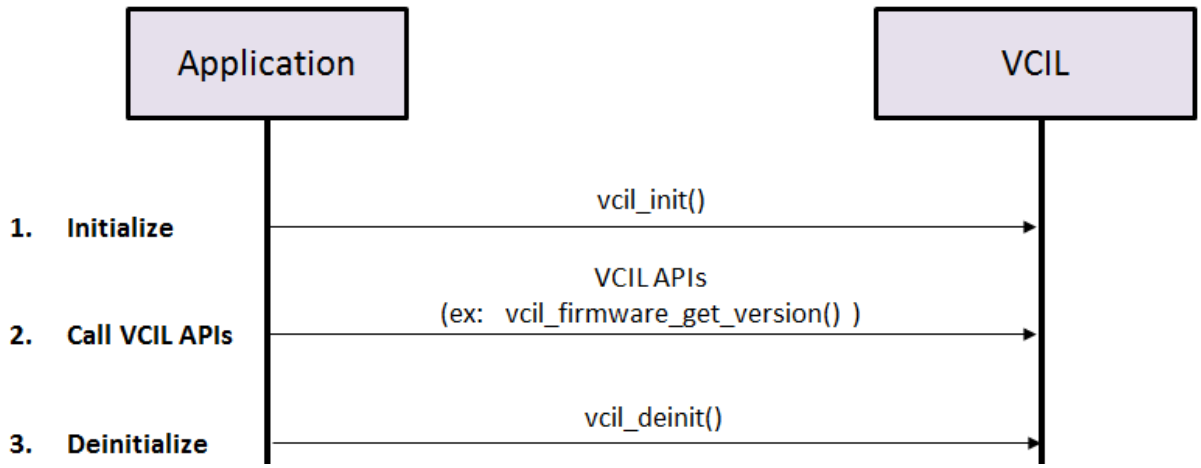
  - Any read type API using the pass by pointer. You should ensure the pointer be not released during the API processing and the pointer is valid.
  - You should always check the return value equal MRM_ERR_NO_ERROR(0), in order to ensure the command working.

2.1.2 Android

- **SDK Namespaces**
  - **mrm.VCIL**
    - The class of VCIL API.
  - **mrm.define.VCIL**
    - The definition of data structures used by VCIL API.
  - **mrm.define.MRM_ENUM**
    - The definition of enumeration values used by VCIL API.
  - **mrm.define.MRM_CONSTANTS**
    - The definition of constants used by VCIL API.
  - **mrm.define.MRM_ERROR**
    - The definition of error codes.

- **Basic API usage**
  - Each VCIL API has a prefix "**vcil**_" immediately followed by the function name and the operation name
  - You must create an instance of **mrm.VCIL** to usage VCIL APIs .
  - To use the VCIL APIs, you must first initialize the library and deinitialize before your APP is closed.

    The flow is described as following:
    1. You must **vcil_init()** before using the other IVCP APIs.
    2. Call VCIL APIs.
    3. You must call **vcil_deinit()** before you APP closed.



  - APIs for reading data need an array for argument to store data. The array should be allocated

before you pass it to the API and the data will be stored at **index 0 of the array.**

- You should always check the return value of APIs for error checking. The value should equal to MRM_ERR_NO_ERROR(0) when success or other value when failed.

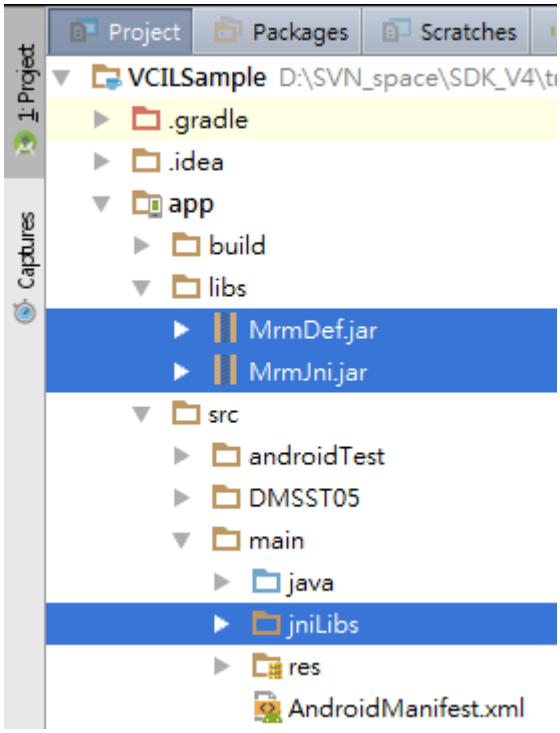
2.2 C++ with Visual Studio

1. Right Click on Project to enter Project Properites
2. Add #include "vcil/vcil.h" to your program
3. Select **Linker** and **Input** page. Set Additional Dependencies "vcil.lib"

2.3 JAVA with Android Studio

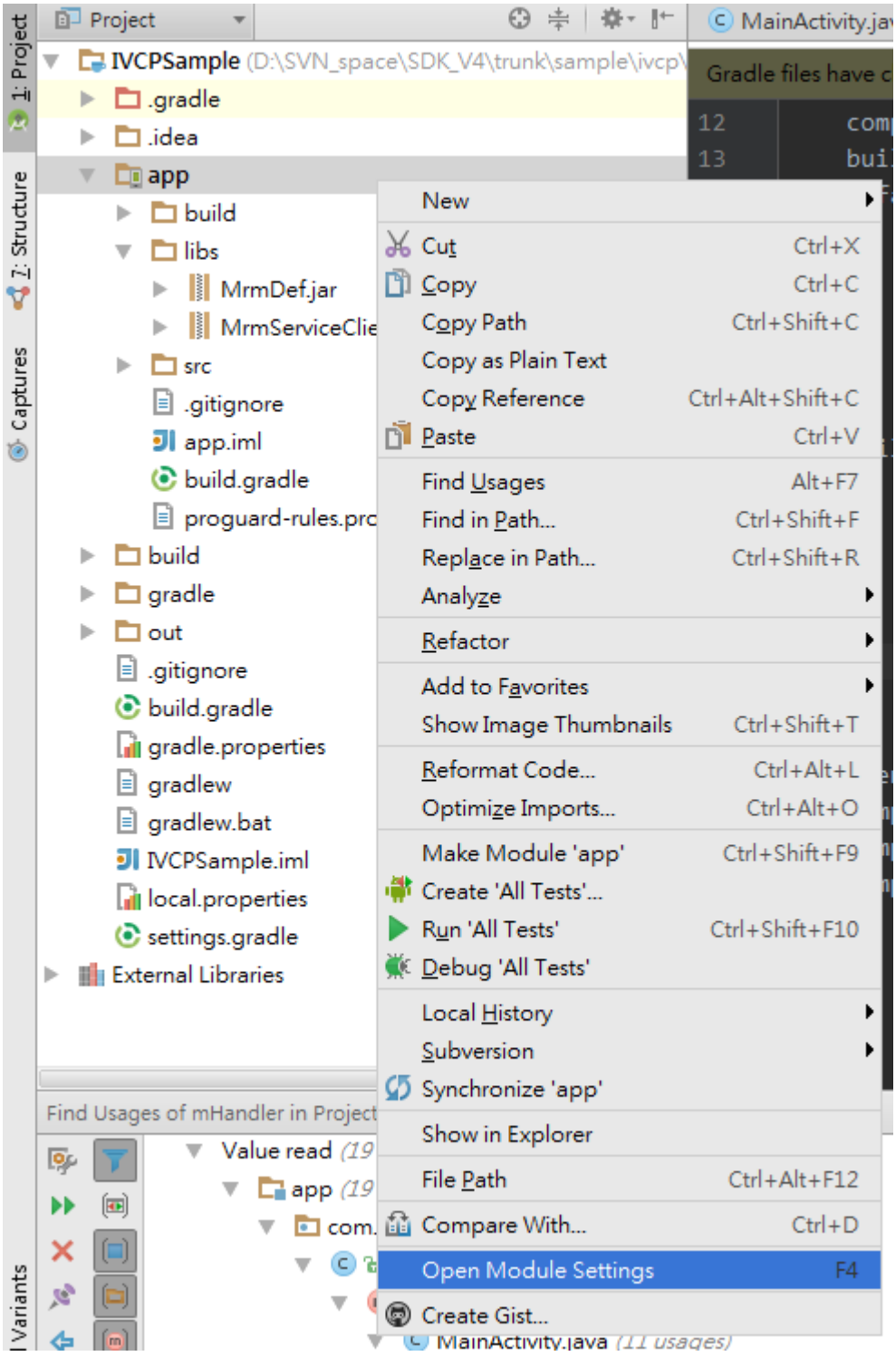
To access VCIL funtions from your APP, you must import the VCIL libraries into you project.

Please find the **MrmJni.jar,** **MrmDef.jar** and **jniLibs/** folder in the MRM SDK package.
Copy the **MrmJni.jar,** **MrmDef.jar** to the directory **/[Module Name]/libs/** in your Android Studio project (the default module name might be "app") and copy the **jniLibs/** folder to the directory **/[Module Name]/src/main/** .

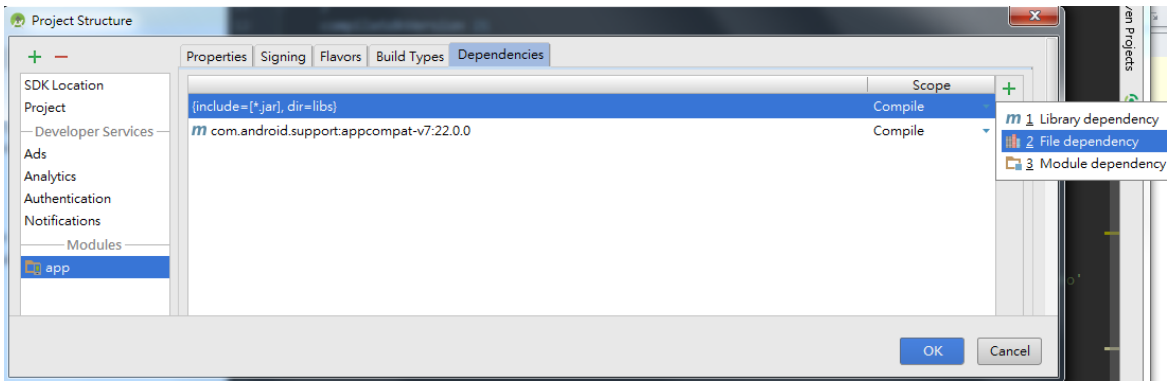


Then import the Java libraries by following the steps below:

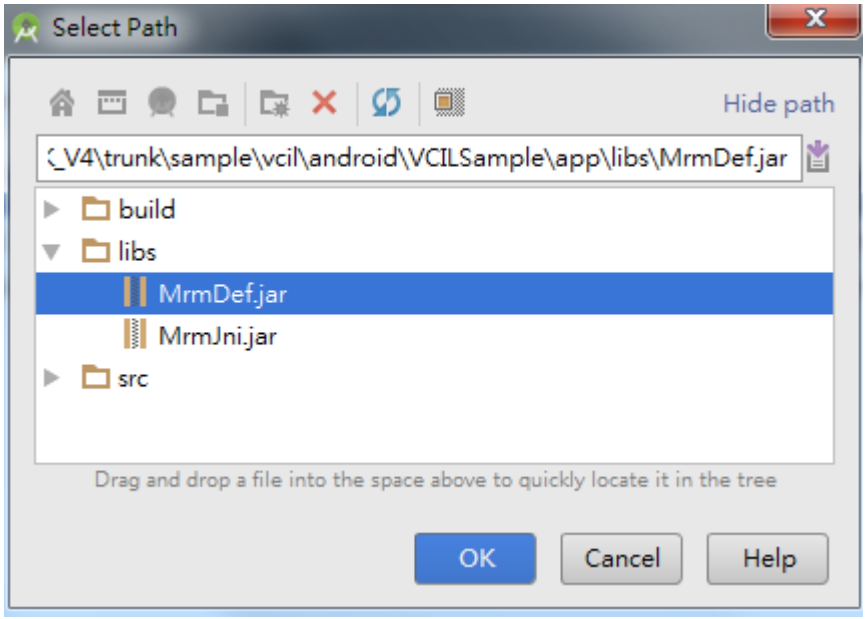
- Right click on you APP module. Click "**Open module settings**"



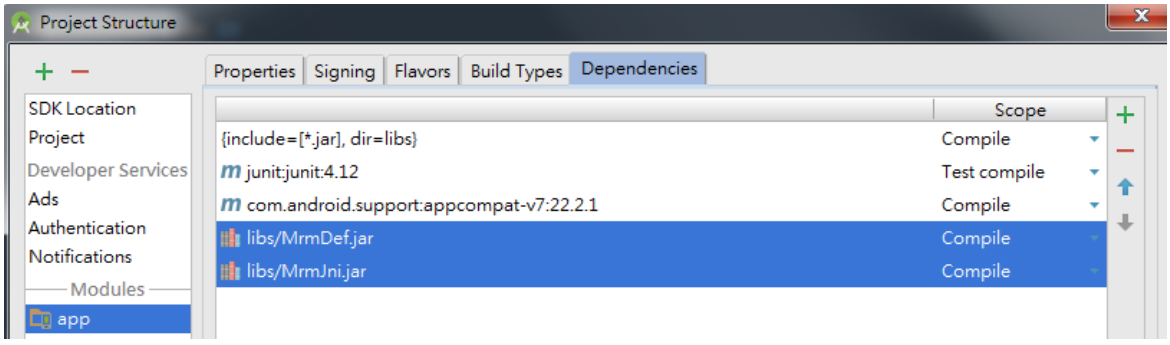
- Click the "**Dependency**" tab. Then click "+" -> "**File dependency**"



- Select the lib file.



- Repeat the above steps to add all libs and you will see all libs are added to the list.



3 Application Programming Interface

3.1 VCIL Management Functions

3.1.1 Usage

3.1.1.1 Basic Usage

Windows/Linux

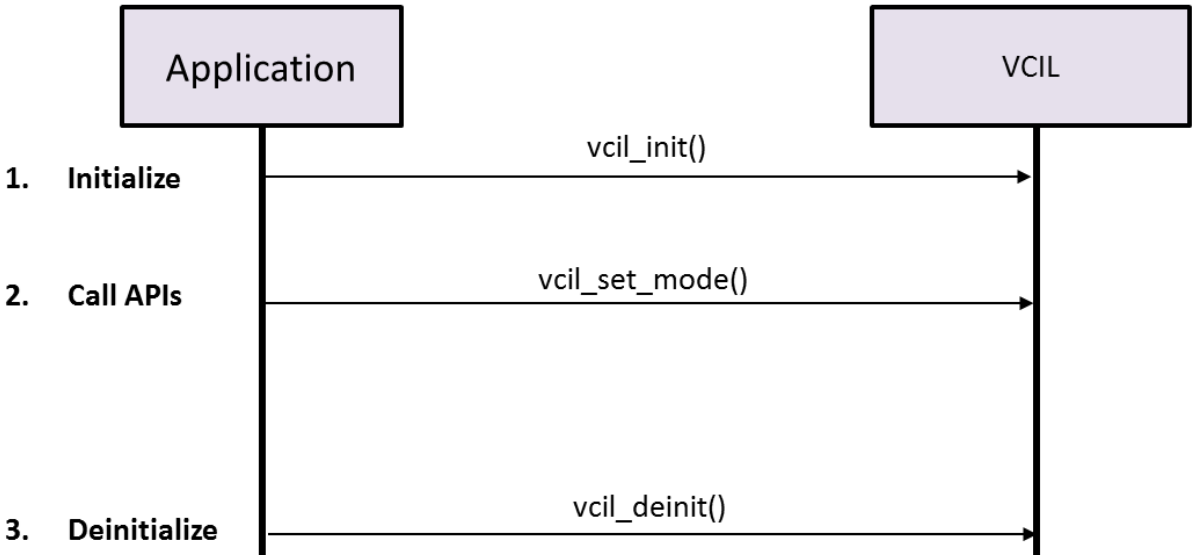
Please refer to the VCIL Conventions for basic usage for Windows/Linux.

Android

Please refer to the VCIL Conventions for basic usage for Android.

3.1.1.2 Protocol Mode Setting

Windows/ Linux/ Android



To start using VCIL APII module and related modules, you should first call vcil_init() before using the other VCIL APIs. To stop using VCIL module, you must call vcil_deinit() to close API.

To set the activated protocols for each port, you can use vcil_set_mode() to set.

The available modes for each CAN Port are as followings:

- (0) **VCIL_MODE_CAN -** CAN protocol **(DEFAULT)**
- (1) **VCIL_MODE_J1939 -** J1939 protocol
- (2) **VCIL_MODE_ODB2 -** OBD2 protocol

The available modes for each J1708 Port are as followings:

- (3) **VCIL_MODE_J1708** - J1708 protocol **(DEFAULT)**
- (4) **VCIL_MODE_J1587** - J1587 protocol

For example,
to activate CAN on CAN port 0, J1939 on CAN port 1 and J1708 on J1708 port 0, you can call the API with following parameters -
vcil_set_mode(VCIL_MODE_CAN, VCIL_MODE_J1939, VCIL_MODE_J1708).

To reset the MCU, you can use vcil_firmware_reset().

3.1.2 Enumeration

3.1.2.1 Windows/Linux

vcil_mode Enum

- (0) **VCIL_MODE_CAN** - Active at RAW CAN mode.
- (1) **VCIL_MODE_J1939** - Active at J1939 mode.
- (2) **VCIL_MODE_OBD2** - Active at OBD2 mode.
- (3) **VCIL_MODE_J1708** - Active at J1708 mode.
- (4) **VCIL_MODE_J1708** - Active at J1708 mode.

3.1.2.2 Android

VCIL_MODE

class:

mrm.define.MRM_ENUM.**VCIL_MODE**

Enum:

Name	Type	Value	Comment
VCIL_MODE_CAN	int	0	Active at CAN mode.
VCIL_MODE_J1939	int	1	Active at J1939 mode.
VCIL_MODE_OBD2	int	2	Active at OBD2 mode.
VCIL_MODE_J1708	int	3	Active at J1708 mode.
VCIL_MODE_J1587	int	4	Active at J1587 mode.

Remark:

Use the method **getValue()** to get the enum value.

ex: MRM_ENUM.VCIL_MODE.VCIL_MODE_J1939**.getValue()** returns 1.

Please refer to sample code for detailed usage.

3.1.3 Constant

3.1.3.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_EVENT_ID_UNKNOWN	int	-1
VCIL_EVENT_ID_RECEIVED_MSG_CAN	int	0
VCIL_EVENT_ID_RECEIVED_MSG_J1939	int	1
VCIL_EVENT_ID_RECEIVED_MSG_OBD2	int	2
VCIL_EVENT_ID_RECEIVED_MSG_J1708	int	3
VCIL_EVENT_ID_RECEIVED_MSG_J1708	int	4

3.1.4 APIs

3.1.4.1 vcil_init

Syntax:

Windows / Linux	mrm_err **vcil_init**(char *port)
Android	int **vcil_init**(String port)

Description:

Initialize the VCIL library.

Parameters:

port [in]

Pointer to a buffer that will hold the string of VCIL device path. For C, the path string is end of '\0'.

Example:

port = "\\.\COM7" for Windows

port = "/dev/ttyA0" for Linux

port = "/dev/ttyA0" for Android (default)

port = "/dev/ttyUSB7" for Android (TREK-734 only)

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

Prior to calling any VCIL API function the library needs to be initialized by calling this function. The return code for all VCIL API function will be **MRM_ERR_LIBRARY_NOT_INIT** unless this function is called.

3.1.4.2 vcil_deinit

Syntax:

Windows / Linux	mrm_err **vcil_deinit**(void)
Android	int **vcil_deinit**()

Description:

Deinitialize the VCIL library.

Parameters:

None.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.1.4.3 vcil_get_version

Syntax:

Windows / Linux	mrm_err **vcil_get_version**(char *version)
Android	int **vcil_get_version**(byte[] version)

Description:

Get the version of SDK.

Parameters:

version [out]

Pointer to a buffer that will hold the version of SDK. The buffer is C string that end of '\0'. The content of unused bytes filled 0x00.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

The maximum length of version string is **IVCP_MAXIMUM_LIBRARY_STRING_LENGTH**(24)

3.1.4.4 vcil_set_mode

Syntax:

Windows / Linux	mrm_err **vcil_set_mode**(vcil_mode can_port0, vcil_mode can_port1, vcil_mode j1708_port0)
Android	int **vcil_set_mode**(int can_port0, int can_port1, int j1708_port0)

Description:

Set the protocol mode of each port.

Parameters:

can_port0 [in]

Setup the CAN port 0 protocol mode.

can_port1 [in]

Setup the CAN port 1 protocol mode.

j1708_port0 [in]

Setup the J1708 port 0 protocol mode

For the definition of mode ID,

For Windows/Linux, please refer to vcil_mode.

For Android, please refer to VCIL_MODE.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.1.4.5 vcil_get_mode

Syntax:

Windows / Linux	mrm_err **vcil_get_mode**(vcil_mode *can_port0, vcil_mode *can_port1, vcil_mode *j1708_port0)
Android	int **vcil_get_mode**(int[] can_port0, int[] can_port1, int[] j1708_port0)

Description:

Get the protocol mode of each port.

Parameters:

can_port0 [out]

The current CAN port 0 protocol mode.

can_port1 [out]

The current CAN port 1 protocol mode.

j1708_port0 [out]

The current J1708 port 0 protocol mode.

For the definition of mode ID,

For Windows/Linux, please refer to vcil_mode.

For Android, please refer to VCIL_MODE.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.2 Firmware Management Functions

3.2.1 Usage

3.2.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.

Android

Please refer to the VCIL Conventions for basic usage for Android.

3.2.2 Constant

3.2.2.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAXIMUM_FIRMWARE_VERSION_LENGTH	int	16

3.2.3 APIs

3.2.3.1 vcil_firmware_get_version

Syntax:

Windows / Linux	mrm_err **vcil_firmware_get_version**(char *version)
Android	int **vcil_firmware_get_version** (byte[] version)

Description:

Get the version of firmware.

Parameters:

version [out]

Pointer to a buffer that will hold the version of firmware. The buffer is C string that end of '\0'. The content of unused bytes filled 0x00.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

The maximum length of version string is **VCIL_MAXIMUM_FIRMWARE_VERSION_LENGTH**(16)

3.2.3.2 vcil_firmware_reset

Syntax:

Windows / Linux	mrm_err **vcil_firmware_reset**(void)
Android	int **vcil_firmware_reset**()

Description:

Reset the VCIL firmware.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

All function mask/filter will be reset after call this function.

3.3 CAN Functions

3.3.1 Usage

3.3.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.

Also, before using CAN related APIs, you must first set the protocol mode of proper CAN port to **CAN mode.** Please refer to the protocol mode setting section.

Android

Please refer to the VCIL Conventions for basic usage for Android.

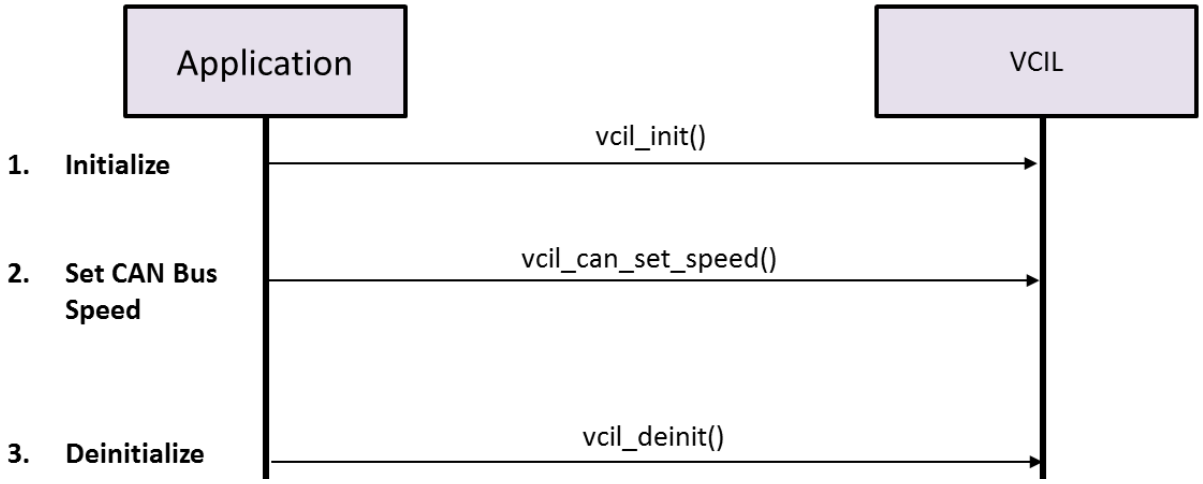
Also, before using CAN related APIs, you must first set the protocol mode of proper CAN port to **CAN mode**. Please refer to the protocol mode setting section.

3.3.1.2 CAN Bus Speed Setting

Windows/ Linux/ Android

To start data transmission of CAN, J1939, OBD2 protocol on CAN bus, you must first configure the CAN bus speed for MCU.

This following figure describes how to set CAN bus speed through SDK.



3.3.1.3 CAN Message Reading

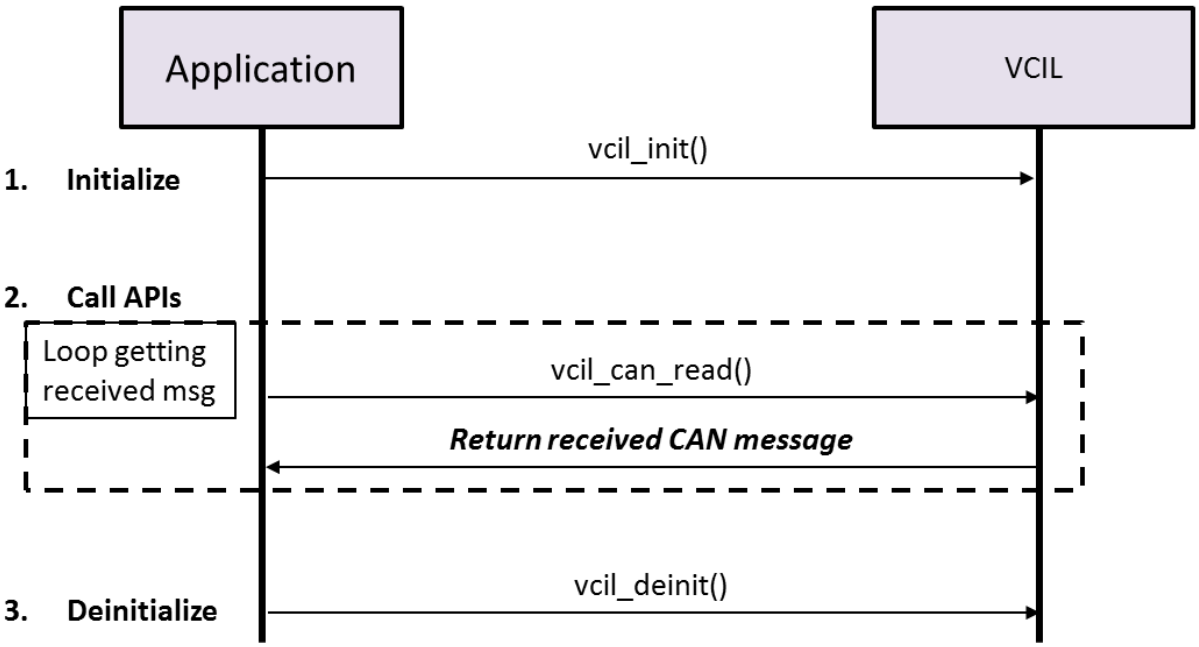
This section describes how to read CAN messages from CAN bus by using SDK APIs.

The CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.

Windows/Linux

You can implement your CAN message reading application in either **Polling** or **Event Handling** style.

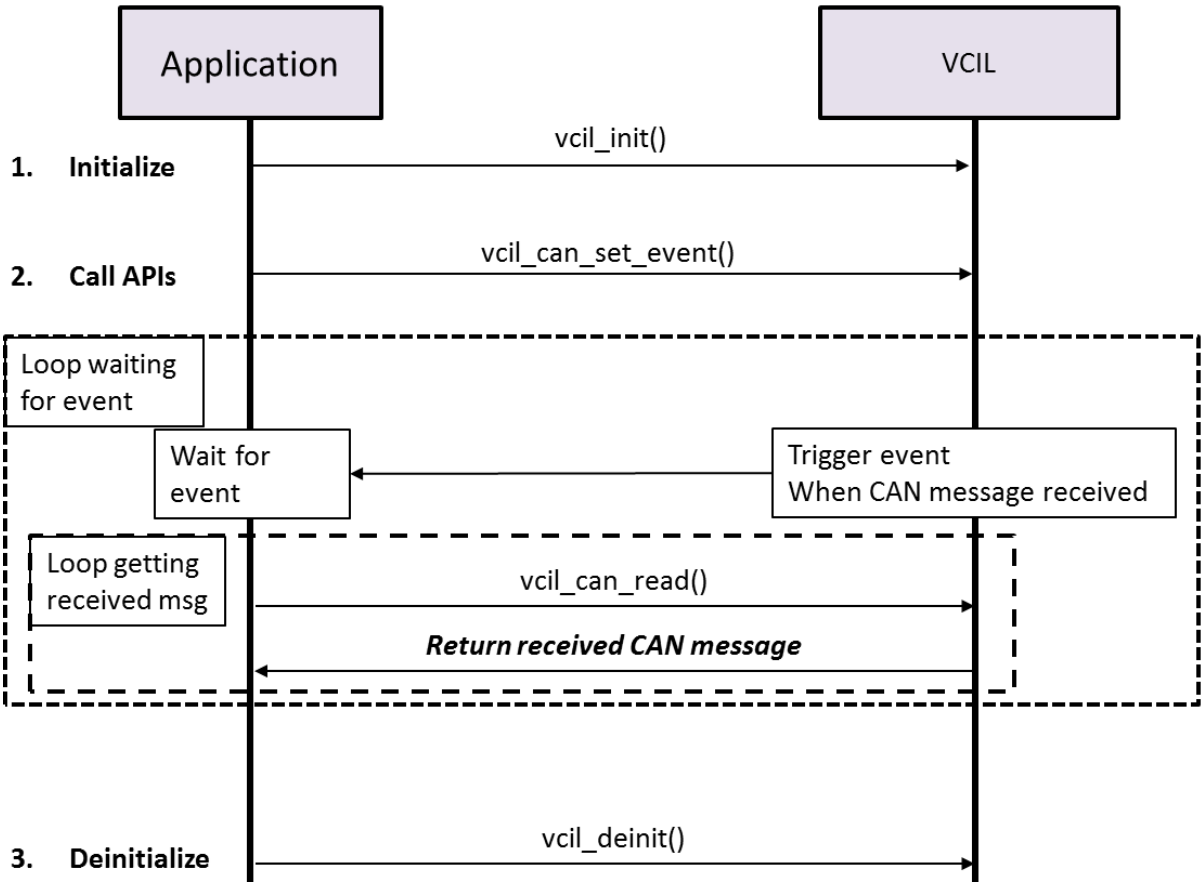
- **Polling:**



The most simple way to get received message is to hold a loop in your APP and keep calling vcil_can_read() to get received message from SDK internal buffer.

The advantage of reading messages in polling style is that it is simple and relatively lower overhead (i.e. no event handling) to read a message.
The disadvantage is that you need to keep your APP's process reading message even when there is actually no message in SDK internal buffer, which may result in unnecessary power consuming.

- **Event Handling:**



To read messages in event handling style, you need to call vcil_can_set_event() first to register an event to CAN API module. When MCU receive a CAN message from CAN bus, SDK adds the received message to internal buffer and trigger the registered event.

You need to hold an event listening loop in your APP to handle the registered event. When the event is triggered, you can then run a loop to calling vcil_can_read() to get and consume the received messages from the internal buffer. After all messages in buffer are consumed, you should keep listening the registered event. Please refer to the sample code for the details of implementation.

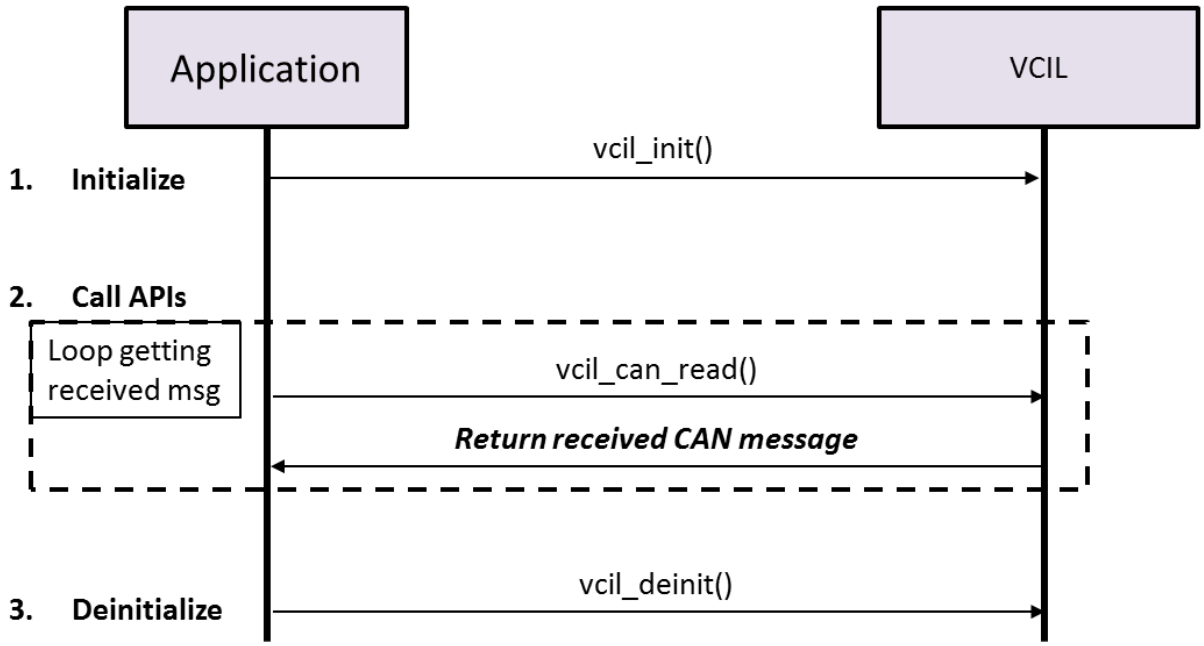
The advantage of reading messages in event handling style is that APP's process only work when there are messages can be read, which relatively cost less system resources and power.
The disadvantage is that there would be some overhead of event handling to read a message.

Please refer to the sample code for implementation details.

Android

You can implement your CAN message reading application in either **Polling** or **Event Handling** style.

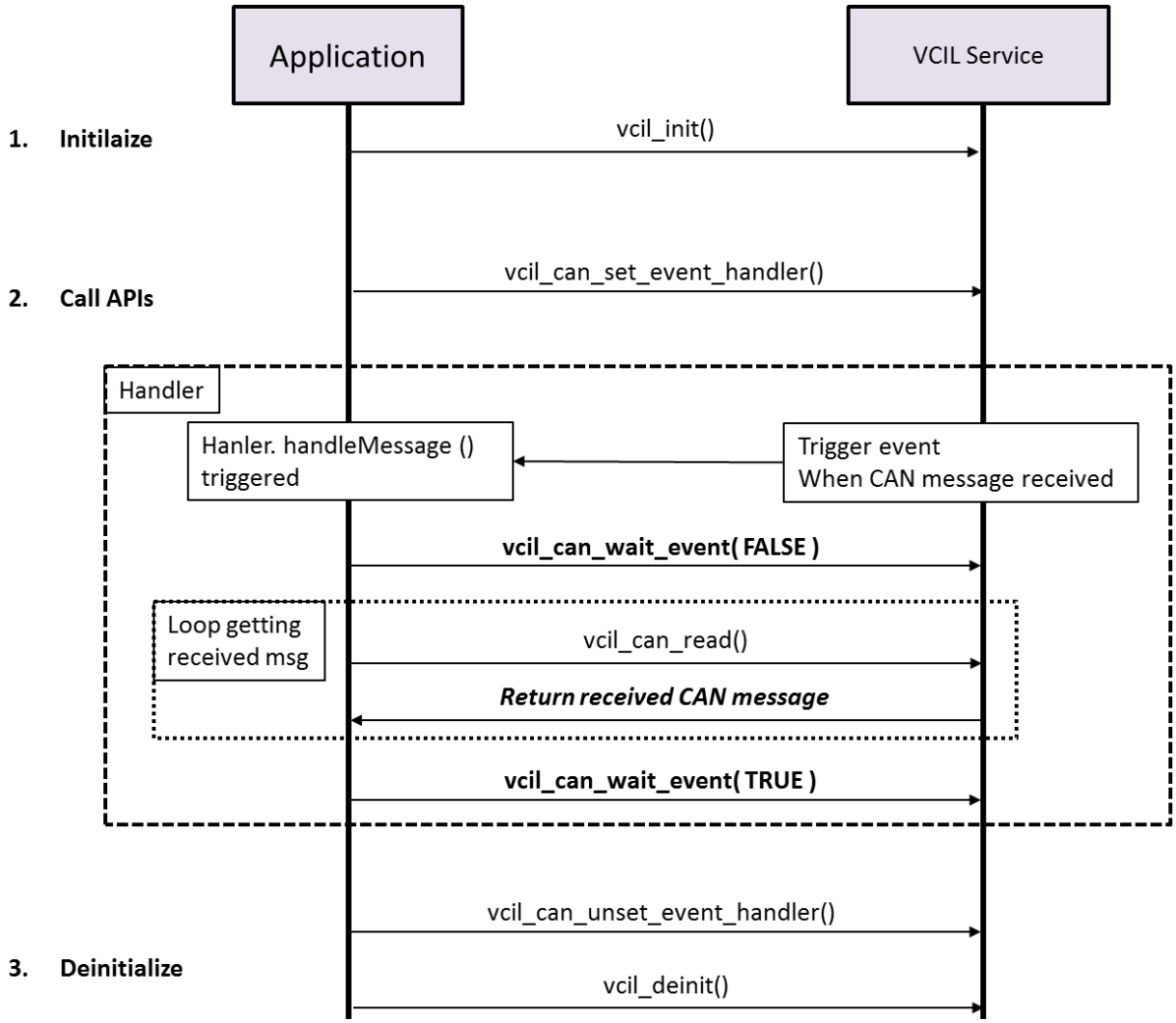
- **Polling:**



The most simple way to get received message is to hold a loop in your application and keep calling vcil_can_read() or vcil_can_read_multi() to get received message(s) from SDK internal buffer.

The advantage of reading messages in polling style is that it is simple and relatively lower overhead (i.e. no event handling) to read message(s).
The disadvantage is that you need to keep your application process reading message even when there is actually no message in SDK internal buffer, which may result in unnecessary power consuming.

- **Event Handling:**



MRM SDK leverage the Android Handler mechanism to inform CAN message receive event.

To read messages in event handling style, you need to create an instance of Handler and call vcil_can_set_event_handler() first to register the Handler instance to VCIL. When MCU receive a CAN message from CAN bus, VCIL adds the received message to internal buffer and trigger the event to inform registered handler.

When the event is triggered, the handleMessage() callback of registered handler instance will be triggered. you can then run a loop to calling vcil_can_read() or vcil_can_read_multi() to get and consume the received messages from the internal buffer.

Due to that VCIL will trigger event whenever it receive a CAN message, to avoid from getting multiple unnecessary events, you should call vcil_can_wait_event(FALSE) to ask VCIL temporarily stop passing event to handler before your APP start the alarm data getting loop.
After all messages in buffer are consumed, you should call vcil_can_wait_event(TRUE) to ask VCIL continue the event informing.

If you do not need to listen to alarm event anymore, you should call vcil_can_unset_event_handler() to

unregister the Handler instance from VCIL.

The advantage of reading messages in event handling style is that application process only work when there are messages can be read, which relatively cost less system resources and power.
The disadvantage is that there would be some overhead of event handling to read a message.

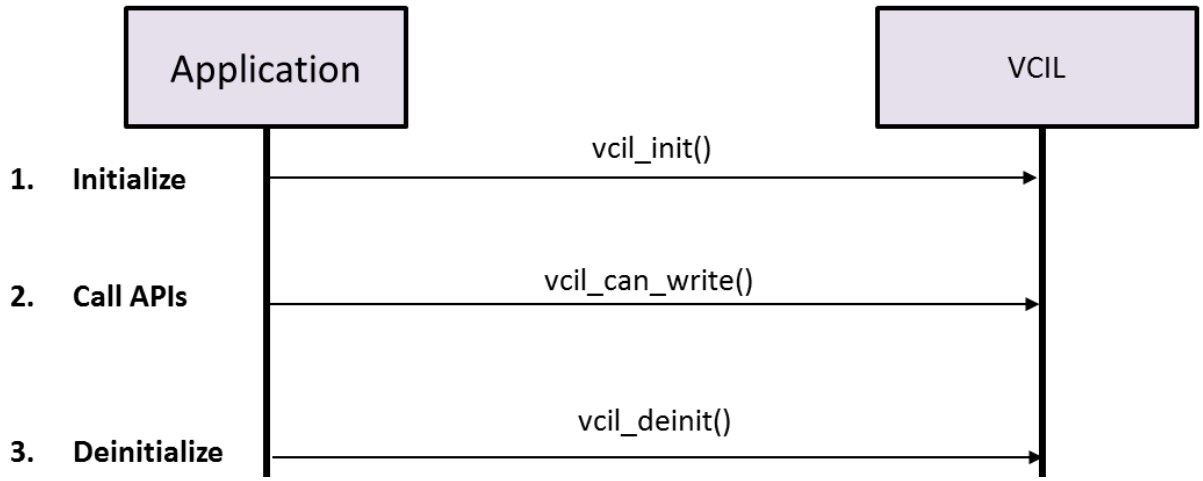
Please refer to the sample code for implementation details.

3.3.1.4 CAN Message Writing

Windows/ Linux/ Android

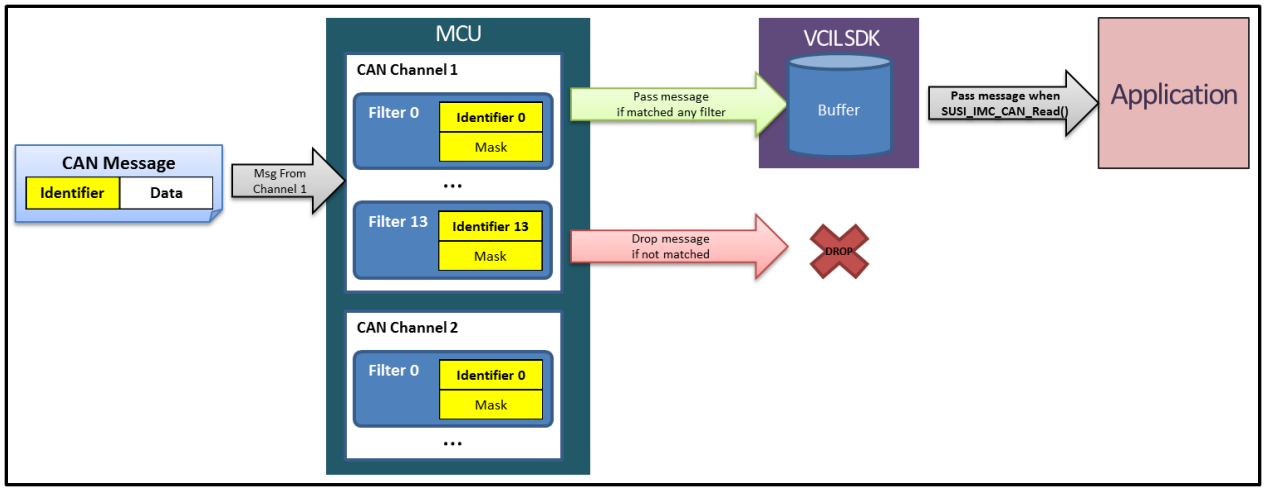
This following figure describes how to write CAN messages to CAN bus by using SDK APIs.

The CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.



3.3.1.5 CAN Acceptance filter Settings

In the car system there might be many nodes on the bus and the number of CAN messages transmitted to the bus might be enormous. Due to the performance and application purpose concerns, it might not be a good idea to try to process all messages on the bus. To focus on the messages you interest in, you can use the filter functions provided by VCIL MCU.



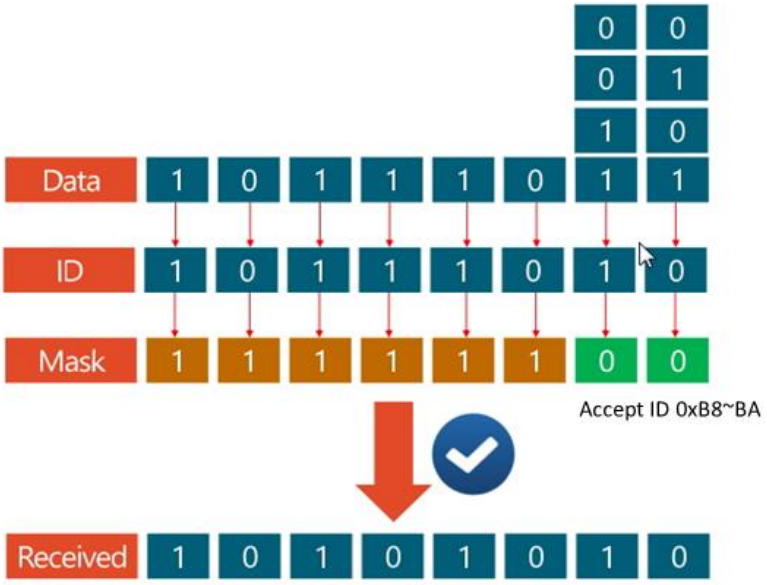
The VCIL MCU provides 14 configurable CAN identifier filter banks for filtering the incoming messages for each CAN channel. The filters act as "white list". If the CAN channel is configured to work with filters, the MCU which will only receive the CAN messages which match the filter conditions and drop others.

For each filter, a identifier value and mask is set. The identifier value defines a desired identifier and the mask defined the bits of identifier the filter care about. If the mask of filter is set to 0, then it means the filter "don't care" any bit of the identifier and all CAN messages from the bus will be passed through this filter. When VCIL MCU get a CAN message from bus, it AND the identifier of message and the identifier of each filter with mask to test whether the message should be passed or not.

The following are examples of filter mask setting.

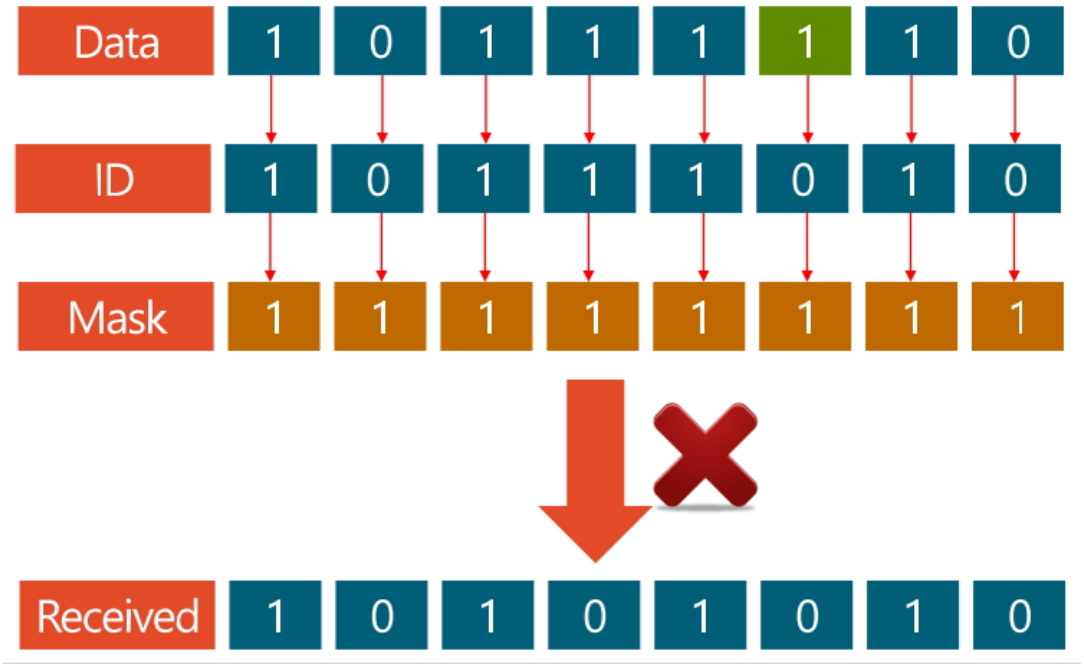
- Accepted example

If we set a filter with mask value = [1 1 1 1 1 1 0 0] (0xFC) and identifier = [1 0 1 1 1 0 1 0](0xBA), then MCU will care the first 6 bits of the identifier.
Thus MCU will only accept messages with identifier 0xB8~BA. (Binary:1011 = Hex:B)



- Unaccepted example

If we set a filter with mask value = [1 1 1 1 1 1 1 1] (0xFF) and identifier = [1 0 1 1 1 0 1 0](0xBA), then MCU will care the all bits of the identifier.
Thus MCU will only accept messages with identifier 0xBA.



The following shows examples of filtering.

- **Example - Single filter**

CAN MASK Configuration	Receive CAN data	Result
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111111b (0xFF)	10111010 (0xBA)	**Accept**
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111111b (0xFF)	10111011 (0xBB)	**Drop**

CAN MASK Configuration	Receive CAN data	Result
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111100b (0xFC)	10111000 (0xB8)	**Accept**
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111100b (0xFC)	10111001 (0xB9)	**Accept**
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111100b (0xFC)	10111010 (0xBA)	**Accept**
Filter 1: CAN ID=10111010b (0xBA) Mask = 11111100b (0xFC)	10111100 (0xBC)	**Drop**

- **Example - Multiple filters**

You can set multiple filter at same time for multiple range of targets. For example, ID 0x28~0x37 and 0x3000~0x31FF and 0x1600

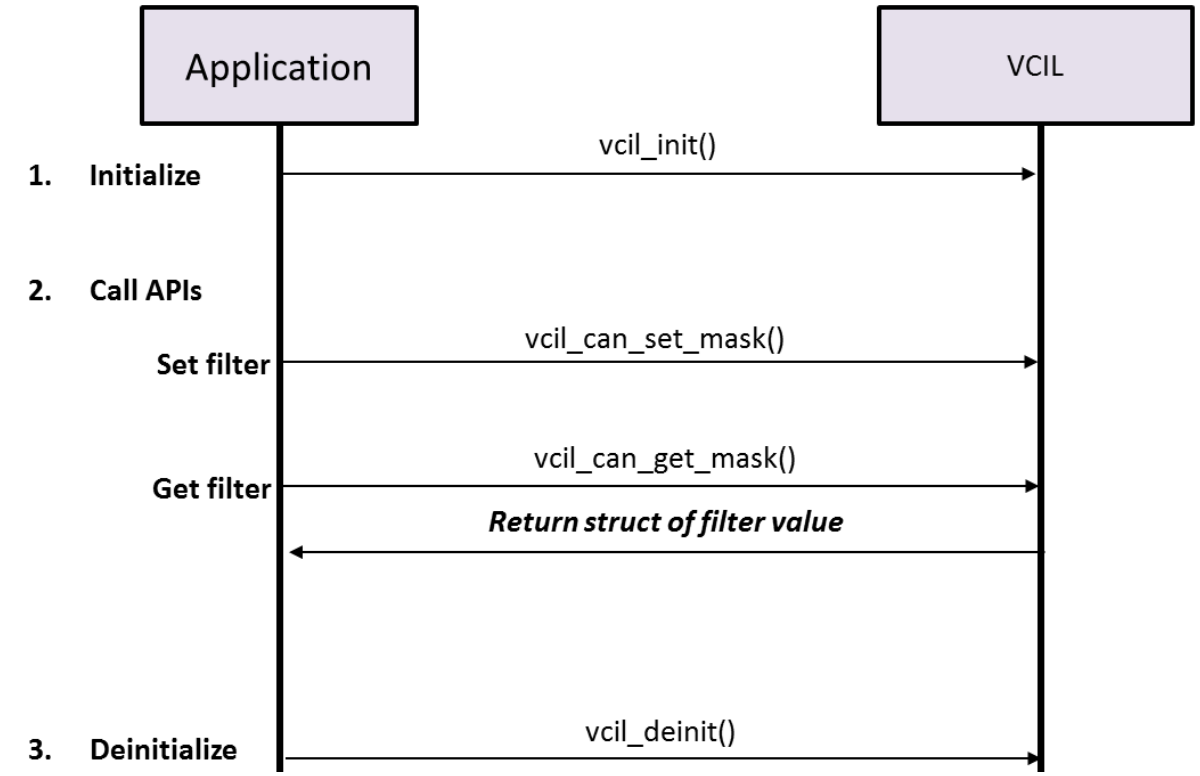
CAN MASK Configuration	Receive CAN data	Result

Filter 1: CAN ID=0000000000101000b (0x0028), Mask = 1111111111111110b (0xFFFE) Filter 2: CAN ID=0000000000110000b (0x0030) Mask = 1111111111111000b (0xFFF8) Filter 3: CAN ID=0011000000000000b (0x3000) Mask = 1111111000000000b (0xFE00) Filter 4: CAN ID=0001011000000000b (0x1600) Mask = 1111111111111111b (0xFFFF)	00101000b(0x29)	**Accept**
Filter 1: CAN ID=0000000000101000b (0x0028) Mask = 1111111111111110b (0xFFFE) Filter 2: CAN ID=0000000000110000b (0x0030) Mask = 1111111111111000b (0xFFF8) Filter 3: CAN ID=0011000000000000b (0x3000) Mask = 1111111000000000b (0xFE00) Filter 4: CAN ID=0001011000000000b (0x1600) Mask = 1111111111111111b (0xFFFF)	01001000b(0x48)	**Drop**
Filter 1: CAN ID=0000000000101000b (0x0028) Mask = 1111111111111110b (0xFFFE) Filter 2: CAN ID=0000000000110000b (0x0030) Mask = 1111111111111000b (0xFFF8) Filter 3: CAN ID=0011000000000000b (0x3000) Mask = 1111111000000000b (0xFE00) Filter 4: CAN ID=0001011000000000b (0x1600) Mask = 1111111111111111b (0xFFFF)	0011000100000010b(0x3102)	**Accept**
Filter 1: CAN ID=0000000000101000b (0x0028) Mask = 1111111111111110b (0xFFFE) Filter 2: CAN ID=0000000000110000b (0x0030) Mask = 1111111111111000b (0xFFF8) Filter 3: CAN ID=0011000000000000b (0x3000) Mask = 1111111000000000b (0xFE00) Filter 4: CAN ID=0001011000000000b (0x1600) Mask = 1111111111111111b (0xFFFF)	0011001000000000b(0x3200)	**Drop**

Windows/ Linux/ Android

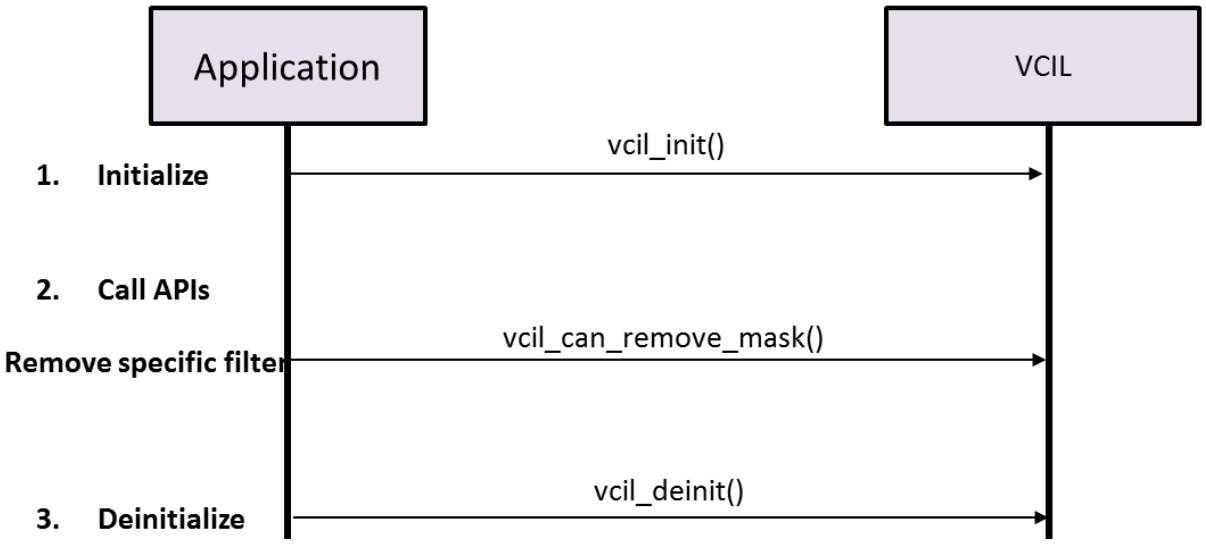
- **To get/set filter:**

Call vcil_can_set_mask() to set the mask and call vcil_can_get_mask() to get mask.



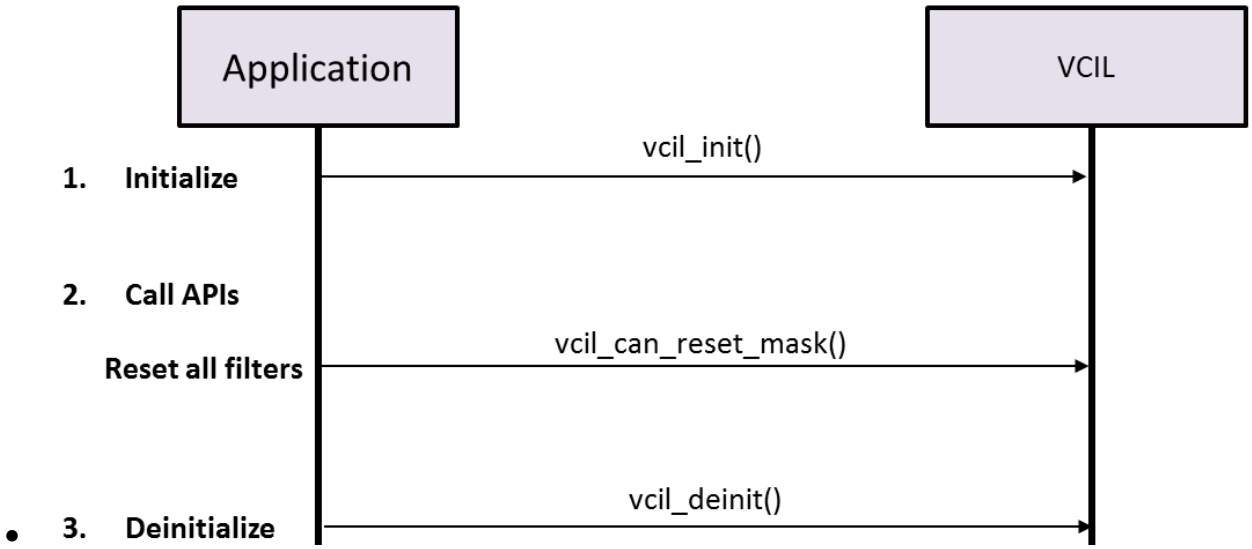
- **To remove specific filter:**

Call vcil_can_remove_mask() to remove filter of specified filter bank.



- **To remove all filter:**

Call vcil_can_reset_mask() to reset all filter.


Application
VCIL
vcil_init()
1. Initialize
2. Call APIs
vcil_can_reset_mask()
Reset all filters
vcil_deinit()
3. Deinitialize

3.3.2 Enumeration

3.3.2.1 Windows/Linux

vcil_can_speed Enum

- (0) **VCIL_CAN_SPEED_125K** - The CAN bus speed at 125Kbits.
- (1) **VCIL_CAN_SPEED_250K** - The CAN bus speed at 250Kbits.
- (2) **VCIL_CAN_SPEED_500K** - The CAN bus speed at 500Kbits.
- (3) **VCIL_CAN_SPEED_1M** - The CAN bus speed at 1Mbits.
- (4) **VCIL_CAN_SPEED_200K** - The CAN bus speed at 200Kbits.
- (5) **VCIL_CAN_SPEED_100K** - The CAN bus speed at 100Kbits.
- (6) **VCIL_CAN_SPEED_800K** - The CAN bus speed at 800Kbits.
- (7) **VCIL_CAN_SPEED_83K** - The CAN bus speed at 83.333Kbits.
- (8) **VCIL_CAN_SPEED_50K** - The CAN bus speed at 50Kbits.
- (9) **VCIL_CAN_SPEED_20K** - The CAN bus speed at 20Kbits.
- (10) **VCIL_CAN_SPEED_10K** - The CAN bus speed at 10Kbits.
- (11) **VCIL_CAN_SPEED_5K** - The CAN bus speed at 5Kbits.
- (0xFF) **VCIL_CAN_SPEED_USER_DEFINE** - current bit-rate is user-define.

vcil_can_bus_mode Enum

- (0) **VCIL_CAN_BUS_NORMAL_MODE** -
  CAN controller operates in normal mode. In normal mode, CAN controller synchronize the bit traffic on the CAN bus and is able to receive/transmit CAN messages.

- (1) **VCIL_CAN_BUS_LISTEN_MODE** -
  CAN controller operates in listen mode. In listen mode, the CAN controller is only able to receive valid data frames and remote request frames and NOT able to transmit. The CAN controller only monitor the bit traffic on bus without interfering the bus (e.g. keep in recessive state) and NO dominant bit will be sent (i.e. Acknowledge Bits, Error Frames) .
  Listen mode can be used to monitor the traffic on a CAN bus without interfering the bus.

- (2) **VCIL_CAN_BUS_INIT_MODE** -
- CAN controller operates in initiation mode. In Initialization Mode, the CAN controller is uninitialized and stops transmitting and receiving to/from the CAN bus, and dose not interfere the bus traffic (keeps bus output in recessive state).

3.3.2.2 Android

VCIL_CAN_SPEED

class:

mrm.define.MRM_ENUM.**VCIL_CAN_SPEED**

Enum:

Name	Type	value	Comment
VCIL_CAN_SPEED_125K	int	0	125 kbit/s
VCIL_CAN_SPEED_250K	int	1	250 kbit/s
VCIL_CAN_SPEED_500K	int	2	500 kbit/s
VCIL_CAN_SPEED_1M	int	3	1M bit/s
VCIL_CAN_SPEED_200K	int	4	200 kbit/s
VCIL_CAN_SPEED_100K	int	5	100 kbit/s
VCIL_CAN_SPEED_800K	int	6	800 kbit/s
VCIL_CAN_SPEED_83K	int	7	83 kbit/s
VCIL_CAN_SPEED_50K	int	8	50 kbit/s
VCIL_CAN_SPEED_20K	int	9	20 kbit/s
VCIL_CAN_SPEED_10K	int	10	10 kbit/s
VCIL_CAN_SPEED_5K	int	11	5 kbit/s
VCIL_CAN_SPEED_USER_DEFINE	int	0xFF	user-defined bit-rate

Remark:

Use the method **getValue()** to get the enum value.

ex: MRM_ENUM.CAN_SPEED.VCIL_CAN_SPEED_250K**.getValue()** returns 1.

Please refer to sample code for detailed usage.

VCIL_CAN_BUS_MODE

class:

mrm.define.MRM_ENUM**.VCIL_CAN_BUS_MODE**

Enum:

Name	Type	value	Comment
VCIL_CAN_BUS_NORMAL_MODE	int	0	CAN controller operates in normal mode. In normal mode, CAN controller synchronize the bit traffic on the CAN bus and is able to receive/transmit CAN messages.
VCIL_CAN_BUS_LISTEN_MODE	int	1	CAN controller operates in listen mode. In listen mode, the CAN controller is only able to receive valid data frames and remote request frames and NOT able to transmit. The CAN controller only monitor the bit traffic on bus without interfering the bus (e.g. keep in recessive state) and NO dominant bit will be sent (i.e. Acknowledge Bits, Error Frames) . Listen mode can be used to monitor the traffic on a CAN bus without interfering the bus.
VCIL_CAN_BUS_INIT_MODE	int	2	CAN controller operates in initiation mode. In Initialization Mode, the CAN controller is uninitialized and stops transmitting and receiving to/from the CAN bus, and keep the bus output in recessive status.

Remark:

Use the method **getValue()** to get the enum value.

ex: MRM_ENUM.VCIL_CAN_BUS_MODE.VCIL_CAN_BUS_LISTEN_MODE**.getValue()** returns 1.

Please refer to sample code for detailed usage.

3.3.3 Constant

3.3.3.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAX_CAN_DATA_SIZE	int	8

3.3.4 Structure/Classes

3.3.4.1 Windows/Linux

vcil_can_message_t Structure

Syntax:

```
typedef struct
{
        unsigned char port;
        char length;
        bool remote_request;
        bool extended_frame;
        unsigned int id;
        unsigned char data[8];
} vcil_can_message_t;
```

Description:

This data structure defines the CAN message.

Members:

port

This message come from/send to which port.

length

The standard CAN message data length. This data length should not over VCIL_MAX_CAN_DATA_SIZE(8).

remote request

This field is used to indicate whether this CAN message is a remote transmit request(RTR) frame.

The value is 1 if the message is a RTR frame(the RTR field of the CAN message identifier is 1).

The value is 0 if the message is not a RTR frame.

extended frame

This field is used to indicate that the message is a standard format(CAN2.0A) or a extended format(CAN2.0B) message.

The value is 1 if the message is a CAN2.0B message (with 29-bits identifier).

The value is 0 if the message is a CAN2.0A message (with 11-bits identifier).

id

The Identifier of CAN.

data

The data array of CAN message.

vcil_can_mask_t Structure

Syntax:

```
typedef struct
{
        unsigned char type;
        unsigned char bank;
        bool remote_request;
        bool extended_frame;
        unsigned int id1;
        unsigned int mask1;
        unsigned int id2;
        unsigned int mask2;
} vcil_can_mask_t;
```

Description:

This data structure defines the hardware CAN mask.

Members:

type

This mask type. This field reserved for furture configuration currently ingnore.

bank

The mask bank, the VCIL supported maximum 14 bank which 0~13. This bank should not over 13.

You can think of the bank is a rule of CAN message hardware filter.

remote request

This field is used to indicate whether this CAN message is a remote transmit request(RTR) frame.

The value is 1 if the message is a RTR frame(the RTR field of the CAN message identifier is 1).

The value is 0 if the message is not a RTR frame.

extended frame

This field is used to indicate that the message is a standard format(CAN2.0A) or a extended format(CAN2.0B) message.

The value is 1 if the message is a CAN2.0B message (with 29-bits identifier).

The value is 0 if the message is a CAN2.0A message (with 11-bits identifier).

id1

The Identifier 1 of bank.

mask1

The mask 1 of the bank.

id2

The Identifier 2 of bank.

This field will be ignored if **extended frame is set to 1.**

mask2

The mask 2 of the bank.

This field will be ignored if **extended frame is set to 1.**

vcil_can_error_status Structure

Syntax:

```
typedef struct
{
        unsigned int rec; // receive error counter
        unsigned int tec; // transmit error counter
        unsigned int last_error_code; // last error code
        unsigned int error_flag;
} vcil_can_error_status;;
```

Description:

This CAN controller error status.

Members:

rec

receive error counter. The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.

tec

Transmit error counter The implementing part of the fault confinement mechanism of the CAN protocol.

last_error_code

This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to '0'. The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status.

000: No Error
001: Stuff Error
010: Form Error
011: Acknowledgment Error
100: Bit recessive Error
101: Bit dominant Error
110: CRC Error
111: Set by software

error_flag

CAN Bus error flag

bit0: Error warning flag

This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96).

bit1: Error passive flag

This bit is set by hardware when the Error Passive limit has been reached (Receive VCIM Command Specification 124 / 153 Error Counter or Transmit Error Counter>127).

bit2: Bus-off flag

This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255

3.3.4.2 Android

VCIL_CAN_MESSAGE

class:

mrm.define.VCIL.**VCIL_CAN_MESSAGE**

Fields:

Field Name	Type	Input/Output	Comment
port	int	in, out	The port ID which this CAN message is received from/sent to.
length	int	in, out	Length of data of the CAN message. This data length should not over VCIL_MAX_CAN_DATA_SIZE(8).
remote_request	boolean	in, out	This field is used to indicate whether this CAN message is a remote transmit request(RTR) frame. The value is TRUE if the message is a RTR frame(the RTR field of the CAN message identifier is 1). The value is FALSE if the message is not a RTR frame.
extended_frame	boolean	in, out	This field is used to indicate that the message is a standard format(CAN2.0A) or a extended format(CAN2.0B) message. The value is TRUE if the message is a CAN2.0B message (with 29-bits identifier). The value is FALSE if the message is a CAN2.0A message (with 11-bits identifier).
id	int	in, out	The Identifier of the CAN message.
data	byte[]	in, out	A byte array containing the data of the CAN message.

VCIL_CAN_MASK

class:

mrm.define.VCIL**.VCIL_CAN_MASK**

Fields:

Field Name	Type	Input/Output	Comment
type	byte	-	The mask type. This field is reserved for future use and currently ignored.
bank	byte	in, out	The mask bank, the VCIL supported maximum 14 bank which 0~13. This bank should not over 13. You can think of the bank is a rule of CAN message hardware filter.
remote_request	boolean	in, out	This field is used to indicate whether this CAN message is a remote transmit request(RTR) frame. The value is TRUE if the message is a RTR frame(the RTR field of the CAN message identifier is 1). The value is FALSE if the message is not a RTR frame.
extended_frame	boolean	in, out	This field is used to indicate that the message is a standard format(CAN2.0A) or a extended format(CAN2.0B) message. The value is TRUE if the message is a CAN2.0B message (with 29-bits identifier). The value is FALSE if the message is a CAN2.0A message (with 11-bits identifier).
id1	int	in, out	The Identifier 1 of bank.

Field Name	Type	Input/Output	Comment
mask1	int	in, out	The mask 1 of the bank.
id2	int	in, out	The Identifier 2 of bank.
mask2	int	in, out	The mask 2 of the bank.

VCIL_CAN_ERROR_STATUS

class:

mrm.define.VCIL.**VCIL_CAN_ERROR_STATUS**

Fields:

Field Name	Type	Input/Output	Comment
rec	int	out	Receive error counter. The implementing part of the fault confinement mechanism of the CAN protocol. In case of an error during reception, this counter is incremented by 1 or by 8 depending on the error condition as defined by the CAN standard. After every successful reception the counter is decremented by 1 or reset to 120 if its value was higher than 128. When the counter value exceeds 127, the CAN controller enters the error passive state.
tec	int	out	Transmit error counter. The implementing part of the fault confinement mechanism of the CAN protocol.
last_error_code	int	out	This field is set by hardware and holds a code which indicates the error condition of the last error detected on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared to ‘0’. The LEC[2:0] bits can be set to value 0b111 by software. They are updated by hardware to indicate the current communication status. 000: No Error 001: Stuff Error 010: Form Error 011: Acknowledgment Error 100: Bit recessive Error

Field Name	Type	Input/Output	Comment
			101: Bit dominant Error 110: CRC Error 111: Set by software
error_flag	int	out	CAN Bus error flag. bit0: Error warning flag This bit is set by hardware when the warning limit has been reached (Receive Error Counter or Transmit Error Counter≥96). bit1: Error passive flag This bit is set by hardware when the Error Passive limit has been reached (Receive VCIM Command Specification 124 / 153 Error Counter or Transmit Error Counter>127). bit2: Bus-off flag This bit is set by hardware when it enters the bus-off state. The bus-off state is entered on TEC overflow, greater than 255

3.3.5 APIs

3.3.5.1 vcil_can_read

Syntax:

Windows / Linux	mrm_err **vcil_can_read**(vcil_can_message_t *message)
Android	int **vcil_can_read**(VCIL_CAN_MESSAGE message)

Description:

Get a CAN message from VCIL library CAN buffer if available otherwise you may get a error code(**MRM_ERR_VCIL_DATA_NOT_READY**) and a invalid CAN message.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_can_message_t struct which is used to store received CAN message

Android:

Instance of VCIL_CAN_MESSAGE which is used to store received CAN message

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive CAN message from each CAN port.

3.3.5.2 vcil_can_read_multi

Syntax:

Windows / Linux	-
Android	int **vcil_can_read_multi**(List<VCIL_CAN_MESSAGE> messages, int desiredReadNum, int[] resultReadNum)

Description:

Read multiple received CAN messages from the SDK internal buffer.

Parameters:

message [out]

Android:

List of VCIL_CAN_MESSAGE which is used to store received CAN messages.

desiredReadNum [in]

The number of CAN message you expect to get.

resultReadNum [out]

An allocated array of size 1 for storing the number of CAN message the SDK actually returned.

The return value will be stored at index 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.3.5.3 vcil_can_write

Syntax:

Windows / Linux	mrm_err **vcil_can_write**(vcil_can_message_t *message)
Android	int **vcil_can_write**(VCIL_CAN_MESSAGE message)

Description:

Write a CAN message to specified CAN port.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_can_message_t struct that store the CAN message to be sent.

Android:

Instance of VCIL_CAN_MESSAGE which stores the CAN message to be sent.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive CAN message from each CAN port.

3.3.5.4 vcil_can_set_speed

Syntax:

Windows / Linux	mrm_err **vcil_can_set_speed**(unsigned char port, vcil_can_speed speed)
Android	int **vcil_can_set_speed**(byte port, int speed)

Description:

Set the specified CAN port bus baud rate.

Parameters:

port [in]

The CAN port. The first port is 0.

speed [in]

The bus baud rate.

For Windows/Linux, please refer to vcil_can_speed.

For Android, please refer to VCIL_CAN_SPEED.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.3.5.5 vcil_can_set_speed_listen_mode

Syntax:

Windows / Linux	mrm_err **vcil_can_set_speed_listen_mode**(unsigned char port, vcil_can_speed speed)
Android	int **vcil_can_set_speed_listen_mode**(byte port, int speed)

Description:

Set the specified CAN port bus baud rate at Listen mode. This mode setup controller only listen data and not ACK bus.

Parameters:

port [in]

The CAN port. The first port is 0.

speed [in]

The bus baud rate. The detail please refer to vcil_can_speed.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

Some of bitrate is user-define value for customize , when you setup the specified bitrate and using vcil_can_get_bitrate may get user-define value

3.3.5.6 vcil_can_get_speed

Syntax:

Windows / Linux	mrm_err **vcil_can_get_speed**(unsigned char port, vcil_can_speed speed, vcil_can_bus_mode *mode)
Android	int **vcil_can_get_speed**(byte port, int[] speed, int[] mode)

Description:

Set the specified CAN port bus baud rate.

Parameters:

port [in]

The CAN port. The first port is 0.

speed [out]

The bus baud rate.

For Windows/Linux, please refer to vcil_can_speed.

For Android, please refer to VCIL_CAN_SPEED.

mode [out]

Current mode.

For Windows/Linux, please refer to vcil_can_bus_mode.

For Android, please refer to VCIL_CAN_BUS_MODE.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.3.5.7 vcil_can_get_bus_error_status

Syntax:

Windows / Linux	mrm_err **vcil_can_get_bus_error_status**(unsigned char port, vcil_can_error_status *status)
Android	int **vcil_can_get_bus_error_status**(byte port, VCIL_CAN_ERROR_STATUS status)

Description:

Get the specified CAN port error status. this API can be using to detect bus error status.

Parameters:

port [in]

The CAN port. The first port is 0.

status [out]

Windows/Linux:

Pointer to vcil_can_error_status struct that will hold the CAN error status.

Android:

Instance of VCIL_CAN_ERROR_STATUS which is used to store the CAN error status

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.3.5.8 vcil_can_set_mask

Syntax:

Windows / Linux	mrm_err **vcil_can_set_mask**(unsigned char port, vcil_can_mask_t *mask)
Android	int **vcil_can_set_mask**(byte port, VCIL_CAN_MASK mask)

Description:

Set the specified CAN message filter to specified filter bank of specified CAN port and enable it.

Parameters:

port [in]

The CAN port. The first port is 0.

mask [in]

The mask configuration.

For Windows/Linux, please refer to vcil_can_mask_t.

For Android, please refer to VCIL_CAN_MASK.

Returns:

- **MRM_ERR_NO_ERROR** - On success.
  Otherwise see the error code list.

- The field values of **mask** should be set base on the type of CAN protocol you want to apply to.
  You can set **two filters** in the a filter bank if you set filter for **CAN2.0A** and
  can set **one filter** in the a filter bank if you set filter for **CAN2.0B**.

  For example,
  If you want to set a **CAN2.0A** message filters to **filter bank 0**,
  you must set the "**bank**" field of **mask** to **0** and set "**extended frame"** field to **0(FALSE).**
  In this case you are allowed to set two filters in filter bank 0.
  Set the values of first filter to **"id1"** and **"mask1"** field and the second filter to "**id2**" and "**mask2**".

  If you want to set a **CAN2.0B** message filters to **filter bank 1**,
  you must set the "**bank**" field of **mask** to 1 and set "**extended frame"** field to **1(TRUE).**
  In this case you are allowed to set only one filter in filter bank 1.
  Set the value of filter to **"id1"** and **"mask1"** field. The values of "**id2**" and "**mask2**" will be ignored.

3.3.5.9 vcil_can_get_mask

Syntax:

Windows / Linux	mrm_err **vcil_can_get_mask**(unsigned char port, vcil_can_mask_t *mask)
Android	int **vcil_can_get_mask**(byte port, VCIL_CAN_MASK mask)

Description:

Get the a CAN message filter from specified filter bank of specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

mask [out]

The mask configuration.

For Windows/Linux, please refer to vcil_can_mask_t.

For Android, please refer to VCIL_CAN_MASK.

Returns:

- **MRM_ERR_NO_ERROR** - On success.

  Otherwise see the error code list.
- You must set the "**bank**" field of **mask** to specified which filter bank you want to get from.

3.3.5.10 vcil_can_remove_mask

Syntax:

Windows / Linux	mrm_err **vcil_can_remove_mask**(unsigned char port, unsigned char bank)
Android	int **vcil_can_remove_mask**(byte port, byte bank)

Description:

Remove a filter from specified filter bank of specified CAN port

Parameters:

port [in]

The CAN port. The first port is 0.

bank [in]

The bank of the mask to be removed.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

- If all bank are removed there will be no rule for passing CAN message. In order words, no CAN message can passed the filter and be received by the APP.

3.3.5.11 vcil_can_reset_mask

Syntax:

Windows / Linux	mrm_err **vcil_can_reset_mask**(unsigned char port)
Android	int **vcil_can_reset_mask**(byte port)

Description:

Reset all filter bank of the specified CAN port

Parameters:

port [in]

The CAN port. The first port is 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After reset, all bank will be cleared and disabled.

The MCU firmware will automatically add a filter of mask 0 and id 0 to filter bank 0 which means pass all CAN message without checking ant bit of the identifier. All CAN message will pass through this filter and be received by the APP.

3.3.5.12 vcil_can_set_event

Syntax:

Windows / Linux	mrm_err **vcil_can_set_event**(void *can_rx_event)
Android	-

Description:

Set a user define event in order to let VCIL library notify the specified event when CAN message is received.

Parameters:

can_rx_event [in]

Pointer to the CAN received event. In windows, the can_rx_event will pointer to a Windows Events HANDLE. In linux, the can_rx_event will pointer to a struct which consists of a pthread_mutex_t and pthread_cond_t.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You should not close the event before deinitialize VCIL library.

3.3.5.13 vcil_can_set_event_handler

Syntax:

Android	int **vcil_can_set_event_handler**(Handler handler)

Description:

Set handler which handles CAN message received event.

Parameters:

handler [in]

An instance of Handler.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.
- On alarm event triggered, the Handler will receive message with the "what" field equals to VCIL_EVENT_ID_RECEIVED_MSG_CAN

3.3.5.14 vcil_can_unset_event_handler

Syntax:

Android	int **vcil_can_unset_event_handler**()

Description:

Unregister handler of CAN message received event.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.

3.3.5.15 vcil_can_wait_event

Syntax:

Android	int **vcil_can_wait_event**(boolean status)

Description:

Allow/Disallow VCIL to pass CAN message received event to registered handler when a message is pushed into SDK internal buffer.

Parameters:

status [in]

The status of whether VCIL should pass CAN message received event to registered handler

TRUE: Inform

FLASE: Not to inform

Returns:

IMC_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample for details.

3.4 J1939 Functions

3.4.1 Usage

3.4.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.
Also, before using J1939 related APIs, you must first set the protocol mode of proper CAN port to **J1939 mode.** Please refer to the protocol mode setting section.

Android

Please refer to the VCIL Conventions for basic usage for Android.
Also, before using J1939 related APIs, you must first set the protocol mode of proper CAN port to **J1939 mode**. Please refer to the protocol mode setting section.

3.4.1.2 J1939 Message Reading

The APIs for reading J1939 messages is similar to CAN APIs usage. Please refer to CAN message reading section, and use corresponding APIs for J1939 instead.

Also, the CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.

3.4.1.3 J1939 Message Writing

The APIs for writing J1939 messages is similar to CAN APIs usage. Please refer to CAN message writing section, and use corresponding APIs for J1939 instead.

Also, the CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.

3.4.1.4 J1939 Acceptance filter Settings

Similar with CAN Acceptance Filter, the VCIL MCU provide acceptance filter for J1939 protocol, which can filter the PGN of J1939 message. The VCIL MCU supports at most 128 PGN filters. A J1939 message with PGN which is matched with any of the filters will be accepted by MCU.

The APIs for J1939 filter setting is similar to CAN APIs usage. Please refer to CAN Acceptance filter Settings section for Windows/Linux and Android APIs usage, and use corresponding APIs for J1939 instead.

The following is an example of filter setting. If we set filter with PGN: 0xFEF6, 0xFE20, 0xF201. When the received J1939 message's PGN value is 0xFEF6, the messages is accepted.

J1939 Filter Configuration	Receive J1939 data's PGN value	Result
Filter PGN=0xFEF6	0xFEF6	**Accept**
Filter PGN=0xFEF6	0xFEF7	**Drop**

J1939 Filter Configuration	Receive J1939 data's PGN value	Result
Filter PGN=0xFEF6 OR 0xFE20 OR 0xF201	0xFEF6	**Accept**
Filter PGN=0xFEF6 OR 0xFE20 OR 0xF201	0xF201	**Accept**
Filter PGN=0xFEF6 OR 0xFE20 OR 0xF201	0xF203	**Drop**
Filter PGN=0xFEF6 OR 0xFE20 OR 0xF201	0xFE22	**Drop**

3.4.2 Constant

3.4.2.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAX_J1939_DATA_SIZE	int	64
VCIL_MAX_J1939_MASK_NUM	int	128

3.4.3 Structure/Classes

3.4.3.1 Windows/Linux

vcil_j1939_message_t Structure

Syntax:

```
typedef struct
{
        unsigned char port;
        unsigned int pgn;
        unsigned char destination;
        unsigned char source;
        unsigned char priority;
        int length;
        unsigned char data[VCIL_MAX_J1939_DATA_SIZE];
} vcil_j1939_message_t;
```

Description:

This data structure defines the J1939 message.

Members:

port

This message come from/send to which port.

pgn

The parameter group number of this message. currently only support 0 to 0x1FFFF

destination

The destination address of this message. 0x00 to 0xFF

source

The source address of this message. This field ignored when J1939 write.

priority

The priority of this message. The priority of range is 0 to 7. Normally set to 6.

length

The standard J1939 message data length. The maximum length of data is **VCIL_MAX_J1939_DATA_SIZE**(64)

data

The data array of J1939 message.

Remark:

SAE-J1939 defined the longest message is 1785 byte but currently library only supported **VCIL_MAX_J1939_DATA_SIZE** for performance considered.

The SAE-J1939-81 defined the preferred address. The 255 is defined global address for broadcast and 254 is defined NULL address for the device without address at start.

vcil_j1939_config_t Structure

Syntax:

```
typedef struct
{
    unsigned char address;
    unsigned char arbitrary_address_capable;
    unsigned char industry_group;
    unsigned char vehicle_system_instance;
    unsigned char vehicle_system;
    unsigned char function;
    unsigned char function_instance;
    unsigned char ecu_instance;
    unsigned short manufacturer_code;
    unsigned int identity_number;
} vcil_j1939_config_t;
```

Description:

This data structure defines the configuration of J1939

Members:

address

The device address. This field will affect the J1939 message source address and J1939 address claiming.

arbitrary_address_capable

The fields of NAME specified in **SAE-J1939-81.**

industry_group

The fields of NAME specified in **SAE-J1939-81.**

vehicle_system_instance

The fields of NAME specified in **SAE-J1939-81.**

vehicle_system

The fields of NAME specified in **SAE-J1939-81.**

function

The fields of NAME specified in **SAE-J1939-81.**

function_instance

The fields of NAME specified in **SAE-J1939-81.**

ecu_instance

The fields of NAME specified in **SAE-J1939-81.**

manufacturer_code

The fields of NAME specified in **SAE-J1939-81.**

identity_number

The fields of NAME specified in **SAE-J1939-81.**

3.4.3.2 Android

VCIL_J1939_MESSAGE

class:

mrm.define.VCIL.**VCIL_J1939_MESSAGE**

Fields:

Field Name	Type	Input/Output	Comment
port	byte	in, out	This message come from/send to which port.
pgn	int	in, out	The parameter group number of this message. currently only support 0 to 0x1FFFF
destination	byte	in, out	The destination address of this message. 0x00 to 0xFF
source	byte	in, out	The source address of this message. This field ignored when J1939 write.
priority	byte	in	The priority of this message. The priority of range is 0 to 7. Normally set to 6.
length	int	in, out	The standard J1939 message data length. The maximum length of data is **VCIL_MAX_J1939_DATA_SIZE**(64)
data	byte[]	in, out	The array of data of the J1939 message.

VCIL_J1939_CONFIG

class:

mrm.define.VCIL.**VCIL_J1939_CONFIG**

Fields:

Field Name	Type	Input/Output	Comment
address	byte	in, out	The device address. This field will affect the J1939 message source address and J1939 address claiming.
arbitrary_address_capable	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
industry_group	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
vehicle_system_instance	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
vehicle_system	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
function	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
function_instance	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
ecu_instance	byte	in, out	The fields of NAME specified in **SAE-J1939-81.**
manufacturer_code	int	in, out	The fields of NAME specified in **SAE-J1939-81.**
identity_number	int	in, out	The fields of NAME specified in **SAE-J1939-81.**

3.4.4 APIs

3.4.4.1 vcil_j1939_read

Syntax:

Windows / Linux	mrm_err **vcil_j1939_read**(vcil_j1939_message_t *message)
Android	int **vcil_j1939_read**(VCIL_J1939_MESSAGE message)

Description:

Get a J1939 message from VCIL library J1939 buffer if available otherwise you may get a error code(**MRM_ERR_VCIL_DATA_NOT_READY**) and a invalid J1939 message.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_j1939_message_t which is used to store received J1939 message

Android:

Instance of VCIL_J1939_MESSAGE which is used to store received J1939 message

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive J1939 message from each CAN port.

3.4.4.2 vcil_j1939_read_multi

Syntax:

Windows / Linux	-
Android	int **vcil_j1939_read_multi**(List<VCIL_J1939_MESSAGE> messages, int desiredReadNum, int[] resultReadNum)

Description:

Read multiple received J1939 messages from the SDK internal buffer.

Parameters:

message [out]

Android:

List of VCIL_J1939_MESSAGE which is used to store received J1939 messages.

desiredReadNum [in]

The number of J1939 message you expect to get.

resultReadNum [out]

An allocated array of size 1 for storing the number of J1939 message the SDK actually returned.

The return value will be stored at index 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.4.4.3 vcil_j1939_write

Syntax:

Windows / Linux	mrm_err **vcil_j1939_write**(vcil_j1939_message_t *message)
Android	int vcil_j1939_write(VCIL_J1939_MESSAGE message)

Description:

Write a J1939 message to specified CAN port.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_j1939_message_t struct that store the J1939 message to be sent.

Android:

Instance of VCIL_J1939_MESSAGE which stores the J1939 message to be sent.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive J1939 message from each CAN port.

3.4.4.4 vcil_j1939_add_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1939_add_mask**(unsigned char port, unsigned int pgn)
Android	int **vcil_j1939_add_mask**(byte port, int pgn)

Description:

Add a PGN mask to specified CAN port. The mask will allow message with specified PGN able to pass through the filter and be received by user APP.

Parameters:

port [in]

The CAN port. The first port is 0.

pgn [in]

The PGN mask.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After VCIL initialized, by default, all message can pass through the VCIL hardware filter.

After you calling this function once, all message will be blocked except the message with specified PGN.

If you want only messages with specified PGN pass through the hardware filter, you can call this function to add more acceptable PGN

The maximum VCIL_MAX_J1939_MASK_NUM (128) masks can be applied.

3.4.4.5 vcil_j1939_get_mask_number

Syntax:

Windows / Linux	mrm_err **vcil_j1939_get_mask_number**(unsigned char port, unsigned int* total)
Android	int **vcil_j1939_get_mask_number**(byte port, int[] total)

Description:

Get the number of J1939 mask of specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

total [out]

The total number of mask PGN.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.4.4.6 vcil_j1939_get_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1939_get_all_mask**(unsigned char port, unsigned int* pgn)
Android	int **vcil_j1939_get_all_mask**(byte port, int[] pgn)

Description:

Get the all J1939 masks from specified CAN port

Parameters:

port [in]

The CAN port. The first port is 0.

pgn [out]

Pointer a PGN array, the array size you can get from vcil_j1939_get_mask_number().

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You must pass sufficient memory place for this function else your will get segmentation fault

3.4.4.7 vcil_j1939_remove_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1939_remove_mask**(unsigned char port, unsigned int pgn)
Android	int **vcil_j1939_remove_mask**(byte port, int pgn)

Description:

Remove specified PGN mask from the specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

pgn [in]

The PGN.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.4.4.8 vcil_j1939_remove_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1939_remove_all_mask**(unsigned char port)
Android	int **vcil_j1939_remove_all_mask**(byte port)

Description:

Remove all J1939 mask from the specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

When you remove all mask, all messages can be received.

3.4.4.9 vcil_j1939_set_config

Syntax:

Windows / Linux	mrm_err **vcil_j1939_set_config**(unsigned char port, vcil_j1939_config_t *config)
Android	int **vcil_j1939_set_config**(byte port, VCIL_J1939_CONFIG config)

Description:

Set the J1939 configuration.

Parameters:

port [in]

The CAN port. The first port is 0.

config [out]

Windows/Linux:

Pointer to vcil_j1939_config_t struct that will hold the J1939 config.

Android:

Instance of VCIL_J1939_CONFIG which is used to store J1939 config

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After calling this message, the device will send a address claiming message to the bus.

3.4.4.10 vcil_j1939_get_config

Syntax:

Windows / Linux	mrm_err **vcil_j1939_get_config**(unsigned char port, vcil_j1939_config_t *config)
Android	int **vcil_j1939_get_config**(byte port, VCIL_J1939_CONFIG config)

Description:

Get the J1939 configuration.

Parameters:

port [in]

The CAN port. The first port is 0.

config [out]

Windows/Linux:

Pointer to vcil_j1939_config_t struct that will hold the J1939 config.

Android:

Instance of VCIL_J1939_CONFIG which is used to store J1939 config

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.4.4.11 vcil_j1939_set_event

Syntax:

Windows / Linux	mrm_err **vcil_j1939_set_event**(void *j1939_rx_event)
Android	-

Description:

Set a user define event in order to let VCIL library notify the specified event when J1939 message is received.

Parameters:

j1939_rx_event [in]

Pointer to the J1939 received event. In windows, the j1939_rx_event will pointer to a Windows Events HANDLE. In linux, the j1939_rx_event will pointer to a struct which consists of a pthread_mutex_t and pthread_cond_t.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You should not close the event before deinitialize VCIL library.

3.4.4.12 vcil_j1939_set_event_handler

Syntax:

Android	int **vcil_j1939_set_event_handler**(Handler handler)

Description:

Set handler which handles J1939 message received event.

Parameters:

handler [in]

An instance of Handler.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.
- On alarm event triggered, the Handler will receive message with the "what" field equals to VCIL_EVENT_ID_RECEIVED_MSG_J1939

3.4.4.13 vcil_j1939_unset_event_handler

Syntax:

Android	int **vcil_j1939_unset_event_handler**()

Description:

Unregister handler of J1939 message received event.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.

3.4.4.14 vcil_j1939_wait_event

Syntax:

Android	int **vcil_j1939_wait_event**(boolean status)

Description:

Allow/Disallow VCIL to pass J1939 message received event to registered handler when a message is pushed into SDK internal buffer.

Parameters:

status [in]

The status of whether VCIL should pass J1939 message received event to registered handler

TRUE: Inform

FLASE: Not to inform

Returns:

IMC_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample for details.

3.5 OBD2 Functions

3.5.1 Usage

3.5.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.
Also, before using OBD2 related APIs, you must first set the protocol mode of proper CAN port to **OBD2 mode.** Please refer to the protocol mode setting section.

Android

Please refer to the VCIL Conventions for basic usage for Android.
Also, before using OBD2 related APIs, you must first set the protocol mode of proper CAN port to **OBD2 mode**. Please refer to the protocol mode setting section.

3.5.1.2 OBD2 Message Reading

The APIs for reading OBD2 messages is similar to CAN APIs usage. Please refer to CAN message reading section, and use corresponding APIs for OBD2 instead.

Also, the CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.

3.5.1.3 OBD2 Message Writing

The APIs for writing OBD2 messages is similar to CAN APIs usage. Please refer to CAN message writing section, and use corresponding APIs for OBD2 instead.

Also, the CAN bus speed must be set correctly before you do read/write operation. Please refer to CAN Bus Speed Setting section for the details.

3.5.1.4 OBD2 Acceptance filter Settings

Similar with CAN Acceptance Filter, the VCIL MCU provide acceptance filter for OBD2 protocol, which can filter the PID of OBD2 message. The VCIL MCU supports at most 256 PID filters. A OBD2 message with PID which is matched with any of the filters will be accepted by MCU.

The APIs for OBD2 filter setting is similar to CAN APIs usage. Please refer to CAN Acceptance filter Settings section for Windows/Linux and Android APIs usage, and use corresponding APIs for OBD2 instead.

The following is an example of filter setting. If we set filter with PID 0x80, 0x56, 0x11 When the received OBD2 message's PID value is 0x56, the messages is accepted.

OBD2 Filter Configuration	Receive OBD2 data's PID value	Result
Filter PID=0x80	0x80	**Accept**
Filter PID=0x80	0x79	**Drop**

OBD2 Filter Configuration	Receive OBD2 data's PID value	Result
Filter PID=0x80 OR 0x56 OR 0x11	0x80	**Accept**
Filter PID=0x80 OR 0x56 OR 0x11	0x56	**Accept**
Filter PID=0x80 OR 0x56 OR 0x11	0x57	**Drop**
Filter PID=0x80 OR 0x56 OR 0x11	0x12	**Drop**

Remark:

The second byte VCIL_OBD2_MESSAGE.data is PID.

3.5.2 Constant

3.5.2.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAX_OBD2_DATA_SIZE	int	64
VCIL_MAX_OBD2_MASK_NUM	int	128
VCIL_OBD2_TYPE_PHYSICAL	int	218 (0xDA)
VCIL_OBD2_TYPE_FUNCTIONAL	int	219 (0xDB)

3.5.3 Structure/Classes

3.5.3.1 Windows/Linux

vcil_obd2_message_t Structure

Syntax:

```
typedef struct
{
        unsigned char port;
        unsigned char type;
        unsigned char destination;
        unsigned char source;
        unsigned char priority;
        int length;
        unsigned char data[VCIL_MAX_OBD2_DATA_SIZE];
} vcil_obd2_message_t;
```

Description:

This data structure defines the OBD2 message.

Members:

port

This message come from/send to which port.

type

The type format of CAN identifier, Functional(0xDB) or Physical(0xDA).

destination

The destination address of this message. The range is 0x00 to 0xFF.

source

The source address of this message. The range is 0x00 to 0xFF.

priority

The priority of this message. The priority of range is 0 to 7. Normally set to 6.

length

The standard OBD2 message data length.

data

The data array of OBD2 message.

Remark:

The maximum length of data is **VCIL_MAX_OBD2_DATA_SIZE**(64). Currently only supported up to 64 bytes OBD2 message. The maximum message length is specified in **ISO 15765-2**. For request messages, the message length is limited to seven (7) data bytes. The other type of message can transmit large logical messages (up to 4095 bytes) using a series of individual CAN frames.

3.5.3.2 Android

VCIL_OBD2_MESSAGE

class:

mrm.define.VCIL.**VCIL_OBD2_MESSAGE**

Fields:

Field Name	Type	Input/Output	Comment
port	byte	in, out	This message come from/send to which port.
type	byte	in, out	The type format of CAN identifier, Functional(0xDB) or Physical(0xDA).
destination	byte	in, out	The destination address of this message. The range is 0x00 to 0xFF.
source	byte	in, out	The source address of this message. The range is 0x00 to 0xFF.
priority	byte	in	The priority of this message. The priority of range is 0 to 7. Normally set to 6.
length	int	in, out	The standard OBD2 message data length.
data	byte[]	in, out	The array of data of the OBD2 message.

Remark:

The maximum length of data is **VCIL_MAX_OBD2_DATA_SIZE**(64). Currently only supported up to 64 bytes OBD2 message. The maximum message length is specified in **ISO 15765-2**. For request messages, the message length is limited to seven (7) data bytes. The other type of message can transmit large logical messages (up to 4095 bytes) using a series of individual CAN frames.

3.5.4 APIs

3.5.4.1 vcil_obd2_read

Syntax:

Windows / Linux	mrm_err **vcil_obd2_read**(vcil_obd2_message_t *message)
Android	int **vcil_obd2_read**(VCIL_OBD2_MESSAGE message)

Description:

Get a OBD2 message from VCIL library OBD2 buffer if available otherwise you may get a error code(**MRM_ERR_VCIL_DATA_NOT_READY**) and a invalid OBD2 message.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_obd2_message_t struct which is used to store received OBD2 message

Android:

Instance of VCIL_OBD2_MESSAGE which is used to store received OBD2 message

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive OBD2 message from each CAN port.

3.5.4.2 vcil_obd2_read_multi

Syntax:

Windows / Linux	-
Android	int **vcil_obd2_read_multi**(List<VCIL_OBD2_MESSAGE> messages, int desiredReadNum, int[] resultReadNum)

Description:

Read multiple received OBD2 messages from the SDK internal buffer.

Parameters:

message [out]

Android:

List of VCIL_OBD2_MESSAGE which is used to store received OBD2 messages.

desiredReadNum [in]

The number of OBD2 message you expect to get.

resultReadNum [out]

An allocated array of size 1 for storing the number of OBD2 message the SDK actually returned.

The return value will be stored at index 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.5.4.3 vcil_obd2_write

Syntax:

Windows / Linux	mrm_err **vcil_obd2_write**(vcil_obd2_message_t *message)
Android	int **vcil_obd2_write**(VCIL_OBD2_MESSAGE message)

Description:

Write a OBD2 message to specified CAN port.

Parameters:

message [in]

Windows/Linux:

Pointer to vcil_obd2_message_t struct that store the OBD2 message to be sent.

Android:

Instance of VCIL_OBD2_MESSAGE which stores the OBD2 message to be sent.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.5.4.4 vcil_obd2_add_mask

Syntax:

Windows / Linux	mrm_err **vcil_obd2_add_mask**(unsigned char port, unsigned int pid)
Android	int **vcil_obd2_add_mask**(byte port, int pid)

Description:

Add a PID mask to specified CAN port. The mask will allow message with specified PID able to pass through the filter and be received by user APP.

Parameters:

port [in]

The CAN port. The first port is 0.

pid [in]

The PID mask.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After VCIL initialized, by default, all message can pass through the VCIL hardware filter.

After you calling this function once, all message will be blocked except the message with specified PID.

To make only messages with specified PID pass through the filters, you can call this function to add more acceptable PID.

3.5.4.5 vcil_obd2_get_mask_number

Syntax:

Windows / Linux	mrm_err **vcil_obd2_get_mask_number**(unsigned char port, unsigned int* total)
Android	int **vcil_obd2_get_mask_number**(byte port, int[] total)

Description:

Get the number of OBD2 mask of specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

total [out]

The total number of mask PID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.5.4.6 vcil_obd2_get_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_obd2_get_all_mask**(unsigned char port, unsigned int* pid)
Android	int **vcil_obd2_get_all_mask**(byte port, int[] pid)

Description:

Get the all OBD2 masks from specified CAN port

Parameters:

port [in]

The CAN port. The first port is 0.

pid [out]

Pointer a PID array, the array size you can get from vcil_obd2_get_mask_number().

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You must pass sufficient memory place for this function else your will get segmentation fault

3.5.4.7 vcil_obd2_remove_mask

Syntax:

Windows / Linux	mrm_err **vcil_obd2_remove_mask**(unsigned char port, unsigned int pid)
Android	int **vcil_obd2_remove_mask**(byte port, int pid)

Description:

Remove specified PID mask from the specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

pid [in]

The PID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.5.4.8 vcil_obd2_remove_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_obd2_remove_all_mask**(unsigned char port)
Android	int **vcil_obd2_remove_all_mask**(byte port)

Description:

Remove all OBD2 mask from the specified CAN port.

Parameters:

port [in]

The CAN port. The first port is 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

When you remove all mask, all messages can be received.

3.5.4.9 vcil_obd2_set_event

Syntax:

Windows / Linux	mrm_err **vcil_obd2_set_event**(void *obd2_rx_event)
Android	-

Description:

Set a user define event in order to let VCIL library notify the specified event when OBD2 message is received.

Parameters:

obd2_rx_event [in]

Pointer to the OBD2 received event. In windows, the obd2_rx_event will pointer to a Windows Events HANDLE. In linux, the obd2_rx_event will pointer to a struct which consists of a pthread_mutex_t and pthread_cond_t.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You should not close the event before deinitialize VCIL library.

3.5.4.10 vcil_obd2_set_event_handler

Syntax:

Android	int **vcil_obd2_set_event_handler**(Handler handler)

Description:

Set handler which handles OBD2 message received event.

Parameters:

handler [in]

An instance of Handler.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.
- On alarm event triggered, the Handler will receive message with the "what" field equals to VCIL_EVENT_ID_RECEIVED_MSG_OBD2

3.5.4.11 vcil_obd2_unset_event_handler

Syntax:

Android	int **vcil_obd2_unset_event_handler**()

Description:

Unregister handler of OBD2 message received event.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.

3.5.4.12 vcil_obd2_wait_event

Syntax:

Android	int **vcil_obd2_wait_event**(boolean status)

Description:

Allow/Disallow VCIL to pass OBD2 message received event to registered handler when a message is pushed into SDK internal buffer.

Parameters:

status [in]

The status of whether VCIL should pass OBD2 message received event to registered handler

TRUE: Inform

FLASE: Not to inform

Returns:

IMC_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample for details.

3.6 J1708 Functions

3.6.1 Usage

3.6.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.
Also, before using J1708 related APIs, you must first set the protocol mode of proper J1708 port to **J1708 mode.** Please refer to the protocol mode setting section.

Android

Please refer to the VCIL Conventions for basic usage for Android.
Also, before using J1708 related APIs, you must first set the protocol mode of proper J1708 port to **J1708 mode**. Please refer to the protocol mode setting section.

3.6.1.2 J1708 Message Reading

The APIs for reading J1708 messages is similar to CAN APIs usage. Please refer to CAN message reading section, and use corresponding APIs for J1708 instead.

3.6.1.3 J1708 Message Writing

The APIs for writing J1708 messages is similar to CAN APIs usage. Please refer to CAN message writing section, and use corresponding APIs for J1708 instead.

3.6.1.4 J1708 Acceptance filter Settings

Similar with CAN Acceptance Filter, the VCIL MCU provide acceptance filter for J1708 protocol, which can filter the MID of J1708 message. The VCIL MCU supports at most 256 MID filters. A J1708 message with MID which is matched with any of the filters will be accepted by MCU.

The APIs for J1708 filter setting is similar to CAN APIs usage. Please refer to CAN Acceptance filter Settings section for Windows/Linux and Android APIs usage, and use corresponding APIs for J1708 instead.

The following is an example of filter setting. If we set filter with MID 0x80, 0x56, 0x11 When the received J1708 message's MID value is 0x56, the messages is accepted.

J1708 Filter Configuration	Receive J1708 data's MID value	Result
Filter MID=0x80	0x80	**Accept**
Filter MID=0x80	0x79	**Drop**

J1708 Filter Configuration	Receive J1708 data's MID value	Result
Filter MID=0x80 OR 0x56 OR 0x11	0x80	**Accept**
Filter MID=0x80 OR 0x56 OR 0x11	0x56	**Accept**
Filter MID=0x80 OR 0x56 OR 0x11	0x57	**Drop**
Filter MID=0x80 OR 0x56 OR 0x11	0x12	**Drop**

3.6.2 Constant

3.6.2.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAX_J1708_DATA_SIZE	int	20
VCIL_MAX_J1708_MASK_NUM	int	256

3.6.3 Structure/Classes

3.6.3.1 Windows/Linux

vcil_j1708_message_t Structure

Syntax:

```
typedef struct
{
        unsigned char mid;
        unsigned char priority;
        int length;
        unsigned char data[VCIL_MAX_J1708_DATA_SIZE];
} vcil_j1708_message_t;
```

Description:

This data structure defines the J1708 message.

Members:

mid

The message identification character. The range is 0x00 to 0xFF.

priority

The priority of this message. The priority of range is 1 to 8. Normally set to 6.

length

The standard J1708 message data length. The range is 0x00 to 0xFF.

data

The data array of J1708 message.

Remark:

The maximum length of data is **VCIL_MAX_J1708_DATA_SIZE**(20). The maximum message length is 21 (include MID+data) specified in **SAE J1708**.

3.6.3.2 Android

VCIL_J1708_MESSAGE

class:

mrm.define.VCIL.**VCIL_J1708_MESSAGE**

Fields:

Field Name	Type	Input/Output	Comment
mid	byte	in, out	The message identification character. The range is 0x00 to 0xFF.
priority	byte	in	The priority of this message. The priority of range is 1 to 8. Normally set to 6.
length	int	in, out	The standard J1708 message data length.
data	byte[]	in, out	The array of data of the J1708 message.

Remark:

The maximum length of data is **VCIL_MAX_J1708_DATA_SIZE**(20). The maximum message length is 21 (include MID+data) specified in **SAE J1708**.

3.6.4 APIs

3.6.4.1 vcil_j1708_read

Syntax:

Windows / Linux	mrm_err **vcil_j1708_read**(vcil_j1708_message_t *message)
Android	int **vcil_j1708_read**(VCIL_J1708_MESSAGE message)

Description:

Get a J1708 message from VCIL library J1708 buffer if available otherwise you may get a error code(**MRM_ERR_VCIL_DATA_NOT_READY**) and a invalid J1708 message.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_j1708_message_t struct which is used to store received J1708 message

Android:

Instance of VCIL_J1708_MESSAGE which is used to store received J1708 message

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.6.4.2 vcil_j1708_read_multi

Syntax:

Windows / Linux	-
Android	int **vcil_j1708_read_multi**(List<VCIL_J1708_MESSAGE> messages, int desiredReadNum, int[] resultReadNum)

Description:

Read multiple received J1708 messages from the SDK internal buffer.

Parameters:

message [out]

Android:

List of VCIL_J1708_MESSAGE which is used to store received J1708 messages.

desiredReadNum [in]

The number of J1708 message you expect to get.

resultReadNum [out]

An allocated array of size 1 for storing the number of J1708 message the SDK actually returned.

The return value will be stored at index 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.6.4.3 vcil_j1708_write

Syntax:

Windows / Linux	mrm_err **vcil_j1708_write**(vcil_j1708_message_t *message)
Android	int **vcil_j1708_write**(VCIL_J1708_MESSAGE message)

Description:

Write a J1708 message to specified CAN port.

Parameters:

message [in]

Windows/Linux:

Pointer to vcil_j1708_message_t struct that store the J1708 message to be sent.

Android:

Instance of VCIL_J1708_MESSAGE which stores the J1708 message to be sent.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.6.4.4 vcil_j1708_add_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1708_add_mask**(unsigned char mid)
Android	int **vcil_j1708_add_mask**(byte mid)

Description:

Add a MID mask to specified CAN port. The mask will allow message with specified MID able to pass through the filter and be received by user APP.

Parameters:

mid [in]

The MID mask.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After VCIL initialized, by default, all message can pass through the VCIL hardware filter.

After you calling this function once, all message will be blocked except the message with specified MID.

To make only messages with specified MID pass through the filters, you can call this function to add more acceptable MID.

3.6.4.5 vcil_j1708_get_mask_number

Syntax:

Windows / Linux	mrm_err **vcil_j1708_get_mask_number**(unsigned int* total)
Android	int **vcil_j1708_get_mask_number**(int[] total)

Description:

Get the number of J1708 mask.

Parameters:

total [out]

The total number of mask MID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.6.4.6 vcil_j1708_get_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1708_get_all_mask**(unsigned char* mid)
Android	int **vcil_j1708_get_all_mask**(byte[] mid)

Description:

Get the all J1708 mask.

Parameters:

mid [out]

Pointer a MID array, the array size you can get from vcil_j1708_get_mask_number().

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You must pass sufficient memory place for this function else your will get segmentation fault

3.6.4.7 vcil_j1708_remove_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1708_remove_mask**(unsigned char mid)
Android	int **vcil_j1708_remove_mask**(byte mid)

Description:

Remove specified MID mask.

Parameters:

mid [in]

The MID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.6.4.8 vcil_j1708_remove_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1708_remove_all_mask**(void)
Android	int **vcil_j1708_remove_all_mask**()

Description:

Remove all J1708 mask.

Parameters:

None.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

When you remove all mask, all messages can be received.

3.6.4.9 vcil_j1708_set_event

Syntax:

Windows / Linux	mrm_err **vcil_j1708_set_event**(void *j1708_rx_event)
Android	-

Description:

Set a user define event in order to let VCIL library notify the specified event when J1708 message is received.

Parameters:

j1708_rx_event [in]

Pointer to the J1708 received event. In windows, the j1708_rx_event will pointer to a Windows Events HANDLE. In linux, the j1708_rx_event will pointer to a struct which consists of a pthread_mutex_t and pthread_cond_t.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You should not close the event before deinitialize VCIL library.

3.6.4.10 vcil_j1708_set_event_handler

Syntax:

Android	int **vcil_j1708_set_event_handler**(Handler handler)

Description:

Set handler which handles J1708 message received event.

Parameters:

handler [in]

An instance of Handler.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.
- On alarm event triggered, the Handler will receive message with the "what" field equals to VCIL_EVENT_ID_RECEIVED_MSG_J1708

3.6.4.11 vcil_j1708_unset_event_handler

Syntax:

Android	int **vcil_j1708_unset_event_handler**()

Description:

Unregister handler of J1708 message received event.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.

3.6.4.12 vcil_j1708_wait_event

Syntax:

Android	int **vcil_j1708_wait_event**(boolean status)

Description:

Allow/Disallow VCIL to pass J1708 message received event to registered handler when a message is pushed into SDK internal buffer.

Parameters:

status [in]

The status of whether VCIL should pass J1708 message received event to registered handler

TRUE: Inform

FLASE: Not to inform

Returns:

IMC_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample for details.

3.7 J1587 Functions

3.7.1 Usage

3.7.1.1 Basic Usage

Windows/Linux

Please refer to the VCIL Conventions for basic usage for Windows/Linux.
Also, before using J1587 related APIs, you must first set the protocol mode of proper J1708 port to **J1587 mode.** Please refer to the protocol mode setting section.

Android

Please refer to the VCIL Conventions for basic usage for Android.
Also, before using J1587 related APIs, you must first set the protocol mode of proper J1708 port to **J1587 mode**. Please refer to the protocol mode setting section.

3.7.1.2 J1587 Message Reading

The APIs for reading J1587 messages is similar to CAN APIs usage. Please refer to CAN message reading section, and use corresponding APIs for J1587 instead.

3.7.1.3 J1587 Message Writing

The APIs for writing J1587 messages is similar to CAN APIs usage. Please refer to CAN message writing section, and use corresponding APIs for J1587 instead.

3.7.1.4 J1587 Acceptance filter Settings

Similar with CAN Acceptance Filter. the VCIL MCU provide acceptance filter for J1587 protocol, which can filter the PID of J1587 message. The VCIL MCU supports at most 512 PID filters. A J1587 message with PID which is matched with any of the filters will be accepted by MCU.

The APIs for J1587 filter setting is similar to CAN APIs usage. Please refer to CAN Acceptance filter Settings section for Windows/Linux and Android APIs usage, and use corresponding APIs for J1587 instead.

The following is an example of filter setting. If we set filter with PID 0x80, 0x56, 0x11 When the received J1587 message's PID value is 0x56, the messages is accepted.

J1587 Filter Configuration	Receive J1587 data's PID value	Result
Filter PID=0x80	0x80	**Accept**
Filter PID=0x80	0x79	**Drop**

J1587 Filter Configuration	Receive J1587 data's PID value	Result
Filter PID=0x80 OR 0x56 OR 0x11	0x80	**Accept**
Filter PID=0x80 OR 0x56 OR 0x11	0x56	**Accept**
Filter PID=0x80 OR 0x56 OR 0x11	0x57	**Drop**
Filter PID=0x80 OR 0x56 OR 0x11	0x12	**Drop**

3.7.2 Constant

3.7.2.1 Android

Class:
mrm.define.**MRM_CONSTANTS**

Fields:

Field Name	Type	Value
VCIL_MAX_J1587_DATA_SIZE	int	20
VCIL_MAX_J1587_MASK_NUM	int	256

3.7.3 Structure/Classes

3.7.3.1 Windows/Linux

vcil_j1587_message_t Structure

Syntax:

```
typedef struct
{
        unsigned char mid;
        unsigned char priority;
        unsigned int pid;
        int length;
        unsigned char data[VCIL_MAX_J1587_DATA_SIZE];
} vcil_j587_message_t;
```

Description:

This data structure defines the J1587 message.

Members:

mid

The message identification character. The range is 0x00 to 0xFF.

priority

The priority of this message. The priority of range is 1 to 8. Normally set to 6.

pid

The parameter Identification. The range is 0x00 to 0x1FE, exclude 0xFF. The detail range please refer to **SAE-J1708**

length

The standard J1587 message data length. exclude pid length.

data

The data array of J1587 message.

Remark:

The maximum length of data is **VCIL_MAX_J1587_DATA_SIZE**(20). The maximum message length is 21 (include MID+data) specified in **SAE J1708**.

3.7.3.2 Android

VCIL_J1587_MESSAGE

class:

mrm.define.VCIL.**VCIL_J1587_MESSAGE**

Fields:

Field Name	Type	Input/Output	Comment
mid	byte	in, out	The message identification character. The range is 0x00 to 0xFF.
priority	byte	in	The priority of this message. The priority of range is 1 to 8. Normally set to 6.
pid	int	in, out	The parameter Identification. The range is 0x00 to 0x1FE, exclude 0xFF. The detail range please refer to SAE-J1708
length	int	in, out	The standard J1587 message data length. exclude pid length.
data	byte[]	in, out	The array of data of the J1587 message.

Remark:

The maximum length of data is **VCIL_MAX_J1587_DATA_SIZE**(20). The maximum message length is 21 (include MID+data) specified in **SAE J1708**.

3.7.4 APIs

3.7.4.1 vcil_j1587_read

Syntax:

Windows / Linux	mrm_err **vcil_j1587_read**(vcil_j1587_message_t *message)
Android	int **vcil_j1587_read**(VCIL_J1587_MESSAGE message)

Description:

Get a J1587 message from VCIL library J1587 buffer if available otherwise you may get a error code(**MRM_ERR_VCIL_DATA_NOT_READY**) and a invalid J1587 message.

Parameters:

message [out]

Windows/Linux:

Pointer to vcil_j1587_message_t struct which is used to store received J1587 message

Android:

Instance of VCIL_J1587_MESSAGE which is used to store received J1587 message

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You can call this function to receive J1587 message from each CAN port.

3.7.4.2 vcil_j1587_read_multi

Syntax:

Windows / Linux	-
Android	int **vcil_j1587_read_multi**(List<VCIL_J1587_MESSAGE> messages, int desiredReadNum, int[] resultReadNum)

Description:

Read multiple received J1587 messages from the SDK internal buffer.

Parameters:

message [out]

Android:

List of VCIL_J1587_MESSAGE which is used to store received J1587 messages.

desiredReadNum [in]

The number of J1587 message you expect to get.

resultReadNum [out]

An allocated array of size 1 for storing the number of J1587 message the SDK actually returned.

The return value will be stored at index 0.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.7.4.3 vcil_j1587_write

Syntax:

Windows / Linux	mrm_err **vcil_j1587_write**(vcil_j1587_message_t *message)
Android	int **vcil_j1587_write**(VCIL_J1587_MESSAGE message)

Description:

Write a J1587 message to specified J1587 port.

Parameters:

message [in]

Windows/Linux:

Pointer to vcil_j1587_message_t struct that store the J1587 message to be sent.

Android:

Instance of VCIL_J1587_MESSAGE which stores the J1587 message to be sent.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.7.4.4 vcil_j1587_add_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1587_add_mask**(unsigned int pid)
Android	int **vcil_j1587_add_mask**(int pid)

Description:

Add a PID mask to specified CAN port. The mask will allow message with specified PID able to pass through the filter and be received by user APP.

Parameters:

pid [in]

The PID mask.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

After VCIL initialized, by default, all message can pass through the VCIL hardware filter.

After you calling this function once, all message will be blocked except the message with specified PID.

To make only messages with specified PID pass through the filters, you can call this function to add more acceptable PID.

3.7.4.5 vcil_j1587_get_mask_number

Syntax:

Windows / Linux	mrm_err **vcil_j1587_get_mask_number**(unsigned int* total)
Android	int **vcil_j1587_get_mask_number**(int[] total)

Description:

Get the number of J1587 mask.

Parameters:

total [out]

The total number of mask PID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.7.4.6 vcil_j1587_get_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1587_get_all_mask**(unsigned int* pid)
Android	int **vcil_j1587_get_all_mask**(int[] pid)

Description:

Get the all J1587 mask.

Parameters:

pid [out]

Pointer a PID array, the array size you can get from vcil_j1587_get_mask_number().

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You must pass sufficient memory place for this function else your will get segmentation fault

3.7.4.7 vcil_j1587_remove_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1587_remove_mask**(unsigned int pid)
Android	int **vcil_j1587_remove_mask**(int pid)

Description:

Remove specified PID mask.

Parameters:

pid [in]

The PID.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

3.7.4.8 vcil_j1587_remove_all_mask

Syntax:

Windows / Linux	mrm_err **vcil_j1587_remove_all_mask**(void)
Android	int **vcil_j1587_remove_all_mask**()

Description:

Remove all J1587 mask.

Parameters:

None.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

When you remove all mask, all messages can be received.

3.7.4.9 vcil_j1587_set_event

Syntax:

Windows / Linux	mrm_err **vcil_j1587_set_event**(void *j1587_rx_event)
Android	-

Description:

Set a user define event in order to let VCIL library notify the specified event when J1587 message is received.

Parameters:

j1587_rx_event [in]

Pointer to the J1587 received event. In windows, the j1587_rx_event will pointer to a Windows Events HANDLE. In linux, the j1587_rx_event will pointer to a struct which consists of a pthread_mutex_t and pthread_cond_t.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

You should not close the event before deinitialize VCIL library.

3.7.4.10 vcil_j1587_set_event_handler

Syntax:

Android	int **vcil_j1587_set_event_handler**(Handler handler)

Description:

Set handler which handles J1587 message received event.

Parameters:

handler [in]

An instance of Handler.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.
- On alarm event triggered, the Handler will receive message with the "what" field equals to VCIL_EVENT_ID_RECEIVED_MSG_J1587

3.7.4.11 vcil_j1587_unset_event_handler

Syntax:

Android	int **vcil_j1587_unset_event_handler**()

Description:

Unregister handler of J1587 message received event.

Parameters:

none

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample code for details.

3.7.4.12 vcil_j1587_wait_event

Syntax:

Android	int **vcil_j1587_wait_event**(boolean status)

Description:

Allow/Disallow VCIL to pass J1587 message received event to registered handler when a message is pushed into SDK internal buffer.

Parameters:

status [in]

The status of whether VCIL should pass J1587 message received event to registered handler

TRUE: Inform

FLASE: Not to inform

Returns:

IMC_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remark:

- Please refer to the usage guide and sample for details.

3.8 Cradle Functions

3.8.1 APIs

3.8.1.1 vcil_cradle_set_detach_event

Syntax:

```
mrm_err vcil_cradle_set_detach_event(void *cradle_event)
```

Description:

Set a user define event in order to let VCIL library notify the specified event when detect cradle detach.

Parameters:

cradle_event [in]

Pointer to the cradle detach event. In windows, the cradle_detach_event will pointer to a Windows Events HANDLE.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

- This function only support for TREK-973.

3.8.1.2 vcil_cradle_set_attach_event

Syntax:

```
mrm_err vcil_cradle_set_attach_event(void *cradle_event)
```

Description:

Set a user define event in order to let VCIL library notify the specified event when detect cradle detach.

Parameters:

cradle_event [in]

Pointer to the cradle attach event. In windows, the cradle_attach_event will pointer to a Windows Events HANDLE.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

- This function only support for TREK-973.

3.8.1.3 vcil_cradle_get_status

Syntax:

```
mrm_err vcil_cradle_get_status(char *status)
```

Description:

Get current status of cradle.

Parameters:

status [out]

cradle detach or attach status. The value 1 is Attach otherwise 0 is detach.

Returns:

MRM_ERR_NO_ERROR - On success.

Otherwise see the error code list.

Remarks:

- This function only support for TREK-973.

4 Error Code List

4.1 Comman Error

- (0x00000000) **MRM_ERR_NO_ERROR** - On success.
- (0x00000001) **MRM_ERR_INVALID_POINTER** - Encounter invalid pointer.
- (0x00000002) **MRM_ERR_INVALID_ARGUMENT** - Encounter invalid argument. Please check out the API parameter.
- (0x00000003) **MRM_ERR_UNSUPPORT_OPERATION** - Encounter unsupport opeartion. Please check out spec. of the platform supported.
- (0x00000005) **MRM_ERR_LIBRARY_NOT_INIT** - Function call before the library init.
- (0x00000010) **MRM_ERR_ILLEGAL_OPERATION** - Encounter illegal operation.
- (0x00000011) **MRM_ERR_LIBRARY_ALREADY_INIT** - Function call before the library init.
- (0x00000012) **MRM_ERR_ARRAY_OUR_OF_RANGE** - Encounter the access array out of range.
- (0x00000013) **MRM_ERR_OPERATION_FAIL** - Encounter the operation fail.
- (0x00000014) **MRM_ERR_DEVICE_NOT_EXIST** - Encounter the cradle not attached.

- (0x10000001) **MRM_ERR_ANDROID_JNI_NULL_POINTER** - Null pointer error occurred on VCIL lib side(JNI).
- (0x10000002) **MRM_ERR_ANDROID_JNI_OUT_OF_MEMORY** - Out of memory error occurred on VCIL lib side(JNI).
- (0x10000003) **MRM_ERR_ANDROID_JNI_EVENT_INIT_FAILED** - Failed to init event handle on VCIL lib side(JNI).
- (0x10000004) **MRM_ERR_ANDROID_JNI_EVENT_DEINIT_FAILED** - Failed to init event handle on VCIL lib side(JNI).
- (0x10000005) **MRM_ERR_ANDROID_JNI_EVENT_LISTENING_THREAD_ALREADY_RUNNING** - Event listening thread is already running(JNI).
- (0x10000006) **MRM_ERR_ANDROID_JNI_EVENT_LISTENING_THREAD_CREATE_FAILED** - Failed to create event listening thread(JNI).

4.2 This following figure describes how to write CAN messages to CAN bus by using SDK APIs.

- (0x03000001) **MRM_ERR_VCIL_DEVICE_NODE_OPEN_FAIL** - Open VCIL device node fail. Please checkout VCIL is exist or the device not use by another application.
- (0x03000002) **MRM_ERR_VCIL_DEVICE_NODE_WRITE_FAIL** - Encounter write operation fail. Please retry operation.

- (0x03000003) **MRM_ERR_VCIL_DEVICE_NODE_READ_FAIL** - Encounter read operation fail. Please retry operation.
- (0x03000004) **MRM_ERR_VCIL_IS_BUSY** - Encounter library is busy. Please retry operation.
- (0x03000008) **MRM_ERR_VCIL_DEVICE_NODE_READ_TIMEOUT** - Encounter read operation timeout. Please retry operation.
- (0x03000009) **MRM_ERR_VCIL_DATA_NOT_READY** - Encounter data buffer empty.