LabVIEW® API for PCAN©-Developer 4.x
by
KDI Kunze Digital Instrumentation

CONTENTS

PURPOSE AND SCOPE

This document describes the easy-to-use LabVIEW® API. This API uses the PCAN©-Developer 4.x API from PEAK-System to establish one or more CAN networks. Using the LabVIEW® API, you can exchange CAN messages via CAN 1.0/2.0B and CAN FD with the PCAN© hardware. CAN FD is supported by the PEAK device driver from version 4. The LabVIEW® API is strongly related to the PCAN©-Developer 4.x help files and supports LabVIEW® 2014 or any higher version.

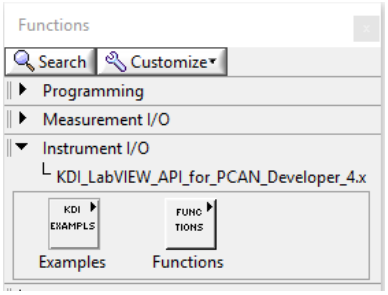
LICENSE CONSTRAINTS

The LabVIEW® PCAN©-Developer 4.x API (API) is a floating license and can be installed on different computers without additional licensing fees. LabVIEW® developers have the right to build applications with this API without additional license fees.

Using the same API license simultaneously on different computers for development purposes require additional licenses. The number of licenses depends on the number of simultaneously used APIs.

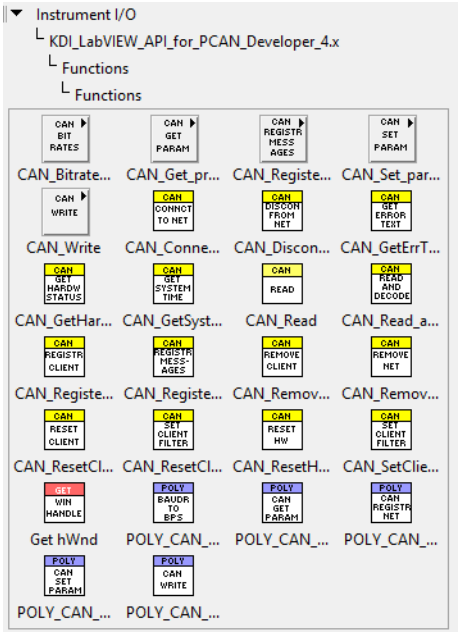
INSTALLATION AND VI LOCATIONS

To install the LabVIEW® API the VI Package Manager is needed. It's part of LabVIEW® since years. To install the API, just make a double click on the kdi_labview_api_for_pcan_developer_4.x.vip file and follow the instructions.



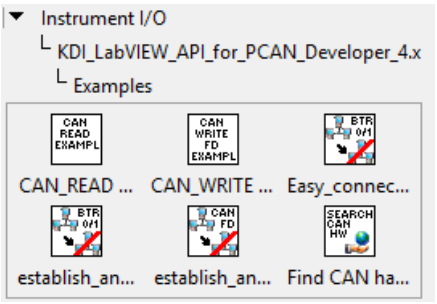
Once installed, the VIs are located in the Instrument I/O function palette of the block diagram palette.

In the KDI_LabVIEW_API_for_PCAN_Developer_4.x palette, you can find the functions (Vis according to the PCAN© documentation paragraph Functions) and examples for CAN 1.0/2.0B and CAN FD.



There are also sub palettes for the Get and Set functions for parameter settings, Bitrate functions, CAN_Register_Messages and CAN_Write. These sub palette functions are part of the polymorphic and available from the lower part of this palette.

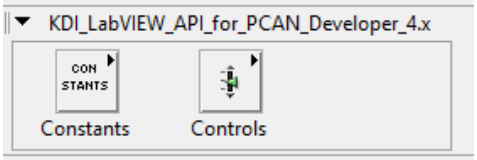
In addition to the PCAN© documentation this palette includes a VI called CAN_Read_and_decode.vi. This VI uses CAN_Read.vi to read ot the while CAN message queue and decodes the messages according to the Can_Record types of the PCAN© documentation for the developer driver 4.x.



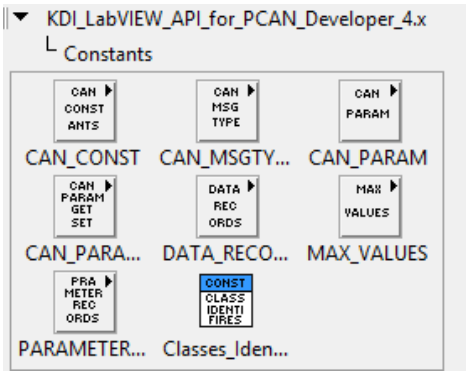
Six examples are included to demonstrate how to establish a CAN network using different methods and how to send and receive CAN messages.

Each and every VI has a context help in English and German. Activate the context help by pressing CTRL + h or STRG + h on the keyboard. The documentation is likewise the same as the PCAN© Developer 4.x documentation.

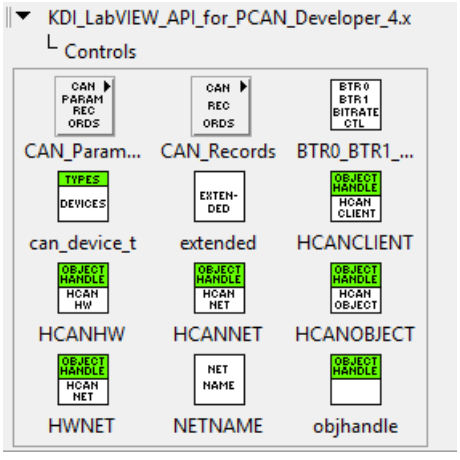
CONTROLS AND CONSTANTS LOCATION



All the controls and constants are located in the front panel palette.



Constants are located in the constants sub palette. Here you'll find all the necessary constants needed to call all the PCAN© Developer driver functions. All names are strictly the same as in the PCAN© Developer documentation.

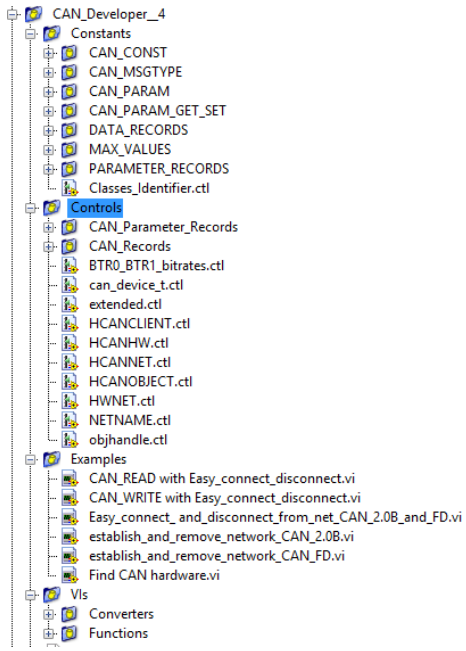


Controls are located in the controls sub palette. Here you'll find all the necessary controls needed to call the PCAN© Developer driver functions. All names are strictly the same as in the PCAN© Developer documentation.

Since it is not target of this documentation to document all the Vis and Controls you have to read the LabVIEW® context help and the PCAN© Developer 4.x documentation for details. This especially if you want to use the Set and Get Vis.

PROJECT OVERVIEW

To get a more detailed insight I'll give you some more information of the project. First of all you may need a general overview.



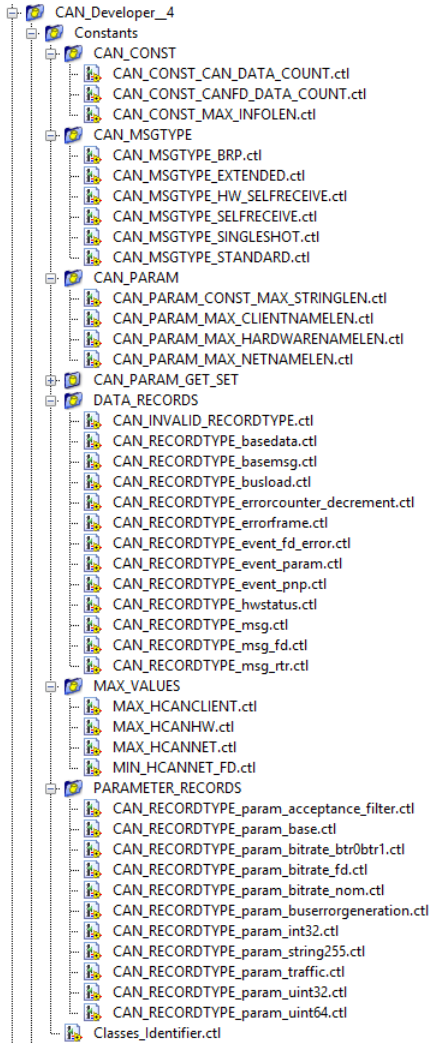
Within the project tree you can see all the controls, constants, examples and Vis. Like described before the constants and controls are strictly the same as described in the PCAN© Developer 4.x documentation. They are separated according to the documentation.

The examples are part of the project and will be explained later.

Within the Vis branch you'll see a converters branch. It contains routines to split and join numbers in different ways.

The functions branch include all Vis to handle the driver communication.

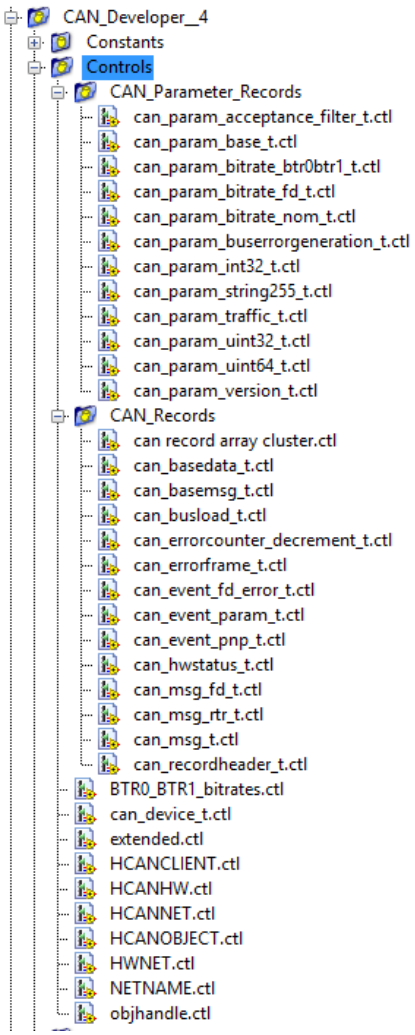
CONSTANTS



Let's have a look into the constants branch.

Here you'll find all the constants, CAN_MSGTYPEs, CAN_PARAMs, CAN_PARAMS for Get and Set functions, DATA_RECORDS, maximum values, PARAMETER_RECORDS and the Classes_Identifier (enum).

Within the explanation of the examples you'll learn how to use them.



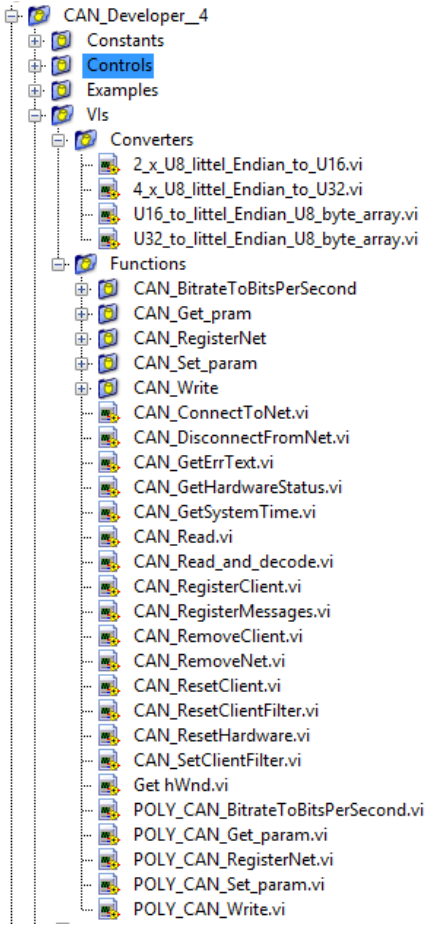
The controls branch include all the controls to use the driver functionality.

The CAN_Parameter_Records are almost used for Get and Set functions.

While CAN_Records are used for communication purposes.

The most common controls are starting with BTR0_BTR1_bitrate control. These controls are always in charge while your program is running.

FUNCTIONS



Within the VIs branch you'll find all the functions and converters used in the project. All the functions have the same names as the names in the PCAN© Developer 4.x documentation, except the Converters, CAN_Read_and_decode and Get_hWnd.

CAN_Read_and_decode is a VI that will receive all CAN data records and decode them. You may edit the block diagram according to your needs.

Get hWnd fetches the Windows handle of the application. The handle is used to register the client.

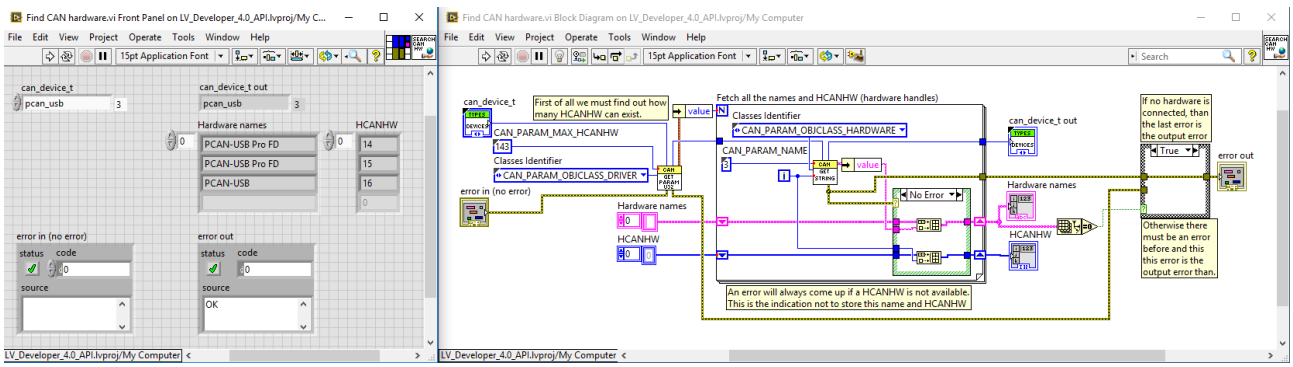
All the polymorphic Vis are located here while the according Vis are located within the corresponding folders above.

EXAMPLES

First of all I have to explain a little bit how the works.

The driver gives you the possibility to connect a virtual CAN bus net to each PCAN© hardware. You can attach this net manually or programmatically to the hardware. You can attach clients to each net. These clients are programs running independently. All the clients will receive all the CAN frames send by another client on the net or from external CAN devices that are connected to the hardware of this internal virtual net.

Seams a little bit difficult but we will start with a small routine searching for the hardware connected to your PC.

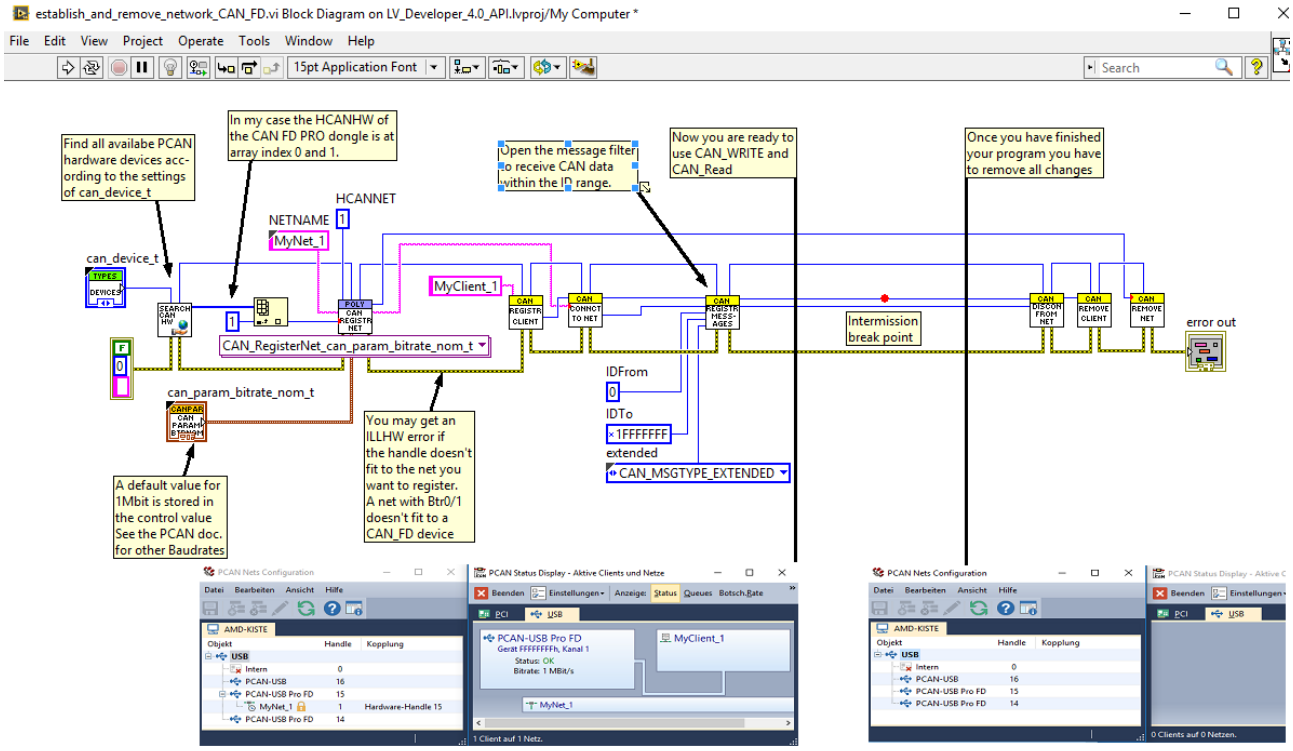


We are using the Get_Param function to find out how many hardware devices can be possibly connected to the PC. Once we have this information we can run a for-loop to find the connected hardware by using Get_Param, CAN_PARAM_NAME (constant). Whenever the feedback string isn't empty the program has found a new hardware and also the corresponding hardware handle. These hardware handles are used to register a net.

In the Front Panel (FP) above you can see the connected hardware found by the VI and the according hardware handles.

ESTABLISH AND REMOVE NETWORK AND CLIENT

Once the program has found all the device handles it is possible to register a new net. Therefore you might pick a hardware handle from the output array of hardware handles (HCANHW). See the block diagram below where the 2'nd element of the HCANHW handles array is used to register a new net.



A new Net must contain a net name, can_param_bitrate_xxx (here nominal for FD), a suggested net handle (HCANNET) and the HCANHW device handle. The parameter can_device_t is always used to deter main the kind of device that is being used.

After registration the net we have to register a new client. Therefore we need the can_device_t information and a client name. Output of register_client.vi is a client handle. In this state or the program the client is registered but not connected to a net. The next step is to connect the client to the net that we have registered before. Therefore the net name, can_device_t and the client handle are required. This is the point where you should see the information displayed in PCAN© Nets Configuration and PCAN© Status Display.

The net and client are now connected but the client still doesn't get messages from the external connected device. The client input filter must be set to deter main the CAN IDs that should be received by the client. In the program I have opened the whole message filter, but it's also possible to use a loop if you have a special bunch of IDs the client shall receive.

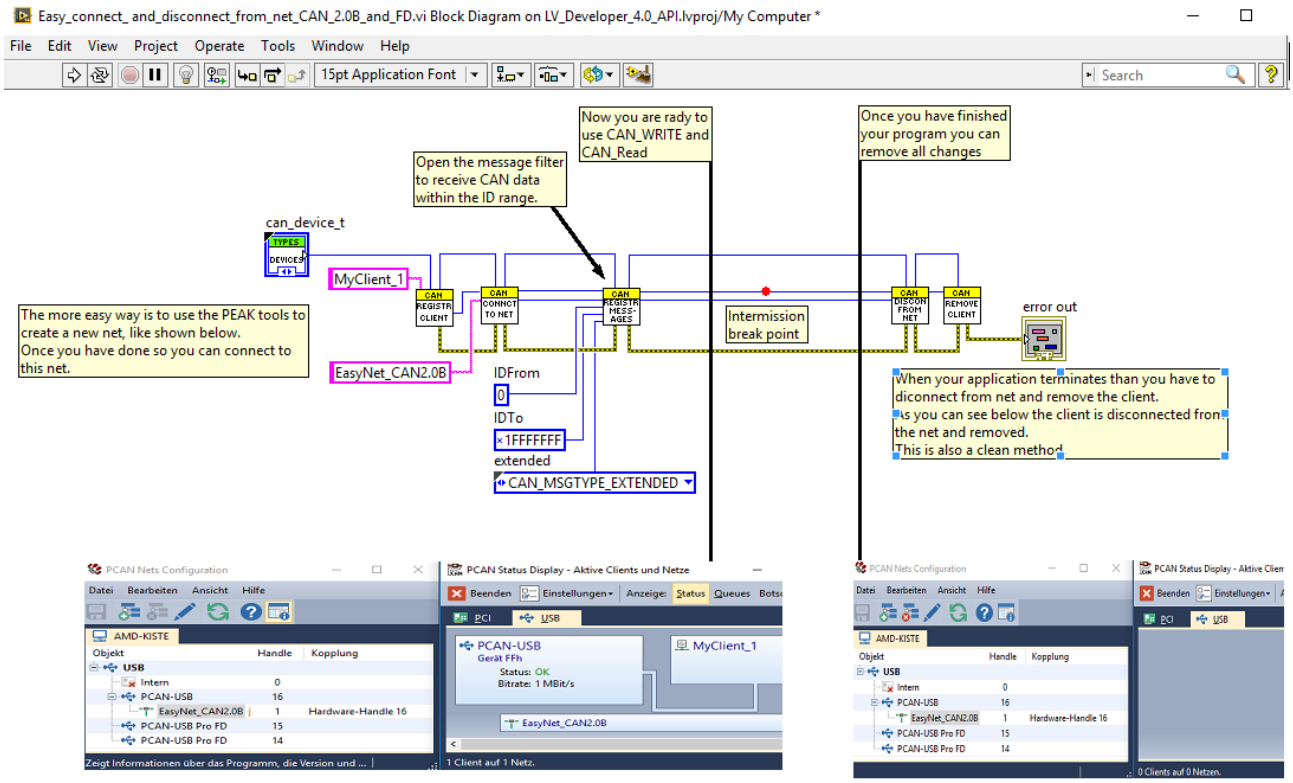
Yet the client is ready to receive CAN data and the program will stop at the break point now.

If you click on continue you'll see that client and net will be removed from the hardware.

This is the way to establish and remove a net it your program starts within an unconfigured environment, but there is a slightly more easy way to work with the driver.

EASY CONNECT AND DISCONNECT FROM NET

A more common way is to configure the nets using PCAN© Nets Configuration. You can configure a net attached to the hardware of your choice. Once you have done this you can use the VI below to register a client and connect the client to the net.



The program does exactly the same as shown in the 'establish and remove network and client' example before except that it uses a predefined net with a given net name.

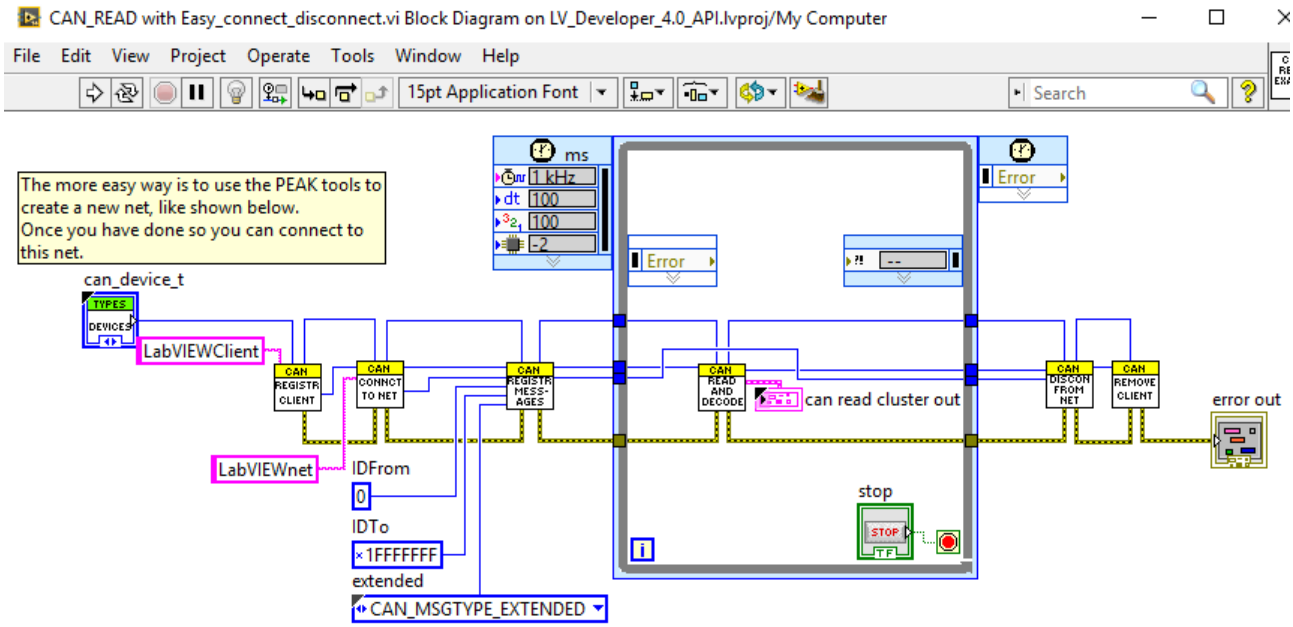
If you compare it with the example before you can see that the net is not removed after the program is finished.

CAN_READ

Using a preconfigured net like shown above I have programed a CAN_read example. The preconfigured net is named 'LabVIEWnet' and the client name is choosen as LabVIEWClient.

This VI will poll the receive queue of the client every 100ms. This means that after 100ms the whole client receive queue will be read out. The CAN records read out from CAN read will be decoded and accumulated within a cluster of cluster arrays. I had to do this because of the limitations of the VI connector pane and the amount of different messages that can be received by CAN_Read.

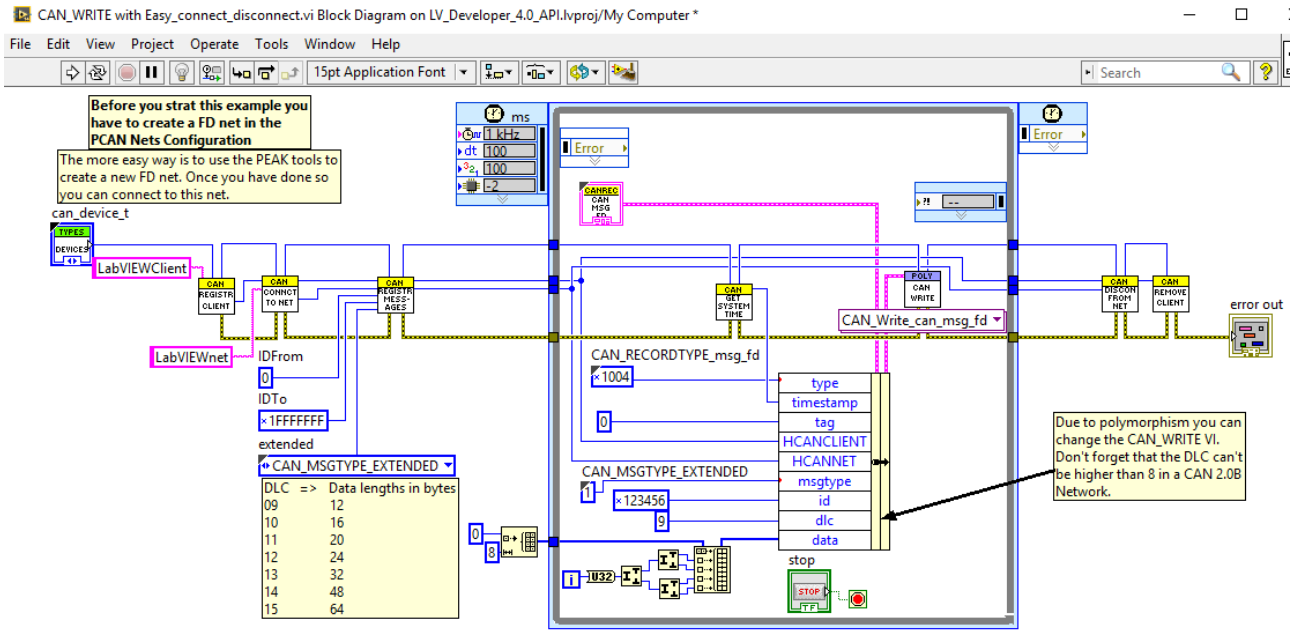
For details of each CAN record see the PCAN© Developer 4.x documentation.



The VI will stop if you press the stop button on the front panel.

CAN_WRITE

According to CAN_Read this example will use a preconfigured net too. The VI will connect to this net and send CAN messages on the net. In this case I have predefined a CAN FD net. As you can see the DLC is 9 witch means that 12 bytes will be written according to the table within the block diagram.



CAN_Write is a little more complex than CAN_Read as you can see. The VI sends CAN FD messages on the bus within a time frame of 100ms and the last four bytes of the CAN data frame is the iteration counter of the while-loop. You can modify this example for CAN 1.0/2.0B using CAN_RECORDTYPE_msg.ctl, reduce the initialized array to four and set DLC to eight.

The timestamp within the example can the zero too. The driver will send messages with a timestamp lower than the actual timestamp immediately. If you want to, you can prepare messages with a predefined timestamp that is higher than the current time and put it into the transmit queue of the driver. Keep in mind, that these CAN messages must be sorted by timestamp (from low to high). If you

put afterwards CAN messages into the transmit queue with a timestamp lower or zero the driver will transmit these messages after the transmission of the CAN messages with the preconfigured timestamp because they are already in the transmit queue.

REFERENCES

LabVIEW®	is a registered trademark of National Instruments
PCAN©	is a registered trademark of PEAK-System Germany

COPYRIGHT

Dipl.-Ing. (FH) Martin Kunze

Fröbelstraße 11
D-26127 Oldenburg
E-Mail: mailto:support@digiinst.de