Fieldbus Access LinX-Base 3

Getting Started

crosscontrol
www.crosscontrol.com

Contents

Revision history

Rev	Date	Author	Comments
1.0	2016-06-10	D. Nisses-Gagnér	First Version
1.1	2016-11-08	D. Nisses-Gagnér	Review for FA 1.0.0 patch release
1.2	2016-11-17	D. Nisses-Gagnér	Workshop findings - update
1.3	2016-12-16	D. Nisses-Gagnér	Merge Conf Tool info from Quick guide
1.4	2016-12-30	D. Nisses-Gagnér	Update/clairification of example + FAQ
1.5	2017-01-09	D. Nisses-Gagnér	Added Widget based example
1.6	2017-05-18	D. Nisses-Gagnér	Error handling info added
2.0	2018-08-08	P.Görling	Updated for VM v3.x

Glossary

Word/Abrevation	Explanation
CrossTecc	A set of applications and components...
crc	Cyclic redundancy check
SAP	Software Application Platform
Qt	Development framework qt.io
RTE	RunTime Environment
API	Application Programming Interface
VDM	Virtual Development Machine
VM	Virtual Machine
GUI	Graphical User Interface

1. Brief Introduction

Fieldbus Access is a communication bridge containing the implementation of the most widely used vehicle fieldbuses (at the moment onlyJ1939 but CANopen will be added in the future) and enables all other LinX Software Suite modules to use them with an easy-to-use interface. The Fieldbus Access UX configuration tool provides an easy interface to manage fieldbuses and which signals to use in your application.

Accessing CANbus signals from your GUI application is simple. Install the FieldbusAccess runtime in the LinX Software Suite development VM and on your target device. Packages are found from our support site, http://support.crosscontrol.com/%20downloads/LinX_Software_Suite

From your GUI application you connect and access your signals as before via Data Engine. With the added Fieldbus Access runtime you have accessibility to the can-bus signals. Mapping between DE signals and can-bus signals is simply done from our configurator that is integrated in the development tool.

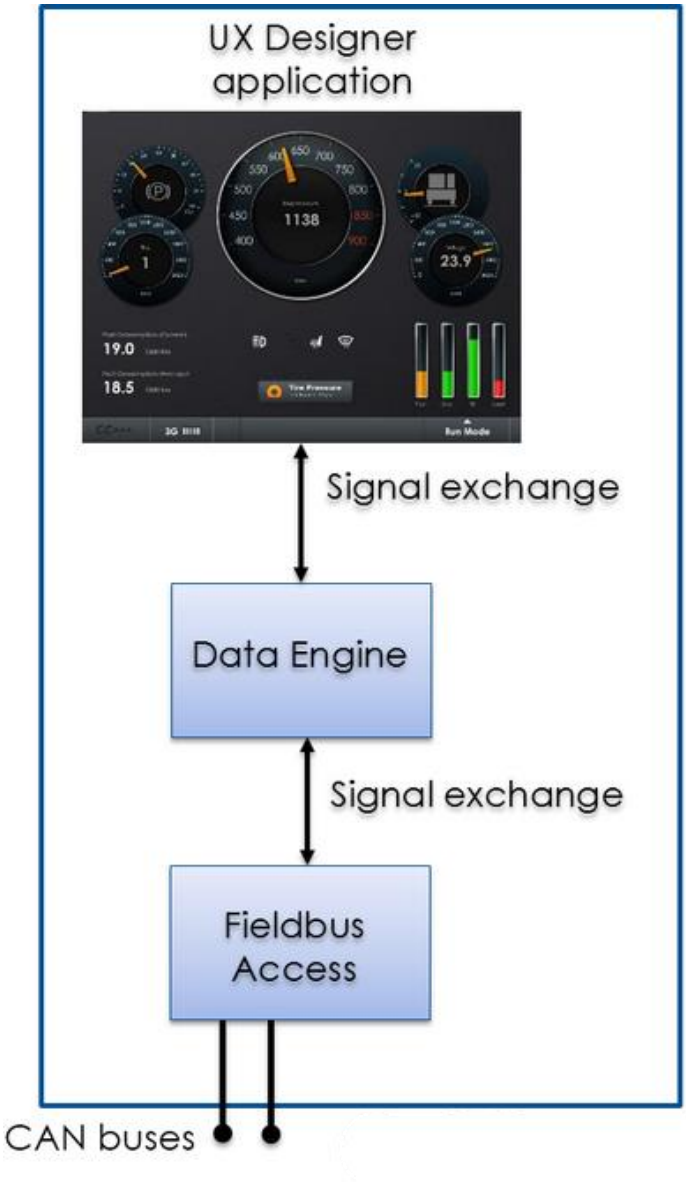


Figure 1 Signal flow between components

The configurator tool, further explained later in this documentation, generates a configuration file that is used both from the development environment and from the deployed GUI application. Also all code connecting and creating signals from the GUI application is now generated and automatically assigned your GUI project from the integrated tool. Booth QML and Widget based code is supported. This enables code completion within your projects. Simply setup your signals from the graphical configuration tool and then you are fully set to use these signals in your applications.

2. Configuration Tool

2.1. LinX Fieldbus Access

A QtCreator plugin, User Interface, managing the setup and configuration of CAN/DE Signals of the system. Aside from generating configuration files for fieldbus layer, this configuration tool can auto generate the communication layer between your Qt application and Data Engine, exposing your configured signals to QML and widgets with a single mouse click.

Files generated by the Fieldbus Configuration Tool:

- qtobserver.h (qtobserver.cpp)
- dataenginesignal.h (dataenginesignal.cpp)
- dataenginebase.h (dataenginebase.cpp)
- dataengine.h (dataengine.cpp)
- fieldbusaccesserror.h (fieldbusaccesserror.cpp)
- fieldbusaccesserrormodel.h (fieldbusaccesserrormodel.cpp)

Since these files are overwritten every time the linked fieldbus configuration is changed, it is not recommended to make any custom edits to these files. Sub-class the DataEngine class (dataengine.h) and override relevant parts instead. If you do so, remember to expose your own derived class to QML or your widgets, instead of the default one.

From the configuration tool you have access to the an extract of the J1939 signal library. Under the tab “Library” to the right, you can simply search and drag-and-drop the desired PGNs/signals to your configuration. Some information such as bit-position, length etc. will be prefilled, other will be filled with an explanatory text and marked with red to indicate user configuration (cycle time etc.).

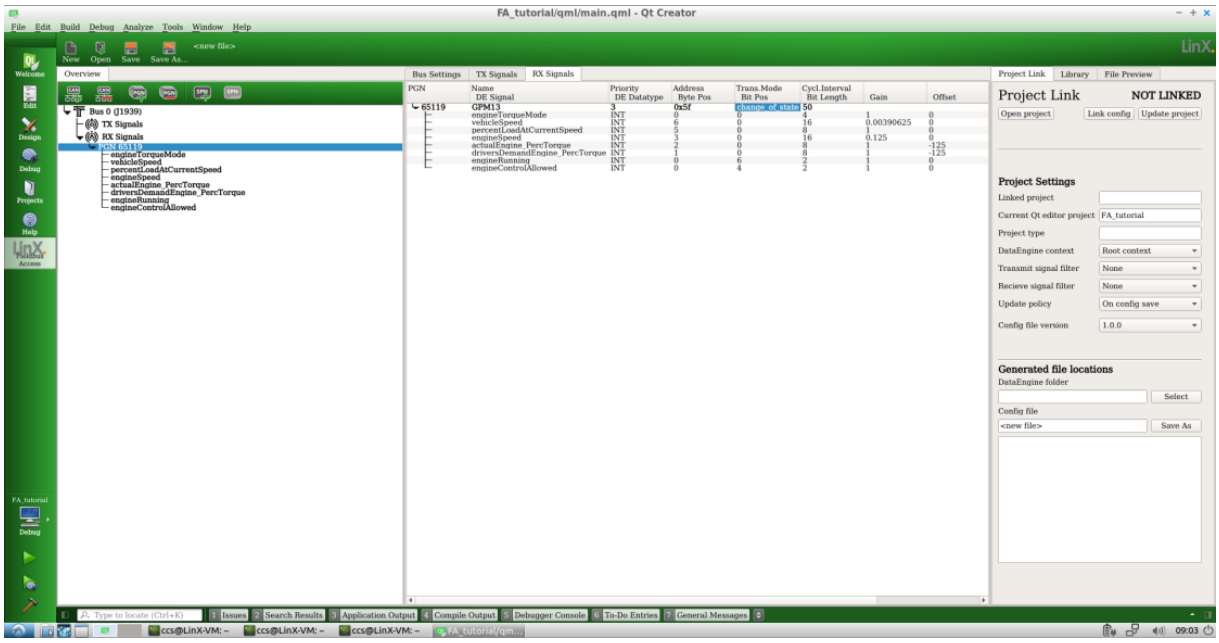


Figure 2 LinX FA Configuration Tool

Figure 2b File menu

2.1.1. Configuration – Left Column

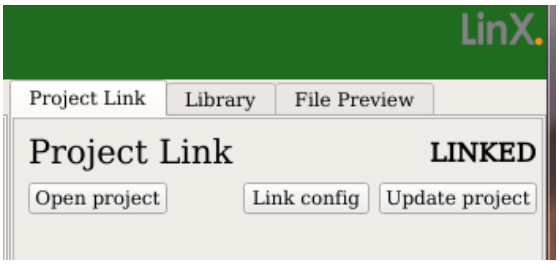
On the top left side you will find menu buttons used to open/close/save etc. the configuration file. Here is also the basic configuration with Bus as root and Signals on second level. Signals are grouped in TX (Transmit from GUI application to CAN bus) and RX (Receive from CAN bus to GUI application). The PGN signals are expandable and hold its SPN's underneath.

2.1.2. Configuration – Middle (Wide) Column

In the wide middle column you find more information and are also able to change different settings. By clicking on the bus (left column) will bring up configuration of the bus in the middle view. Here you are able to set bus number, baud rate, node identifier name etc. If clicking instead on some of the signals (left column) will change the view in the middle where configuration of signals are possible (similar to picture above). PGN/SPN Byte/Bit position are indexed between 0-7. Raw can-messages uses Gain 1.0 with Offset 0.

2.1.3. Configuration – Right Column

There are three tabs in the Configuration column to the right:



Project Link **(see ch 2.1.3.1)**

Library **(see ch 2.1.3.2)**

File Preview **(see ch 2.1.3.3)**

2.1.3.1. Project Link tab

The Project Link tab contains three buttons:

Open Project

Selects/opens the configuration file (.json) that is linked/connected to the Qt-project.

Link config

Link the FA configuration to your Qt project. Only used when creating a new configuration and want to link (connect) that configuration to the project currently selected.

Update Project

When you are done changing the desired signals and CAN busses, save the file and link it to your application project. The .json configuration file is created/updated and also some cpp code is generated that is used to create and link your application to the data engine.

2.1.3.2. Library tab

All the PGNs present in the J1939 signal library extract are found in this tab. Search and filter the signals for an easy overview of the signals you need and then drag them to the left for configuration. You are able to create and import your own custom library (.csv or .dbc file) for convenient use in all your projects.

2.1.3.3. File Preview tab

This tab shows a preview of the created .json file that will be linked to the current project.

3. Simple Example

A normal workflow could look something similar to this example. The *LinX Software Suite - Getting Started.pdf* guide found on our supportsite explains further how to setup a project and compile/deploy and execute on target.

3.1. Create CAN bus settings and signals

This chapter describes how to create a new LinX FA configuration to a Qt project.
First, start with creating a new GUI application or open an existing Qt project.

- In the menu press File->New File or Open Project
- We use the mtcc QML template in this example.
- Name your project and save it (usually /home/ccs/qt/MyProjects)
- For resolution, refer to your devices technical manual. This can be changed manually in your project later!
- Kit Selection, chose *CCpilot ARM (VS or VI2) Qt-5.7.1* and *Virtual Development Qt5*

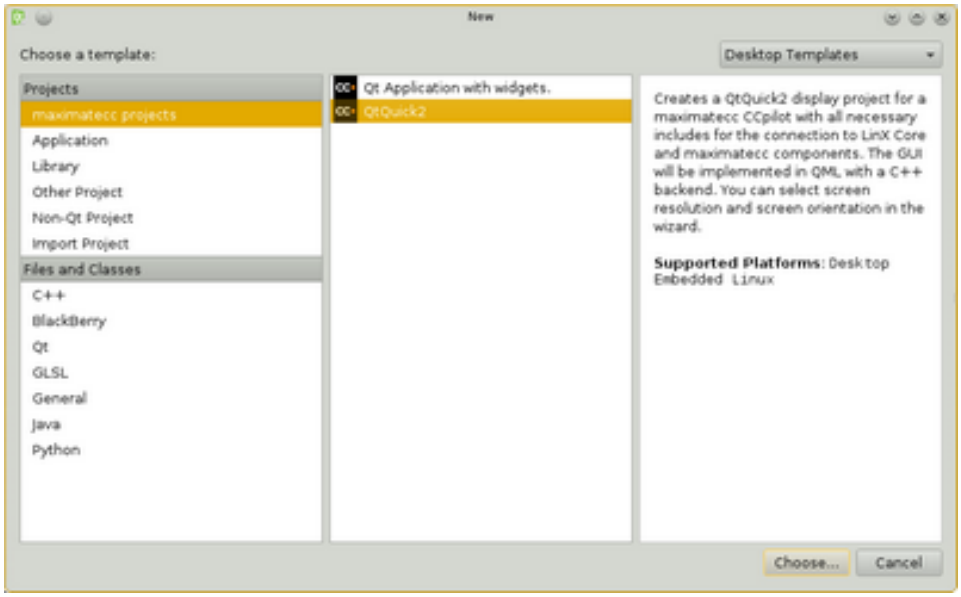


Figure 3 QtQuick (QML) template

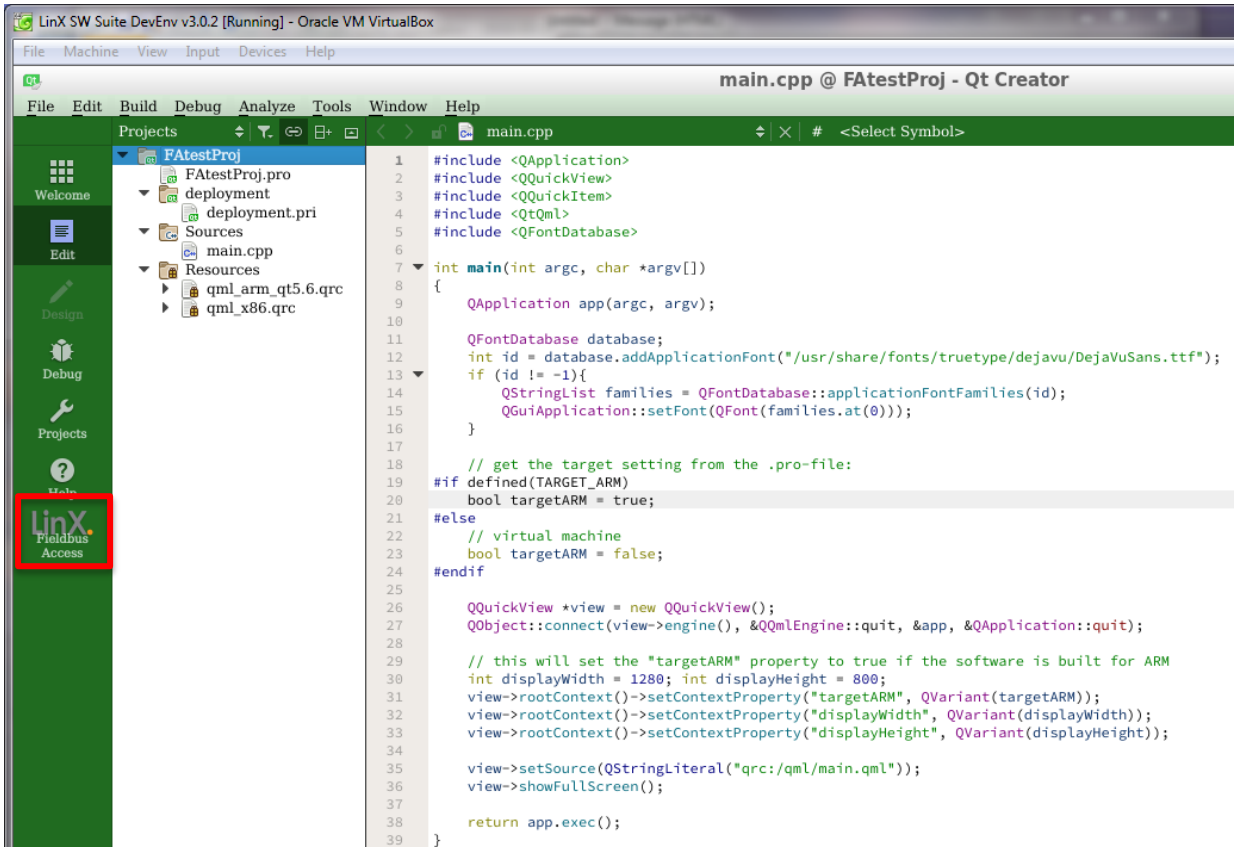


Figure 4 Quick project from template

Press the LinX Fieldbus Access Configuration tool (see above) and setup/manage your signals.

- A CAN bus is added by default and the Rx/Tx tree structures are empty.
- In the right panel, press the “Library” tab and search for the signals you need, i.e. “Enginespeed”. Drag PGN 65119 (0xFE5F) to RX Signals on the left panel. It is also possible to drag PGNs or SPNs to the middle info panel (if TX or RX tab is selected).

For easier overview when we test the application later we change the Trans. Mode to On Change (default for this signal is cyclic).

- Double click on cyclic and choose “change_of_state” from the drop down.

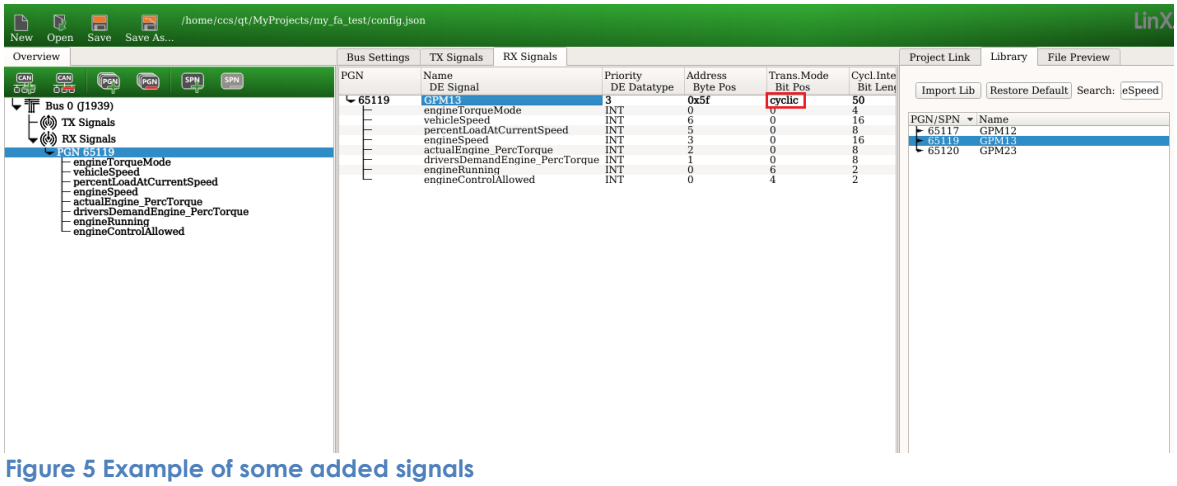
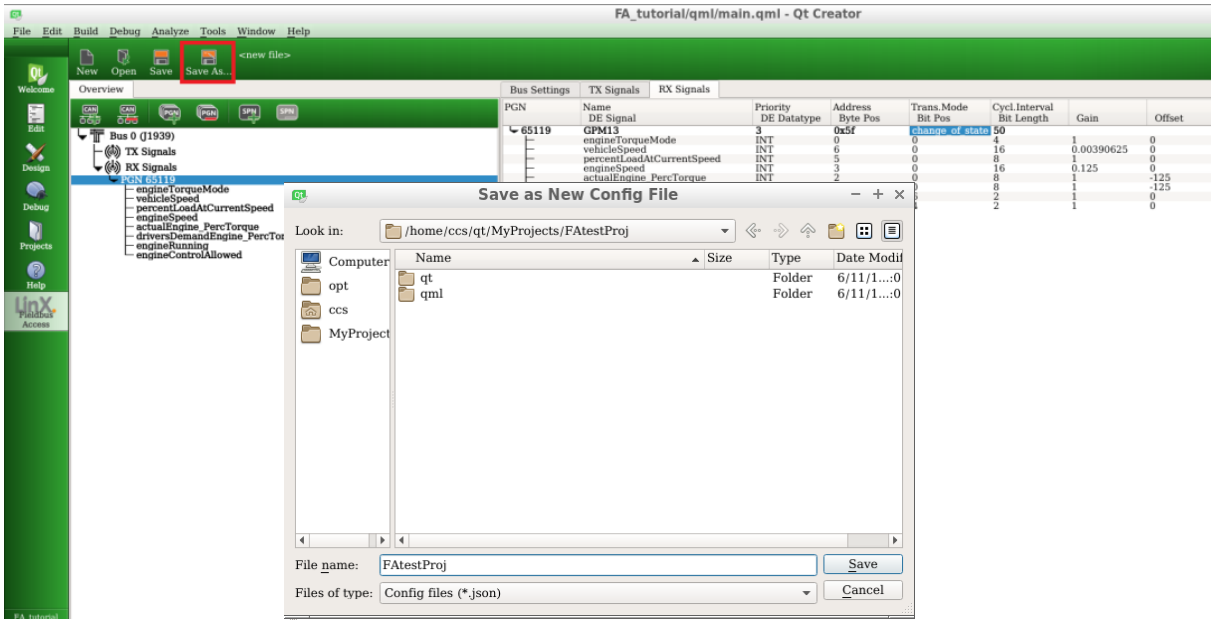


Figure 5 Example of some added signals

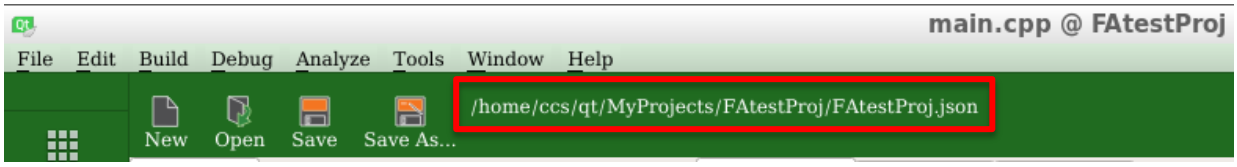
When you have finished the updating of Tx/Rx signals /PGNs, you must save the configuration.

Click” Save As” and type the name of your config file (.json) to save it in your project folder.



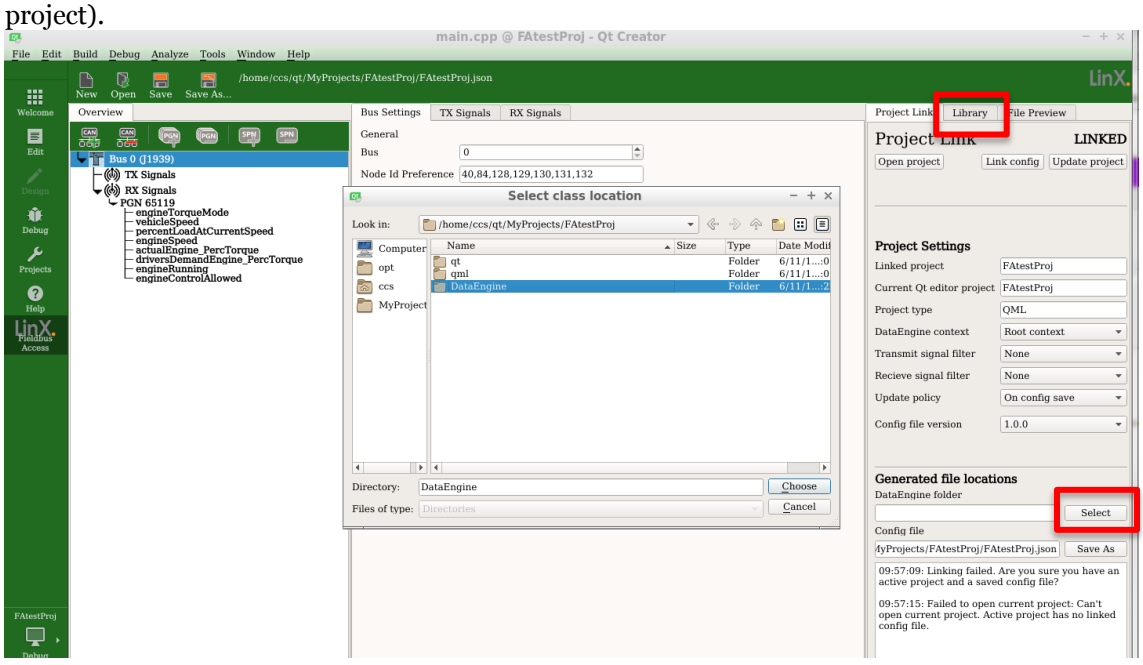
Click “Save” in the new config file window.

The name and path of your configuration file will now be visible in the upper green frame (see below).



The configuration file (.json) now exists in your project structure, but it has not been linked (connected) to the current project.

Choose tab “Project Link” and click “Select” to create a dataengine folder where you want the auto-generated files to be stored (you can use your project root or create a new dataengine folder in your project).



Verify that the correct project is stated in the box “Current Qt editor project” and click “Choose”. Then click the button “Link config”. Now, you have linked/connected a configuration to your project.

Now, finalize the configuration by adding/changing the Tx/Rx signals in your project.

To end the configuration session, click the “Save” icon or the button “Update project”. Make sure the project is updated successfully by checking the messages in the bottom right box.

By updating the configuration some new files are included and added to your project. The data engine files will be stored in the subfolder that was defined in the “Select” step above.

3.2. Update an existing LinX FA configuration

If you open an existing Qt project, containing a LinX FA configuration and press the LinX FA icon in QtCreator, the FA configuration is not open automatically.

1. Press the button “Open project” in the “Project Link” area.
2. Do the updates/changes to the configuration that you want to do.
3. Save the configuration, either with the “Save” icon in the top menu or press the button “Update project” in the “Project Link” area.

The tool will automatically recognize what project you currently have active and link the configuration to that. If you have many projects open and switches between them the signal configuration will change according to what file you have linked. If you have problems linking your project, make sure that you have one of the files in the active project selected (Qt Edit menu).

After linking dialog, switch back to Edit mode in Qt. We should now find auto created files added in the directory “DataEngine” (or in the folder we choose in the linker dialog). Our configured signals are now available within our project.

By simply changing our configuration file in the tool and save it, these classes will automatically update and make sure you have a connection to data engine. In dataengine.h we find our added SPN signals from the configuration.

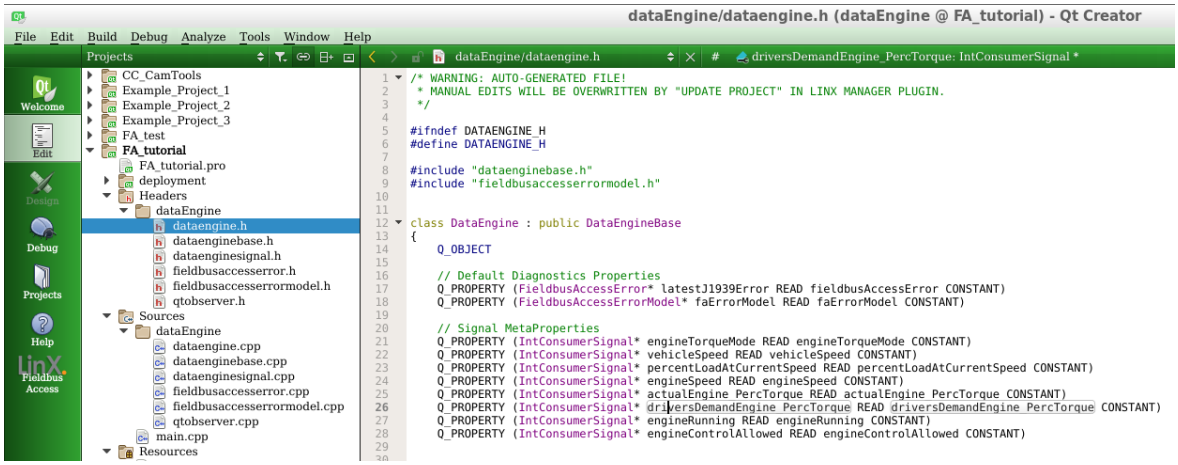


Figure 6 Example project linked with our configuration file.

3.3. Use signals in the gui application (Quick/QML)

In main.cpp add the code marked with a red box

```
1  #include <QApplication>
2  #include <QQuickView>
3  #include <QQuickItem>
4  #include <QtQml>
5  #include <QFontDatabase>
6
7  #include "dataEngine/dataengine.h"
8
9  int main(int argc, char *argv[])
10 {
11     QApplication app(argc, argv);
12
13     QFontDatabase database;
14     int id = database.addApplicationFont("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf");
15     if (id != -1){
16         QStringList families = QFontDatabase::applicationFontFamilies(id);
17         QGuiApplication::setFont(QFont(families.at(0)));
18     }
19
20     // get the target setting from the .pro-file:
21 #if defined(TARGET_ARM)
22     bool targetARM = true;
23 #else
24     // virtual machine
25     bool targetARM = false;
26 #endif
27
28     QQuickView *view = new QQuickView();
29     QObject::connect(view->engine(), &QQmlEngine::quit, &app, &QApplication::quit);
30
31     // this will set the "targetARM" property to true if the software is built for ARM
32     int displayWidth = 800; int displayHeight = 480;
33     view->rootContext()->setContextProperty("targetARM", QVariant(targetARM));
34     view->rootContext()->setContextProperty("displayWidth", QVariant(displayWidth));
35     view->rootContext()->setContextProperty("displayHeight", QVariant(displayHeight));
36
37     qmlRegisterUncreatableType<DataEngine>("Crossconrol", 1, 0, "dataEngine", "Cannot create dataEngine as a qml object");
38     DataEngine dataEngine;
39     view->rootContext()->setContextProperty("dataEngine", &dataEngine);
40
41     view->setSource(QStringLiteral("qrc:/qml/main.qml"));
42     view->show();
43
44     return app.exec();
45 }
46
```

This code will register your newly created DataEngine class for easy access in your qml code. We may now change our main.qml to display one of our receiving signals.

The code in file main.cpp will instantiate our DataEngine class for us so we may use it in our qml-file.

If we want to display an incoming signal value in the application we can use the default text filed that currently displays our project name and change it to our value.

- In the Text block in main.qml, replace (row 34):

```
text: "Tutorial"
```

Tutorial is the project name, if you have named your application otherwise that will be shown here.

- Replace with:

```
text: dataEngine.engineSpeed.value
```

DataEngine is the signal manager runtime and after you type ‘dataEngine.e’ there will be a dropdown list with available signals from your configuration for easy auto completion and finally select “value” the actual value of the signal.

```
// a text in the center of the application
Text {
    anchors.centerIn: parent
    font.pointSize: 20
    color: "white"
    text: dataEngine.e
}
                    engineControlAllowed
                    engineRunning
// a simple exit-bu  engineSpeed                er
Rectangle {         engineTorqueMode
    width: 120;  height: 120
    anchors.right: parent.right
```

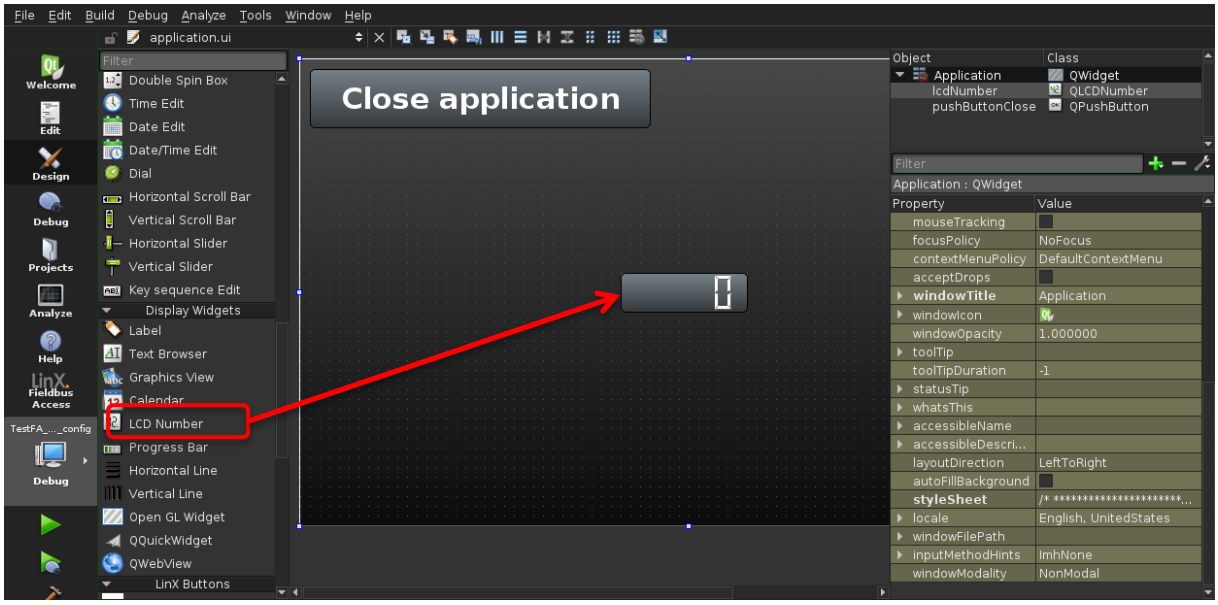
```
// a text in the center of the application
Text {
    anchors.centerIn: parent
    font.pointSize: 20
    color: "white"
    text: dataEngine.engineSpeed.v
}                                   value
```

The application will display the value of EngineSpeed signal whenever received on the CAN-bus.

3.4. Use signals in the gui application (Widget)

If you instead of QML choose a widget based application the data engine context takes pure c++ classes only. The wizard does not update main.cpp for widget applications.

To connect a fieldbus value to a widget, please follow these steps.



Add a LCD Number Widget to your ui file (application.ui)

Next step is to instantiate the DataEngine class so we have access to our signals.

In **application.cpp**, add:

```
#include "DataEngine/dataengine.h"
…
// ADD USER CODE HERE
DataEngine *de = new DataEngine();
connect(de, SIGNAL(de->engineSpeed()), ui->lcdNumber, SLOT(display(int)));
…
```

The application will display the value of EngineSpeed signal whenever received on the can-bus.

3.5. Run the example in development VM

To verify that the application runs we will setup the environment in our VM and start the application there. First we will make sure that the DataEngine and FieldbusAccess runtime are running. On the VM Desktop you find ICONS for quick access.

- Start "Firebird DB startup" (Starts the persistant signals db server). If you are prompted to enter a password you should enter "default". The DataBase server will start and run in the background.
- Start "DataEngineServer", you can start it from the desktop shortcut **or** open a new console and run following command

  ```
  # /opt/bin/dataengineserverApp --debug
  ```

This will display current DataEngine version and what ip:port you are running on. Before we start the fieldbusaccess runtime we need to make sure that we have our CAN-bus interface running in our VM. Included in the runtime-installation package there is a script that sets up a virtual CAN interface.

- `# /home/ccs/setupVCAN can0`

To verify that the interface is running you could run

- `# ifconfig`

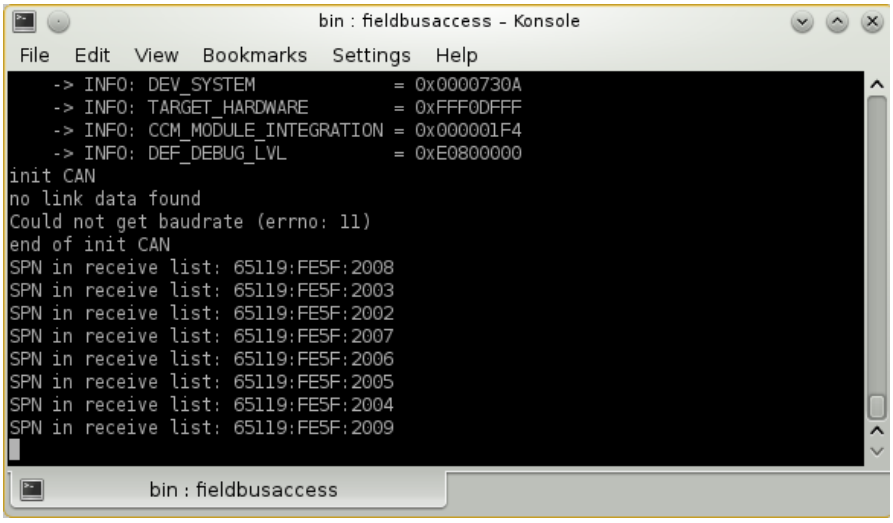
This output will look like this:

```
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP RUNNING NOARP  MTU:16  Metric:1
RX packets:5 errors:0 dropped:0 overruns:0 frame:0
TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:30 (30.0 B)  TX bytes:30 (30.0 B)
```

Next step is to start our fieldbusaccess runtime and that requires our configuration file (.json) we created earlier in the example section. Open a new console and start the runtime with following command (add path and filename to your configuration file as argument):

- # /opt/bin/fieldbusaccess /home/ccs/qt/MyProject/tutorial/FA_Signal_Configuration.json
- fieldbusaccess takes a configuration file of format json as it's first argument, this is the only required argument. Type fieldbusaccess –h for further help.

If the FA runtime is started from a terminal window, the following debug information will be listed, showing the signals from the configuration file.



Every SPN creates a unique data engine signal.

SPN in receive list: 65119:FE5F:2008

- 65119 PGN in decimal form
- FE5F PGN in hex form
- 2008 Data Engine signal index

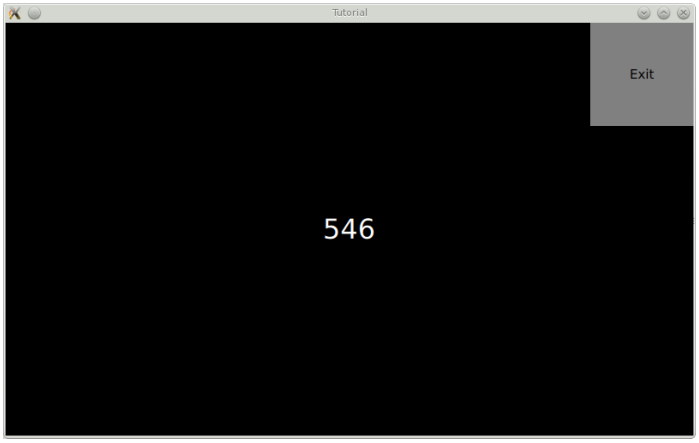
Now it's time to start our application from QtCreator. Initially we will have a black screen with a grey Exit button on the upper right corner and a signal value in the middle. To simulate sending a can-signal we can use a tool called '*cansend*' in the VM. If the command is not present, use a terminal window to install the toolkit:

- `# sudo apt-get install can-utils`

Send a can message with following command

- `# cansend can0 18FE5F08#1111111111111111`

This will produce the signal value 546 in DataEngine index 2005 and will now be shown in our application (see below).



3.6. Signalvalue explained

We are sending an 8 byte value with 0x11 {0001 0001} as value in every byte.

{0001 0001} {0001 0001} {0001 0001} { etc.

In the config file we find that signal EngineSpeed is mapped on byte 3, bit 0 with length 16.

Settings | TX Signals | RX Signals

PGN SPN	Name DE Signal	Priority DE Datatype	Address Byte Pos	Trans.Mode Bit Pos	Cycl.Interval Bit Length	Gain	Offset
▾ 65119	GPM13	3	0x5f	change_of...	50		
0	EngineTorqueMode	INT	0	0	4	1	0
0	VehicleSpeed	INT	6	0	16	0,003906...	0
0	PercentLoadAtCurrentSpeed	INT	5	0	8	1	0
0	EngineSpeed	INT	3	0	16	0,125	0
0	ActualEngine_PercTorque	INT	2	0	8	1	−125
0	DriversDemandEngine_PercTorque	INT	1	0	8	1	−125
0	EngineRunning	INT	0	6	2	1	0
0	EngineControlAllowed	INT	0	4	2	1	0

Byte index 3 Byte index 4 Byte index 5

{0001 0001} {0001 0001} {0001 0001}

0001 0001 0001 0001 = 0x1111 = 4369 dec

Scaling this value with Gain 0,125 and offset 0 we get 4369*0.125+0 = 546,125. But since we send it as a DataEngine INT value the number is truncated to 546.

3.7. Scaling of values

Many signals in J1939 library is scaled according to configuration. In the configuration tool you will set this information in the Gain and Offset fields. The sent values will be scaled with the following formula

rawValue * gain + offset

To send raw can values set gain to 1 and offset to 0.

3.8. Run the example on a CCpilot ARM device.

Make sure that your CCpilot device is prepared for running fieldbus access, Data Engine version 3.0.1 or newer and LinX-Base 2.1.0 or newer is required. Install the

The correct LinX-Base RT can be found on our support site.
http://support.maximatecc.com/downloads/LinX_Software_Suite

The FieldbusAccess runtime is autostarted on the device and you must make sure that it is started with the correct configuration file (.json) as input. Your configuration file must be present in the Linux file system on the target device. The runtime executable and the default configuration file can be found in directory: `/opt/bin`

Change the /opt/bin/FieldbusAccess_ConfigFile.json to a softlink, pointing to your deployed configuration file instead. Do the following:

- `# ssh root@<ipaddress>  (pw:suseroot)`
- `# ln -sf /opt/tutorial/bin/FA_Signal_ConfigFile.json /opt/bin/FieldbusAccess_ConfigFile.json`
- `reboot`

After reboot of your CCpilot device the fieldbusaccess runtime will restart automatically and use the correct .json file.

Manual stop/start of LinX runtimes can be done with following commands:

- /opt/etc/init.d/StartupDataEngineServer stop/start
- /opt/etc/init.d/StartupFieldbusAccess stop/start

Next we will deploy our GUI to a CCpilot device.

First, change the build configuration in QtCreator by selecting CCpilot ARM.

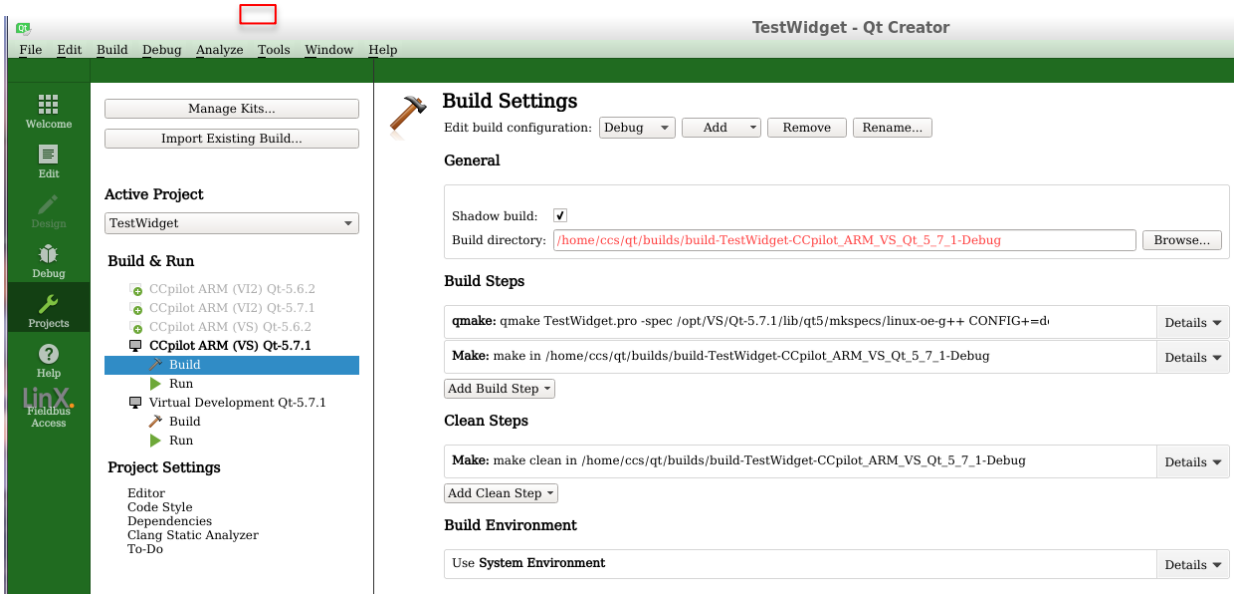
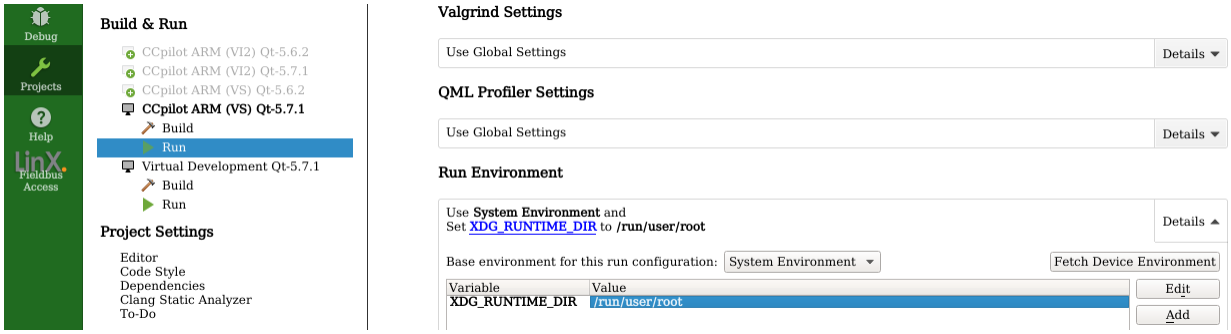


Figure 7 Project settings in QtCreator

In the “Run Settings”, add Run Environment variable:

XDG_RUNTIME_DIR /run/user/root



Set the device IP address to the target device. Select “Tools → Options… and then “Devices”.

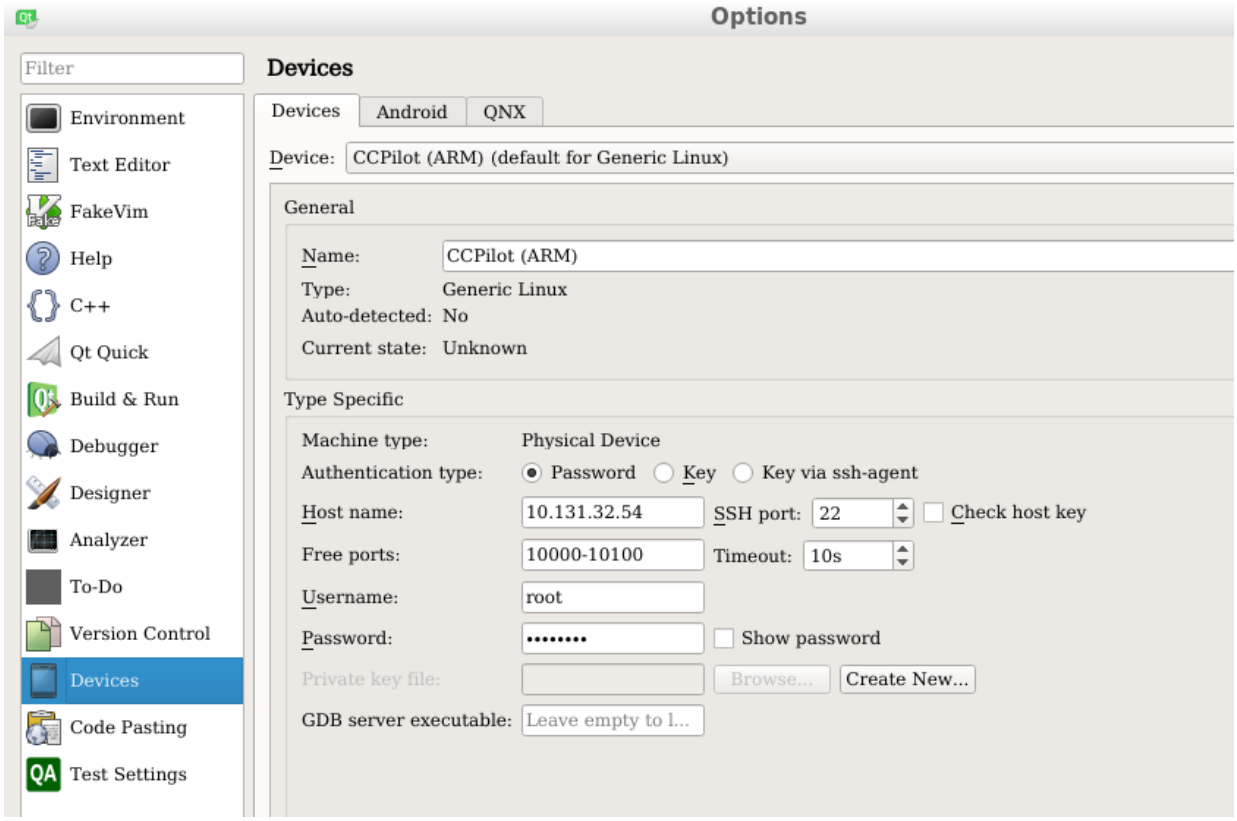
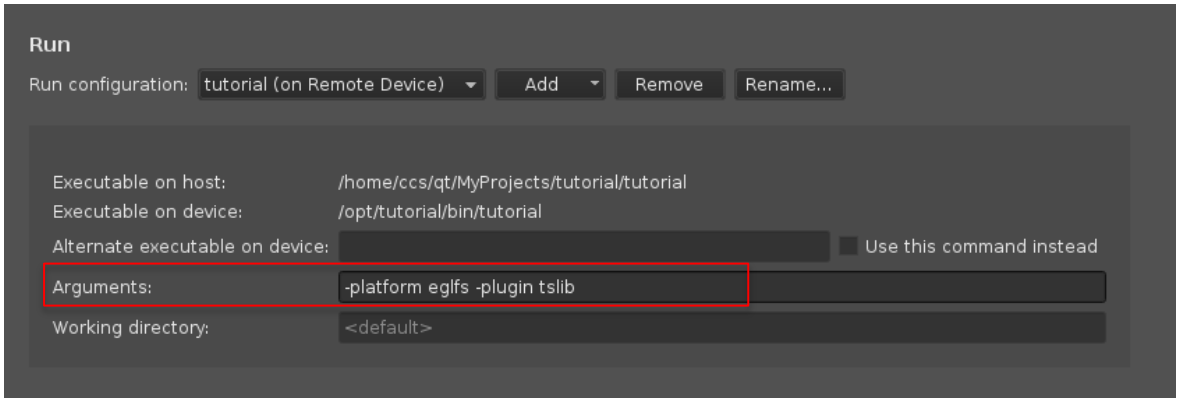


Figure 8 Add IP address of your CCpilot device here

Add application arguments to projects Run Settings



For widget based projects add:

- -platform eglfs -plugin tslib

For QML based projects add:

- …

In the "Run Settings", add Run Environment variable:

XDG_RUNTIME_DIR /run/user/root

Then rebuild the project and deploy/run on target.

- Build->Clean Project "Tutorial"
- Build->Build Project "Tutorial" (Ctrl+B)
- Build->Run (Ctrl+R)

Verify the connection by sending a value on EngineSpeed

- `cansend can0 18FE5F08#1111111111111111`

This will produce the signal value 546 in DataEngine index 2005 and will now be shown in our application.

Figure 9 Demo application with value sent from can-bus (QML)

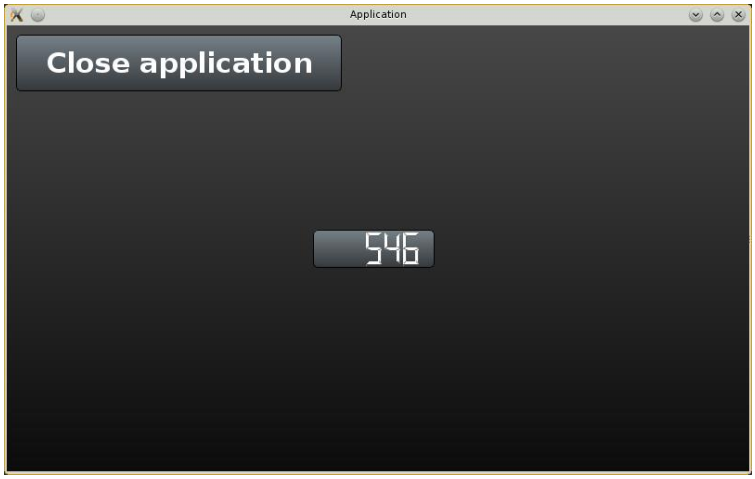


Figure 10 Demo application with value sent from can-bus (Widget)

4. Error Handling

The J1939 stack produces error messages that is transmitted via Data Engine to your application. The default data engine signal name is “fieldbusAccess_J1939Error”. If you would like to change it you may start the fieldbusaccess runtime with the argument --name “InstanceName”. Please note that the signal name should start with lower case letter (this will be automatically changed by the runtime). Starting Fieldbus Access like

./fieldbusaccess --name InstanceName FieldbusAccess_ConfigFile.json

This would create the data engine signal “instanceName_J1939Error” available for the application.

4.1. J1939 Stack Error

The errors produced from the J1939 stack is sent on following format.

** 16 bit error code:

** +-----------+-------------+---------+------+-----------+

** | Bit | 15 | 14/13 | 12-8 | 7-0 |

** +-----------+-------------+---------+------+-----------+

** | Element | Reserved | Class | ID | Number |

** +-----------+-------------+---------+------+-----------+

The error code is divided in three categories, class, id and number.

These categories are further explained in the complete list added in Appendix A

If Fieldbus Access runtime is executed from command prompt error messages are displayed in the runtime output.

Ex.

Error 20 of class 1 in unit 3 with info 64768 has occurred

Looking in the Appendix A we find that above is of

Error Class type Error

Error ID 3 (TPL_ERR_ID)

Error Number 20 TPL_ERR_RX_TIMEOUT To: Timer overrun for Rx Timeout To.

A More descriptive error text is generated in the application!

4.2. Application error handling

When a configuration file is created and linked to a project (your application) various backend code is generated to create, connect and update values within your application. In addition an error handling class is created which will show the error with a more descriptive text of the error followed with what PGN number the error message refers too (if any such information is available for current error).

4.2.1. Cyclic timeout monitor detection

When an Rx (receive) signal is created and specified as Cyclic in the configuration tool it will be automatically monitored for cyclic timeouts (No values received on the CAN bus since last received value + cycle time x 3). As soon as a cycle timeout is detected for any given signal an error message is sent to the application (through DE). Every period (cycle time x 3) a new error will be sent. When fieldbus access starts to receive values again from the CAN bus an error message is sent once with same code and PGN info except that the Error class is of type ERR_OK (3). This way it's possible to validate the status of the values received in your application.

5. FAQ

5.1. How do I log in to my device

From console in your virtual machine

- ssh root@<ipaddress> (pw: suseroot)

5.2. Application running but no value is updated

Check that dataengineserverApp, fieldbusaccess and your application is running

- ps
  (if one of above applications is not found in the process list try start them manually)
- ifconfig
  (can0 should be shown in the list of network interfaces)
- Make sure that FA has been started with the correct .json configuration file
- Verify that correct node id's and addresses are configured in the .json configuration file

5.3. Application not starting on device

On the device, execute following command,

- ldd <Application Name>

If you get "not a dynamic executable" your binary is compiled for another platform. Try to change build kit in your project settings.

5.4. How to verify that LinX-Base runtimes is installed on Device

On the device, execute following command,

- ls /opt/packages

In the list of installed components you should see "LinX-Base_VA_device_v2.1.0"

5.5. Is Data Engine running

On the device, execute following command,

- ps | grep dataengineserverApp

  2024 root 118m S /opt/bin/dataengineserverApp

5.6. Is Fieldbus Access runtime running

- ps | grep fieldbusaccess

  2409 root 38068 S /opt/bin/fieldbusaccess /opt/bin/FieldbusAccess_Config

5.7. Manual start of FieldbusAccess or DataEngine

Log in to the device and execute following command:

- /opt/etc/init.d/StartupDataEngineServer {start|stop}

- /opt/etc/init.d/StartupFieldbusAccess {start|stop}
    - FieldbusAccess will start with the default /opt/bin/FieldbusAccess_Configfile.json as its argument if none is given.

Manual start of Fieldbus Access runtime with arguments,

- /opt/bin/fieldbusaccess /path_to_application_config/config_name
- /opt/bin/fieldbusaccess -h (for additional help)

5.8. Not sure if FieldbusAccess starts with my Configuration file

On the device, execute following command:

- ps | grep fieldbusaccess

  2409 root 38068 S /opt/bin/fieldbusaccess /opt/bin/FieldbusAccess_Config

Fieldbus Access runtime is running with /opt/bin/FieldbusAcces_ConfigFile.json as it's configuration file.

- ls -l /opt/bin/FieldbusAccess_ConfigFile.json

  /opt/bin/FieldbusAccess_ConfigFile.json -> /opt/Tutorial/bin/FieldbusAccess_ConfigFile.json

The link /opt/bin/FieldbusAccess_ConfigFile.json points to your configuration
-> /opt/Tutorial/bin/FieldbusAccess_ConfigFile.json

Another way to make sure what configuration file is executed by Fieldbus Access is to run it manually.

- /opt/etc/init.d/StartupFieldbusAccess stop
- /opt/bin/fieldbusaccess /path_to_application_config/config_name &

This starts Fieldbus Access runtime and a short summary of the configuration parsed is shown in the start of the output

- Number of can-bus connections
- Number of mappings (PGN's)
- Number of SPN's
- List of PGN's connected to Data Engine

6. Appendix A

J1939 Stack errors

** **Structure of an error**.
**
** The 16 bit error code is made up of the following elements:
** +---------+----------+----------+---------+--------+
** | Bit | 15 | 14/13 | 12-8 | 7-0 |
** +---------+----------+----------+---------+--------+
** | Element | Reserved | Class | ID | Number |
** +---------+----------+----------+---------+--------+
**

6.1. Error Class

The errors are allocated to the three following classes:

+------------+---+
| Class | Description |
+------------+---+

Warning A function (in the sense of a task) could not be performed, as a pre-requisite or an external condition was not fulfilled. It may also be that the execution of the function currently is not possible but can be repeated later. In general, this type of error can be ignored in a running system or is handled by the application.

Statistical analysis or recording of the errors may be useful.

Examples of these errors are:

- CAN controller in warning level
- transmit queue full
- communication aborted by the remote station

+---------+---+

Error An error has occurred in the system that does not inhibit continuous operation but that cannot be ignored in general. These are often errors caused by overload or incorrect input data. This error can lead to subsequent errors. Depending on the application, the system must be set to an error state or shut down completely. Continued operation of the system in est/debug mode may be advisable.

Examples of these errors are:

- data overrun in the receive queue
- protocol aborts due to temporary memory problems
- timeouts of real-time tasks

+---------+---+

Fatal A fatal error has occurred in the system, which, in most cases, inhibits further software operation. Depending on the application, the system must be shut down completely or rebooted.

Examples of these errors are:

- stack overrun
- CAN controller in Bus-Off
- no more free memory available
- error in memory- or HW-monitoring
- invalid system states

+---------+---+

Ok No error. Used to acknowledge errors.

Example could be:

- send ERR_OK when cyclic reception monitoring starts to receive signals again!

+---------+---+

6.2. Error Id

```
/** Error Identifiers:
** The error identifiers of the several units, which use the error unit.
** It's possible to use the error unit also for application errors.
** Therefore the error identifier of the application has to be added.
*/

ERR_ERR_ID      0     /* error unit identifier                */
DLL_ERR_ID      1     /* data link layer identifier           */
MEM_ERR_ID      2     /* memory unit identifier               */
TPL_ERR_ID      3     /* transport protocol identifier        */
NWM_ERR_ID      4     /* network management identifier        */
APL_ERR_ID      5     /* application layer identifier         */
CYC_ERR_ID      6     /* cycle unit identifier                */
REQ_ERR_ID      7     /* request unit identifier              */
L2C_ERR_ID      8     /* layer 2 unit identifier              */
ADR_ERR_ID      9     /* addressing unit identifier           */
NMEA_ERR_ID     10    /* NMEA 2000 module                     */
CDP_ERR_ID      11    /* CAN Diagnostic Protocol              */
DIAG_ERR_ID     12    /* J1939 Diagnostic Module              */
USR_ERR_ID      13    /* user specific code (generated)       */
```

6.3. Error Number

```
/** Errors of the APL unit:
** In the following all synchronous and asynchronous errors of the APL
** unit are defined.
*/
/*---------------------------------------------------------------------------
/  Synchronous Errors
/--------------------------------------------------------------------------*/
/** APL_ERR_TIME_EXCEEDED:
** APL_Main() was not called in the required sequence
*/

#define APL_ERR_TIME_EXCEEDED   ERR_BUILD_ERROR_CODE \
                                (ERR_ERROR   , APL_ERR_ID, 1)
```

```
/** APL_ERR_TX_LONG:
** The data field of the transmit message is too long
*/

#define APL_ERR_TX_LONG       ERR_BUILD_ERROR_CODE \
                              (ERR_FATAL   , APL_ERR_ID, 2)

/** APL_ERR_NO_MEMORY:
** No memory for the data field of the transmit message allocated
*/

#define APL_ERR_NO_MEMORY     ERR_BUILD_ERROR_CODE \
                              (ERR_ERROR   , APL_ERR_ID, 3)

/** APL_ERR_REG_REQUEST:
** An error occurred during registration of a request PGN
*/

#define APL_ERR_REG_REQUEST   ERR_BUILD_ERROR_CODE \
                              (ERR_ERROR   , APL_ERR_ID, 4)

/** APL_ERR_TX_NO_ADDR:
** Device has no address claimed
*/

#define APL_ERR_TX_NO_ADDR    ERR_BUILD_ERROR_CODE \
                              (ERR_ERROR   , APL_ERR_ID, 5)

/** APL_ERR_INVALID_PARAM:
** Invalid parameter passed to an API function
*/

#define APL_ERR_INVALID_PARAM ERR_BUILD_ERROR_CODE \
                              (ERR_WARNING , APL_ERR_ID, 6)

/*---------------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------------*/
/** APL_ERR_REQ_RESPONSE:
** A request message couldn't be answered from the request handler due to
** a queue overrun.
*/

#define APL_ERR_REQ_RESPONSE  ERR_BUILD_ERROR_CODE(ERR_WARNING, APL_ERR_ID, 11)

/** Errors of the CYC unit:
** In the following all synchronous and asynchronous errors of the CYC
** unit are defined.
*/
/*---------------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------------*/
/** CYC_ERR_NOT_INITIALIZED:
** The unit is not yet initialized.
*/

#define CYC_ERR_NOT_INITIALIZED ERR_BUILD_ERROR_CODE(ERR_FATAL, CYC_ERR_ID, 1)

/** CYC_ERR_RX_REGISTRATION:
** The maximal number of receive messages is exceeded.
*/

#define CYC_ERR_RX_REGISTRATION ERR_BUILD_ERROR_CODE(ERR_FATAL, CYC_ERR_ID, 2)
```

```
/** CYC_ERR_TX_REGISTRATION:
** The maximal number of transmit messages is exceeded.
*/

#define CYC_ERR_TX_REGISTRATION ERR_BUILD_ERROR_CODE(ERR_FATAL, CYC_ERR_ID, 3)

/** CYC_ERR_TIME_EXCEEDED:
** CYC_Main() was not called in the required sequence.
*/

#define CYC_ERR_TIME_EXCEEDED   ERR_BUILD_ERROR_CODE(ERR_ERROR, CYC_ERR_ID, 4)

/*---------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------*/
/** CYC_ERR_RX_TIMEOUT:
** A registered message wasn't received in the given time.
*/

#define CYC_ERR_RX_TIMEOUT      ERR_BUILD_ERROR_CODE(ERR_ERROR, CYC_ERR_ID, 11)

/** CYC_ERR_RX_TIMEOUT_OK:
** A registered message were received in the given time.
*/

#define CYC_ERR_RX_TIMEOUT_OK   ERR_BUILD_ERROR_CODE(ERR_OK, CYC_ERR_ID, 11)

/** CYC_ERR_TX_INVALID_HDL:
** A message couldn't be sent due to an invalid address handle.
*/

#define CYC_ERR_TX_INVALID_HDL  ERR_BUILD_ERROR_CODE(ERR_ERROR, CYC_ERR_ID, 12)

/** CYC_ERR_TX_QUEUE_OVRN:
** A message couldn't be sent due to an overrun of the transmit queue.
*/

#define CYC_ERR_TX_QUEUE_OVRN   ERR_BUILD_ERROR_CODE(ERR_WARNING, CYC_ERR_ID, 13)

/** CYC_ERR_RX_INVALID_LENGTH:
** The length of the received message is too long for the receive buffer.
*/

#define CYC_ERR_RX_INVALID_LENGTH ERR_BUILD_ERROR_CODE(ERR_ERROR, CYC_ERR_ID, 14)

/** Errors of the NWM unit:
** In the following all synchronous and asynchronous errors of the NWM
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** NWM_ERR_DEVICE_REG:*
** The device is already registered in the device list.
*/

#define NWM_ERR_DEVICE_REG      ERR_BUILD_ERROR_CODE(ERR_ERROR, NWM_ERR_ID, 1)

/** NWM_ERR_INVALID_HDL:
** The device handle is invalid.
*/


#define NWM_ERR_INVALID_HDL     ERR_BUILD_ERROR_CODE(ERR_ERROR, NWM_ERR_ID, 2)
```

```
/** NWM_ERR_INVALID_ADDR:
** The device address is invalid.
*/
#define NWM_ERR_INVALID_ADDR    ERR_BUILD_ERROR_CODE(ERR_ERROR, NWM_ERR_ID, 3)
/*---------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------*/
/** NWM_ERR_NO_COMM:
** The device is not able to start the CAN communication (starting CAN failed).
*/
#define NWM_ERR_NO_COMM       ERR_BUILD_ERROR_CODE(ERR_FATAL, NWM_ERR_ID, 11)
/** NWM_ERR_NO_ADDR:
** The device is not able to claim an address and is therefore not allowed
** to take part at network communication any longer.
*/
#define NWM_ERR_NO_ADDR       ERR_BUILD_ERROR_CODE(ERR_FATAL, NWM_ERR_ID, 12)

/** NWM_ERR_LIST_FULL:
** The maximum number of nodes in the network (CNF_NWM_MAX_NODES_IN_NET)
** is exceeded.
*/
#define NWM_ERR_LIST_FULL     ERR_BUILD_ERROR_CODE(ERR_FATAL, NWM_ERR_ID, 13)


/** Errors of the TPL unit:
** In the following all synchronous and asynchronous errors of the TPL
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** TPL_ERR_TX_OVRN:
** No free Entry in the TxQueue of the TPL available.
*/
#define TPL_ERR_TX_OVRN       ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID, 1)
/** TPL_ERR_CONF_FILTER:
** Too many PGNs tried to register in the config filter.
*/
#define TPL_ERR_CONF_FILTER    ERR_BUILD_ERROR_CODE(ERR_FATAL, TPL_ERR_ID, 2)
```

```
/*--------------------------------------------------------------------------
/  Asynchronous Errors
/--------------------------------------------------------------------------*/
/** TPL_ERR_UNEXP_BAM_FRM:
** Unexpected BAM frame received.
*/
#define TPL_ERR_UNEXP_BAM_FRM   ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,11)
/** TPL_ERR_UNEXP_RTS_FRM:
** Unexpected RTS frame received
*/
#define TPL_ERR_UNEXP_RTS_FRM   ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,12)
/** TPL_ERR_UNEXP_CTS_FRM:
** Unexpected CTS frame received.
*/
#define TPL_ERR_UNEXP_CTS_FRM   ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,13)
/** TPL_ERR_UNEXP_EOM_FRM:
** Unexpected EOM frame received.
*/
#define TPL_ERR_UNEXP_EOM_FRM   ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,14)
/** TPL_ERR_UNEXP_CA_FRM:
** Unexpected CA frame received.
*/
#define TPL_ERR_UNEXP_CA_FRM    ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,15)
/** TPL_ERR_UNEXP_DT_FRM:
** Unexpected DT frame received.
*/
#define TPL_ERR_UNEXP_DT_FRM    ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,16)
/** TPL_ERR_RX_OVRN:
** No free Rx Msg Buffer in the TPL for a global message available.
*/
#define TPL_ERR_RX_OVRN        ERR_BUILD_ERROR_CODE(ERR_ERROR  , TPL_ERR_ID,17)
/** TPL_ERR_TX_TIMEOUT0:
** Timer overrun for Tx Timeout T0.
*/
#define TPL_ERR_TX_TIMEOUT0     ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,18)
/** TPL_ERR_TX_TIMEOUT2:
** Timer overrun for Tx Timeout T2.
*/
#define TPL_ERR_TX_TIMEOUT2     ERR_BUILD_ERROR_CODE(ERR_WARNING, TPL_ERR_ID,19)
/** TPL_ERR_RX_TIMEOUT0:
** Timer overrun for Rx Timeout T0.
*/
#define TPL_ERR_RX_TIMEOUT0     ERR_BUILD_ERROR_CODE(ERR_ERROR  , TPL_ERR_ID,20)
```

```
/** TPL_ERR_RX_TIMEOUT1:
** Timer overrun for Rx Timeout T1.
*/

#define TPL_ERR_RX_TIMEOUT1    ERR_BUILD_ERROR_CODE(ERR_ERROR , TPL_ERR_ID,21)

/** TPL_ERR_RX_TIMEOUT2:
** Timer overrun for Rx Timeout T2.
*/

#define TPL_ERR_RX_TIMEOUT2    ERR_BUILD_ERROR_CODE(ERR_ERROR , TPL_ERR_ID,22)

/** TPL_ERR_RX_LONG:
** The length of the received message is too long for a seg msg.
*/

#define TPL_ERR_RX_LONG      ERR_BUILD_ERROR_CODE(ERR_FATAL , TPL_ERR_ID,23)

/** TPL_ERR_SEND_MSG:
** Error by Sending: Can't send a segmented message.
*/

#define TPL_ERR_SEND_MSG      ERR_BUILD_ERROR_CODE(ERR_ERROR , TPL_ERR_ID,24)

/** TPL_ERR_SEND_CA:
** Sending a CA message failed.
*/

#define TPL_ERR_SEND_CA      ERR_BUILD_ERROR_CODE(ERR_ERROR , TPL_ERR_ID,25)

/** TPL_ERR_SEND_NACK:
** Sending a NACK message failed.
*/

#define TPL_ERR_SEND_NACK     ERR_BUILD_ERROR_CODE(ERR_ERROR , TPL_ERR_ID,26)

/** Errors of the DLL unit:
** In the following all synchronous and asynchronous errors of the DLL
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** DLL_ERR_TX_OVRN:
** overrun in the TxQueue.
*/

#define DLL_ERR_TX_OVRN     ERR_BUILD_ERROR_CODE(ERR_WARNING, DLL_ERR_ID, 1)

/** DLL_ERR_START_CAN:
** starting CAN failed.
*/

#define DLL_ERR_START_CAN    ERR_BUILD_ERROR_CODE(ERR_FATAL , DLL_ERR_ID, 2)

/** DLL_ERR_RESET_CAN:
** reset CAN failed.
*/

#define DLL_ERR_RESET_CAN    ERR_BUILD_ERROR_CODE(ERR_FATAL , DLL_ERR_ID, 3)

/** DLL_ERR_INIT_CAN:
** init CAN  failed.
*/
```

```
#define DLL_ERR_INIT_CAN     ERR_BUILD_ERROR_CODE(ERR_FATAL , DLL_ERR_ID, 4)

/*---------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------*/
/** DLL_ERR_BOFF:
** bus error interrupt occurs on CAN.
*/

#define DLL_ERR_BOFF         ERR_BUILD_ERROR_CODE ( ERR_FATAL   , DLL_ERR_ID, 11)

/** DLL_ERR_EPAS:
** error warning interrupt occurs on CAN
*/

#define DLL_ERR_EPAS         ERR_BUILD_ERROR_CODE ( ERR_WARNING , DLL_ERR_ID, 12)

/** DLL_ERR_EACT:
** CAN controller is in status error active.
*/

#define DLL_ERR_EACT         ERR_BUILD_ERROR_CODE ( ERR_WARNING , DLL_ERR_ID, 13)

/** DLL_ERR_DOI:
** data overrun interrupt occurs on CAN.
*/

#define DLL_ERR_DOI          ERR_BUILD_ERROR_CODE ( ERR_ERROR   , DLL_ERR_ID, 14)

/** DLL_ERR_RX_OVRN:
** overrun in the RxQueue.
*/

#define DLL_ERR_RX_OVRN      ERR_BUILD_ERROR_CODE ( ERR_ERROR   , DLL_ERR_ID, 15)

/** Errors of the MEM unit:
** In the following all synchronous and asynchronous errors of the MEM
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/


/*---------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------*/
/** MEM_ERR_ALLOC_MEMORY:
** Allocate memory failed.
**/

#define MEM_ERR_ALLOC_MEMORY   ERR_BUILD_ERROR_CODE ( ERR_FATAL, MEM_ERR_ID, 11)

/** MEM_ERR_FREE_MEMORY:
** Free memory failed.
*/

#define MEM_ERR_FREE_MEMORY    ERR_BUILD_ERROR_CODE ( ERR_WARNING, MEM_ERR_ID, 12)

/** Errors of the ERR unit:
** In the following all synchronous and asynchronous errors of the ERR
** unit are defined.
*/
```

```
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** ERR_ERR_INVALID_PARAM:
** Invalid parameter passed to an API function
*/
#define ERR_ERR_INVALID_PARAM   ERR_BUILD_ERROR_CODE (ERR_WARNING , ERR_ERR_ID, 1)

/** Errors of the USR unit (generated by the J1939 Designer):
** In the following all synchronous and asynchronous errors of the USR
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Asynchronous Errors
/---------------------------------------------------------------------*/
/** USR_ERR_REQ_RESPONSE:
** A request message couldn't be answered from the request handler due to
** a queue overrun.
*/
#define USR_ERR_REQ_RESPONSE   ERR_BUILD_ERROR_CODE(ERR_WARNING, USR_ERR_ID, 11)

/** Errors of the REQ unit:
** In the following all synchronous and asynchronous errors of the REQ
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** REQ_ERR_INVALID_PARAM:
** Invalid parameter passed to an API function
*/
#define REQ_ERR_INVALID_PARAM   ERR_BUILD_ERROR_CODE (ERR_WARNING , REQ_ERR_ID, 1)

/** Errors of the DIAG unit:
** In the following all synchronous and asynchronous errors of the DIAG
** unit are defined.
*/
/*---------------------------------------------------------------------
/  Asynchron Errors
/---------------------------------------------------------------------
/  Description                    * Location
/---------------------------------------------------------------------*/
/** DIAG_ERR_MEM_ABORTED_TX:
** A memory access session was aborted due to a transmit problem
** (sending a memory access message failed).
*/
#define DIAG_ERR_MEM_ABORTED_TX   ERR_BUILD_ERROR_CODE\
                                  (ERR_ERROR, DIAG_ERR_ID, 11)

/** DIAG_ERR_MEM_ABORTED_RX:
** A memory access session was aborted due to a receive problem
** (a memory access message wasn't received in the expected time).
*/
#define DIAG_ERR_MEM_ABORTED_RX   ERR_BUILD_ERROR_CODE\
                                  (ERR_ERROR, DIAG_ERR_ID, 12)
```

```
/** DIAG_ERR_MEM_ABORTED_KEY:
** A memory access session was aborted due to security reasons
** (an invalid key was received).
*/

#define DIAG_ERR_MEM_ABORTED_KEY  ERR_BUILD_ERROR_CODE\
                                  (ERR_WARNING, DIAG_ERR_ID, 13)

/** DIAG_ERR_MEM_ABORTED_DATA:
** A memory access session was aborted due to addressing problems
** (invalid data length was received).
*/

#define DIAG_ERR_MEM_ABORTED_DATA ERR_BUILD_ERROR_CODE\
                                  (ERR_WARNING, DIAG_ERR_ID, 14)

/** DIAG_ERR_MEM_BUSY:
** A memory access session was declined because the device is busy
** (another session is in process).
*/

#define DIAG_ERR_MEM_BUSY       ERR_BUILD_ERROR_CODE\
                                  (ERR_WARNING, DIAG_ERR_ID, 15)

/** DIAG_ERR_MEM_BUSY_FAILED:
** Transmit a memory access response 'busy' failed.
*/

#define DIAG_ERR_MEM_BUSY_FAILED  ERR_BUILD_ERROR_CODE\
                                  (ERR_WARNING, DIAG_ERR_ID, 16)

/** DIAG_ERR_MEM_INVALID_MSG:
** Invalid message received (the received PGN was not registered for the
** diagnostic unit).
*/

#define DIAG_ERR_MEM_INVALID_MSG  ERR_BUILD_ERROR_CODE\
                                  (ERR_ERROR, DIAG_ERR_ID, 17)

/** Errors of the CDP unit:
** In the following all synchronous and asynchronous errors of the CDP
** unit are defined.
*/
/*-------------------------------------------------------------------------
/  Asynchronous Errors
/-------------------------------------------------------------------------*/
/** CDP_ERR_TIME_EXCEEDED:
** CDP_Main() was not called in the required sequence
*/

#define CDP_ERR_TIME_EXCEEDED   ERR_BUILD_ERROR_CODE \
                                  (ERR_ERROR   , CDP_ERR_ID, 1)

/** CDP_ERR_ADDR_FORMAT:
** The addressing format of the CAN diagnostic protocol is invalid.
*/

#define CDP_ERR_ADDR_FORMAT    ERR_BUILD_ERROR_CODE\
                                 (ERR_ERROR  , CDP_ERR_ID, 2)
```

```
/** CDP_ERR_CONF_PGN:
** There is no PGN entry free to register a CDP message.
*/
#define CDP_ERR_CONF_PGN      ERR_BUILD_ERROR_CODE\
                              (ERR_ERROR  , CDP_ERR_ID, 3)

/** CDP_ERR_UNKNOWN_PGN:
** PGN is not registered.
*/
#define CDP_ERR_UNKNOWN_PGN    ERR_BUILD_ERROR_CODE\
                               (ERR_ERROR  , CDP_ERR_ID, 4)

/** CDP_ERR_STATE_READY_TX:
** PGN is not ready for transmitting a CAN diagnostic message.
*/
#define CDP_ERR_STATE_READY_TX ERR_BUILD_ERROR_CODE\
                              (ERR_WARNING, CDP_ERR_ID, 5)

/** CDP_ERR_STATE_READY_RX:
** PGN is not ready for receiving a CAN diagnostic message.
*/
#define CDP_ERR_STATE_READY_RX ERR_BUILD_ERROR_CODE\
                              (ERR_ERROR  , CDP_ERR_ID, 6)

/** CDP_ERR_NO_MEMORY:
** No memory for the data field allocated.
*/
#define CDP_ERR_NO_MEMORY     ERR_BUILD_ERROR_CODE \
                              (ERR_ERROR  , CDP_ERR_ID, 7)

/*---------------------------------------------------------------------
/  Synchronous Errors
/---------------------------------------------------------------------*/
/** CDP_ERR_ALLOC_MEMORY:
** Allocate memory failed.
*/
#define CDP_ERR_ALLOC_MEMORY   ERR_BUILD_ERROR_CODE\
                              (ERR_ERROR  , CDP_ERR_ID, 11)


/** CDP_ERR_FREE_MEMORY:
** Free memory failed.
*/
#define CDP_ERR_FREE_MEMORY    ERR_BUILD_ERROR_CODE\
                              (ERR_ERROR, CDP_ERR_ID, 12)


/** CDP_ERR_RX_LENGTH:
** The length of the received CAN diagnostic protocol message is too large
** for the available receive buffer.
*/
#define CDP_ERR_RX_LENGTH     ERR_BUILD_ERROR_CODE\
                              (ERR_ERROR  , CDP_ERR_ID, 13)
```

```
/** CDP_ERR_RX_TIMEOUT_FC:
** The CAN diagnostic protocol flow control message wasn't received in the
** given time.
*/
#define CDP_ERR_RX_TIMEOUT_FC  ERR_BUILD_ERROR_CODE\
                               (ERR_ERROR  , CDP_ERR_ID, 14)


/** CDP_ERR_RX_TIMEOUT_CF:
** The CAN diagnostic protocol consecutive frame message wasn't received in
** the given time.
*/
#define CDP_ERR_RX_TIMEOUT_CF ERR_BUILD_ERROR_CODE\
                               (ERR_ERROR  , CDP_ERR_ID, 15)


/** CDP_ERR_TX_TIMEOUT:
** The CAN diagnostic protocol message wasn't sent in the given time.
*/
#define CDP_ERR_TX_TIMEOUT    ERR_BUILD_ERROR_CODE\
                               (ERR_ERROR  , CDP_ERR_ID, 16)


/** CDP_ERR_PCI_INVALID:
** The CAN diagnostic protocol control information was invalid.
*/
#define CDP_ERR_PCI_INVALID   ERR_BUILD_ERROR_CODE\
                               (ERR_ERROR  , CDP_ERR_ID, 17)
```