iqInterface User Manual Documentation

Release 1.1.0.9

Andrey Zagrebin

April 04, 2016

Contents

Release 1.1.0.9

Date April 04, 2016

CHAPTER 1

Introduction

iqInterface is a device to support development and testing of IO-Link masters and devices. It can be configured for two operation modes:

- IO-Link master mode
- simulation of IO-Link device with a settable configuration (generic device feature).

For more information about IO-Link® technology see its specification.

iqInterface can be *connected* to PC with Windows OS over comport, USB or Ethernet. You can manage it using *iqTool GUI* or *driver DLL* written in C.

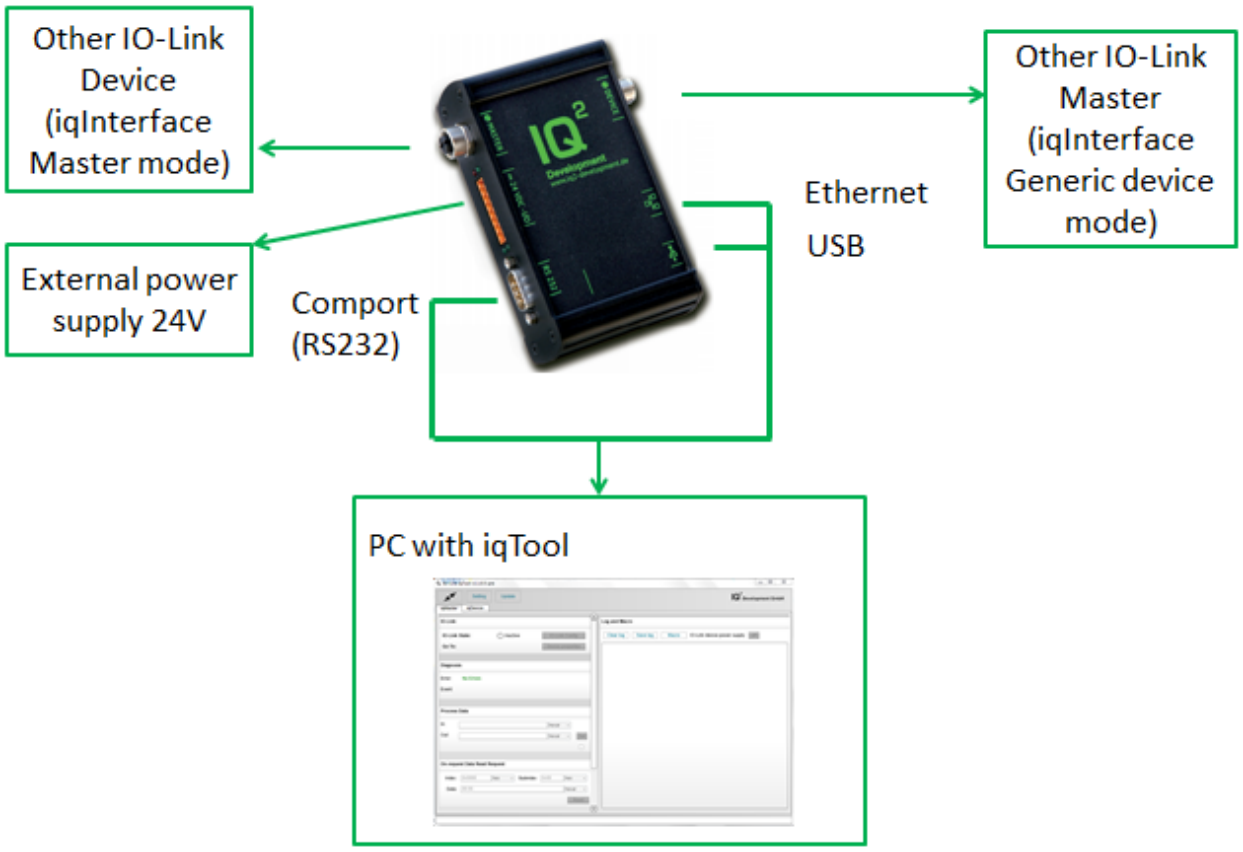
1.1 Get Started

- Connect iqInterface to PC with Windows OS over comport, USB or Ethernet (see *Physical connection*)
- Download last version of user bundle with related software and documentation or use USB flash drive content delivered with iqInterface
- Install iqTool GUI to manage iqInteface manually (see *iqTool GUI*)
- Use C driver DLL source code for c/c++ projects or last built iqcomm.dll to write automated procedures (see *C driver DLL*)
- Check for last software and firmware updates of iqInterface on its download page (see also *Update and upgrade*)
- See version changelogs for *iqInterface firmware*, *iqTool GUI* and *C driver DLL*.

CHAPTER 2

Physical connection

2.1 Connection overview



2.2 PC connection

There are three alternatives to connect iqInterface to PC:

- Comport (RS232) connection
- USB connection (virtual comport)
- Ethernet cable (tcp/ip).

USB connection is actually a virtual comport connection. To use it you should install a USB driver for Windows OS, which can be found on the USB flash drive delivered with iqInterface under "usb_driver\Windows XP_7_8". You can also visit the driver homepage for updates.

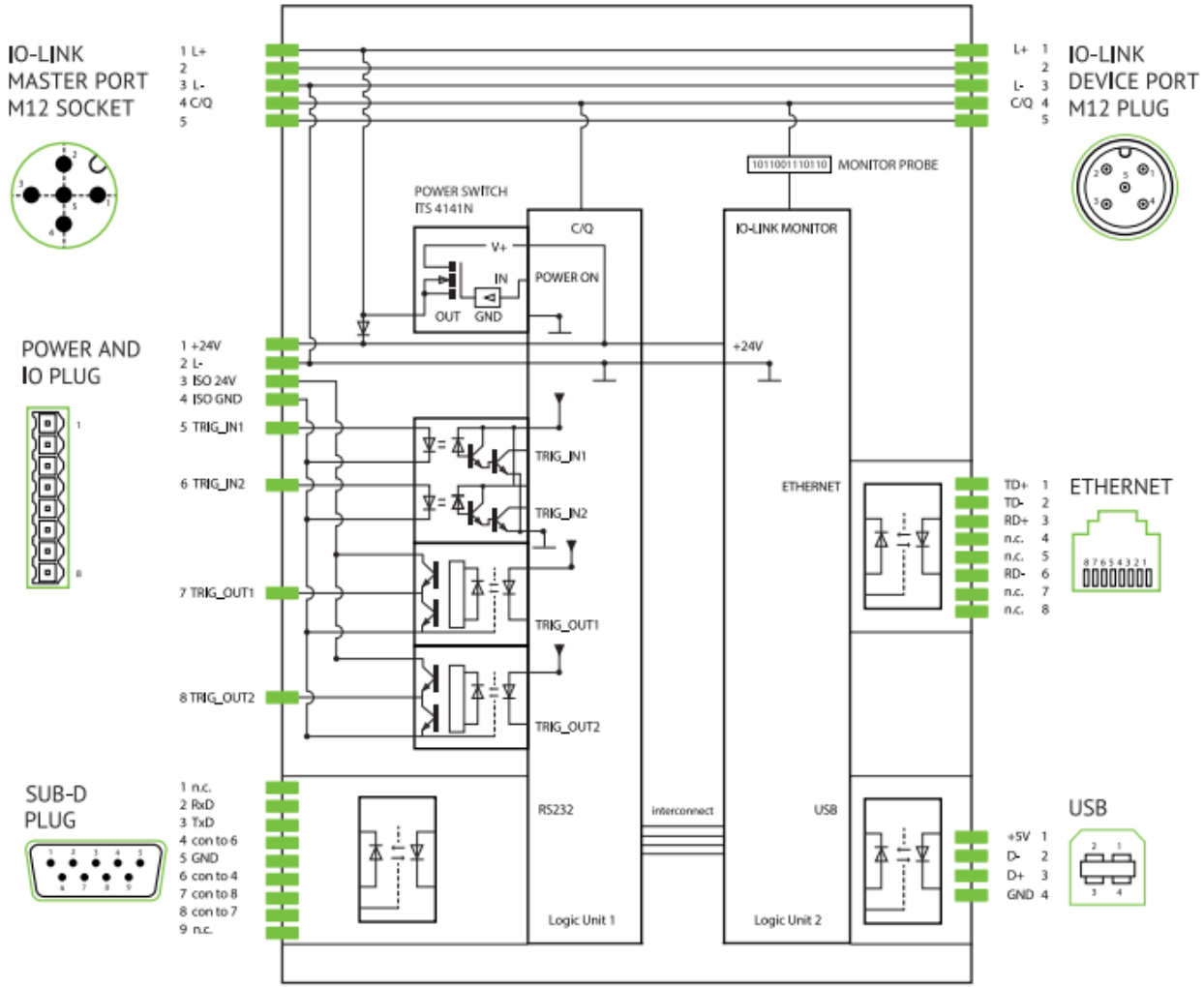
Ethernet (LAN) communication works over UDP and TCP/IP protocols using 30718 and 10001 ports by default. It is tested and suitable only for networks with automatic DHCP configuration. If you have some specific settings in your local network where you are going to use iqInterface, contact your system administrator in case of iqInterface detection problems.

2.3 IO-Link connection

iqInterface has two IO-Link M12 connectors:

- Master port for IO-Link master mode to connect it to other IO-Link device
- Device port for IO-Link generic device mode to connect it to other IO-Link master

2.4 Pin connection diagram



CHAPTER 3

iqTool GUI

The iqTool GUI is dedicated to the manual management of iqInterface device. You can use it to perform the following actions:

- *configure* connection between iqInterface and PC with Windows OS
- standard IO-Link actions in *IO-Link master mode*
- simulation of IO-Link device in *generic device mode*
- monitor IO-Link communication status in both master/device modes
- observe communication activity with detailed description in log text box on the right side
- *update and upgrade* iqInterface firmware

For installation and tool update details see *Installation guide*.

3.1 Installation guide

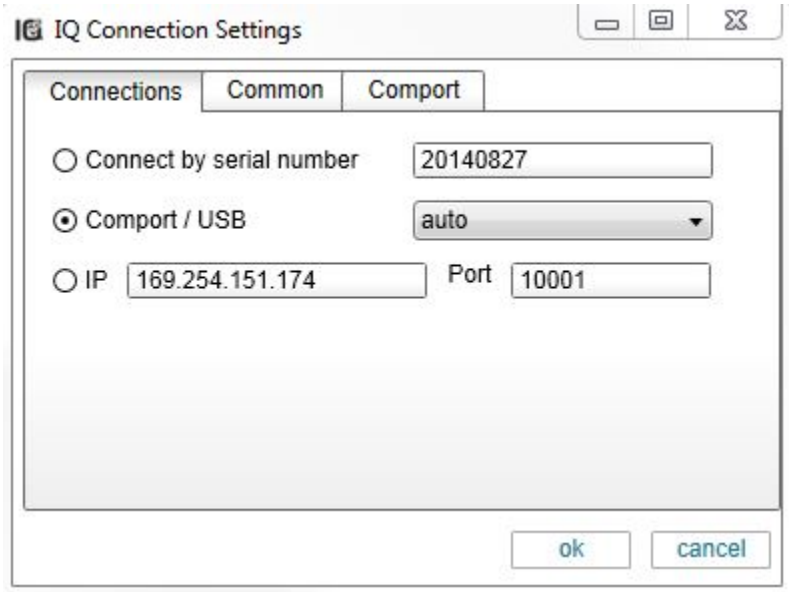
To install the iqTool GUI perform the following actions:

- connect iqInterface to PC with OS Windows XP or 7 (see *Physical connection*)
- install USB virtual port driver if USB connection to PC is going to be used (you can find it on USB flash drive delivered with iqInterface or driver homepage)
- install .Net framework version 4 or higher (it can also be found on the USB flash drive under *iqInterface_GUI_Tool* folder)
- uninstall the old version of iqTool if it has been installed
- download the latest version of iqTool coming with iqInterface bundle on its download page or take the installer from the USB flash drive under *iqInterface_GUI_Tool\XXXX_XX_XX_iqTool_VX.X.X.X* folder (see also *version changelog*)
- Launch the installer and install the iqTool GUI following the instructions

3.2 PC connection configuration

You can setup connection parameters before connecting PC with iqTool to iqInterface using the *settings* button at the top panel.

In the leftmost tab *Connections* you can select connection type which you use to connect PC with iqInterface.



You have three options:

- Connect to iqInterface with a given **serial number**. You can find the serial number of your iqInterface on its box. In this case iqInterface can be connected to PC anyway. iqTool will try all available connection in OS to find iqInterface with a given serial number.
- **Comport/USB connection**. If you know the comport number (virtual in case of USB) of connected iqInterface you can manually select it. Otherwise you can select *auto* port and iqTool will try to connect to iqInterface over each OS registered comport.
- **Ethernet connection** (tcp/ip). If you know the IP address and port number (10001 by default) of connected iqInterface you can manually input them.

Default connection type is auto comport/usb.

Common and *Comport* tabs contains specific communication settings which should be normally not changed. Request timeout in *Common* tab can be increased in case of long IO-Link startup in master mode (e.g. data storage on COM2 or COM1 baudrates).

The connection settings are saved in the user system *Application Data* folder. For example, for Windows 7 it is:

C:\Users\USER_NAME\AppData\Roaming\iqTool\settings.xml

After its setup the connection with iqInterface can be established pressing “Connect” button at the top leftmost corner. The same button can be used to disconnect iqInterface (free OS resources: comport or tcp/ip connection).

3.3 IO-Link master mode

In master mode iqInterface can be connected to other IO-Link device to perform standard IO-Link master actions. You can configure master system manager parameters (cycle time, device check level etc.) and setup data storage. After configuring the master you can change IO-Link operation mode (inactive, preoperate, operate, auto), receive IO-Link

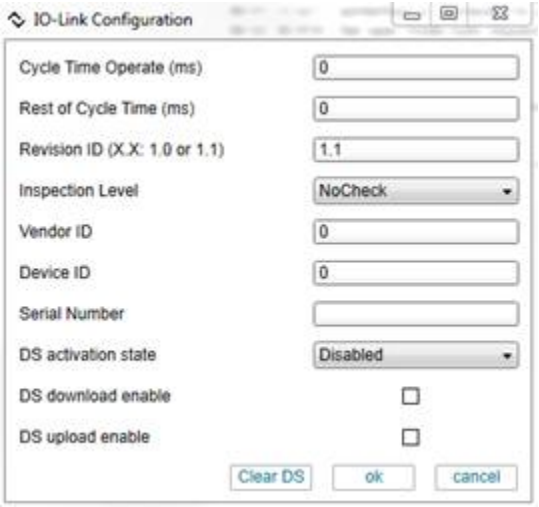
events, set/get process data and read/write ISDU parameters. You can also record macro actions sequences to replay them automatically at once. To switch into master mode:

- disconnect iqTool from iqInterface if connected (see *PC connection*)
- select "iqMaster" tab and connect iqTool to iqInterface again

3.3.1 Master Configuration

After connecting iqTool to iqInterface and pressing *IO-Link config* button you can configure the following parameters of IO-Link master mode:

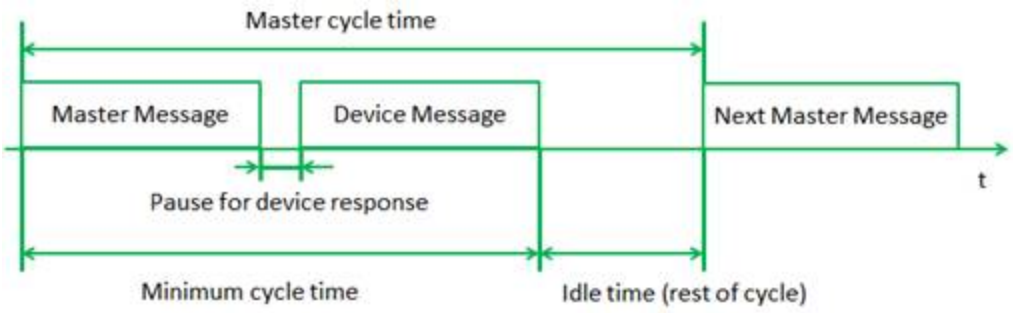
- master cycle time in milliseconds
- inspection level and parameters to be checked in startup
- data storage



Inspection level influences which configured parameters in device will be checked by master system manager according to IO-Link specification during communication startup (revision/device/vendor id and serial number, see also *mst_ConfigT*).

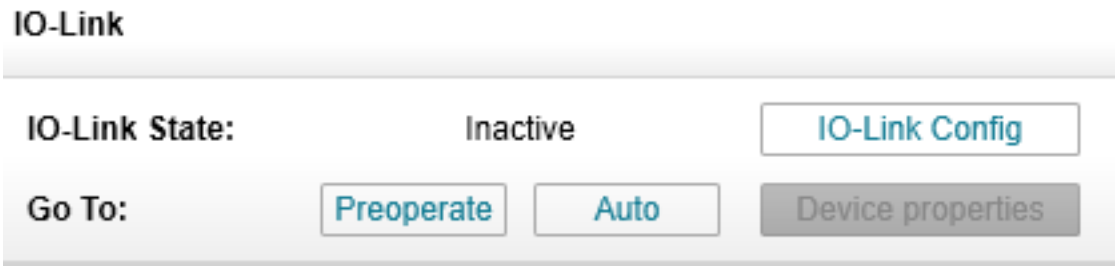
Data storage mechanism can be enabled or disabled during communication startup as well as its upload and download directions. You can also clean up data storage in master to reset it and use it with another connected device (otherwise a new device will be rejected because of its id and its vendor id check with already saved ones).

If you set **master cycle time** then master will try to communicate with device using exactly this cycle time and will not consider "rest of cycle time" parameter. If you leave it zero then master system manager calculates automatically the minimum time necessary to send master message and receive device response during a communication cycle. In case of setting non-zero *rest of cycle time* master system manager adds it to the previously calculated minimum cycle time and will use it for communication. If you set *rest of cycle time* to zero then master system manager will double minimum cycle time and use this value to communicate (*rest of cycle time* equals minimum cycle time).



3.3.2 Change IO-Link operation mode

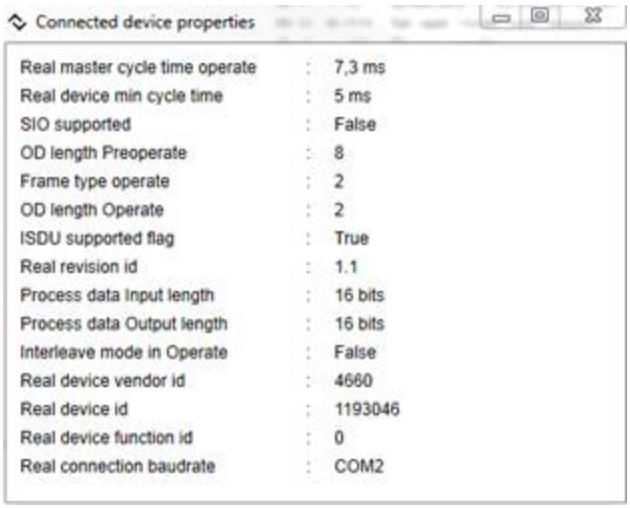
Every time you connected iqTool to iqInterface in master mode you can see status of current IO-Link communication. It is located after label "IO-Link state" in "IO-Link" group box (Inactive, Preoperate, Operate or Check fault). There are also possible transfers from this state after "Go to" label (Inactive, Preoperate, Operate or Auto). "Check fault" state means that you have configured master to check some parameters during startup (see *Master Configuration*) and they failed. "Check fault" state is equivalent to preoperate but you cannot go to operate from it. "Auto" mode means that iqInterface will try constantly to switch to operate with any connected device even after a communication lost with last one. In contrast "Operate" mode means that iqInterface will make only one trial to establish communication with a device according to IO-Link specification.



3.3.3 Active IO-Link communication

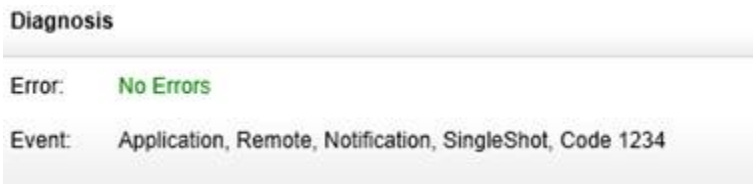
Current IO-Link state

If you have established an active IO-Link communication (state "preoperate" or "operate") you can see its current state and properties of connected device pressing "Device properties" button in the "IO-Link" group box.



Incoming events from device

You can also see last event sent by the connected device in the "Diagnosis" group box (former arrived events still stay in log text box).



Read/Write ISDU requests

To read/write ISDU (Indexed Service Data Unit) parameters or commands input corresponding index and subindex of ISDU and data in case of write operation. Then press "read" or "write" buttons. The controls are located in "On-request data" group boxes.

Read reuest:

Write reuest:

Monitor/Set Process Data Input/Output

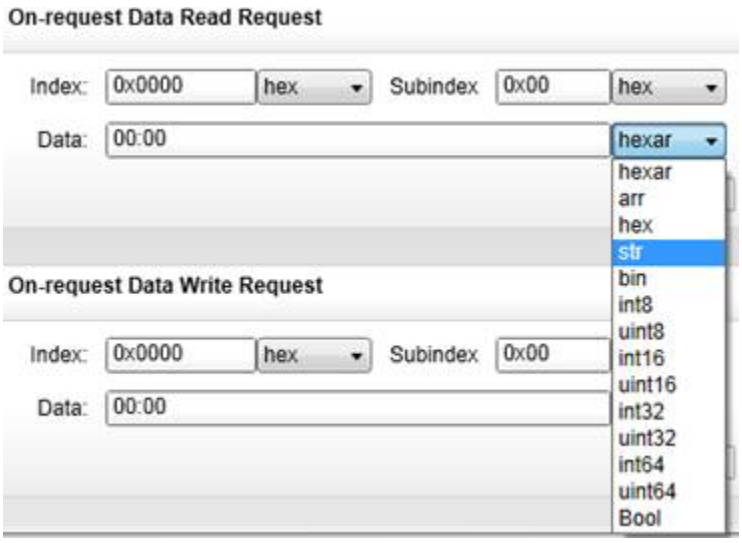
In operate state you can see current process data input which device sends to master and set current process data output which master will send to device. You can also set process data output validity using check box under the “Set” button (checked – valid, not checked - invalid).

Data view

Process data, ISDU index, subindex and its data can be seen in different representations.

- “hexar” is semicolon separated 2-symbol hex byte array like “12:A4:CD” (default)
- ASCII string
- Boolean (one byte: 0 – false; not 0 – true, packed as 0xFF) type
- signed/unsigned integers with various lengths 8-64 bits.

Boolean or integer types can be viewed only if data length corresponds to them. To send their values iqTool pack them into corresponding bytes number according to IO-Link specification (MSB is sent first).



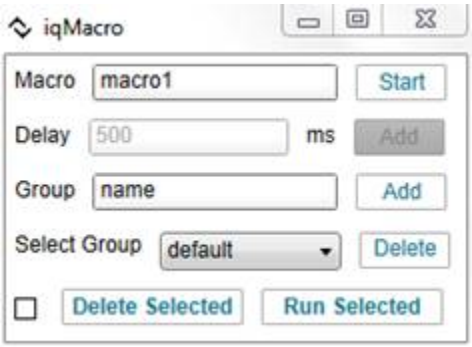
3.3.4 Macro sequences

After connecting iqTool to iqInterface in master mode you can record macro sequences. The macro sequences consist of the standard IO-Link actions:

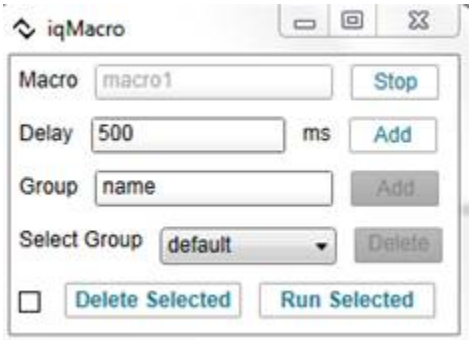
- change operatation mode (for example, go to operate)
- set process data to some value, change its validity
- write/read ISDU with given index/subindex”

To activate macro manager press “Macro” button at the top panel of the “Log and Macro” group box.

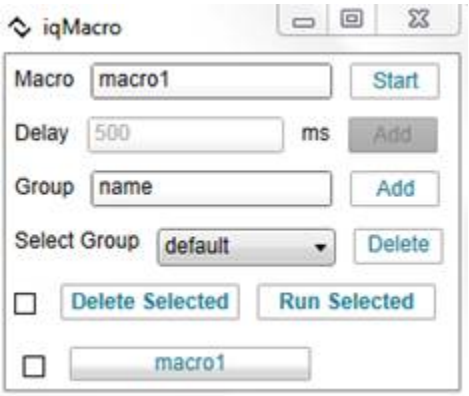
To start macro recording enter new macro name and press the “Start” button in the opened macro manager window.



Perform desired actions (change IO-Link mode, set process data, read/write ISDU). You can also add delays between actions. To do it input delay value to “Delay”-labeled text box and press “Add” button. The iqTool will automatically wait the delay after last action before adding delay to perform the next one. After all actions are done, press “Stop” button to finish the macro recording.

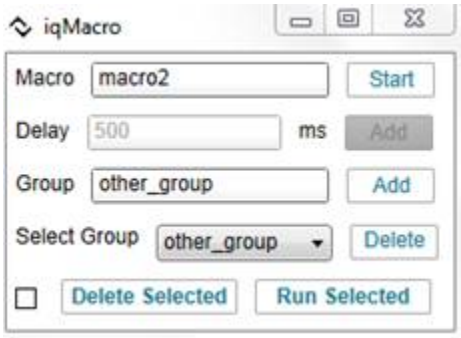


You can replay the recorded sequence pressing the button labeled with inputted macro name.



To delete a macro select it and press “Delete” button. You can also select several macros and delete them pressing “Delete Selected” or run one after another pressing “Run Selected”.

You can also group macros logically. There is always default macro group. To create a new group you can input a new group name in the "Group" text box and press "Add" button. Then recorded macros will be assigned to the new group. Later you can switch between them using "Select Group" combo box.



The recorded macro sequences can be easily transferred between different PCs. They are saved in the "macros" folder under the user system "Application Data" folder. For example, for windows 7 it is:

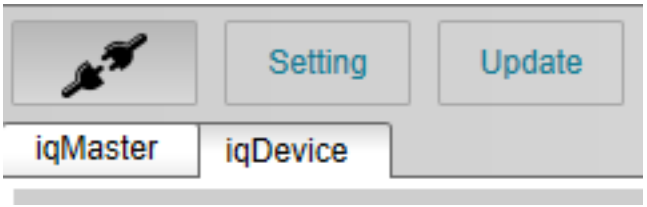
C:\Users\USER_NAME\AppData\Roaming\iqTool\macros

3.4 IO-Link generic device mode

In generic device mode iqInterface can be connected to other IO-Link master and simulate an IO-Link device with a given configuration. You can set/get process data, change values of configured ISDU parameters and send events to the master during active IO-Link communication. All actions can be monitored in log text box with detailed description.

To switch into generic device mode:

- disconnect iqTool from iqInterface if connected (see *PC connection*)
- select "iqDevice" tab and connect iqTool to iqInterface again



3.4.1 Device Configuration

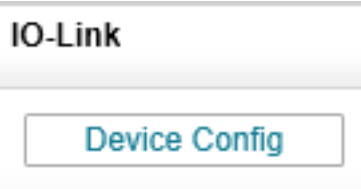
After connecting iqTool to iqInterface you can configure its generic device mode. You can set IO-Link device communication parameters (baudrate, min cycle time, frame etc.) and add ISDU parameters before starting IO-Link on the master side.

Important: After changing any parameters or loading them (see Backup configuration) you should press the "Download" button at the bottom of "ISDU" group box to apply their new values in the iqInterface. To load the current configuration from iqInterface to iqTool press the neighbor "Upload" button.

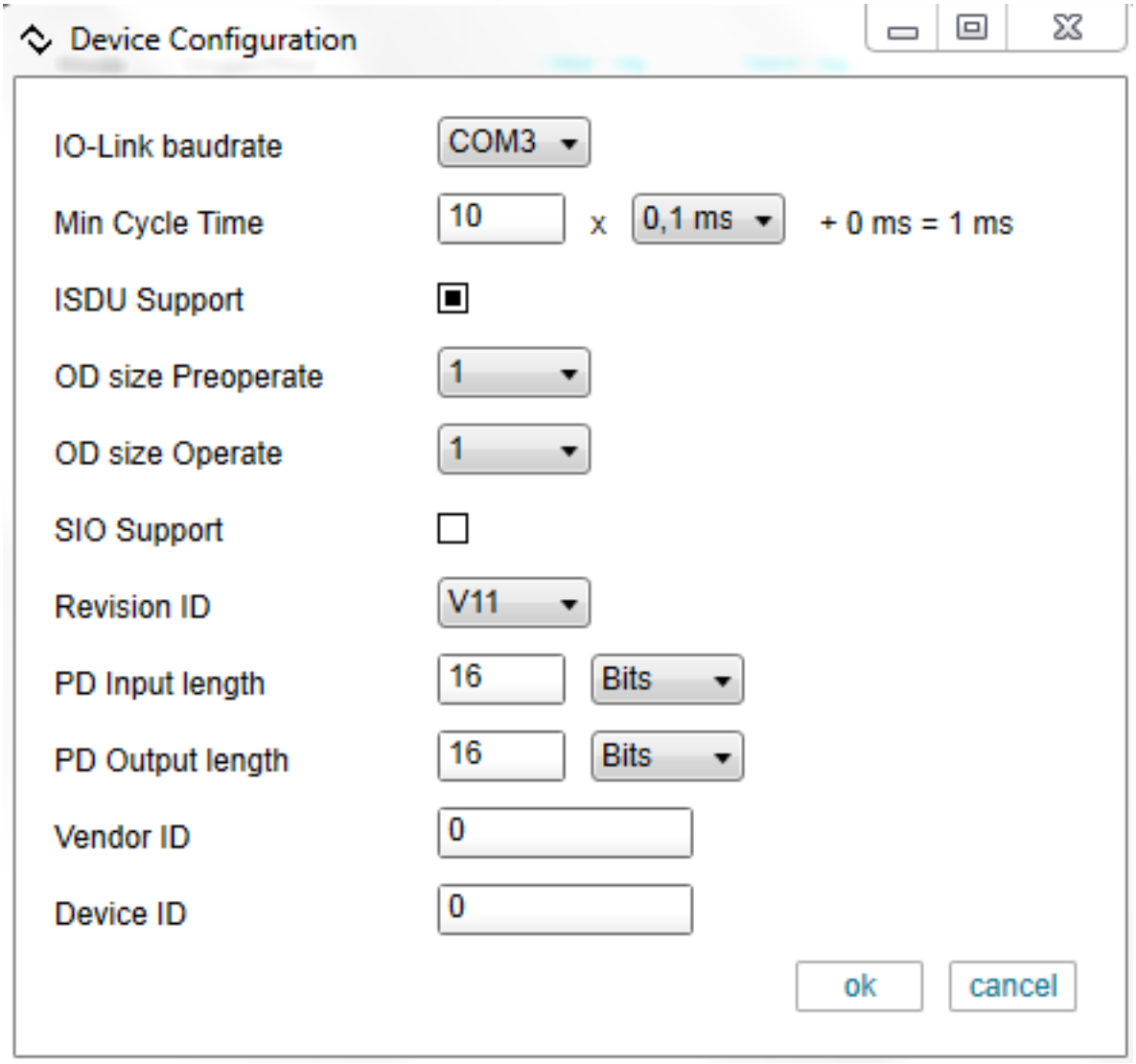
These buttons are active only if iqInterface is connected to iqTool and there is no current IO-Link communication with iqInterface as an IO-Link device. If IO-Link state is not inactive (preoperate or operate, you can see it in the "IO-Link" group box) you have to stop IO-Link communication from the master side or disconnect iqInterface from the master to configure it. Without a master you have to use an external power supply.

IO-Link device communication parameters

To configure IO-Link device communication parameters press the "Device Config" button at the top leftmost corner of the "IO-Link" group box:



Configuration dialog:



Min cycle time should be sufficient to send/receive all master/device message bytes with the selected baudrate:

*Tcycle time = (Tmaster message + OD + Tdevice message) * 14 * Tbit + Tidle*

where:

Tmaster message = 2 + Process data output byte count

OD = On-request data byte count (from master or device, depending on r/w direction)

Tdevice message = Process data input byte count + 1 + 1

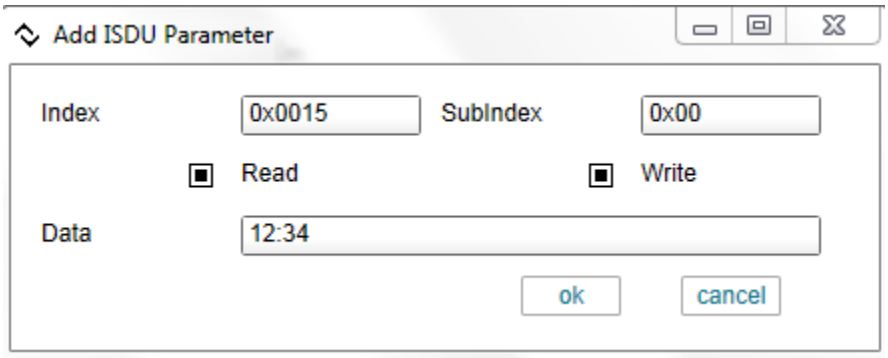
Tbit = 0.00434ms (for COM3), 0.02604ms (COM2), 0.20833ms (COM1)

Tidle = some pause for an idle time can be roughly the same as the previous part.

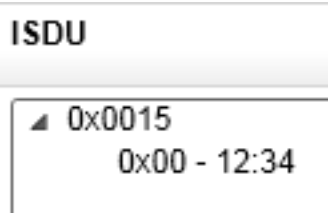
ISDU Parameters

To add new ISDU (Indexed Service Data Unit) parameter press “Add” button at the bottom of *ISDU* group box.

Then input its index, subindex, byte sequence value (2-symbol hex values separated by ":"), set read/write permission and press *ok*.



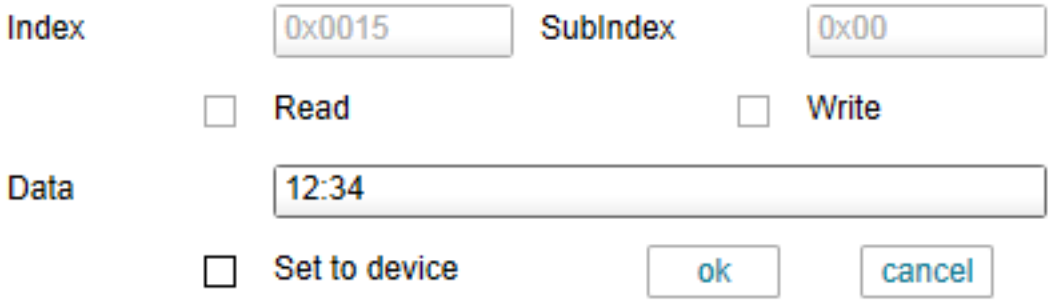
After successfully adding new ISDU parameter you can see it in the *ISDU* index/subindex tree structure.



To remove an ISDU parameter, select it in the *ISDU* tree and press *Remove* button.

All ISDU parameters must have a unique index and a unique subindex within a given index. Subindices can *only* go in consecutive order from 0 to some value less than or equal 0xFF without missing values.

You can change a value of an ISDU parameter (also during active IO-Link communication). To do it press *edit* button, change byte sequence value and press *ok*. You can also set a "Set to device" option and a changed value will be also transferred to iqInterface if it is connected to iqTool.

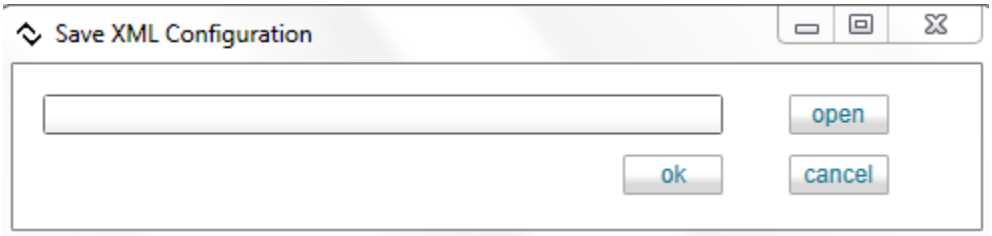


If some ISDU parameters have been read or written from master side during an active IO-Link communication you can see detailed descriptions of these actions in the log text box. To see changed values in the "ISDU" tree press "Upload" button.

Backup configuration

You can back up your current generic device configuration from iqTool to a file on your PC (in xml format but NOT "IODD"). To do it press the "Save" button at the bottom of the "ISDU" group box:

then press "open" button to select a file to save to and then "ok" button.



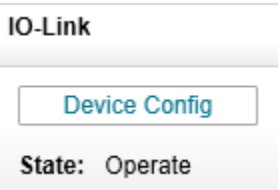
To restore a backup configuration press "Load" button, select a file with it and press "ok" button.

3.4.2 Active IO-Link communication

If IO-Link communication is activated from the master side (Preoperate or Operate) you can make the following actions:

Monitor IO-Link State

You can see IO-Link communication state in the "IO-Link" group box.



Monitor/Set Process data Output/Input

You can see a current process data output which the master transfers to the generic device and set a current process data input which the generic device will transfer to the master. To set process data input valid/invalid, set/unset a check box under the "Set" button.

Send IO-Link Event

You can send an IO-Link event from the generic device to the master. To do it select type and mode of the event, input its code and press "Send Event" button.

3.5 Update and upgrade iqInterface firmware

To update version of iqInterface firmware or upgrade its functions you have to use the iqTool GUI. iqInterface **must be connected to PC over USB** (or comport if USB is not available) as described in *physical connection section*. Launch iqTool and press "Update" button at the top panel to open an Update/Upgrade dialog.

3.5.1 Update

To update firmware version of iqInterface follow the instructions:

- select "Update" tab in the opened Update/Upgrade dialog
- press "open" button to find a file with the new firmware version
- press "Update" button
- wait until the update process is accomplished.

Warning: To make update process successful do not cancel it or interrupt any other way, do not switch off the iqInterface power supply. If iqInterface update process has not succeeded inform the distributor of device.

3.5.2 Upgrade

If not all features are currently activated at your iqInterface you can upgrade it to support them. The following features can be upgraded:

- IO-Link master mode
- IO-Link device mode

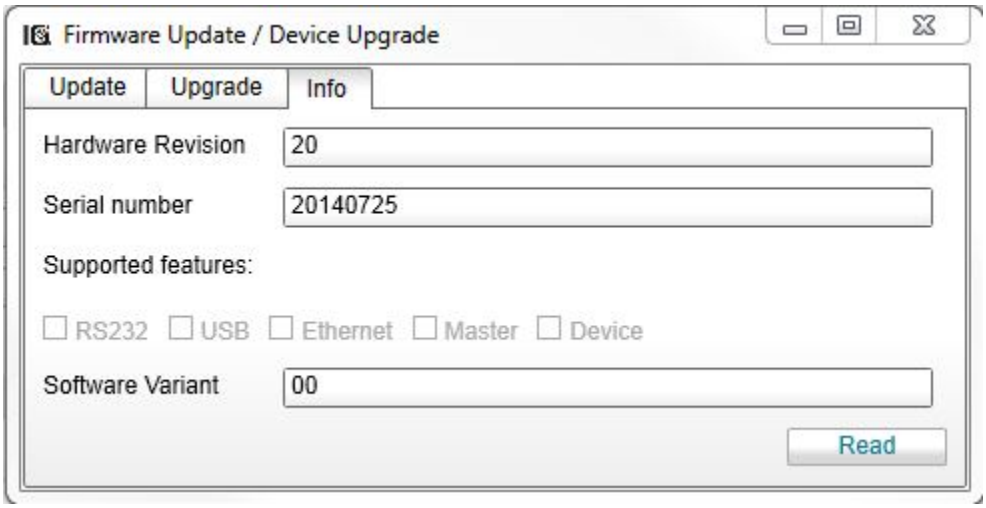
- Comport, USB or Ethernet connection (if hardware ports are available)

To upgrade iqInterface follow the instructions:

- ask the distributor of iqInterface for upgrade code
- select “Upgrade” tab in the opened Update/Upgrade dialog
- input an upgrade code from the distributor and press “Upgrade” button

3.5.3 Device info

You can see information about your currently connected iqInterface if you select the *info* tab in the opened Update/Upgrade dialog and press *Read* button. This information is to report to support service in case of questions or problems. It also contains the currently supported (activated) features of iqInterface (see *Upgrade*).



3.6 iqTool changelog

3.6.1 Version 1.1.0.1 (04.03.2013)

- First basic working version with IO-Link data type controls

3.6.2 Version 1.1.0.2 (04.07.2013)

- add data storage settings
- add index/subindex tips from connected device data storage list
- add macros
- add save and clear log

- add first version of IQ Generic Device

3.6.3 Version 1.1.0.3 (23.10.2013)

- add iqInterface firmware update and upgrade of supported modes using code

3.6.4 Version 1.1.0.4 (25.08.2014)

- clear last event label after switching into new IO-Link communication
- set request/mode timeout to 10sec because of somitimes long DS startup
- add active IO-Link communication indication
- make iq communication asyncron
- add ethernet/tcp ip connection support
- new communication settings for USB/comport/Ethernet
- add io-link device power supply on/off function for master mode
- log data bytes number of ISDU read/write requests

3.6.5 V1.1.0.5 (15.04.2015)

- minor bug fixes

CHAPTER 4

C driver DLL

4.1 Get started

iqInterface can be programmatically managed using C DLL for Windows to write automated tests of IO-Link masters or devices for example. You can use a DLL with its API as a black box or integrate its source code to your existing C/C++ project, for example CVI, Teststand or MS Visual Studio.

You can find the DLL *iqcomm.dll* on the USB flash drive delivered with iqInterface under *dll* folder. Source code of DLL, examples of CVI and MS Visual Studio projects can be found under *dll\CVI_and_Visual_Studio_demo_projects\201X_XX_XX_iqcomm_dll_vX.X.X.X*. The last version of DLL and its source code can be also downloaded from its page (see also *version changelog*). The file *iqcomm_example.c* in the source code folder contains example code of standard iqInterface usage with detailed description in comments.

iqInterface must be properly connected to PC over comport/USB or Ethernet (tcp/ip) interface. For USB usage the driver should be installed (see *physical connection*).

To use DLL or its source you should always include *iqi_define.h* header, *iqi_mst.h* for master mode, *iqi_dev.h* for device mode and *iqi_ethernet.h/iqi_comm.h* for Ethernet (tcp/ip) connection. See further more detailed API description of these modules.

All functions return some non-negative value as a result or negative error code in case of failure. *Error codes* are enumerated in *iqi_define.h* with *ERR_* prefix. Each function also takes a character buffer pointer *cpErrMsg* for an error description in case of failure. The error buffer should be at least 256 bytes. Most of functions also take a port number to identify a port to which iqInterface is connected. This port number is obtained from connect functions, like *mst_Connect*/*mst_EthernetConnect* or *dev_Connect*/*dev_EthernetConnect*.

4.2 API description

4.2.1 Common

Data types

boolean – boolean value: TRUE or FALSE

char08 – one byte signed integer or ASCII character

uchar08 - one byte unsigned integer or ASCII character

int16 – 2 byte signed integer

uint16 - 2 byte unsigned integer

long32 - 4 byte signed integer

ulong32 - 4 byte unsigned integer

long64 - 8 byte signed integer

ulong64 - 8 byte unsigned integer

EventQualifierT (IO-Link event qualifier)

EventQualifierT has the following fields:

- **Instance - received event instance, can be**
  - *EVENT_INSTANCE_PL* – from device physical layer
  - *EVENT_INSTANCE_DL* - from device data link layer
  - *EVENT_INSTANCE_AL* - from device application layer
  - *EVENT_INSTANCE_APPL* - from device application
- **Source - received event source, can be**
  - *EVENT_SOURCE_REMOTE* - from device
  - *EVENT_SOURCE_LOCAL* - from master
- **Type - received event type, can be**
  - *EVENT_TYPE_NOTIFICATION*
  - *EVENT_TYPE_WARNING*
  - *EVENT_TYPE_ERROR*
- **Mode - received event mode, can be**
  - *EVENT_MODE_SINGLE_SHOT* - appears only once
  - *EVENT_MODE_APPEARS* - event appears to disappear later
  - *EVENT_MODE_DISAPPEARS* - event disappears

Error return values

```
#define ERR_NONE                 0  // no error
#define ERR_INTERNAL            -1  // internal error
#define ERR_NOTFOUND            -2  // iqInterface is found for connection
#define ERR_BUFFER_OVERFLOW     -3  // not enough buffer size
#define ERR_PARAMETER           -4  // wrong input function parameter
#define ERR_OPERATION           -5  // operation failed
#define ERR_DATA                -6  // wrong data received or to send
#define ERR_CUSTOM              -7  // specific error
```

4.2.2 Master mode

The master mode API is located in *iqi_mst.h*.

mst_Connect - connect to iqInterface over comport/USB in master mode

mst_EthernetConnect - connect to iqInterface over Ethernet (tcp/ip) in master mode

mst_Disconnect - disconnect iqInterface in master mode

mst_GetGadgetID - get the master device id

mst_GetConfig - get master mode configuration

mst_SetConfig - set master mode configuration

mst_ConfigT - master mode configuration data type

mst_SetOperatingMode - set IO-Link operation mode

mst_GetStatus - get process data, event and on-request data status

mst_SetPDValue - set master process data output

mst_SetPDValidity - set master process data output validity

mst_WaitODRsp - wait for ISDU response

mst_StartReadOD - start ISDU asynchronous read request

mst_GetReadODRsp - get ISDU read response

mst_ReadOD - ISDU read request

mst_StartWriteOD - start asynchronous ISDU write request

mst_GetWriteODRsp - get ISDU write response

mst_WriteOD - ISDU write request

mst_ReadEvent - receive an IO-Link event from device

mst_Connect (connect to iqInterface over comport/USB in master mode)

Function

```
int16 mst_Connect(uchar08 ucFromPortNo, uchar08 ucUntilPortNo, char08 *cpErrMsg);
```

Description

To connect to iqInterface over comport or USB in the master mode call *mst_Connect* function with start and stop comport numbers (virtual in case of USB) which define a range where to look for iqInterface. Save the returned port number which will be a handle to perform all other operations.

Input Parameters

- **uchar08 ucFromPortNo** – (virtual) comport number to start looking for iqInterface with (inclusively)
- **uchar08 ucFromPortNo** – (virtual) comport number to stop looking for iqInterface (inclusively)

Output parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

First successfully found (virtual) comport number to connect or negative error number. The comport number is an established connection handle for all other functions.

Example

```
uchar08 ucPortNoStart = 0;
uchar08 ucPortNoFinish = 10;
int16 iPortNo = mst_Connect(ucPortNoStart, ucPortNoFinish, cpErrMsg);
```

mst_EthernetConnect (connect to iqInterface over Ethernet (tcp/ip) in master mode)

Function

```
int16 mst_EthernetConnect(enet_ConnectionT * connection, uint16 uiSendRecvTimoutMs, char08 *cpErrMsg)
```

Description

If you know the IP address of iqInterface in your network and port (10001 by default) you can use variable of type *enet_ConnectionT* to set the IP and port. To connect to iqInterface in master mode call *mst_EthernetConnect* with the IP/port variable and send/receive timeout in milliseconds as arguments instead of using *mst_Connect*. If you do not know the IP address you can try to search it using *comm_EthernetSearch* function before connecting. See also *Ethernet connection*.

Input Parameters

- **enet_ConnectionT * connection** – pointer to Ethernet connection variable
- **uint16 uiSendRecvTimoutMs** – send/receive timeout in milliseconds used for underlying tcp/ip connection

Output parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

Virtual handle number of successful Ethernet (tcp/ip) connection or negative error number. The virtual number is an established connection handle for all other functions the same as for comport/USB connection (see *mst_Connect*).

Example

```
enet_ConnectionT connection;
uint16 uiSendRecvTimoutMs = 2000; // 2 sec
int16 iPortNo;
strcpy(connection. cpIqInterfaceIpAddr, "XXX.XXX.XXX.XXX"); // IPv4
connection.uiPort = 10001;
iPortNo  = mst_EthernetConnect(&connection, uiSendRecvTimoutMs, cpErrMsg);
mst_GetConfig(iPortNo, &config, cpErrMsg);
```

mst_Disconnect (disconnect iqInterface in master mode)

Function

```
int16 mst_Disconnect(uint16 uiPortNo, char08 *cpErrMsg);
```

Description

To disconnect iqInterface (free OS resources) call *mst_Disconnect* with a previously connected port number.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions

Output parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
mst_Disconnect(iPortNo, cpErrMsg);
```

mst_GetGadgetID (get the master id)

Function

```
int16 mst_GetGadgetID(uint16 uiPortNo, uchar08 *ucpGadgetID, char08 *cpErrMsg);
```

Description

To read the master id call *mst_GetGadgetID* with a previously connected port number.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions

Output parameters

- **uchar08 *ucpGadgetID** – pointer to one byte buffer to return the master id
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
uchar08 id;

mst_GetGadgetID(iPortNo, &id, cpErrMsg);
```

mst_GetConfig and mst_SetConfig (get/set master mode configuration)

Function

```
int16 mst_GetConfig(uint16 uiPortNo, mst_ConfigT *pConfig, char08 *cpErrMsg);
int16 mst_SetConfig(uint16 uiPortNo, mst_ConfigT *pConfig, char08 *cpErrMsg);
```

Description

You can load the master configuration in a variable of type *mst_ConfigT*, manipulate it and save it again to iqInterface before starting an IO-Link communication using *mst_GetConfig* and *mst_SetConfig* functions.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **mst_ConfigT *pConfig** – pointer to variable of type *mst_ConfigT* to set configuration from or get to

Output parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
mst_ConfigT config;
mst_GetConfig(iPortNo, &config, cpErrMsg);
// manipulate config ...
mst_SetConfig(iPortNo, &config, cpErrMsg);
```

mst_ConfigT (master mode configuration data type)

Description

This structure holds the configuration parameters of iqInterface master mode. It is used to get or set them in *mst_GetConfig* and *mst_SetConfig* functions.

Fields

- **uint16 stackVersion (readonly)** – 2 number version of iqStack master in connected iqInterface firmware (X.X: MSB – high number, LSB – low number)
- **IOLBaudrateT realBaudrate (readonly)** – baudrate of current or last successful IO-Link communication, can be *IOL_BAUDRATE_COM1* (4,8 kbit/s), *IOL_BAUDRATE_COM2* (38,4 kbit/s) or *IOL_BAUDRATE_COM3* (230,4 kbit/s)
- **uint16 cycleTimeOperate** – IO-Link master cycle time in milliseconds in operate state (see Master cycle time)
- **uint16 restOfCycleTimeOperate** – idle period of IO-Link master cycle time in milliseconds in operate state (see Master cycle time)

The following parameters manage device checking during the startup according to the state machine of Master System Manager in the IO-Link specification. The configured *device ID*, *vendor ID* and *serial number* (index 0x0015, subindex 0x00) are compared with these parameters in connected IO-Link device. They are checked depending on the setting of the *inspection level*.

- **mst_InspectionLevelT inspectionLevel** – inspection level, can be *MST_IL_NO_CHECK* (check only revision ID), *MST_IL_TYPE_COMP* (check also device and vendor ID) or *MST_IL_IDENTICAL* (check also serial number)
- **IOLRevisionIDT revisionID** – configured IO-Link revision ID, can be *IOL_REV_ID_V1_0* (v1.0) or *IOL_REV_ID_V1_1* (v1.1)

Important: iqInterface cannot emulate legacy v1.0 master. The parameter *revisionID* can be used as a configured revision ID. It is always compared with the revision ID of connected IO-Link device during the startup.

- **uint16 deviceVendorID** – configured vendor ID
- **ulong32 deviceID** – configured device ID
- **uchar08 deviceSerialNumber[MST_DEV_SER_NUM_MAX_LEN + 1]** – configured device serial number
- **uchar08 deviceSerialNumberLen** – length of deviceSerialNumber

The following parameters manage activation state of data storage and its upload/download during the next startup in master mode.

- **mst_DSActivStateT dsActivState** – activation state of data storage, can be *MST_DS_ACTIV_STATE_ENABLED*, *MST_DS_ACTIV_STATE_DISABLED* or *MST_DS_ACTIV_STATE_CLEARED* (contains is cleared and data storage is enabled)
- **boolean dsUploadEnable** – TRUE – upload is enabled, FALSE - upload is disabled
- **boolean dsDownloadEnable** – TRUE – download is enabled, FALSE - download is disabled

mst_SetOperatingMode (change IO-Link operating mode)

Function

```
int16 mst_SetOperatingMode(uint16 uiPortNo, uchar08 ucOperMode, uchar08 ucStateToWait, uchar08 *ucpSt
```

Description

After configuring the master mode you can change IO-Link operation mode to preoperate, operate or auto (see also Change IO-Link operation mode) using *mst_SetOperatingMode* function. You should give it as arguments: an operation mode, communication state to wait for (normally corresponds to mode) and pointer to a variable where actually achieved state will be saved. In *iqi_mst.h* you can find all possible modes with prefix *MST_OPERATING_MODE_* you can switch to and states modes with prefix *MST_STATE_*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uchar08 ucOperMode** – IO-Link operating mode, can be *MST_OPERATING_MODE_INACTIVE*, *MST_OPERATING_MODE_AUTO*, *MST_OPERATING_MODE_PREOPERATE* or *MST_OPERATING_MODE_OPERATE*
- **uchar08 ucStateToWait** – state to wait, can be *MST_STATE_INACTIVE*, *MST_STATE_CHK_FAULT*, *MST_STATE_PREOPERATE* or *MST_STATE_OPERATE*

Output Parameters

- **uchar08 *ucpState** – pointer to variable where to return actually achieved state, which value can be *MST_STATE_INACTIVE*, *MST_STATE_CHK_FAULT*, *MST_STATE_PREOPERATE* or *MST_STATE_OPERATE*
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
mst_StateT state;
uchar08 ucState;
mst_SetOperatingMode(iPortNo, MST_OPERATING_MODE_OPERATE, MST_STATE_OPERATE, &ucState, cpErrMsg);
state = (mst_StateT)ucState;
```

To deactivate the IO-Link communication switch to the inactive mode using *mst_SetOperatingMode* function (send a fallback command to IO-Link device)

```
mst_SetOperatingMode(iPortNo, MST_OPERATING_MODE_INACTIVE, MST_STATE_INACTIVE, &ucState, cpErrMsg);
```

mst_GetStatus (get status of iqInterface and PD input in master mode)

Function

```
int16 mst_GetStatus(uint16 uiPortNo, uchar08 *pStatus, uchar08 *ucpPDInBuf, uint16 uiPDInBufSize, cha
```

Description

You can read master status with a current IO-Link state, process data input from IO-Link device and its validity using *mst_GetStatus* function with a pointer to master status variable and process data input buffer. You can use a bit field of type *mst_StatusT* (see example) to extract current IO-Link communication state, process data input validity (0 – valid,

1 - invalid) and whether a new event has come from device (0 – no events, 1 – new event). If a new event has come you can read it using *mst_ReadEvent* function.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiPDInBufSize** – size of process data input buffer *ucpPDInBuf* (32 bytes length is recommended)

Output Parameters

- **uchar08 *pStatus** – pointer to variable where to return a status byte, it can be decoded using *mst_StatusT* data type (see example)
- **uchar08 *ucpPDInBuf** – process data input sent by IO-Link device
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

Actual size of process data input written to *ucpPDInBuf* or negative error number.

Example

```
uchar08 pdIn[32];
mst_StatusT status;
uchar08 pdInValidity;
uchar08 isNewEvent;
mst_GetStatus(iPortNo, (uchar08*)(&status), pdIn, 32, cpErrMsg);
state = (mst_StateT)(status.State);
pdInValidity = status.PDInInvalid; // 0 - valid, 1 - invalid
isNewEvent = status.Event; // 0 - no events, 1 - new event
```

mst_SetPDValue: (set process data output)

Function

```
int16 mst_SetPDValue(uint16 uiPortNo, uchar08 *ucpPDOutBuf, uint16 uiBytesCount, char08 *cpErrMsg);
```

Description

You can set process data output which will be transferred to IO-Link device using *mst_SetPDValue* function with a pointer to process data output buffer.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uchar08 *ucpPDOutBuf** – process data output buffer to set
- **uint16 uiBytesCount** – byte size of process data output buffer to set *ucpPDOutBuf*

Output Parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
uchar08 pdOut[32];
// set pdOut ...
mst_SetPDValue(iPortNo, pdOut, 2, cpErrMsg);
```

mst_SetPDValidity (set process data output validity)

Function

```
int16 mst_SetPDValidity(uint16 uiPortNo, boolean bValidity, char08 *cpErrMsg);
```

Description

To set validity of current process data output use "mst_SetPDValidity" function.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **boolean bValidity** – process data output validity (*TRUE* – valid, *FALSE* - invalid)

Output Parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
mst_SetPDValidity(iPortNo, TRUE, cpErrMsg);
```

mst_WaitODRsp (Wait for ISDU response)

Function

```
int16 mst_WaitODRsp(uint16 uiPortNo, char08 *cpErrMsg);
```

Description

This function waits ISDU response from device with timeout *MST_OD_TIMEOUT* after sending ISDU request with *mst_StartReadOD* or *mst_StartWriteOD*. The function is used to make asynchronous ISDU read/write requests (see example). The response can be read by *mst_GetReadODRsp* and *mst_GetWriteODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions

Output Parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number (*ERR_INTERNAL* – wait timeout *MST_WRITE_OD_TIMEOUT*, see code in *uipISDUErr*)

Example

```
int16 error_or_size;
// asynchronous ISDU read request example
uchar08 ucpBuf[255]; // buffer with request result
uint16 index = 0x0015;
uchar08 subindex = 0x00;
uint16 uiISDUErr = 0; // possible ISDU error
mst_StartReadOD(iPortNo, index, subindex, cpErrMsg);
if (mst_WaitODRsp(uiPortNo, cpErrMsg) == ERR_NONE)
{
    error_or_size = mst_GetReadODRsp(iPortNo, ucpBuf, 255, &uiISDUErr, cpErrMsg)
    if (error_or_size >= ERR_NONE)
    {
        // reponse ok, process 'error_or_size' number of data bytes in ucpBuf
    }
    else
    {
        // ISDU error in uiISDUErr
    }
}
else
{
    // ISDU response waiting timeout
}
// asynchronous ISDU write request example
uchar08 ucpBuf[255]; // buffer to write
uchar08 lengthToWrite = 5; // write 5 first bytes of *ucpBuf* buffer
uint16 index = 0x0015;
uchar08 subindex = 0x00;
uint16 uiISDUErr = 0; // possible ISDU error
// fill *ucpBuf* buffer with data bytes to write ...
mst_StartWriteOD(iPortNo, index, subindex, ucpBuf, lengthToWrite, cpErrMsg);
if (mst_WaitODRsp(uiPortNo, cpErrMsg) == ERR_NONE)
{
    error_or_size = mst_GetWriteODRsp(iPortNo, &uiISDUErr, cpErrMsg)
    if (error_or_size >= ERR_NONE)
    {
        // reponse ok
    }
    else
    {
        // ISDU error in uiISDUErr
    }
}
else
{
    // ISDU response waiting timeout
}
```

mst_StartReadOD (Start asynchronous ISDU read request)

Function

```
int16 mst_StartReadOD(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, char08 *cpErrMsg);
```

Description

Starts asynchronous ISDU read request. Device response can be waited for by *mst_WaitODRsp* and read by *mst_GetReadODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiIndex** – ISDU index to read
- **uchar08 ucSubindex** – ISDU subindex to read

Output Parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number

Example

See example of *mst_WaitODRsp*.

mst_GetReadODRsp (Get ISDU read response)

Function

```
int16 mst_GetReadODRsp(uint16 uiPortNo, uchar08 *ucpBuf, uint16 uiBufSize, uint16* uipISDUErr, char08
```

Description

Reads ISDU read response from device started by *mst_StartReadOD* and waited by *mst_WaitODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiBufSize** – size of On-request data buffer *ucpBuf*

Output Parameters

- **char08 *ucpBuf** – pointer to On-request data buffer to return read ISDU data
- **uint16 *uipISDUErr** – pointer to variable to return possible ISDU error code
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

Negative error number (*ERR_OPERATION* – ISDU error, see code in *uipISDUErr*) or size of response data in buffer *ucpBuf*

Example

See example of *mst_WaitODRsp*.

mst_ReadOD (ISDU read request)

Function

```
int16 mst_ReadOD(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBuf, uint16 uiBufSi
```

Description

To make ISDU read request use "mst_ReadOD" function with an ISDU index, subindex, response data buffer and a pointer to possible ISDU error. The function is a short hand for call sequence of ISDU read described in example of *mst_WaitODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiIndex** – ISDU index to read
- **uchar08 ucSubindex** – ISDU subindex to read
- **uint16 uiBufSize** – size of On-request data buffer *ucpBuf*

Output Parameters

- **char08 *ucpBuf** – pointer to On-request data buffer to return read ISDU data
- **uint16 *uipISDUErr** – pointer to variable to return possible ISDU error code
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

Negative error number (*ERR_OPERATION* – ISDU error, see code in *uipISDUErr*) or size of response data in buffer *ucpBuf*

Example

```
uchar08 ucpBuf[255]; // buffer with request result
uint16 index = 0x0015;
uchar08 subindex = 0x00;
uint16 uiISDUErr = 0; // possible ISDU error
mst_ReadOD(iPortNo, index, subindex, ucpBuf, 255, &uiISDUErr, cpErrMsg);
```

mst_StartWriteOD (Start asynchronous ISDU Write request)

Function

```
int16 mst_StartWriteOD(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBuf, uint16 u
```

Description

Starts asynchronous ISDU Write request. Device response can be waited for by *mst_WaitODRsp* and read by *mst_GetWriteODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiIndex** – ISDU index to Write
- **uchar08 ucSubindex** – ISDU subindex to Write
- **char08 *ucpBuf** – pointer to On-request data buffer to write
- **uint16 uiBytesCount** – bytes number in On-request data buffer *ucpBuf* to write

Output Parameters

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number

Example

See example of *mst_WaitODRsp*.

mst_GetWriteODRsp (Get ISDU Write response)

Function

```
int16 mst_GetWriteODRsp(uint16 uiPortNo, uint16* uipISDUErr, char08 *cpErrMsg);
```

Description

Reads ISDU Write response from device started by *mst_StartWriteOD* and waited by *mst_WaitODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions

Output Parameters

- **uint16 *uipISDUErr** – pointer to variable to return possible ISDU error code
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number (*ERR_OPERATION* – ISDU error, see code in *uipISDUErr*)

Example

See example of *mst_WaitODRsp*.

mst_WriteOD (ISDU write request)

Function

```
int16 mst_WriteOD(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBuf, uint16 uiByte
```

Description

To make ISDU write request use *mst_WriteOD* function with an ISDU index, subindex, data buffer to write, its length and a pointer to possible ISDU error. The function is a short hand for call sequence of ISDU write described in example of *mst_WaitODRsp*.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions
- **uint16 uiIndex** – ISDU index to write
- **uchar08 ucSubindex** – ISDU subindex to write
- **char08 *ucpBuf** – pointer to On-request data buffer to write
- **uint16 uiBytesCount** – bytes number in On-request data buffer *ucpBuf* to write

Output Parameters

- **uint16 *uipISDUErr** – pointer to variable to return possible ISDU error code

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number (*ERR_OPERATION* – ISDU error, see code in *uipISDUErr*)

Example

```
uchar08 ucpBuf[255]; // buffer to write
uchar08 lengthToWrite = 5; // write 5 first bytes of *ucpBuf* buffer
uint16 index = 0x0015;
uchar08 subindex = 0x00;
uint16 uiISDUErr = 0; // possible ISDU error
// fill *ucpBuf* buffer with data bytes to write ...
mst_WriteOD(iPortNo, index, subindex, ucpBuf, lengthToWrite, &uiISDUErr, cpErrMsg);
```

mst_ReadEvent (receive an IO-Link event from device)

Function

```
int16 mst_ReadEvent(uint16 uiPortNo, uchar08 *ucpQualifier, uint16 *uipCode, char08 *cpErrMsg);
```

Description

If you await an event from IO-Link device you can receive it using *mst_ReadEvent* function with a pointer to a qualifier and a code of received event. A *Fields* attribute of variable of type *EventQualifierT* contains event instance, source, type and mode (see example and *IO-Link event qualifier*). You can also receive IO-Link communication lost event (code 0xFF22) this way in case of IO-Link communication interruption. To monitor whether a new event has come from device use *mst_GetStatus* function.

Input Parameters

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *mst_Connect* or *mst_EthernetConnect* functions

Output Parameters

- **uchar08 *ucpQualifier** – pointer to byte with received event qualifier where event instance, source, type and mode are encoded according to IO-Link specification, variable of type *EventQualifierT* can be used to decode these event attributes (see example)
- **uchar08 * uipCode** – pointer to received event code
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number (*ERR_INTERNAL* is returned if timeout occurred and no event was received)

Example

```
EventQualifierT eventQualifier;
uchar08 ucEventQualifier;
uint16 uiEventCode;
mst_ReadEvent(iPortNo, &ucEventQualifier, &uiEventCode, cpErrMsg));
eventQualifier.Byte = ucEventQualifier;

// eventQualifier.Instance: received event instance, can be
// EVENT_INSTANCE_PL - from device physical layer
// EVENT_INSTANCE_DL - from device data link layer
// EVENT_INSTANCE_AL - from device application layer
// EVENT_INSTANCE_APPL - from device application
```

```
// eventQualifier.Source: received event source, can be
// EVENT_SOURCE_REMOTE - from device
// EVENT_SOURCE_LOCAL - from master

// eventQualifier.Type: received event type, can be
// EVENT_TYPE_NOTIFICATION
// EVENT_TYPE_WARNING
// EVENT_TYPE_ERROR

// eventQualifier.Mode: received event mode, can be
// EVENT_MODE_SINGLE_SHOT - appears only once
// EVENT_MODE_APPEARS - event appears to disappear later
// EVENT_MODE_DISAPPEARS - event disappears
```

4.2.3 Device mode

The device mode API is located in *iqi_dev.h*.

dev_Connect - connect to iqInterface over comport/USB in device mode

dev_EthernetConnect - connect to iqInterface over Ethernet (tcp/ip) in generic device mode

dev_Disconnect - disconnect iqInterface in generic device mode

dev_GetConfig - get generic device mode configuration

dev_SetConfig - set generic device mode configuration

dev_ConfigT - generic device mode configuration data type

dev_GetODRequest - wait and read ISDU request from master

dev_ODResponse - send ISDU response to master

dev_AddISDU - add new ISDU parameter to generic device

dev_RemoveAllISDU - remove all ISDU parameters from generic device

dev_GetISDUValue - get ISDU parameter value

dev_SetISDUValue - set ISDU parameter value

dev_GetStatus - get generic device status

dev_SetPDIn - set process data input and its validity

dev_SendEvent - send IO-Link event to master

dev_Connect (connect to iqInterface over comport/USB in generic device mode)

Function:

```
int16 dev_Connect(uchar08 ucFromPortNo, uchar08 ucUntilPortNo, char08 *cpErrMsg);
```

Description:

To connect to iqInterface over comport or USB in the device mode call *dev_Connect* function with start and stop comport numbers (virtual in case of USB) which define a range where to look for iqInterface. Save the returned port number which will be a handle to perform all other operations.

Input Parameters:

- **uchar08 ucFromPortNo** – (virtual) comport number to start looking for iqInterface with (inclusively)
- **uchar08 ucUntilPortNo** – (virtual) comport number to stop looking for iqInterface (inclusively)

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

First successfully found (virtual) comport number to connect or negative error number. The comport number is an established connection handle for all other functions.

Example:

```
uchar08 ucPortNoStart = 0;
uchar08 ucPortNoFinish = 10;
int16 iPortNo = dev_Connect(ucPortNoStart, ucPortNoFinish, cpErrMsg);
```

dev_EthernetConnect (connect to iqInterface over Ethernet (tcp/ip) in generic device mode)

Function:

```
int16 dev_EthernetConnect(enet_ConnectionT * connection, uint16 uiSendRecvTimoutMs, char08 *cpErrMsg)
```

Description:

If you know the IP address of iqInterface in your network and port (10001 by default) you can use variable of type *enet_ConnectionT* to set the IP and port. To connect to iqInterface in generic device mode call *dev_EthernetConnect* with the IP/port variable and send/receive timeout in milliseconds as arguments instead of using *dev_Connect*. If you do not know the IP address you can try to search it using *comm_EthernetSearch* function before connecting. See also *Ethernet connection*.

Input Parameters:

- **enet_ConnectionT * connection** – pointer to Ethernet connection variable
- **uint16 uiSendRecvTimoutMs** – send/receive timeout in milliseconds used for underlying tcp/ip connection

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

Virtual handle number of successful Ethernet (tcp/ip) connection or negative error number. The virtual number is an established connection handle for all other functions the same as for comport/USB connection (see dev_Connect).

Example:

```
enet_ConnectionT connection;
uint16 uiSendRecvTimoutMs = 2000; // 2 sec
int16 iPortNo;
strcpy(connection. cpIqInterfaceIpAddr, "XXX.XXX.XXX.XXX"); // IPv4
connection.uiPort = 10001;
iPortNo  = dev_EthernetConnect(&connection, uiSendRecvTimoutMs, cpErrMsg);
dev_GetConfig(iPortNo, &config, cpErrMsg);
```

dev_Disconnect (disconnect iqInterface in generic device mode)

Function:

```
int16 dev_Disconnect(uint16 uiPortNo, char08 *cpErrMsg);
```

Description:

To disconnect iqInterface (free OS resources) call *dev_Disconnect* with a previously connected port number.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
dev_Disconnect(iPortNo, cpErrMsg);
```

dev_GetConfig and dev_SetConfig (get/set generic device mode configuration)

Function:

```
int16 dev_GetConfig(uint16 uiPortNo, dev_ConfigT *pConfig, char08 *cpErrMsg);
int16 dev_SetConfig(uint16 uiPortNo, dev_ConfigT *pConfig, char08 *cpErrMsg);
```

Description:

You can load the generic device configuration in a variable of type *dev_ConfigT*, manipulate it and save it again to iqInterface before starting an IO-Link communication from master side.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **dev_ConfigT *pConfig** – pointer to variable of type *dev_ConfigT* to set configuration from or get to

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number (it is not allowed to configure generic device during active IO-Link communication from master side).

Example:

```
dev_ConfigT config;
dev_GetConfig(iPortNo, &config, cpErrMsg);
// manipulate config...
dev_SetConfig(iPortNo, &config, cpErrMsg);
```

dev_ConfigT (generic device mode configuration data type)

Description:

This structure holds the configuration parameters of iqInterface generic device mode. It is used to get or set them in *dev_GetConfig* and *dev_SetConfig* functions.

Fields

- **uint16 stackVersion (readonly)** – 2 number version of iqStack device in connected iqInterface firmware (X.X: MSB – high number, LSB – low number)
- **IOLBaudrateT supportedBaudrate** – IO-Link communication baudrate, can be *IOL_BAUDRATE_COM1* (4,8 kbit/s), *IOL_BAUDRATE_COM2* (38,4 kbit/s) or *IOL_BAUDRATE_COM3* (230,4 kbit/s)
- **uint16 masterCycleTime (readonly)** – IO-Link master cycle time in milliseconds used by master in current or last successful operate state
- **uint16 minCycleTime** – device minimum cycle time in milliseconds in operate state
- **IOLRevisionIDT revisionID** – IO-Link revision ID, can be *IOL_REV_ID_V1_0* (v1.0) or *IOL_REV_ID_V1_1* (v1.1), version of IO-Link communication to emulate
- **uchar08 ODSizePreoperate** – On-request data bytes number in IO-Link communication message in preoperate state
- **uchar08 ODSizeOperate** – On-request data bytes number in IO-Link communication message in operate state
- **boolean isPDInLengthBytes** – *TRUE*: PDInLength in bytes, *FALSE*: in bits
- **uchar08 PDInLength** – process data input byte number in IO-Link communication message in operate state
- **boolean isPDOutLengthBytes** – *TRUE*: PDOutLength in bytes, *FALSE*: in bits
- **uchar08 PDOutLength** – process data output byte number in IO-Link communication message in operate state
- **uint16 deviceVendorID** – vendor ID
- **ulong32 deviceID** – device ID
- **uchar08 deviceSerialNumber[MST_DEV_SER_NUM_MAX_LEN + 1]** – configured device serial number
- **dev_ISDURspTypeT ISDURspType** – see description of *dev_GetODRequest*

dev_GetODRequest (wait and read ISDU request from master)

Function:

```
int16 dev_GetODRequest(uint16  uiPortNo, uchar08 *ucType, uint16  *uiIndex, uchar08 *ucSubindex, ucha
```

Description:

In order that this function works the *dev_ConfigT* **.ISDURspType** should be set to **OD_RSP_TYPE_AUTO** or **OD_RSP_TYPE_MANUAL** by *dev_SetConfig*.

The default setting for **ISDURspType** is **OD_RSP_TYPE_AT_ONCE** after iqInterface reset. In this case iqInterface responds immediately to master according to configured (saved) ISDU parameters and this function is useless because it never gets a notification about ISDU request.

In case of **OD_RSP_TYPE_AUTO** this function makes iqInterface respond with configured (saved) ISDU parameters and confirm ISDU request to master which will not receive ISDU response without call of this function.

In case of **OD_RSP_TYPE_MANUAL** iqInterface does not response with configured ISDU parameters, the function just returns last received and pending ISDU request, reponse should be sent by *dev_ODResponse*.

ISDU requests to direct page 1 are not indicated by this function, because system manager of device stack is responsible for it.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uint16 ucBufSize** – max size of buffer **ucpBuf**

Output Parameters:

- **uint16 *ucType** – type of ISDU request can be **DEV_ODREQ_TYPE_WRITE**, **DEV_ODREQ_TYPE_WRITE_ERR**, **DEV_ODREQ_TYPE_READ**, **DEV_ODREQ_TYPE_READ_ERR**
- **uint16 *uiIndex** – index of ISDU request
- **uchar08 *ucSubindex** – subindex of ISDU request
- **uchar08 *ucpBuf** – data bytes of ISDU request
- **uchar08 *uiISDUErrCode** – possible ISDU error code
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

Negative error number (ERR_PARAMETER - **ucBufSize** is too small for received ISDU request, ERR_INTERNAL - ISDU request waiting timeout) or size of ISDU data saved in **ucpBuf**.

Example:

```
uint16 uiIndex;
uchar08 ucSubindex;
uchar08 ucODType;
uchar08 ucpBuf[2], ucState;
uint16 uiISDUErr = 0;
int16 isduSize = dev_GetODRequest(iPortNo, &ucODType, &uiIndex, &ucSubindex, ucpBuf, 2, &uiISDUErr, c
```

dev_ODResponse (send ISDU response to master)

Function:

```
int16 dev_ODResponse(uint16 uiPortNo, uchar08 ucType, uchar08 *ucpBuf, uint16 uiBytesCount, uint16 ui
```

Description:

This function sends ISDU response received by *dev_GetODRequest* in case of setting *dev_ConfigT* **.ISDURspType** to **OD_RSP_TYPE_MANUAL**. It also confirms the response to master which will not receive the response without this function call.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uint16 *ucType** – type of ISDU request can be **DEV_ODREQ_TYPE_WRITE**, **DEV_ODREQ_TYPE_WRITE_ERR**, **DEV_ODREQ_TYPE_READ**, **DEV_ODREQ_TYPE_READ_ERR**
- **uchar08 *ucpBuf** – data bytes of ISDU read response if **ucType** is **DEV_ODREQ_TYPE_READ**
- **uint16 uiBytesCount – size of ISDU read response in buffer **ucpBuf** if **ucType** is **DEV_ODREQ_TYPE_READ**
- **uchar08 *uiISDUErrCode** – possible error code of ISDU response if **ucType** is **DEV_ODREQ_TYPE_WRITE_ERR** or **DEV_ODREQ_TYPE_READ_ERR**

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
// config generic device before starting IO-Link communication
dev_ConfigT config;
config.ISDURspType = OD_RSP_TYPE_MANUAL;
dev_SetConfig(iPortNo, &config, cpErrMsg);
// ... IO-Link communication is being started
uint16 uiIndex;
uchar08 ucSubindex;
uchar08 ucODType;
uchar08 ucpBuf[2], ucState;
uint16 uiISDUErr = 0;
int16 isduSize = dev_GetODRequest(iPortNo, &ucODType, &uiIndex, &ucSubindex, ucpBuf, 2, &uiISDUErr, c
// ... process received ISDU request
// ISDU read ok response
uchar08 ucpBuf[2]; // ... fill ucpBuf with response data
dev_ODResponse(iPortNo, DEV_ODREQ_TYPE_READ, ucpBuf, 2, 0, cpErrMsg);
// ISDU read error response
dev_ODResponse(iPortNo, DEV_ODREQ_TYPE_READ_ERR, 0, 0, 0x8011, cpErrMsg);
// ISDU write ok response
dev_ODResponse(iPortNo, DEV_ODREQ_TYPE_WRITE, 0, 0, 0, cpErrMsg);
// ISDU write error response
dev_ODResponse(iPortNo, DEV_ODREQ_TYPE_WRITE_ERR, 0, 0, 0x8011, cpErrMsg);
```

dev_AddISDU (add new ISDU parameter to generic device)

Function:

```
int16 dev_AddISDU(uint16 uiPortNo, uint16 uiFlag, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBu
```

Description:

You can add new ISDU parameters using *dev_AddISDU* function with an ISDU index, subindex, data, data length and a read/write flag as arguments. Master can read and write the ISDU parameter value depending on its access level. The data length of added ISDU will be fixed and checked for exact equality in case of write operation from master. You cannot delete one previously added ISDU parameter but you can delete all of them using *dev_RemoveAllISDU* function (clear ISDU structure) and add all remained ISDU parameters again. The ISDU structure (subindexes within one index) must always stay consistent, see *ISDU Parameters*.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uint16 uiFlag** – flag where ISDU read/write access is decoded, variable of type *dev_ISDU_FlagT* to encode access (see example)
- **uint16 uiIndex** – index of ISDU parameter to add
- **uchar08 ucSubindex** – subindex of ISDU parameter to add
- **uchar08 *ucpBuf** – data bytes of ISDU parameter value to add
- **uint16 uiBytesCount** – size of ISDU parameter value "ucpBuf" to add

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
dev_ISDU_FlagT ISDUFlag;
uchar08 od[256];
uchar08 ISDULength = 2; // ISDU length is 2 bytes
uint16 index = 0x0015;
uchar08 subindex = 0
dev_RemoveAllISDU(iPortNo, cpErrMsg);
ISDUFlag.Data = 0;
ISDUFlag.Bits.IsRead = TRUE;
ISDUFlag.Bits.IsWrite = TRUE;
// fill "od" buffer with ISDU data ...
dev_AddISDU(iPortNo, ISDUFlag.Data, index, subindex, od, ISDULength, cpErrMsg);
```

dev_RemoveAllISDU (remove all ISDU parameters from generic device)

Function:

```
int16 dev_RemoveAllISDU (uint16 uiPortNo, char08  *cpErrMsg);
```

Description:

Remove all ISDU parameters previously added by *dev_AddISDU*.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
dev_RemoveAllISDU(iPortNo, cpErrMsg);
```

dev_GetISDUValue (get ISDU parameter value)

Function:

```
int16 dev_GetISDUValue(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBuf, uint16 u
```

Description:

Every time (also during active IO-Link communication) you can get ISDU parameter current value which could be changed by master. To get it use *dev_GetISDUValue* function with ISDU index, subindex, buffer and its size to return value as arguments. The function returns an actual size of ISDU parameter value returned to the buffer.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uint16 uiIndex** – index of ISDU parameter to get value
- **uchar08 ucSubindex** – subindex of ISDU parameter to get value
- **uint16 uiBufSize** – size of buffer *ucpBuf* for ISDU parameter value to get

Output Parameters:

- **uchar08 *ucpBuf** – pointer to buffer to return ISDU parameter value to get
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
uchar08 odBufferSize = 256;
uchar08 odActualSize = dev_GetISDUValue(iPortNo, index, subindex, od, odBufferSize, cpErrMsg);
```

dev_SetISDUValue (set ISDU parameter value)

Function:

```
int16 dev_SetISDUValue(uint16 uiPortNo, uint16 uiIndex, uchar08 ucSubindex, uchar08 *ucpBuf, uint16 u
```

Description:

You can also set ISDU parameter current value using *dev_SetISDUValue* function with ISDU index, subindex, buffer with value to set and its size as arguments. The value length must equal its length at the moment of adding the ISDU parameter.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uint16 uiIndex** – index of ISDU parameter to set value
- **uchar08 ucSubindex** – subindex of ISDU parameter to set value
- **uchar08 *ucpBuf** – data bytes of ISDU parameter value to set
- **uint16 uiBytesCount** – size of ISDU parameter value *ucpBuf* to set

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
uchar08 odSizeToSet = 5;
// fill "od" buffer with value to set
dev_SetISDUValue(iPortNo, index, subindex, od, odSizeToSet, cpErrMsg));
```

dev_GetStatus (get generic device status)

Function:

```
int16 dev_GetStatus(uint16 uiPortNo, uchar08 *pStatus, uchar08 *ucpPDOutBuf, uint16 uiPDOutBufSize,
```

Description:

You can get current state of IO-Link communication, process data output which master sends and its validity using *dev_GetStatus* function with a pointer to status byte and process data output buffer to return it. Then you can use a *dev_StatusT* structure to extract current IO-Link state of type *dev_StateT* and process data output validity (0 – valid, 1 - invalid). Possible IO-Link states are enumerated in the *iqi_dev.h* header file with a prefix *DEV_STATE_* (see also example).

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uchar08 *pStatus** – pointer to byte where generic device status is encoded
- **uint16 uiPDOutBufSize** – size of process data output buffer *ucpPDOutBuf*

Output Parameters:

- **uchar08 *ucpPDOutBuf** – pointer to buffer to return process data output
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

Actual data bytes number (process data output size) written to *ucpPDOutBuf* or negative error number.

Example:

```
dev_StatusT status;
dev_StateT state;
uchar08 validity;
uchar08 pdOut[32];
dev_GetStatus(iPortNo, (uchar08*)&status, pdOut, 32, cpErrMsg));
state = (dev_StateT)(status.State); // DEV_STATE_INACTIVE, DEV_STATE_SIO, DEV_STATE_STARTUP, DEV_STA
validity = status.PDInInvalid; // 0 - valid, 1 - invalid
```

dev_SetPDIn (set process data input and its validity)

Function:

```
int16 dev_SetPDIn(uint16 uiPortNo, uchar08 *ucpPDInBuf, uint16 uiBytesCount, boolean bValidity, char
```

Description:

You can set current process data input to send to master in IO-Link operate state using *dev_SetPDIn* function with process data input buffer to set, its size and validity flag as arguments.

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uchar08 *ucpPDInBuf** – pointer to buffer with process data input to set
- **uint16 uiBytesCount** – data bytes number in process data input buffer "ucpPDInBuf" to set

- **boolean bValidity** – process data input validity to set (*TRUE* – valid, *FALSE* - invalid)

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
uchar08 pdIn[32];
uchar08 pdInSize = 2;
boolean validity = TRUE; // (TRUE - valid, FALSE - invalid)
// fill "pdIn" buffer with data to set ...
dev_SetPDIn(iPortNo, pdIn, pdInSize, validity, cpErrMsg);
```

dev_SendEvent (send IO-Link event to master)

Function:

```
int16 dev_SendEvent(uint16  uiPortNo, uchar08 ucQualifier, uint16  uiCode, char08  *cpErrMsg);
```

Description:

To send an IO-Link event to master in preoperate or operate state use *dev_SendEvent* function with the event qualifier byte and code as arguments. The event qualifier byte must be encoded according to IO-link specification with the event instance, source, type and mode. It can be constructed using a variable of type *EventQualifierT* and corresponding enumerations from *iqi_dev.h* with prefixes *EVENT_INSTANCE_*, *EVENT_SOURCE_*, *EVENT_TYPE_* and *EVENT_MODE_* (see example and *IO-Link event qualifier*).

Input Parameters:

- **uchar08 uiPortNo** – (virtual) comport number to disconnect obtained previously from *dev_Connect* or *dev_EthernetConnect* functions
- **uchar08 ucQualifier** – qualifier byte of event to send with encoded event attributes according to IO-Link specification
- **uint16 uiCode** – code of event to send

Output Parameters:

- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value:

ERR_NONE or negative error number.

Example:

```
EventQualifierT eventQualifier;
eventQualifier.Byte = 0;
eventQualifier.Fields.Instance = EVENT_INSTANCE_APPL;
eventQualifier.Fields.Source = EVENT_SOURCE_REMOTE;
eventQualifier.Fields.Type = EVENT_TYPE_NOTIFICATION;
eventQualifier.Fields.Mode = EVENT_MODE_SINGLE_SHOT;
uiEventCode = 0x1234;
dev_SendEvent(iPortNo, eventQualifier.Byte, uiEventCode, cpErrMsg);
```

4.2.4 Ethernet connection (tcp/ip)

Startup Ethernet connection

If you are going to use iqInterface over Ethernet/LAN connection you should once startup Ethernet communication beforehand using *enet_Startup* function declared in *iqi_ethernet.h*

```
enet_Startup(cpErrMsg);
```

At the end of Ethernet communication usage you can free OS resources bound to it calling *enet_Cleanup* function also declared in *iqi_ethernet.h*

```
enet_Cleanup(cpErrMsg);
```

comm_EthernetSearch (search iqInterface IP in LAN)

Function

```
int16 comm_EthernetSearch(enet_ConnectionT * pFoundConnections,  uchar08 ucMaxConnectionsNumber,  uch
```

Description

If you do not know the IP address you can try to search it using *comm_EthernetSearch* function declared in *iqi_comm.h*. Call it with a buffer to return found connections of type *enet_ConnectionT*, its size, a pointer to found connections number and send/receive timeout in milliseconds as arguments. The UDP protocol with port number 30718 is used to discover iqInterfaces in LAN. You can use the found connections further in *mst_EthernetConnect* or *dev_EthernetConnect* function to connect to found iqInterface.

Input Parameters

- **uchar08 ucMaxConnectionsNumber** – size of connection buffer *pFoundConnections*

Output parameters:

- **enet_ConnectionT * pFoundConnections** – pointer to buffer to return found connections of type "enet_ConnectionT"
- **uchar08 * ucpFoundConnectionsNumber** – number of connections actually found and saved to buffer "pFoundConnections"
- **uint16 uiSendRecvTimoutMs** – send/receive timeout used for UDP packets to discover iqInterfaces
- **char08 *cpErrMsg** – pointer to buffer to return error message (minimum 256 bytes)

Return value

ERR_NONE or negative error number.

Example

```
enet_ConnectionT connections[5];
uchar08 ucFoundConnectionNumber = 0;
uint16 uiSendRecvTimoutMs = 500; // 0,5 sec
int16 iPortNo;
comm_EthernetSearch(connections, 5, &ucFoundConnectionNumber, uiSendRecvTimoutMs, cpErrMsg);
// inspect found connections: connections[0].cpIqInterfaceIpAddr ...
uiSendRecvTimoutMs = 2000; // 2 sec
iPortNo = mst_EthernetConnect(&(connections[0]), uiSendRecvTimoutMs, cpErrMsg);
// or
iPortNo  = dev_EthernetConnect(&(connections[0]), uiSendRecvTimoutMs, cpErrMsg);
```

enet_ConnectionT (Ethernet connection type)

Fields

- **char08 cpIqInterfaceIpAddr[17]** - ip address of iqInterface TCP connection over ethernet: "xxx.xxx.xxx.xxx" (string in dec format)
- **uint16 uiPort** - port of iqInterface TCP connection over ethernet
- **char08 cpIqInterfaceMacAddr[19]** - MAC address of iqInterface ethernet connection: "HH:HH:HH:HH:HH:HH" (string in hex format)
- **ulong64 ul64SerialNumber** - serial number of iqInterface
- **char08 cpLocalIpAddr[17]** - local (PC) ip address of network adapter in which subnet iqInterface is availiable

4.3 C driver DLL changelog

4.3.1 Version 1.1.0.1 (15.01.2014)

- first version of iqInterface control c dll includes master and generic device functions to control iqInterface over comport with porting example for MS Visual Studio 2010 and NI CVI 2010

4.3.2 Version 1.1.0.2 (25.08.2014)

- switch all size/count parameters' type from 'uchar08' to 'uint16'
- fix ISDU read buffer size error
- fix bug with check of device min cycle time setting
- add ethernet communication
- add iq command send/receive retry in case of wrong checksum or response receive timeout

4.3.3 Version 1.1.0.3 (06.07.2015)

- Generic device: ISDU request notification ISDU response from PC with custom data

CHAPTER 5

Firmware changelog

5.1 Version 1.1.0.1 (04.03.2013)

- first hardware and bootloader launch
- first master stack integration and IO-Link communication

5.2 Version 1.1.0.2 (04.07.2013)

- add IQ Generic Device
- non-volatile memory configuration save

5.3 Version 1.1.0.3 (12.10.2013)

- add first version of bootloader with update and supported modes upgrade functions via RS232 and USB
- change iq communication frame (swap checksum and length byte, set confirm bit 7 in command byte)

5.4 Version 1.1.0.4 (25.11.2013)

- direct mode for RS232

5.5 Version 1.1.0.5 (03.12.2013)

- add USB support for direct mode

5.6 Version 1.1.0.6 (25.08.2014)

- add system command reset for generic device (index 2 subindex 0)
- prohibit adding indeces 0, 1, 2, 3, 0x0C to generic device ISDU structure
- allow switching from operate to auto and check fault to preoperate, because the state is already achieved

- add io-link power supply on/off iq parameter/command
- make Master ISDU read/write requests and generic device send event commands repeatable (idempotent)

5.7 V1.1.0.7 (15.04.2015)

- minor bug fixes

5.8 Version 1.1.0.8 (12.05.2015)

- Bug with EMC error counter reading collision fixed. Bug appeared on COM3 Baudrate when PC EMC_Utility was polling the error counter and therefore broke the IO-Link communication;

5.9 Version 1.1.0.9 (06.07.2015)

- Generic device: ISDU request notification ISDU response from PC with custom data

5.10 Version 1.1.0.10 (21.07.2015)

- master stack: fix bug with next buffered event signaling/confirmation from DL to AL

CHAPTER 6

Troubleshooting

6.1 No connection with iqInterface or USB driver installion fails

If you cannot connect to iqInterface from PC over comport or USB (or USB driver installation fails), try to clean unused comports following the next steps for Windows 7:

- press the start button, right click on “Computer” in the right column of start menu and select “Properties”
- system information window should be opened
- press “Advanced system settings” on left panel
- press “Enviroment variables” button
- add new user enviroment variable with name **devmgr_show_details** and value 1
- add another user enviroment variable with name **devmgr_show_nonpresent_devices** and also value 1
- press ok to close all settings windows
- press “Device Manager” on left panel of system information window
- go to “View” in main menu and activate “Show hidden devices”
- now under “Ports (COM & LPT)” subtree you can see all inactive comports
- right click on each of them and select “uninstall”, also including maybe currently connected iqinterface
- disconnect the iqInterface cable from PC and connect it again
- if neended try to reinstall USB driver from *usb_driver\Windows XP_7_8\CDM20830_Setup.exe* on USB stick shipped with iqInterface