CANopen Scripting Interpreter - API Reference

```
util.print("Test of simple device");
nmt.preopNetwork();
nmt.startNode(32);
i = 0;
util.print("We are in " + util.pwd());

// set node id for SDO access
sdo.setNodeId(32);

// loop over objects 0x4000 to 0x04010
for (object = 0x4000; object < 0x4005; object ++) {
    // write value to object 0x4000..
    result = sdo.write(object, 0x0, UNSIGNED32, i);
    if (result == "SDO_OK") {
        util.print(" Write OK");
    } else {
        util.print(" Write NOT OK");
    }
    // read from 0x4100.. and expect same value
    result = sdo.read(object+0x100, 0x0, 0x07);
    if (result == i) {
        util.print("Read OK");
    } else {
        util.print(" Read NOT OK");
    }
    i++;
}
```

Version History

Version	Changes	Date	Editor
V1.0.3	Commands for CAN access and file access added, util methods for access to QTableWidget cells added	2013/01/10	ged
V1.1.2	Command to configure and start SYNC added	2013/03/16	ged
V2.0	Some minor commands added	2014/02/04	ged
V2.6.2	Additional commands added	2017/04/03	ged
V 2.8.0	Additional commands added	2018/03/15	ri
V 2.16.1	Added link to reference	2024/02/01	ri

Copyright

Fritz-Haber-Str. 9
D-06217 Merseburg
Germany
Tel. +49 3461/79416-0
Fax. +49 3461/79416-10
service@emotas.de
http://www.emotas.de

Table of Contents

1 General Hints

The Scripting language of the CANopen DeviceExplorer is QtScript with additional CAN/CANopen commands. Qt Script is based on the language ECMAScript, as defined in standard ECMA-262.

Microsoft's JScript, and Netscape's JavaScript are also based on the ECMAScript standard. If you are not familiar with the ECMAScript language, there are several tutorials and books available that cover that topic.

A short reference can be found at

https://doc.qt.io/qt-5/ecmascript.html

A detailed reference can be found at

https://tc39.es/ecma262/

2 UTIL commands

2.1 util.print

Print a text string into the script logging window-

Parameters:

string - string to print into script logging windows

Return value:

nothing

2.2 util.msleep

Sleep for given number of milliseconds.

Parameters:

sleeptime - time in milliseconds to sleep

Returns value:

nothing

2.3 util.clear

Clear context of script logging window.

Parameters:

none

Returns value:

nothing

2.4 util.pwd

Return current working directory path.

Parameters:

none

Returns value:

String containing current working directory

2.5 util.load

Load a JavaScript file

Parameters:

filename - path to the file with JavaScript code

Example:

```
util.load("testdata.js");
```

2.6 util.loadUIFile

Load a UI file created by QtDesigner to use this GUI for scripting. See "Creating new UI windows" for details.

Parameters:

fileName - path to UI file

uiVariableName - name of UI variable to address UI in scripts

Returns:

true - UI loaded

false - UI not found

2.7 util.hideMainWindow

Hide the main window of CANopen DeviceExplorer. This is useful if the scripted application shall be the only visible part of the UI.

Parameters:

none

Return value:

nothing

2.8 util.showMainWindow

Show the main window of CANopen DeviceExplorer. This is useful in combination with util.hideMainWindow.

Parameters:

none

Return value:

nothing

2.9 util.getNodeId

Return the CANopen node-Id as configures in the main window.

Parameters:

none

Return value:

CANopen node ID from 1 to 127, or 0 if not set

2.10 util.after

Execute a script command after a given time. (One-shot timer)

Parameters:

milliseconds	-	time to wait for command execution in milliseconds
script	-	valid QtScript to be executed

Return value:

int timerId	-	unique ID of the timer e.g. to stop it later

2.11 util.every

Start a cyclic timer for a cyclic script execution after a given period of time.

Parameters:

milliseconds	-	time period between to executions of the script
script	-	valid QtScript to be executed

Return value:

int timerId	-	unique ID of the timer e.g. to stop it later

2.12 util.deleteTimer

Delete a given cyclic timer.

Parameter:

timerId	-	timerID of the timer to be deleted

Return value:

nothing

2.13 util.deleteAllTimers

Delete all cyclic timers.

Parameters:

none

Return value:

nothing

2.14 util.getTableItemValue

Returns the value of a cell in a QTableWidget.

Parameters:

tableName	-	name of the QTableWidget
row	-	number of the row
column	-	number of the column

Return value:

String containing the value of the specified cell.

2.15 util.isCheckBoxChecked

Checks if a checkbox in a user UI is checked.

Parameters:

checkBoxName	-	name of checkbox in .ui file

Return value:

true	checkbox is checked
false	checkbox is not checked

2.16 util.setCheckBoxText

Modify the label of a checkbox (from .ui file).

Parameters:

checkBoxName	-	name of checkbox in .ui file
text	-	new label of checkbox

2.17 util.configureSync

Configure the properties of the SYNC producer

Parameters:

id	-	COB-ID of SYNC producer

interval - SYNC interval in µs

syncCounter - max. SYNC counter value (0 == disabled)

Return value:

none

2.18 util.startSync

Return value:

true SYNC started

false SYNC not started

2.19 util.stopSync

Return value:

true SYNC stopped

false SYNC not stopped

2.20 util.exitProgram

Exit the program (CDE).

2.21 util.fileExists

Check if a given file (path) exists.

Parameters:

filePath - path to a file

Return value:

true file exists

false file does not exist

2.22 util.canReplay

Replays a CAN logging (.txt/.ecm). It tries to keep the timing the same as good as possible and additionally some CAN-IDs might be ignored.

Parameters:

skipCanIds - list of CAN-IDs to be ignored (e.g. 1,4,0x300-0x400)

fileName - path to CAN logging (.txt/.ecm)

return value:

replayId returns a unique ID to stop it later

2.23 util.stopReplay

Stops a running CAN replay.

Parameters:

replayId - Id of running CAN replay

2.24 util.stopTimersSleepReplay

Stop a running or scheduled actions suchs as timers, sleeps or running CAN replays.

2.25 util.cd

Change working directory

Parameters:

directory - path desired directory

Example:

```
util.cd("C:\\temp\\");
util.cd("..");
```

return value:

True on success otherwise false

2.26 util.getDir

Returns the current working directory.

return value:

The current working directory.

2.27 util.callOnClose(QString functionCallOnClose)

CDE will call this function after the loaded ui is closed. The ui file have to be loaded before calling this function.

return value:

True on success otherwise false (if ui was not loaded before)

2.28 util.setWidgetAttribute(QString "widgetName", Qt::WidgetAttribute, bool)

Set the attribute of a widget to the wanted value.

The ui file have to be loaded and the widget with the name must exist

return value:

True on success otherwise false.

2.29 util.u32ToFloat(quint32 u32)

Converts a u32 value to a float value

return value:

The converted float value.

2.30 util.floatToU32(float f)

Converts a float value to a u32 value

return value:

The converted u32 value.

2.31 util.getTextEditText(QString "textEditName")

To get the text of a textedit this function should be called with the name of the textedit as parameter

return value:

The text of the textedit

2.32 util.getTextEditHtml(QString "textEditName")

To get the html text of a textedit this function should be called with the name of the textedit as parameter

return value:

The html text of the textedit

2.33 util.sendTime(days, ms)

Send the days and ms as CAN message. If days and ms are 0, the current time will be send.

2.34 util.runExternal(path_to_executable, arguments)

Run external program or batch file (blocking)

exitCode = util.runExternal("/path/to/myscript.sh", "-a -x logging.txt");

return value:

The return code of the external program

2.35 util.runExternal("path_to_executable", "", true);

Start external program or batch file (non-blocking)

2.36 util.playSound();

Play a general sound

2.37 util.playSound("filepath_to_.wav_file");

Play a specific wav file

2.38 util.setTableItemValue("tableName", row, column, stringValue);

Sets the stringvalue to the row and column in in the table with the name "tableName".

2.39 userMessage(QString caption, QString text, int type)

Shows a messagebox to the user.

The type defines how it is shown.

0 – is a simple info box

1 – is a "Yes" "No" box

2 – is a "Ok" "Cancel" box

3 CAN commands

3.1 can.sendBaseFrame

Send a base(standard) CAN message with the specified ID, length and data.

Parameters:

id	-	CAN-ID of message
dlc	-	length of CAN message (0 – 8 bytes)
d0 – d7		data bytes of the CAN message

Return value:

true	CAN message sent
false	CAN message not sent

3.2 can.sendBaseRTRFrame

Send a base(standard) RTR CAN message with the specified ID and length.

Parameters:

id	-	CAN-ID of message
dlc	-	length of CAN message (0 – 8 bytes)

Return value:

true	CAN message sent
false	CAN message not sent

3.3 can.sendExtendedFrame

Send an extended CAN message with the specified ID, length and data.

Parameters:

id	-	CAN-ID of message
dlc	-	length of CAN message (0 – 8 bytes)
d0 – d7		data bytes of the CAN message

Return value:

true	CAN message sent
false	CAN message not sent

3.4 can.sendExtendedRTRFrame

Send an extended RTR CAN message with the specified ID and length.

Parameters:

id - CAN-ID of message

dlc - length of CAN message (0 – 8 bytes)

Return value:

true CAN message sent

false CAN message not sent

3.5 can.registerCanEvent

Register a callback function to be called when a CAN message with given CAN id is received.

Parameters:

id - CAN-ID of message to receive

func - function name of call back function to be called

Return value:

true CAN event has been registered

false CAN event has not been registered

Example:

```
function foo (id, rtrFlag, dlc, d0, d1, d2, d3, d4, d5, d6, d7) {
    util.print( id );
    util.print( rtrFlag );
}

can.registerCanEvent( 0x200, "foo" );
```

3.6 can.registerCanEventWithTimeStamp

Register a callback function to be called when a CAN message with given CAN id is received. This callback function will include the time stamp of the received CAN message.

Parameters:

id - CAN-ID of message to receive

func - function name of call back function to be called

Return value:

true CAN event has been registered

false CAN event has not been registered

Example:

```
function foo (id, seconds, micro, rtrFlag, dlc, d0, d1, d2, d3, d4,
d5, d6, d7) {
    util.print ( seconds)
    util.print( id );
    util.print( rtrFlag );
}

can.registerCanEvent( 0x200, "foo" );
```

3.7 can.unregisterCanEvent

Parameters:

id - CAN-ID of message to receive

Return value:

true CAN event has been unregistered

false CAN event has not been unregistered

3.8 can.unregisterAllCanEvents

Remove all registered CAN events(callbacks).

Parameter:

none

Return value:

true all CAN events have been removed

3.9 can.wait(quint16 canId, quint16 timeout)

Wait for a given CAN id.

Paramter:

canId - CAN-id for registration

timeout - timeout in milliseconds to wait for (0 .. wait forever);

return value:

return true on success otherwise false

4 CANopen SDO commands

4.1 sdo.setNodeId

Define the remote CANopen NodeId for following sdo.read and sdo.write commands.

Parameter:

nodeId remote node id for SDO communication

Return value:

nothing

4.2 sdo.getNodeId

Returns previously set Node ID.

Return value:

node-id

4.3 sdo.read

Read a value of a CANopen device by SDO. This function is blocking and returns after the response from the remote device has been received or after a time-out.

Parameter:

index - index of object to be read

sub - sub index ob object to be read

datatype CANopen index of datatype of the object (e.g. 0x07 for UNSIGNED32)

Return value:

Received value or SDO error code starting with "SDO_ERROR: "

4.4 sdo.read (via routing)

Read a value of a CANopen device by SDO via a CANopen gateway. This function is blocking and returns after the response from the remote device has been received or after a time-out.

Parameter:

networkID - target network ID

nodeID - target node ID

index - index of object to be read

sub - sub index ob object to be read

datatype CANopen index of datatype of the object (e.g. 0x07 for UNSIGNED32)

Return value:

Received value or SDO error code starting with "SDO_ERROR: "

4.5 sdo.write

Write a value of a CANopen device by SDO. This function is blocking and returns after the response

from the remote device has been received or after a time-out.

Parameter:

index	-	index of object to be read
sub	-	sub index ob object to be read
datatype		CANopen index of datatype of the object (e.g. 0x07 for UNSIGNED32)
value		new value to be transmitted

Return value:

SDO_OK or SDO error code starting with "SDO_ERROR:"

4.6 sdo.write (via routing)

Write a value of a CANopen device by SDO via a CANopen gateway. This function is blocking and returns after the response from the remote device has been received or after a time-out.

Parameter:

networkID	-	target network ID
nodeID	-	target node ID
index	-	index of object to be read
sub	-	sub index ob object to be read
datatype		CANopen index of datatype of the object (e.g. 0x07 for UNSIGNED32)
value		new value to be transmitted

Return value:

SDO_OK or SDO error code starting with "SDO_ERROR:"

4.7 sdo.writeDomainFile

Write a file to a specific object

Parameter:

index	-	index of object to be read
sub	-	sub index ob object to be read
filePath	-	path to a file

4.8 sdo.writeDomainFile (via routing)

Write a file to a specific object

Parameter:

networkID - target network ID

nodeID - target node ID

index - index of object to be read

sub - sub index ob object to be read

filePath - path to a file

4.9 sdo.setLocalRouter (for routing)

Sets the node-ID of the CANopen-CANopen gateway in local network for SDO routing.

Parameter:

routerId - node-id of local router

4.10 sdo.setTimeout(timeout in ms)

Sets the timeout of SDO access in ms.

4.11 getByteLengthOfLastResult

Get the byte length of the last sdo result

4.12 getBytesOfLastResult(int i)

Get value byte i of the last sdo result

5 CANopen NMT commands

5.1 nmt.startNetwork

Send NMT command 'Start' to complete network.

Parameters:

none

Return value:

nothing

5.2 nmt.preopNetwork

Send NMT command 'Enter Pre-operational' to complete network.

Parameters:

none

Return value:

nothing

5.3 nmt.stopNetwork

Send NMT command 'Stop' to complete network.

Parameters:

none

Return value:

nothing

5.4 nmt.resetCommNetwork

Send NMT command 'Reset Communication' to complete network.

Parameters:

none

Return value:

nothing

5.5 nmt.resetApplNetwork

Send NMT command 'Reset node' to complete network.

Parameters:

none

Return value:

nothing

5.6 nmt.startNode

Send NMT command 'Start' to specified node.

Parameters:

node - addressed CANopen device

Return value:

nothing

5.7 nmt.preopNode

Send NMT command 'Enter Pre-operational' to specified node.

Parameters:

node - addressed CANopen device

Return value:

nothing

5.8 nmt.stopNode

Send NMT command 'Stop' to specified node.

Parameters:

node - addressed CANopen device

Return value:

nothing

5.9 nmt.resetCommNode

Send NMT command 'Reset communication' to specified node.

Parameters:

node - addressed CANopen device

Return value:

nothing

5.10 nmt.resetApplNode

Send NMT command 'Reset Node' to specified node.

Parameters:

node - addressed CANopen device

Return value:

nothing

6 CANopen LSS commands

6.1 lss.switchModeGlobal

Send LSS switch mode global command. It will be received by all LSS-capable devices and thus only one of these devices should be connected.

Parameters:

configuration	-	true	enable configuration mode
		false	disable configuration mode

6.2 lss.switchModeSelective

Send LSS switch mode selective command to specified LSS slave to switch it into configuration mode.

Parameters:

vid	-	Vendor ID (0x1018:1)
pc	-	product code (0x1018:2)
rn	-	revision number (0x1018:3)
sn	-	serial number (0x1018:4)

Return values:

true	success
false	no response

6.3 lss.configureNodeId

Assign a new node ID to LSS node in configuration mode. Configuration mode must be left afterwards.

Parameters:

newNodeId	- new node ID to be set

6.4 lss.doFastScanAndSetNodeId

Perform LSS fastscan with LSS address 0x00000000000000000000000000 (find all) and it finds the first unconfigured device and assigns a node-ID to it. Configuration mode will be left afterwards. The process can be repeated, but you will not know in advance, which device gets which node ID if there is more than 1 device.

Parameters:

fastScanTimeout_ms	-	timeout in milliseconds
newNodeID	-	new node ID

7 Integrated UI components

The scripting interpreter windows contains 10 freely configurable buttons and 2 checkboxes. These UI elements can be used in scripts without having to define an own UI.

7.1 button.setName

Configure the name of a button.

Parameters:

number	-	number of the button (1-10)
text	-	new text on the button

Return value:

nothing

7.2 button.setCommand

Configure the script command of a button

Parameters:

number	-	number of the button (1-10)
script	-	script to be executed when button is clicked

Return value:

nothing

7.3 button.setEnabled

Enables or disables a button.

Parameters:

number	-	number of the button (1-10)
bool	-	true/false if button shall be enabled or disabled

Return value:

nothing

7.4 option.setName

Configure the name of a option checkbox

Parameters:

number	-	number of the checkbox (1-2)
text	-	new text on the button

Return value:

nothing

7.5 option.setEnabled

Enables or disables an option checkbox

Parameters:

number	-	number of the checkbox (1-2)
bool	-	true/false if checkbox shall be enabled or disabled

Return value:

nothing

7.6 option.isChecked

Return if the checkbox is checked or not.

Parameters:

number	-	number of checkbox(1-2)

Return value:

true	-	checkbox is checked
false	-	checkbox is not checked

7.7 label.setText

Configure the text of a label

Parameters:

number	-	number of the label (1-3)
text	-	new text on the label

Return value:

nothing

8 File selection dialogs

8.1 filedialog.getSaveFileName

Open a dialog to select a file name to save something.

Parameters:

caption	-	title of dialog
filepath	-	proposal for a file name and path
filter	-	filter definition for the selection of file types

Return value:

path to selected file

Example:

```
filepath = filedialog.getSaveFileName("select save file",
            "/home/urke/vorschlag.txt","Texts (*.txt)");
```

8.2 filedialog.getOpenFileName

Open a dialog to select a existing file

Parameters:

caption	-	title of dialog
path	-	proposal for a path
filter	-	filter definition for the selection of file types

Return value:

path to selected file

Example:

```
filepath = filedialog.getSaveFileName("select open file",
            "/home/urke/path","Texts (*.txt)");
```

9 Access to CAN

9.1 canconnection.configureDialog

Open the CAN configuration dialog to select the CAN interface and bitrate. The configured settings will take effect at the next CAN connection (by canconnection.connect() or click in the tool).

Parameters:

none

Return values:

none

9.2 canconnection.connect

Establish a connection to the configured CAN interface.

Parameters:

none

Return values:

true	CAN connection is open
false	CAN connection failed

9.3 canconnection.disconnect

Close a connection to the CAN interface.

Parameters:

none

Return values:

true CAN connection closed

false CAN connection not closed

9.4 canconnection.reset

Reset an open CAN connection is case of an error of the CAN interface.

Parameters:

none

Return values:

true CAN connection is open

false CAN connection failed

N.B. After a reset of a CAN connection some functions have to be reconfigured. These functions are:

- sdo.setNodeId()
- all callback functions registered using can.registerCanEvent.

9.5 canconnection.isConnected

Check is a CAN connection is opened.

Parameters:

none

Return values:

true CAN connection is open

false CAN connection failed

10 Access to text files

The class TextFile provides access to text files to read or write its content. Multiple instances(objects) can be created by a script. The following example shows the usage:

```
file = new TextFile("/home/urke/logfile.txt");
file.open(2 | 4);          // take care of open() modes
file.appendString("CAN FD will extend the life span of CANopen\n");
file.flush();
file.close();
```

10.1 TextFile(filepath)

Constructor of new text file open.

Parameters:

filepath path to text file (may exist or not)

Return value:

instance of TextFile

10.2 TextFile.read()

Read complete content of the text file and return it as a string.

Parameters:

none

Return value:

content of text file

10.3 TextFile.open(openMode)

Open the text file for reading or writing.

Parameters:

openMode - mode to open the file:

0x01 read only

0x02 write only

0x04 append

0x08 truncate

0x10 text file(end of line rules), otherwise binary

mode flags can be combined, like (2 | 4)

for more see http://qt-project.org/doc/qt-5/QIODevice.html#OpenModeFlag-enum

Return value:

true - file opened

false - file could not be opened

10.4 TextFile.appendString(string)

Append a string to text file.

Parameters:

string - string to append

Return value:

nothing

10.5 TextFile.flush()

Write content of text file to disk.

Parameters:

none

Return value:

true - successful

false - an error has occured

10.6 TextFile.close()

Close the file.

Parameters:

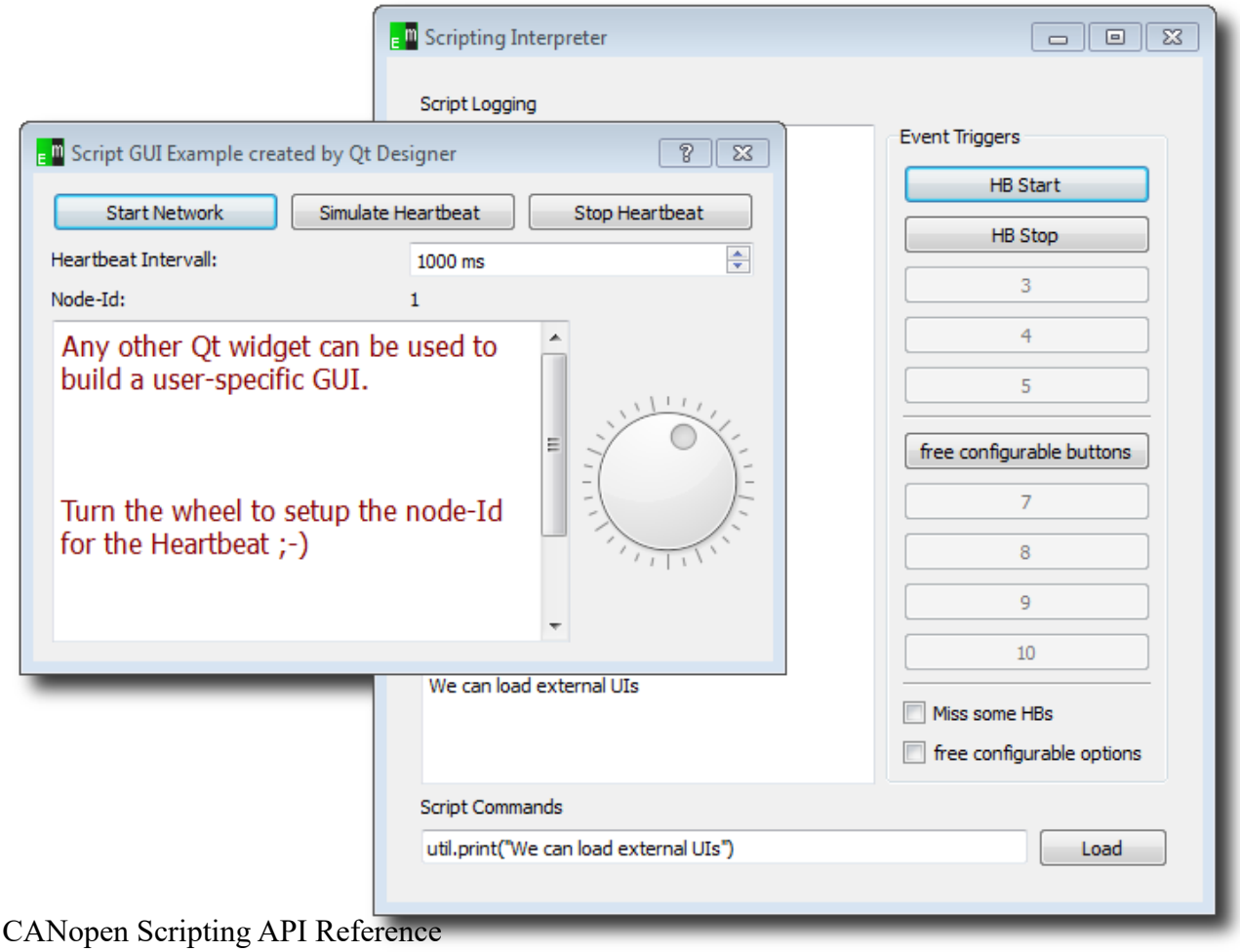
none

Return value:

none

11 Creating new UI windows

It is possible to create new UI windows using the Qt Designer and to use these UI windows in own



scripts. A UI file can be loaded using the command util.loadUIFile(scriptPath, variableName). Using the specified variableName the dialog can be accessed in scripts as shown in the following example:

```
// load UI file
if (util.loadUIFile("scriptexamples/example2.ui", "myDialog") == true) {
    // find UI button
    var b1 = myDialog.findChild("btnStart");
    var b2 = myDialog.findChild("btnStartHb");
    var b3 = myDialog.findChild("btnStopHb");
    var dial = myDialog.findChild("dial");
    var lbl = myDialog.findChild("lblNode");

    // connect buttons and functions
    b1.clicked.connect(nmt.startNetwork);
    b2.clicked.connect(startHB);
    b3.clicked.connect(stopHB);
    dial.valueChanged.connect(lbl.setText);

} else {
    util.print("Error: Cannot load UI file");
}
```

All Qt widgets (version 4.8) can be used in the UI and modified from the QtScript. But there is a limitation that only the slots and signals of a Qt widget can be accessed from the script but only 'normal' public methods.

E.g. QLabel::setText() can be called from a script as the method is defined as a public slot. In contrast to that QLabel::setTextFormat() **cannot** be used from scripts. To overcome this limitation some util-methods have been implemented to provide access to often used widgets. See section 2 (util commands).