CAN Bus Agent (MCU-based) API Manual

1 Introduction

The document introduces the CAN Bus Agent (MCU-based) API library, and how to import the library onto ECUs to detect abnormal or malicious CAN Bus traffic.

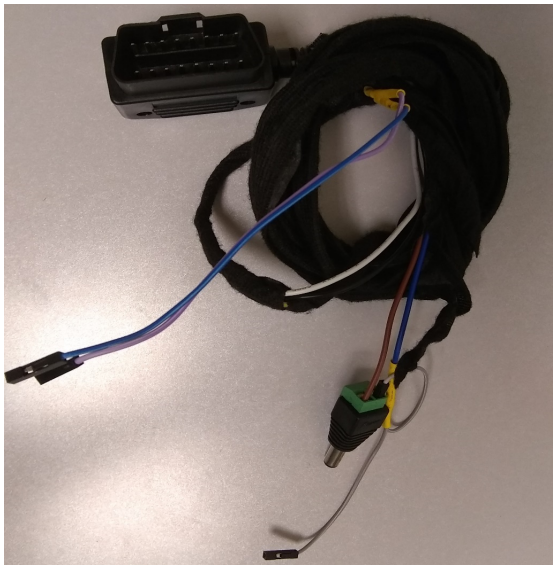
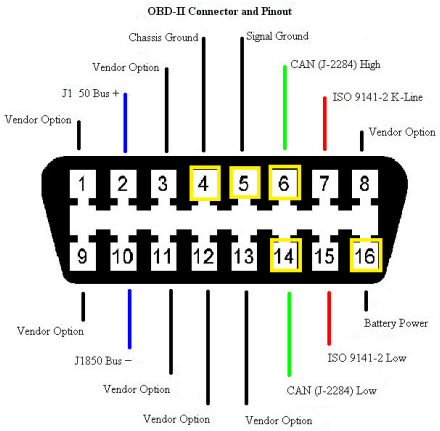


Figure 1-1 OBD II cable



OBD II Pins	CAN Pin of DevKit
6	CANH - P5-01
14	CANL - P5-02
4 and 5	GND - P5-04
4 and 5	GND of Power supply
16	+12V of Power supply

Figure 1-2 Pin layout of OBD port

2 API Library and Customization

Agent library contains the "**libvtAgent_Z4.a"** library file and **vt_fw_if.h** header file.

2.1 API Macros

The API macros are configurable for the agent customization.

VT_VECTOR_MAX_CAN_ID

#define	Max number of CAN ID supported in the signatures. The number depends on the hardware resources on the target board.	48

VT_MAX_DATA_BYTE_LENGTH

#define	Max CAN message payload length supported.	8 bytes

VT_XMINUTE_TIME

#define	Time window of CAN traffic monitoring in the Agent library.	60 seconds

2.2 Agent Engine Status Definitions

vt_status_t (enumeration)

VT_STATUS_SUCCESS	0	Agent success status
VT_STATUS_ERROR	-1	Agent error status
VT_STATUS_BUSY	-2	Agent busy status
VT_STATUS_TIMEOUT	-3	Agent timeout status
VT_STATUS_MEM_OVERFLOW	-4	
VT_STATUS_QUEUE_FULL	-5	Agent full status
VT_STATUS_INVALID	-6	Agent invalid status
VT_STATUS_SEND_ERROR	-7	Agent send-error status
VT_STATUS_RCV_ERROR	-8	Agent receive-error status

2.3 APIs

2.3.1 vt_fw_init

Syntax	void vt_fw_init (const uint8_t *rule_content, const uint8_t *vector_content)
Input parameters	const uint8_t *rule_content; // Policy rules content.
	const uint8_t *vector_content; // Vector content.

Output parameters	None.
Input & Output parameters	None.
Return Value	Void
Description	Initialize agent with signatures.

2.3.2 vt_fw_get_all_can_id_used

Syntax	vt_status_t vt_fw_get_all_can_id_used (uint32_t *id_buffer, int *len)
Input parameters	uint32_t *id_buffer; // Pointer to CAN ID buffer.
Output parameters	None.

Input & Output parameters	int *len; // ID Buffer length.
	Input mode: the size of CAN ID buffer;
	Output mode: the number of CAN IDs in result buffer;

Return Value	vt_status_t
Description	Fetch all CAN IDs from agent signatures.

2.3.3 vt_rule_set_slot_time

Syntax	void vt_rule_set_slot_time (uint16_t itv)
Input parameters	uint16_t itv ; // Slot time.
Output parameters	None.
Input & Output parameters	None.
Return Value	void
Description	Set time unit, e.g. 200us

2.3.4 vt_fw_rcv_msg

Syntax	void vt_fw_rcv_msg (uint32_t id, uint8_t len, uint8_t *databuff)
Input parameters	uint32_t id; // CAN ID.
	uint8_t len; // Length of data buffer.
	uint8_t *databuff; // pointer to data buffer.
Output parameters	None.
Input & Output parameters	None.
Return Value	Void
Description	Add a CAN message to Agent queue.

2.3.5 vt_rule_install_slot_callback

Syntax	void vt_rule_install_slot_callback (vt_rule_slot_callback callback)
Input parameters	vt_rule_slot_callback callback; // user-defined - slot callback function.
Output parameters	None.
Input & Output parameters	None.
Return Value	Void
Callback Definition	typedef vt_status_t (*vt_rule_slot_callback) (vt_slot_matched_t *matched_t)
Description	callback function to report matched slot data to the cloud or print out locally. The callback function is to be extended by users.

2.3.6 vt_rule_install_category_callback

Syntax	void vt_rule_install_category_callback (vt_rule_category_callback callback)
Input parameters	vt_rule_category_callback callback ; // user-defined - category callback function.
Output parameters	None.
Input & Output parameters	None.
Return Value	Void

Callback Definition	typedef vt_status_t (*vt_rule_category_callback) (vt_category_matched_t *matched_t)
Description	callback function to report matched category data to the cloud or print out. This callback function is to be extended by users.

2.3.7 vt_rule_install_vector_callback

Syntax	void vt_rule_install_vector_callback (vt_vector_callback callback)
Input parameters	vt_vector_callback callback ; // user-defined - vector callback function.
Output parameters	None.
Input & Output parameters	None.
Return Value	Void
Callback Definition	typedef vt_status_t (* vt_vector_callback) (vt_vector_result_t *vector_t)
Description	callback function to report matched vector data to server or print out. This callback function is implemented by user.

2.3.8 vt_fw_process

Syntax	void vt_fw_process(void)
Input parameters	None.
Output parameters	None.
Input & Output parameters	None.
Return Value	void
Description	Start Agent. This function will be called in a main loop function.

3 Porting Process

The following codes initializes system clock, configure GPIO, PIT (periodic interrupt timer), RTC, UART, CAN, etc.

```
int main(void)
{
  /*
   * System initilization.
   */

  /* System clock initializtion. */
  CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                 g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
  CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

  /* GPIO, PIN function initialization. */
  PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

  /* Initialize UART PAL over LINFlexD */
  UART_Init(INST_UART_PAL1, &uart_pal1_Config0);

  /* Send the greeting message to console */
  UART_SendData(INST_UART_PAL1, (uint8_t*)"VT Agent \r\n", strlen("VT Agent \r\n"));

  /* Initialize Leds */
  vt_init_leds();

  /* Initialize RTC, alarm repeat interval in 1 second */
  vt_rtc_init(VT_RTC_TIMER, &vt_rtcTimer_StartTime, &vt_rtcTimer_AlarmConfig);

  /* Initialize PIT to create time stamp. Default interval time is 200us */
  vt_timer_init(VT_INST_PIT, &vt_pit_ChnConfig0);

  /* Initialize CAN bus */
  vt_init_can(VT_INST_CAN, bitrate, vt_rcv_callback, NULL);

  /*
   * Other implementations
   */
  // ...
}
```

3.1 Agent Increase Time

vt_fw_increase_time() API should be put in alarm callback function of RTC. In this example, RTC is set to 1 second for the alarm interrupt user Callback, so vt_fw_increase_time() will be invoked every 1 second. Below is the example code:

```
void vt_rtc_timer_callback(void * callbackParam)
{
        /* Put your code here */
        vt_fw_increase_time();
        vt_toggle_led(leds[VT_RTC_LED]);
}
```

3.2 Agent Increase Timestamp

vt_rule_increase_timestamp() API should be put in PIT handler to increase tick count of timestamp for Agent. In this example, PIT is set to trigger an interrupt in 200 us period, so Vt_rule_increase_timestamp() will be invoked every 200 us. Below is the example code:

```
void PIT_Ch0_IRQHandler(void)
{
        /* Count 1 a time */
        vt_rule_increase_timestamp();
        PIT_DRV_ClearStatusFlags(VT_INST_PIT, vt_pit_ChnConfig0.hwChannel);    /* Clear channel 0
interrupt flag */
}
```

3.3 Feed Agent with CAN Data

vt_fw_rcv_msg() API should be put in CAN frame reception handler to add CAN message to Agent queue. Below is the example code.

```
void vt_rcv_callback(uint8_t instance, flexcan_event_type_t eventType, flexcan_state_t *flexcanState)
{
        flexcan_msgbuff_t * msg = NULL;
        (void)flexcanState;

        switch(eventType)
        {
        case FLEXCAN_EVENT_RXFIFO_COMPLETE:
                msg = _vt_get_msg(instance);
                vt_fw_rcv_msg(msg->msgId, msg->dataLen, msg->data);
                vt_toggle_led(leds[VT_CAN_RX_LED]);
                break;
        case FLEXCAN_EVENT_TX_COMPLETE:
                vt_toggle_led(leds[VT_CAN_TX_LED]);
                break;
        default:
                break;
        }
}
```

3.4 Initialize Agent

Below is the example code to initialize Agent. The ID filter is an option.

```
#include "vt_fw_if.h"
void vt_fw_oem_init(void)
{
   vt_status_t status;
   uint32_t id_buffer[VT_VECTOR_MAX_CAN_ID];
   int i, len = VT_VECTOR_MAX_CAN_ID;

   /* Initialize firewall */
   vt_fw_init(mkz_policy, mkz_trans_matrix);

   /* Set slot time to rule, default 200 us */
   vt_rule_set_slot_time(VT_PIT_PERIOD);

   /* Install call-back functions. These call-back functions is implemented by OEM.
     After VT_XMINUTE_TIME, these callback functions will get called */
   vt_rule_install_slot_callback(vt_fw_slot_report_matched);
   vt_rule_install_category_callback(vt_fw_category_report_matched);
   vt_rule_install_vector_callback(vt_fw_vector_report_matched);
}
```

vt_fw_init(car_policy, car_vector) will initialize the Agent with signatures (**car_policy** and **car_vector)**. Agent then monitor for any abnormal or attacked traffic. Customers can be able to customize the callback functions, e.g., report the alerts or block. Below is an example of callback function definition, it simply outputs the debug messages via UART.

```
/*!
 * @brief  This API will send matched of category data to report to server or print out.
 * @param [in]   *matched_t - pointer to vt_category_matched_t structure.
 * @return      status.
 */
vt_status_t vt_fw_category_report_matched(vt_category_matched_t *matched_t)
{
        char st[256];

        if(matched_t == NULL)
                return VT_STATUS_NULL;
        /* Put your code here */
        memset(st,'\0',256);
        sprintf(st,"Matched Category Rule Id: 0x%08lx, CAN_ID: 0x%08lx\r\n[0: 0x%08lx];[1:
0x%08lx];[2: 0x%08lx];[3: 0x%08lx];[4: 0x%08lx];[5: 0x%08lx];[6: 0x%08lx];[7: 0x%08lx]\r\n",
                        matched_t->rule_id, matched_t->can_id, matched_t->cat_bits[0], matched_t-
>cat_bits[1], matched_t->cat_bits[2], matched_t->cat_bits[3], matched_t->cat_bits[4], matched_t-
>cat_bits[5], matched_t->cat_bits[6], matched_t->cat_bits[7]);
        UART_SendDataBlocking(INST_UART_PAL1, (const uint8_t*)st, strlen(st),30);

        return VT_STATUS_SUCCESS;
}
```

3.5 Agent Processing.

vt_fw_process() API is used to start the agent. Firstly we use an auto-detect CAN baud-rate function.

```
void main(void)
{
 /*
  * System Initialization Code.
  */
  // ...
  // ...

 vt_fw_oem_init();

 /* Polling for Firewall processing */
 while(1)
 {
            if(bitrate == VT_BITRATE_UNKNOWN )
            {
                        /*
                         *  Automatically detect baudrate on CAN bus.
                         */
                        bitrate = vt_autodetect_bitrate(VT_INST_CAN);

                        if(bitrate < VT_BITRATE_UNKNOWN)
                        {
                                    vt_set_filter_rxfifo(VT_INST_CAN);
                                    vt_start_rcv(VT_INST_CAN);
                        }
            }
            else
            {

                        vt_fw_process();
            }
 }
}
```