www.vscom.de

VSCAN J1939 Manual

Edition: January 2018

Tel: +49 40 528 401 0
Fax: +49 40 528 401 99
Web: www.visionsystems.de
Support: faq.visionsystems.de

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Trademarks

VScom is a registered trademark of Vision Systems GmbH. All other trademarks and brands are property of their rightful owners.

Disclaimer

Vision Systems reserves the right to make changes and improvements to its product without providing notice.

Vision Systems provides this document "as is", without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Vision Systems reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Vision Systems assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Contents

1 Introduction **4**

2 Installation **5**
2.1 Windows . . . 5
2.2 Linux . . . 5

3 VSCAN API Functions **6**

4 VSCAN J1939 Functions **6**
4.1 VSCAN_J1939_Init . . . 6
4.2 VSCAN_J1939_Free . . . 7
4.3 VSCAN_J1939_Write . . . 7
4.4 VSCAN_J1939_Request . . . 8
4.5 VSCAN_J1939_Read . . . 8
4.6 VSCAN_J1939_SetPgnCallback . . . 9
4.7 VSCAN_J1939_Read_BufferCount . . . 10
4.8 VSCAN_J1939_Read_BufferClear . . . 10
4.9 VSCAN_J1939_ApiVersion . . . 10

5 VSCAN J1939 Types and Structures **11**
5.1 VSCAN_J1939_STATUS . . . 11
5.2 VSCAN_J1939_MSG . . . 11
5.3 VSCAN_J1939_NAME . . . 12
5.4 VSCAN_J1939_VERSION . . . 12
5.5 VSCAN_J1939_ADDR . . . 12
5.6 VSCAN_J1939_PGN . . . 12

6 Source Code Example **13**

7 Test Programs **15**
7.1 vs_j1939_dump . . . 15
7.2 vs_j1939_send . . . 15

8 Node.js and Node-RED Module **17**
8.1 Installation . . . 17
8.2 Node.js Module . . . 17
8.3 Node-RED Module and Example . . . 18

1 Introduction

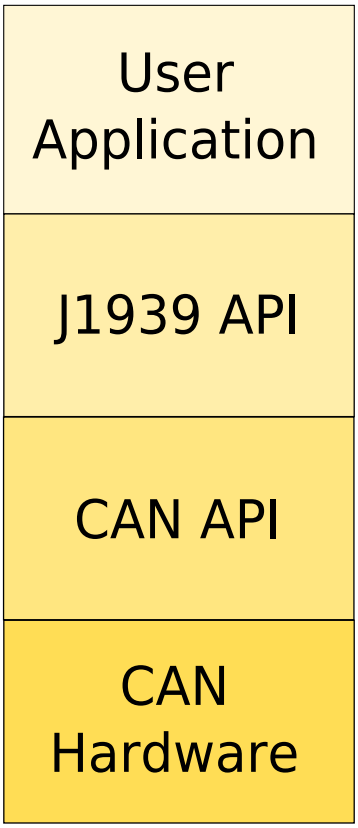
The SAE J1939 protocol is commonly used in the commercial vehicle area for communication in the commercial vehicle. But there are also other protocols based on J1939 like NMEA2000 for ships, ISOBUS for agricultural vehicles, MilCAN for military use and in the FMS (Fleet Management System) standard.

The Controller Area Bus (CAN) is used as the underlying hardware layer which is robust and has a priority based message protocol. It is also used in retail customer vehicles, but mostly with proprietary higher level protocols.

The CAN standard was upgraded from 11 bit to 29 bit identifiers to fulfill the need to embed all J1939 general message information (Protocol Data Unit) in the CAN identifier field. And a more complex layer was implemented to send more than 8 bytes per J1939 message to its destination. It is the Transport Protocol which is natively supported by the VSCAN J1939 stack.

You could use all these J1939 features with the rich family set of VScom CAN converters, which will allow you to use your J1939 device over the USB bus or from all over the world with our VScom NET-CAN and its VPN functionality.

This chart will show you all the layers needed for the use of the VScom J1939 stack:



2 Installation

2.1 Windows

Copy the files *vs_can_j1939.dll* and *vs_can_j1939.lib* into your project directory and add the library definition file (*vs_can_j1939.lib*) with the DLL symbols to your project. You will also need the VSCAN API (*vs_can_api.dll*) which you will find on the CD or on our website. Finally you must include the header files *vs_can_j1939.h* and *vs_can_api.h*.

2.2 Linux

You can chose between a static and shared library. Therefore include the header files *vs_can_j1939.h* and *vs_can_api.h* in your project. In the linker stage add one of *libvs_can_j1939.so* (shared) or *libvs_can_j1939.a* (static) to the linker options.

3 VSCAN API Functions

The J1939 API relies on the underlying VSCAN API and before you can use the J1939 functions, you have to open the CAN channel with these calls to get a handle to your VSCAN converter. Please take a look at our VSCAN Manual for further information.

Example:

```
VSCAN_HANDLE handle = VSCAN_Open("COM3", VSCAN_MODE_NORMAL);

VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_SPEED, VSCAN_SPEED_250K);

VSCAN_J1939_...();

VSCAN_Close(handle);
```

4 VSCAN J1939 Functions

4.1 VSCAN_J1939_Init

The `VSCAN_J1939_Init` function initializes the VSCAN J1939 stack.

```
VSCAN_J1939_STATUS VSCAN_J1939_Init(VSCAN_HANDLE Handle,
                                    VSCAN_J1939_NAME Name,
                                    VSCAN_J1939_ADDR_CALLBACK Callback);
```

Parameters:

`Handle`

[in] The handle of the CAN device

`Name`

[in] Describes the name field which will be used to uniquely identify nodes (ECUs) and their function in a J1939 network. This information will be used in the address claim procedure (ArbAddrCapable set to 1). Then a lower name field will win an arbitration process and claims the chosen address. If the ArbAddrCapable is set to 0, the device is not supporting an arbitration mechanism.

`Callback`

[in] A callback function which will return addresses for the address claim procedure. If the first address could not be acquired, the callback is called again and you have to return a second one. This step could be repeated if the next one also failed.

Example:

```
VSCAN_J1939_ADDR callback()
{
    return 252;
}
```

```
VSCAN_J1939_NAME name;

name.ArbAddrCapable = 1;
name.IndustryGroup = 0x01;
name.VehicleSys = 0x01;
name.Func = 0xff;
name.ManCode = 0x66;

VSCAN_J1939_Init(handle, name, callback);
```

4.2 VSCAN_J1939_Free

The `VSCAN_J1939_Free` function deallocates all resources that where acquired for the J1939 stack

```
VSCAN_J1939_STATUS VSCAN_J1939_Free(VSCAN_HANDLE Handle);
```

Parameters:

`Handle`

[in] The handle of the CAN device

4.3 VSCAN_J1939_Write

The `VSCAN_J1939_Write` function writes J1939 frames to the CAN bus.

```
VSCAN_J1939_STATUS VSCAN_J1939_Write(VSCAN_HANDLE Handle,
                                     VSCAN_J1939_MSG *Buf,
                                     DWORD Size,
                                     DWORD *Written);
```

Parameters:

`Handle`

[in] The handle of the CAN device

`Buf`

[in] An array of `VSCAN_J1939_MSG`'s that should be written to the CAN bus.

`Size`

[in] The element size of the `Buf` parameter.

`Written`

[out] Size of written J1939 frames to the CAN bus.

Example:

```
VSCAN_J1939_MSG msg[1];
DWORD written;

msg[0].Dst = 247;
msg[0].Priority = 0x5;
msg[0].PGN = 65262;
msg[0].Data = "testdata";
msg[0].Size = 8;
msg[0].Status = VSCAN_J1939_STATUS_FILL_IN; // make only sense for
                                            // transport protocol (>8 bytes)

VSCAN_J1939_Write(handle, msg, 1, &written);
```

4.4 VSCAN_J1939_Request

The `VSCAN_J1939_Request` function requests a PGN from another device. You can read the answer with `VSCAN_J1939_Read` or prior set up a callback function with `VSCAN_J1939_SetPgnCallback`.

```
VSCAN_J1939_STATUS VSCAN_J1939_Request(VSCAN_HANDLE Handle,
                                       VSCAN_J1939_PGN Pgn,
                                       VSCAN_J1939_ADDR Addr);
```

Parameters:

`Handle`

[in] The handle of the CAN device

`Pgn`

[in] The PGN which should be requested from the destination device.

`Addr`

[in] The address of the destination device.

Example:

```
VSCAN_J1939_Request(handle, 65262, 247);
```

4.5 VSCAN_J1939_Read

The `VSCAN_J1939_Read` function receive J1939 messages for the device.

```
VSCAN_J1939_STATUS VSCAN_J1939_Read(VSCAN_HANDLE Handle,
                                    VSCAN_J1939_MSG *Buf,
                                    DWORD Size,
                                    DWORD *Read);
```

Parameters:

`Handle`

[in] The handle of the CAN device

`Buf`

[out] An array of `VSCAN_J1939_MSG`'s which is filled with received messages

`Size`

[in] The element size of the `Buf` parameter

`Read`

[out] Size of received messages in `Buf`

Example:

```
VSCAN_J1939_MSG msg[32];
DWORD read;

VSCAN_J1939_Read(handle, msg, 32, &read);
```

4.6 VSCAN_J1939_SetPgnCallback

The callback function set over the `VSCAN_J1939_SetPgnCallback` function receive J1939 messages for the device. If you have set a callback and want to remove it, you could set the `Callback` parameter to `NULL`.

```
VSCAN_J1939_STATUS VSCAN_J1939_SetPgnCallback(VSCAN_HANDLE Handle,
                                              VSCAN_J1939_PGN Pgn,
                                              VSCAN_J1939_PGN_CALLBACK Callback);
```

Parameters:

`Handle`

[in] The handle of the CAN device

`Pgn`

[in] PGN number for which a user callback function is set

`Callback`

[in] A pointer to the user callback function for the specified PGN.

Example:

```
VOID MyPgnCallback(VSCAN_J1939_MSG *msg)
{
    // work with the message
}

VSCAN_J1939_SetPgnCallback(handle, 65262, MyPgnCallback);

// remove the callback function for the PGN
VSCAN_J1939_SetPgnCallback(handle, 65262, NULL);
```

4.7 VSCAN_J1939_Read_BufferCount

The `VSCAN_J1939_Read_BufferCount` function return the count of messages in the J1939 receive buffer.

```
DWORD VSCAN_J1939_Read_BufferCount(VSCAN_HANDLE Handle);
```

Parameters:

`Handle`

[in] The handle of the CAN device

Example:

```
DWORD count_of_received_messages = VSCAN_J1939_Read_BufferCount(handle);
```

4.8 VSCAN_J1939_Read_BufferClear

The `VSCAN_J1939_Read_BufferClear` function deletes all messages in the receive buffer.

```
VOID VSCAN_J1939_Read_BufferClear(VSCAN_HANDLE Handle);
```

Parameters:

`Handle`

[in] The handle of the CAN device

Example:

```
VSCAN_J1939_Read_BufferClear(handle);
```

4.9 VSCAN_J1939_ApiVersion

The `VSCAN_J1939_ApiVersion` function return the VSCAN_J1939_VERSION structure with version information.

```
VSCAN_J1939_VERSION VSCAN_J1939_ApiVersion();
```

Example:

```
VSCAN_J1939_VERSION version = VSCAN_J1939_ApiVersion();
```

5 VSCAN J1939 Types and Structures

5.1 VSCAN_J1939_STATUS

```
typedef int VSCAN_J1939_STATUS;
```

The type definition VSCAN_J1939_STATUS can have one of the following status values.

- **VSCAN_J1939_ERR_OK** - indicates that everything is okay
- **VSCAN_J1939_ERR_ERR** - indicates a general error
- **VSCAN_J1939_ERR_INVALID_HANDLE** - indicates that the handle which is used is not valid (e.g. CAN channel closed)
- **VSCAN_J1939_ERR_INVALID_PARAMETER** - a parameter which was passed to the function was invalid
- **VSCAN_J1939_ERR_MEMORY** - indicates that there is not enough memory to complete the function
- **VSCAN_J1939_ERR_SUBAPI** - indicates that an error occurred in a subordinated library
- **VSCAN_J1939_ERR_ADDR_UNAVAILABLE** - indicates that no free address was returned from the users callback function
- **VSCAN_J1939_ERR_ADDR_CLAIM** - indicates that the address claiming process failed
- **VSCAN_J1939_ERR_DATA_TO_SMALL** - indicates that the passed data was to small (must be >= 8 bytes)

5.2 VSCAN_J1939_MSG

The structure is used for the information of each J1939 message which will be received or transmitted.

```
typedef struct
{
    UINT32 PGN;       // Parameter Group Number
    UINT8 *Data;      // data which should be transmitted
    UINT16 Size;      // size of the data (max 1785 bytes / 7 * 255)
    UINT8 Dst;        // destination address
    UINT8 Src;        // our source address
    UINT8 Priority;   // message priority (max value of 7 / 0 = highest priority)
    UINT8 Status;   // Tx: one of VSCAN_J1939_STATUS_...
} VSCAN_J1939_MSG;
```

When you transmit a J1939 message, you must not set the *Src* field. If you send more than 8 bytes over the transport protocol, the field *Status* is used for the progress of the

message on the CAN bus, which persists of multiple CAN messages. To retrieve the *Status*, you must set it to *VSCAN_J1939_STATUS_FILL_IN* prior the VSCAN_J1939_Write call. Then the *Status* will change during the write to the actual one.

```
#define VSCAN_J1939_STATUS_SENDING  0x01  // send is in progress
#define VSCAN_J1939_STATUS_SENT     0x02  // send was okay
#define VSCAN_J1939_STATUS_ERROR    0x03  // error during send
#define VSCAN_J1939_STATUS_FILL_IN  0xff  // use this status field during write
```

5.3 VSCAN_J1939_NAME

The structure is used for the initialization of the J1939 stack. Take a look at VSCAN_J1939_Init for further information.

```
typedef struct
{
    UINT32 Id;              // Identity Number
    UINT16 ManCode;         // Manufacturer Code
    UINT8 EcuInst;          // ECU (Electronic Control Unit) Instance
    UINT8 FuncInst;         // Function Instance
    UINT8 Func;             // Function
    UINT8 VehicleSys;       // Vehicle System
    UINT8 VehicleSysInst;   // Vehicle System Instance
    UINT8 IndustryGroup;    // Industry Group
    UINT8 ArbAddrCapable;   // Arbitrary Address Capable
} VSCAN_J1939_NAME;
```

5.4 VSCAN_J1939_VERSION

The structure retrieves the J1939 API version information over the function

VSCAN_J1939_ApiVersion.

```
typedef struct
{
    UINT8 Major;
    UINT8 Minor;
    UINT8 SubMinor;
} VSCAN_J1939_VERSION;
```

5.5 VSCAN_J1939_ADDR

```
typedef UINT8 VSCAN_J1939_ADDR;
```

5.6 VSCAN_J1939_PGN

```
typedef UINT32 VSCAN_J1939_PGN;
```

6 Source Code Example

```
VOID MyPgnCallback(VSCAN_J1939_MSG *msg)
{
    int i;

    printf("Src: %d\n", msg->Src);
    printf("Dst: %d\n", msg->Dst);
    printf("Priority: %d\n", msg->Priority);
    printf("PGN: %d\n", msg->PGN);
    printf("Size: %d\n", msg->Size);

    printf("Data: ");
    for (i = 0; i < msg->Size; i++)
        printf("%02X", msg->Data[i]);
    printf("\n");
}

int main(int argc, char* argv[])
{
    VSCAN_HANDLE handle;
    VSCAN_J1939_NAME name;
    VSCAN_J1939_MSG msg[32];
    DWORD written, read;
    VSCAN_J1939_VERSION ver = VSCAN_J1939_ApiVersion();

    memset(&name, 0, sizeof(name));

    printf("API Version: %d.%d.%d\n", ver.Major, ver.Minor, ver.SubMinor);

    handle = VSCAN_Open("COM3", VSCAN_MODE_NORMAL);
    if (handle <= 0)
    {
        printf("Could not open VSCAN device\n");
        return -1;
    }

    if (VSCAN_Ioctl(handle, VSCAN_IOCTL_SET_SPEED, VSCAN_SPEED_250K) != VSCAN_ERR_OK)
    {
        printf("Setting CAN speed failed\n");
        return -1;
    }

    name.ArbAddrCapable = 1;
    name.IndustryGroup = 0x01;
    name.VehicleSys = 0x01;
    name.Func = 0xff;
    name.ManCode = 0x66;

    if (VSCAN_J1939_Init(handle, name, callback) != VSCAN_ERR_OK)
    {
        printf("VSCAN_J1939_Init failed\n");
        return -1;
    }

    if (VSCAN_J1939_SetPgnCallback(handle, 65262, MyPgnCallback) != VSCAN_ERR_OK)
        printf("VSCAN_J1939_SetPgnCallback failed\n");

    if (VSCAN_J1939_Request(handle, 65262, 247) != VSCAN_ERR_OK)
        printf("VSCAN_J1939_Request failed\n");
```

```
    // we wait 5 seconds to receive the answer
    Sleep(5000);

    VSCAN_J1939_Free(handle);
    VSCAN_Close(handle);

    return 0;
}
```

7 Test Programs

7.1 vs_j1939_dump

The *vs_j1939_dump.exe* will open a VScom CAN channel, read and display all J1939 messages to the console.

```
Usage: vs_j1939_dump.exe <COM Port/IP Address:Port> <Speed-Code>
Speed-Code in kb/s:
1 = 20
2 = 50
3 = 100
4 = 125
5 = 250
6 = 500
7 = 800
8 = 1000
```

Example:

```
vs_j1939_dump.exe COM3 5

Version:       1.0
API Version:   1.8.2
J1939 Version: 1.0.0

Source,Destination,Priority,PGN,DataLength,Data
S:000,D:255,P:3,PGN:61443,L:8,FF00FFFFFFFFFFFF
S:000,D:255,P:3,PGN:65132,L:8,FFFFFFFFFFFF4001
S:000,D:255,P:3,PGN:61444,L:8,FFFFFF4001FFFFFF
S:000,D:255,P:3,PGN:61443,L:8,FF00FFFFFFFFFFFF
S:000,D:255,P:3,PGN:65132,L:8,FFFFFFFFFFFF4001
S:000,D:255,P:3,PGN:61443,L:8,FF00FFFFFFFFFFFF
S:000,D:255,P:3,PGN:65132,L:8,FFFFFFFFFFFF4001
S:000,D:255,P:3,PGN:61444,L:8,FFFFFF0001FFFFFF
S:000,D:255,P:3,PGN:61443,L:8,FF00FFFFFFFFFFFF
S:000,D:255,P:3,PGN:65132,L:8,FFFFFFFFFFFF4001
S:000,D:255,P:3,PGN:61443,L:8,FF00FFFFFFFFFFFF

Received 11 frames
```

7.2 vs_j1939_send

The *vs_j1939_send.exe* send a J1939 test frame over a VScom channel to the CAN bus.

```
Usage: %s <COM Port/IP Address:Port> <Speed-Code> [Loop]
Speed-Code in kb/s:
1 = 20
2 = 50
3 = 100
4 = 125
5 = 250
6 = 500
7 = 800
8 = 1000
```

Example:

```
vs_j1939_test.exe COM3 5 1000

Version:       1.0
API Version:   1.8.2
J1939 Version: 1.0.0

Sent: 238
```

8 Node.js and Node-RED Module

8.1 Installation

To install Node.js (JavaScript runtime), you can use the package manager of your distribution (e.g. *„apt-get install nodejs"*). Many distributions will not supply a recent version, but you can install a newer one directly from the packages of the Node.js site.

Node.js itself has a package manager called „Node Package Manager" (npm), which will be installed with the main Node.js package.

Then you can install Node-RED with this package manager globally over *„sudo npm install -g –unsafe-perm node-red"*. For detailed information please take a look at this page.

Now you can download and extract the VScom Node-RED package *„node-red-contrib-vscan-wrapper-1.0.0.tgz"*. If you want to install it globally, you could run *„sudo npm install -g <directory>"*. If you want to install it locally for your user, you should change in the directory *„˜/.node-red/"* and run *„npm install <directory>"*. Then you will find the module in the sub-directory *„node_modules"*.

Afterwards you must install the VScom package *„node-red-contrib-vscan-1.0.0.tgz"* and like the package before, you could install it globally or locally. The wrapper package contains native Linux libraries, which you can copy to your global library path or set an environment variable to tell your system, that it should also look in this directory, with *„set LD_LIBRARY_PATH=˜./node-red/node_modules/node-red-contrib-vscan-wrapper"*.

Now you're ready to start node-red for a test with *„node-red -v"* and make a connection to the server with the browser of your choice (*„http://<ip address>:1880"*). There you will see the vscan node in the input nodes list.

8.2 Node.js Module

You can use all Node.js methods in the same way as you do with the general API functions in the chapter VSCAN J1939 Functions.

Example:

```
var ref = require('ref');
var ffi = require('ffi');
var vscan = require("./node.js_wrapper.js")

var addrCallback = ffi.Callback(vscan.vsJ1939Addr, [],
    function() {
      return 252;
    });

// make an extra reference to the callback pointer to avoid GC
process.on('exit', function() {
  addrCallback
});
```

```
// switch on debugging on console
//var status = vscan.VSCAN_Ioctl(0, VSCAN_IOCTL_SET_DEBUG, VSCAN_DEBUG_HIGH);
//status = vscan.VSCAN_Ioctl(0, VSCAN_IOCTL_SET_DEBUG_MODE, VSCAN_DEBUG_MODE_CONSOLE);

var version = vscan.VSCAN_J1939_ApiVersion();
console.log(`J1939 Version: ${version.Major}.${version.Minor}.${version.SubMinor}`);

var canHandle = vscan.VSCAN_Open('/dev/ttyUSB0', 0);
if (canHandle < 0) {
  console.log('could not open CAN channel');
  process.exit(-1);
}

var codeMask = new vscan.vsCanCodeMask;
codeMask.Code = 0;
codeMask.Mask = 0xffffffff;
status = vscan.VSCAN_Ioctl(canHandle, vscan.VSCAN_IOCTL_SET_ACC_CODE_MASK, codeMask.ref());
status = vscan.VSCAN_Ioctl(canHandle, vscan.VSCAN_IOCTL_SET_SPEED, vscan.VSCAN_SPEED_250K);

var name = new vscan.vsJ1939Name;
name.ArbAddrCapable = 0;
name.IndustryGroup = 0x01;
name.VehicleSys = 0x01;
name.Func = 0xff;
name.ManCode = 0x66;

status = vscan.VSCAN_J1939_Init(canHandle, name, addrCallback);

var msg = new vscan.vsJ1939Msg;
var read = ref.alloc('uint32');
for (;;) {
  var cnt = vscan.VSCAN_J1939_Read_BufferCount(canHandle);
  if (cnt > 0) {
    vscan.VSCAN_J1939_Read(canHandle, msg.ref(), 1, read);
    console.log("S:%d,D:%d,P:%d,PGN:%d,L:%d,%s",
      msg.Src, msg.Dst, msg.Priority, msg.Pgn, msg.Size, msg.Data.toString('hex', 0, msg.Size));
  }
}

vscan.VSCAN_J1939_Free(canHandle);
vscan.VSCAN_Close(canHandle);
```

Listing 1: Node.js Testprogram

8.3 Node-RED Module and Example

The following example use a hardware curcuit, which emulates a J1939 truck.

The VSCAN node will output all J1939 messages as a VSCAN_J1939_MSG structure. You can retrieve the message in a function node and work with the data programmatically. You will find an example in the Node-RED menu *„Import/Examples/vscan"* (Figure *1*).

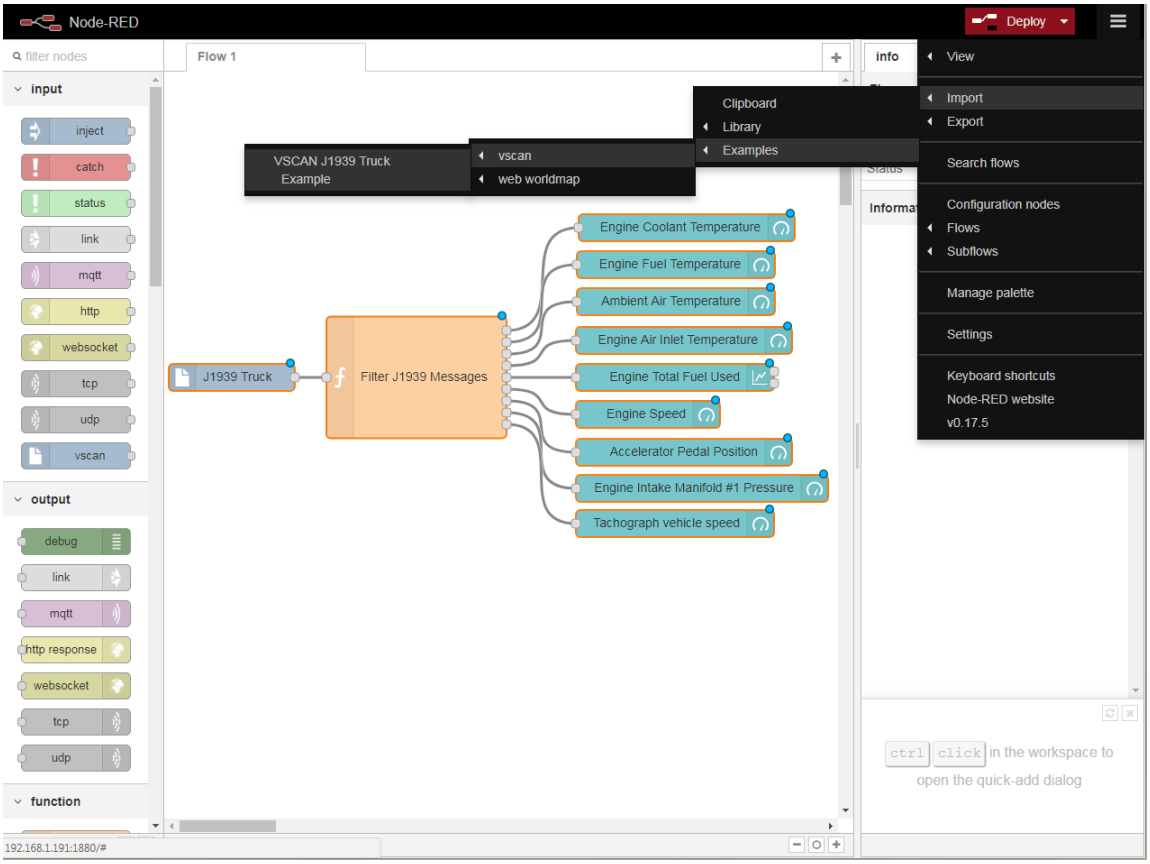


Figure 1: VSCAN J1939 Truck Example (Import)

If you have placed it on your working area, you should set-up the VSCAN node (Figure 2). Therefore you must double-click it and enter the serial port of your USB-CAN or enter the IP address and port number of your NET-CAN. When you use the Node-RED daemon as a normal user, ensure that you are in the group that has access to tty interfaces, when you use the USB-CAN. In the setting tab, you could also change the interval at which new messages are passed to the function routine.

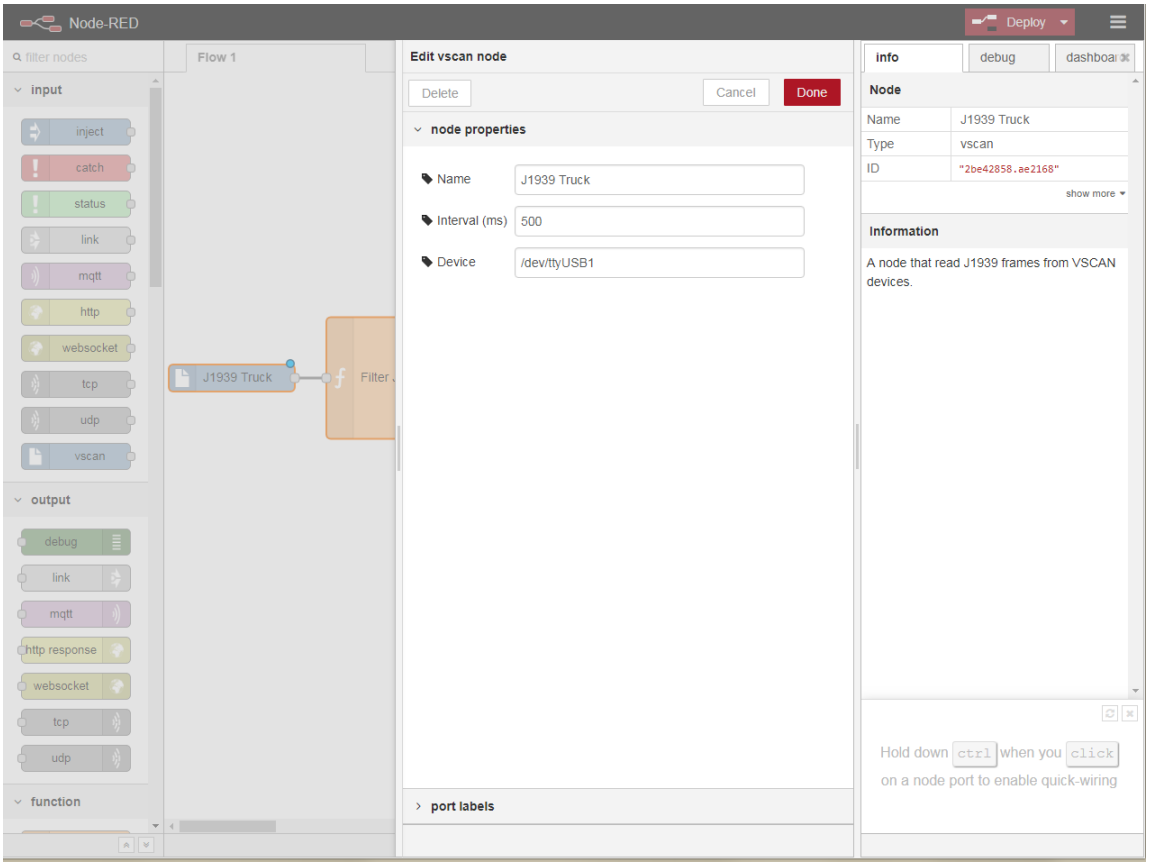


Figure 2: VSCAN J1939 Truck Example (Config)

To take a look at the filtering code based on the PGNs, please double-click the function node (Figure 3).

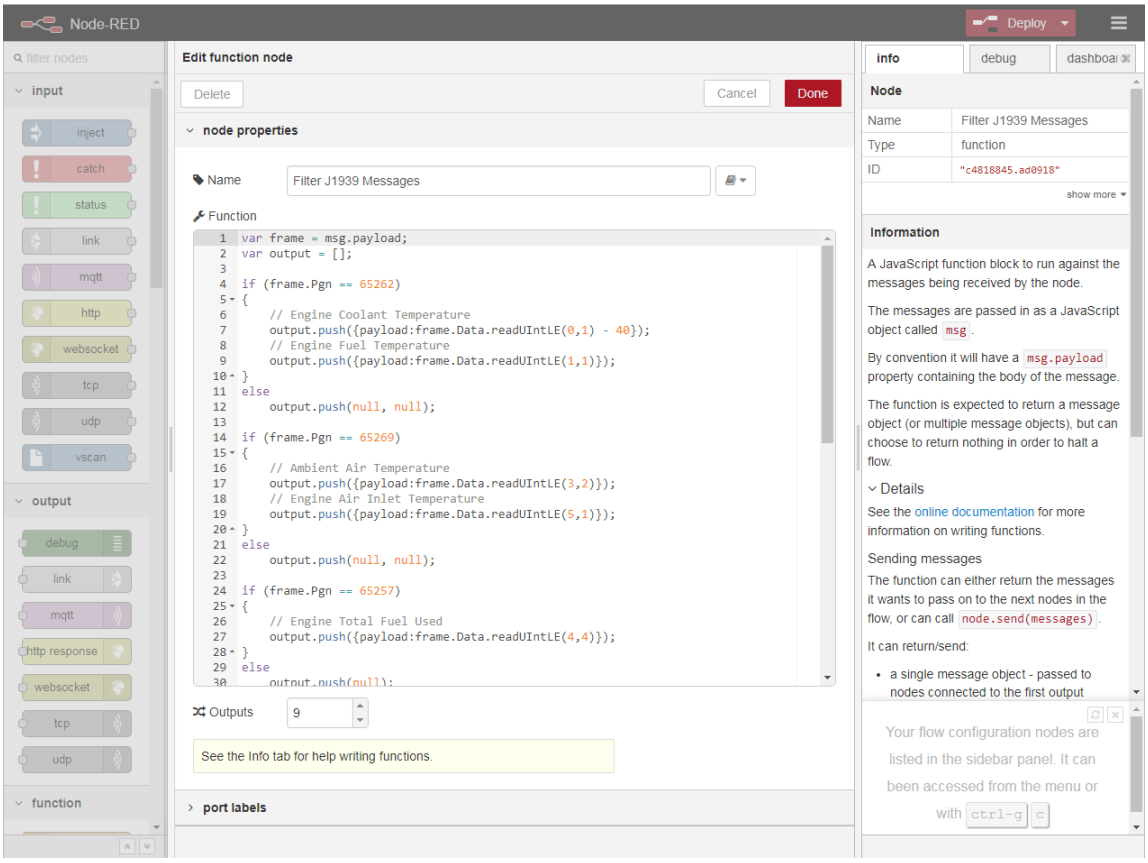


Figure 3: VSCAN J1939 Truck Example (Code)

If you have everything set-up and ready, you could deploy the flow to the Node-RED server and switch over to the dashboard (*„http://<ip address>:1880/ui"*) to see the output (Figure).

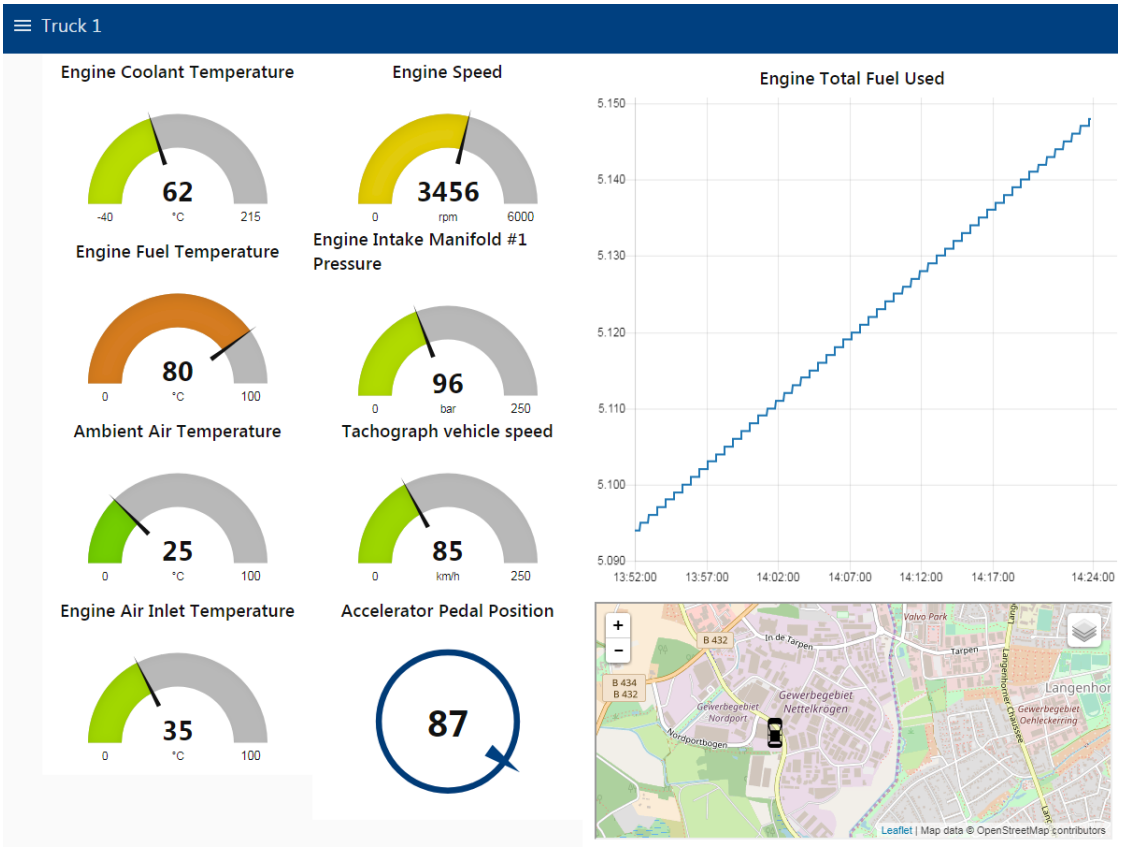


Figure 4: VSCAN J1939 Truck Example (Dashboard)