Technologie Management Gruppe
Technologie und Engineering GmbH

Zur Gießerei 10
76227 Karlsruhe

Telefon +49 (0)721-82 806 0
Telefax +49 (0)721-82 806 10
E-Mail info@tmgte.de
Internet www.tmgte.de

TMG USB IO-Link Master V2 - DLL

Reference Manual

Version 2.3.1 7. July 2025

Contents

Introduction

The TMG-USB-IO-Link-Master-V2-DLL contains the functionality to access the functions which are implemented in the TMG IO-Link USB Master V2. The DLL has a standard Windows C-Interface which can be used by nearly all kind of development systems. Note that the interface of the functions uses structures which are packed because they transport the binary content of the USB Master functions. For this reason the structures are not aligned. If the DLL shall be used in environments where an alignment is usual (like Delphi or dotnet), a so called marshalling must be used to adopt these structures.
the function interface is defined in the folowing header files: TMGIOLUSBIF20.h main functions of the DLL TMGIOLBlob.h BLOB support TMGIOLFwUpdate.h firmware update support the header files can be used together with the library file for easy access (early binding) of the DLL.

Module Documentation

2.1 TMG IO-Link USB Master V2 - DLL interface

Modules

- Global Definitions
- Process Data Logging

2.1.1 Detailed Description

```
\brief     USB IO-Link Master V2 - DLL Interface
\file      TMGIOLUSBIF20.h
```

This file contains the interface definitions for the TMG USB IO-Link Master.
The DLL interface is a standard C interface to the DLL for the TMG IO-Link USB Master V2

```
\file
\brief     BLOB-transfer
```

This file contains definitions for reading/writing BLOBs from/to IO-Link-devices.
The DLL interface is a standard C interface to the DLL for the TMG IO-Link USB Master V2

```
\file
\brief     Firmware Update over Io-Link
```

This file contains definitions for executing the IO-Link firmware update procedure
The DLL interface is a standard C interface to the DLL for the TMG IO-Link USB Master V2

2.2 Global Definitions

Modules

- USB interface management
- Port Configuration
- Process Data Handling
- ISDU handling
- Event handling
- Data Storage
- Firmware Update Functions
- BLOB functions
- Statistic Functions
- Transparent Mode

Sensor Status Bit definitions

Some of the functions return a sensor status, which contains some status bits. The following definitions define the different informations which are shown by the status

- #define MASK_SENSORSTATE
- #define BIT_CONNECTED
- #define BIT_PREOPERATE
- #define BIT_WRONGSENSOR
- #define BIT_EVENTAVAILABLE
- #define BIT_PDVALID
- #define BIT_SENSORSTATEKNOWN

Return codes which are used in the library functions.

These return codes define the reaction of the library functions. This doesn't include the error codes which are returned by the IO-Link devices during ISDU access Codes less than zero are reported from the DLL. The commands have not been transmitted to the IO-Link master if these codes occur. Codes from 1 to 100 are reported from the IO-Link master. They occur if a service which has been received from the DLL cannot be executed due to some reason. all other codes are coming from the IO-Link device as defined in the standard

- #define RETURN_FIRMWARE_NOT_COMPATIBLE
- #define RETURN_FUNCTION_CALLEDFROMCALLBACK
- #define RETURN_FUNCTION_DELAYED
- #define RETURN_FUNCTION_NOT_IMPLEMENTED
- #define RETURN_STATE_CONFLICT
- #define RETURN_WRONG_COMMAND
- #define RETURN_WRONG_PARAMETER
- #define RETURN_WRONG_DEVICE
- #define RETURN_NO_EVENT
- #define RETURN_UNKNOWN_HANDLE
- #define RETURN_UART_TIMEOUT
- #define RETURN_CONNECTION_LOST
- #define RETURN_OUT_OF_MEMORY
- #define RETURN_DEVICE_ERROR
- #define RETURN_DEVICE_NOT_AVAILABLE
- #define RETURN_INTERNAL_ERROR
- #define RETURN_OK
- #define RESULT_STATE_CONFLICT
- #define RESULT_NOT_SUPPORTED
- #define RESULT_SERVICE_PENDING
- #define RESULT_WRONG_PARAMETER_STACK
- #define RESULT_ABORT

2.2.1 Detailed Description

These common definitions are used for several functions in the interface.

2.2.2 Macro Definition Documentation

2.2.2.1 MASK_SENSORSTATE

```
#define MASK_SENSORSTATE
```

1= Sensor Found, 0 = Sensor Lost, 2 = Sensor in Preoperate, 0x10 = wrong sensor connected, validation failed

2.2.2.2 BIT_CONNECTED

```
#define BIT_CONNECTED
```

0x01 Sensor is connected and in state OPERATE

2.2.2.3 BIT_PREOPERATE

```
#define BIT_PREOPERATE
```
0x02 Sensor is connected and in state PREOPERATE

2.2.2.4 BIT_WRONGSENSOR

```
#define BIT_WRONGSENSOR
```
0x03 Sensor is connected, but the validation failed, and a WRONG_SENSOR event has been received

2.2.2.5 BIT_EVENTAVAILABLE

```
#define BIT_EVENTAVAILABLE
```
1 means that there are Events to be read, 0 if there is no event

2.2.2.6 BIT_PDVALID

```
#define BIT_PDVALID
```
1 means Process datas are valid, 0 if not

2.2.2.7 BIT_SENSORSTATEKNOWN

```
#define BIT_SENSORSTATEKNOWN
```
1 means State of Sensor is known, 0 if not. (at start of set mode)

2.2.2.8 RETURN_FIRMWARE_NOT_COMPATIBLE

```
#define RETURN_FIRMWARE_NOT_COMPATIBLE
```
the firmware needs a firmware update because some of the functions are not implemented

2.2.2.9 RETURN_FUNCTION_CALLEDFROMCALLBACK

```
#define RETURN_FUNCTION_CALLEDFROMCALLBACK
```
calling a DLL function from inside a callback is not allowed

2.2.2.10 RETURN_FUNCTION_DELAYED

```
#define RETURN_FUNCTION_DELAYED
```
a callback has been defined, so the result may come later with the callback

2.2.2.11 RETURN_FUNCTION_NOT_IMPLEMENTED

```
#define RETURN_FUNCTION_NOT_IMPLEMENTED
```
the function is not implemented in the connected IO-Link Master

2.2.2.12 RETURN_STATE_CONFLICT

```
#define RETURN_STATE_CONFLICT
```
the function cannot be used in the actual state of the IO-Link Master

2.2.2.13 RETURN_WRONG_COMMAND

```
#define RETURN_WRONG_COMMAND
```
a wrong answer to a command has been received from the IO-Link Master

2.2.2.14 RETURN_WRONG_PARAMETER

```
#define RETURN_WRONG_PARAMETER
```

one of the function parameters is invalid

2.2.2.15 RETURN_WRONG_DEVICE

```
#define RETURN_WRONG_DEVICE
```

the device name was wrong or the device which is connected is not supported

2.2.2.16 RETURN_NO_EVENT

```
#define RETURN_NO_EVENT
```

a Read Event was called, but there is no event

2.2.2.17 RETURN_UNKNOWN_HANDLE

```
#define RETURN_UNKNOWN_HANDLE
```

the handle of the function is unknown

2.2.2.18 RETURN_UART_TIMEOUT

```
#define RETURN_UART_TIMEOUT
```

a timeout has been reached because there as no answer to a command

2.2.2.19 RETURN_CONNECTION_LOST

```
#define RETURN_CONNECTION_LOST
```

the USB master has been unplugged during communication

2.2.2.20 RETURN_OUT_OF_MEMORY

```
#define RETURN_OUT_OF_MEMORY
```

no more memory available

2.2.2.21 RETURN_DEVICE_ERROR

```
#define RETURN_DEVICE_ERROR
```

error in accessing the USB device driver

2.2.2.22 RETURN_DEVICE_NOT_AVAILABLE

```
#define RETURN_DEVICE_NOT_AVAILABLE
```

the device is not available at this moment

2.2.2.23 RETURN_INTERNAL_ERROR

```
#define RETURN_INTERNAL_ERROR
```

internal library error. Please restart the program

2.2.2.24 RETURN_OK

```
#define RETURN_OK
```

sucessful end of the function

2.2.2.25 RESULT_STATE_CONFLICT

`#define RESULT_STATE_CONFLICT`
the command is not applicable in the actual state

2.2.2.26 RESULT_NOT_SUPPORTED

`#define RESULT_NOT_SUPPORTED`
the command is not supported on this device

2.2.2.27 RESULT_SERVICE_PENDING

`#define RESULT_SERVICE_PENDING`
a Service is pending. A new service must wait for the end of the pending service

2.2.2.28 RESULT_WRONG_PARAMETER_STACK

`#define RESULT_WRONG_PARAMETER_STACK`
a parameter has been rejected by the USB master

2.2.2.29 RESULT_ABORT

`#define RESULT_ABORT`
a service has been aborted

2.3 USB interface management

Data Structures

- struct TDeviceIdentification
- struct TMasterInfo
- struct TDllInfo
- struct TDLLCallbacks
- struct THardwareInfo

Functions

- LONG __stdcall IOL_Create (char *Device)
- LONG __stdcall IOL_Destroy (LONG Handle)
- LONG __stdcall IOL_GetUSBDevices (TDeviceIdentification *pDeviceList, LONG MaxNumberOfEntries)
- LONG __stdcall IOL_GetMasterInfo (LONG Handle, TMasterInfo *pMasterInfo)
- LONG __stdcall IOL_GetDLLInfo (TDllInfo *pDllInfo)
- LONG __stdcall IOL_SetCallbacks (LONG Handle, TDLLCallbacks *pDLLCallbacks)
- LONG __stdcall IOL_GetHWInfo (LONG Handle, THardwareInfo *pInfo)

2.3.1 Detailed Description

These functions are used to manage the access to USB devices. There are function to list all connected devices, and to connect or disconnect to a special device.

2.3.2 Function Documentation

2.3.2.1 IOL_Create()

```
LONG __stdcall IOL_Create (
            char * Device )

  \brief     Creates and initializes the communication port and handle
```

This function opens the referred COM Port and initializes the internal Datastructures. If the return value is greater than 0 it is the Handle by which the connected Master and its structures are referenced. It shall be used with further calls to functions in this Library.

Parameters

Device	COM Port to open as string (e.g. COM1, COM2, ...)

Return values

RETURN_DEVICE_NOT_AVAILABLE	the Device referred by the string parameter "Device" is not available or busy
RETURN_COMM_TIMEOUT	the device did not respond in time
RETURN_OUT_OF_MEMORY	no more Handles can be assigned
RETURN_WRONG_PARAMETER	the device name was wrong
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed
RETURN_FIRMWARE_NOT_-COMPATIBLE	the firmware needs a firmware update because some of the functions are not implemented

Returns

if greater than 0 the returnvalue is a Handle

2.3.2.2 IOL_Destroy()

```
LONG __stdcall IOL_Destroy (
            LONG Handle )

  \brief     Closes the communication port and discards the Handle
```

This function closes the COM Port referred by the Handle. And also frees all the Memory coresponding to the Handle.

Note

This function has to be called, once the Programm using this DLL is about to terminate. Otherwise, when not unloading the DLL one might risk an OUT_OF_MEMORY error.

Parameters

Handle	Handle to work on/with

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.3.2.3 IOL_GetUSBDevices()

```
LONG __stdcall IOL_GetUSBDevices (
            TDeviceIdentification * pDeviceList,
            LONG MaxNumberOfEntries )

  \brief    Looks for USB devices which are plugged into the PC
```

This function looks for USB IO-Link masters in the windows device manager and returns a list of these devices. The information which is achieved from the device manager contains name and product informaion of the device.

Note

The memory containing the resulting list must be allocated by the applicaation. The library cannot check if the size is big enough, therefore the application must ensure the size

Parameters

pDeviceList	pointer to a buffer for the result
MaxNumberOfEntries	max number of entries which can be put in the buffer

Return values

number	of USB devices which are found

2.3.2.4 IOL_GetMasterInfo()

```
LONG __stdcall IOL_GetMasterInfo (
            LONG Handle,
            TMasterInfo * pMasterInfo )

  \brief    Get information from the USB Gateway Module
```

This function gets version and type information from the USB Gateway Module. The module type is contained in the Version string (Standard or Development Version).

Parameters

Handle	Handle to work on/with

Parameters

pMasterInfo	Pointer to TMasterinfo structure

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.3.2.5 IOL_GetDLLInfo()

```
LONG __stdcall IOL_GetDLLInfo (
            TDllInfo * pDllInfo )

  \brief     Get information about the DLL
```

This function returns the Version information from the DLL

Parameters

pDllInfo	Pointer to TDllInfo structure

Return values

RETURN_OK	Everything worked out allright

2.3.2.6 IOL_SetCallbacks()

```
LONG __stdcall IOL_SetCallbacks (
            LONG Handle,
            TDLLCallbacks * pDLLCallbacks )

  \brief     set the callbacks for a connection
```

This function sets the callbacks for a given connection handle.

Parameters

Handle	Handle to work on/with
pDLLCallbacks	Pointer to the list of callback functions

Return values

RETURN_OK	Everything worked out allright

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.3.2.7 IOL_GetHWInfo()

```
LONG __stdcall IOL_GetHWInfo (
            LONG Handle,
            THardwareInfo * pInfo )

  \brief     gets actual hardware information
```

This function retreives some hardware information of the actual connected master

Parameters

Handle	Handle to work on/with
pInfo	Pointer to the resulting container for the information

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_NOT_-IMPLEMENTED	Everything worked out allright

2.4 Port Configuration

Data Structures

- struct TPortConfiguration
- struct TInfo
- struct TInfoEx

Functions

- LONG __stdcall IOL_SetPortConfig (LONG Handle, DWORD Port, TPortConfiguration *pConfig)
- LONG __stdcall IOL_GetPortConfig (LONG Handle, DWORD Port, TPortConfiguration *pConfig)
- LONG __stdcall IOL_GetMode (LONG Handle, DWORD Port, TInfo *pInfo)
- LONG __stdcall IOL_SetCommand (LONG Handle, DWORD Port, DWORD Command)
- LONG __stdcall IOL_GetSensorStatus (LONG Handle, DWORD Port, DWORD *Status)
- LONG __stdcall IOL_GetModeEx (LONG Handle, DWORD Port, TInfoEx *pInfoEx, BOOL OnlyStatus)

PortModus port modi which are used for TargetMode in SetPortConfig

- #define SM_MODE_RESET
- #define SM_MODE_IOLINK_PREOP
- #define SM_MODE_SIO_INPUT
- #define SM_MODE_SIO_OUTPUT
- #define SM_MODE_IOLINK_PREOP_FALLBACK
- #define SM_MODE_IOLINK_OPER_FALLBACK
- #define SM_MODE_IOLINK_OPERATE
- #define SM_MODE_IOLINK_FALLBACK

Port Commands. Commands which are used to switch the actual state of a port via the function SM_SetCommand.

Note that not all state changes are allowed at any time

- #define SM_COMMAND_FALLBACK
- #define SM_COMMAND_PD_OUT_VALID
- #define SM_COMMAND_PD_OUT_INVALID
- #define SM_COMMAND_OPERATE
- #define SM_COMMAND_RESTART

Port Mode details for SIO output mode.

These values define the mode of a digital output.

- #define SM_MODE_SIO_PP_SWITCH
- #define SM_MODE_SIO_HS_SWITCH
- #define SM_MODE_SIO_LS_SWITCH

Port Mode details for SIO input mode.

These values define the mode of a digital input.

- #define SM_MODE_NORMAL_INPUT
- #define SM_MODE_DIAGNOSTIC_INPUT
- #define SM_MODE_INVERT_INPUT
- #define SM_MODE_SIO_TYPE_2

Validation Mode, used in SetPortConfig.

These values define the validation mode.

- #define SM_VALIDATION_MODE_NONE
- #define SM_VALIDATION_MODE_COMPATIBLE
- #define SM_VALIDATION_MODE_IDENTICAL

Commands which are used in the DSConfigure parameter in SetPortConfig.

These values define the behavior of the parameter server. The values can be combined. If the data storage shall be enabled, the bit DS_CFG_ENABLED shall be set. If (in addition) the automatic upload mode shall be used, the bit DS_CFG_UPLOAD_ENABLED shall be set in addition to DS_CFG_ENABLED. if the data storage shall not be used, use the value DS_CFG_DISABLED.

- #define DS_CFG_DISABLED
- #define DS_CFG_ENABLED
- #define DS_CFG_UPLOAD_ENABLED

Baud rates. Speed of the connection if it's established

- #define SM_BAUD_19200
- #define SM_BAUD_38400
- #define SM_BAUD_230400

SensorStateDefinitions for TInfo

The SensorState in TInfo structure is different from other state definitions. This is due to historical use of this function. it will still work, but the functions IOL_GetSensorState and IOL_GetModeEx have some advantages over the function IOL_GetMode the value is only useful for the IO-Link mode. In other modes the state will show always the value STATE_DISCONNECTED_GETMC

- #define STATE_DISCONNECTED_GETMODE
- #define STATE_PREOPERATE_GETMODE
- #define STATE_WRONGSENSOR_GETMODE
- #define STATE_OPERATE_GETMODE

2.4.1 Detailed Description

These functions are used to set the specific mode of an IO-Link port, and to get information about connected sensors.

2.4.2 Macro Definition Documentation

2.4.2.1 SM_MODE_RESET

```
#define SM_MODE_RESET
```

Port is deactivated

2.4.2.2 SM_MODE_IOLINK_PREOP

```
#define SM_MODE_IOLINK_PREOP
```

Port is in IO-Link mode and stops in Preoperate

2.4.2.3 SM_MODE_SIO_INPUT

```
#define SM_MODE_SIO_INPUT
```

Port is in SIO Input mode

2.4.2.4 SM_MODE_SIO_OUTPUT

```
#define SM_MODE_SIO_OUTPUT
```
Port is in SIO Output mode

2.4.2.5 SM_MODE_IOLINK_PREOP_FALLBACK

```
#define SM_MODE_IOLINK_PREOP_FALLBACK
```
io-link to preoperate, fallback allowed

2.4.2.6 SM_MODE_IOLINK_OPER_FALLBACK

```
#define SM_MODE_IOLINK_OPER_FALLBACK
```
io-link to operate, fallback allowed

2.4.2.7 SM_MODE_IOLINK_OPERATE

```
#define SM_MODE_IOLINK_OPERATE
```
Io-Link, but go into operate automatically

2.4.2.8 SM_MODE_IOLINK_FALLBACK

```
#define SM_MODE_IOLINK_FALLBACK
```
io-link to preoperate, then automatically to fallback

2.4.2.9 SM_COMMAND_FALLBACK

```
#define SM_COMMAND_FALLBACK
```
switch Device from IO-Link mode back to SIO

2.4.2.10 SM_COMMAND_PD_OUT_VALID

```
#define SM_COMMAND_PD_OUT_VALID
```
send outputs_valid to device

2.4.2.11 SM_COMMAND_PD_OUT_INVALID

```
#define SM_COMMAND_PD_OUT_INVALID
```
send outputs_invalid to device

2.4.2.12 SM_COMMAND_OPERATE

```
#define SM_COMMAND_OPERATE
```
switch from preoperate to operate state

2.4.2.13 SM_COMMAND_RESTART

```
#define SM_COMMAND_RESTART
```
restart the connection

2.4.2.14 SM_MODE_SIO_PP_SWITCH

```
#define SM_MODE_SIO_PP_SWITCH
```
Digital output works in Push/Pull mode

2.4.2.15 SM_MODE_SIO_HS_SWITCH

```
#define SM_MODE_SIO_HS_SWITCH
```

Digital output works as High Side Switch

2.4.2.16 SM_MODE_SIO_LS_SWITCH

```
#define SM_MODE_SIO_LS_SWITCH
```

Digital output works as Low Side Switch

2.4.2.17 SM_MODE_NORMAL_INPUT

```
#define SM_MODE_NORMAL_INPUT
```

Digital input works as a normal input

2.4.2.18 SM_MODE_DIAGNOSTIC_INPUT

```
#define SM_MODE_DIAGNOSTIC_INPUT
```

Digital input works as a diagnostic input

2.4.2.19 SM_MODE_INVERT_INPUT

```
#define SM_MODE_INVERT_INPUT
```

Digital input works as a inverted input

2.4.2.20 SM_MODE_SIO_TYPE_2

```
#define SM_MODE_SIO_TYPE_2
```

this bit can be set in addition to the mode to run the Port in IEC Type 2. The default of the SIO is Type 1

2.4.2.21 SM_VALIDATION_MODE_NONE

```
#define SM_VALIDATION_MODE_NONE
```

no validation, each combination of device and vendor id is allowed

2.4.2.22 SM_VALIDATION_MODE_COMPATIBLE

```
#define SM_VALIDATION_MODE_COMPATIBLE
```

device and vendor ID will be checked

2.4.2.23 SM_VALIDATION_MODE_IDENTICAL

```
#define SM_VALIDATION_MODE_IDENTICAL
```

device and vendor ID and the serial number will be checked

2.4.2.24 DS_CFG_DISABLED

```
#define DS_CFG_DISABLED
```

the data storage mechanism is disabled

2.4.2.25 DS_CFG_ENABLED

```
#define DS_CFG_ENABLED
```

the data storage is enabled.

2.4.2.26 DS_CFG_UPLOAD_ENABLED

```
#define DS_CFG_UPLOAD_ENABLED
```
the automatical upload is enabled

2.4.2.27 SM_BAUD_19200

```
#define SM_BAUD_19200
```
speed of the connection is 19200 baud

2.4.2.28 SM_BAUD_38400

```
#define SM_BAUD_38400
```
speed of the connection is 38400 baud

2.4.2.29 SM_BAUD_230400

```
#define SM_BAUD_230400
```
speed of the connection is 230400 baud

2.4.2.30 STATE_DISCONNECTED_GETMODE

```
#define STATE_DISCONNECTED_GETMODE
```
there is no device connected

2.4.2.31 STATE_PREOPERATE_GETMODE

```
#define STATE_PREOPERATE_GETMODE
```
the connection is still in PREOPERATE state

2.4.2.32 STATE_WRONGSENSOR_GETMODE

```
#define STATE_WRONGSENSOR_GETMODE
```
a wrong device has been conntected. This may appear if the validation mode is set

2.4.2.33 STATE_OPERATE_GETMODE

```
#define STATE_OPERATE_GETMODE
```
the connection has been established

2.4.3 Function Documentation

2.4.3.1 IOL_SetPortConfig()

```
LONG __stdcall IOL_SetPortConfig (
            LONG Handle,
            DWORD Port,
            TPortConfiguration * pConfig )

  \brief     Sets the Mode according to the Parameters
```

This function sets the Port on the USB IO-Link Master Gateway to the desired Mode, specified by the parameters of pConfig.

- TargetMode defines the mode of the port which is used. possible Values are:
  - SM_MODE_RESET Port is deactivated
  - SM_MODE_IOLINK_PREOP Port is in IO-Link mode and stops in Preoperate
  - SM_MODE_SIO_INPUT Port is in SIO Input mode
  - SM_MODE_SIO_OUTPUT Port is in SIO Output mode
  - SM_MODE_IOLINK_PREOP_FALLBACK io-link to preoperate, fallback allowed
  - SM_MODE_IOLINK_OPER_FALLBACK io-link to operate, fallback allowed
  - SM_MODE_IOLINK_OPERATE Io-Link, but go into operate automatically
  - SM_MODE_IOLINK_FALLBACK io-link to preoperate, then automatically to fallback
- PortModeDetails sets additional information for the port mode. The content depends on the TargetMode:
  - in IO-Link Modes SM_MODE_IOLINK_xxx the value contains the cycle time. The format of the cycle time is defined in the IO-Link specification. A value of 0 means "free running" mode where the maximum of (min cycle time of the device and min cycle time of the master) will be used as the real cycle time.
  - in SM_MODE_SIO_INPUT the value defines the behavior of the input value. Possible values are:
    - SM_MODE_NORMAL_INPUT Digital input works as a normal input
    - SM_MODE_DIAGNOSTIC_INPUT Red if Open, diagnostic input
    - SM_MODE_INVERT_INPUT Digital input works as a inverted input
  - in SM_MODE_SIO_OUTPUT the value defines the physical mode of the output circuit
    - SM_MODE_SIO_PP_SWITCH Digital output works in Push/Pull mode
    - SM_MODE_SIO_HS_SWITCH Digital output works as High Side Switch
    - SM_MODE_SIO_LS_SWITCH Digital output works as Low Side Switch
- CRID defines the Configured revision ID. This Value defines the IO-Link version which will be used to communicate. If the sensor does not support this version, the connection will fail. Possible values are:
  - 0x11 The Port will be used in V11 Mode. Devices based on Specification 1.0 will accessed with V1.0 Frames. Devices based on V1.1 Spec will be accessed with V1.1 Frames.
  - 0x10 The Port will run in V10 Mode. Devices based on V11 Specification will be automatically switched to V10 if they are capable to do this.
  - 0 there will be no check against the revision number. both V10 and V11 devices with the same vendor and device ID will be connected. However, if Data storage is enabled, the revision numer must be set to 0x11. Otherwise the service will be respond with WRONG_PARAMETER, because only V11 devices support the data storage.
- DSConfigure configuration of the Data storage. The Values can be combined. Possible values are:
  - DS_CFG_ENABLED defines that the data storage is enabled.
  - DS_CFG_UPLOAD_ENABLED defines that the automatically upload is enabled. If not set, the Upload must be done manually.

- InspectionLevel defines the amount of validation which is done at connecting to the device. If one of the validation parameters does not match the parameters in the device, the connection will fail
  - SM_VALIDATION_MODE_NONE there is no validation. Each device can be connected without validating anything. The parameters VendorID, DeviceID and SerialNumber can be left empty
  - SM_VALIDATION_MODE_COMPATIBLE defines the mode where a given device can be exchanged with a device of the same type. the Parameter VendorID and DeviceID must be set and are checked against the parameters of the device. If the device matches these values (this includes compatible devices which can switch the device ID), the connection will be successful. Otherwise it will fail.
  - SM_VALIDATION_MODE_IDENTICAL defines the mode where the exact device will be checked which is connected. All Parameters VendorID, DeviceID and SerialNumber will be checked. Of course the device has to support the Parameter SerialNumber which is not mandatory.
- InputLength defines the input data length of the application. The normal value is 32 so that each device can be connected. If the application doesn't support 32 byte, it can reduce this. If the device needs more data, the connection will fail
- OutputLength defines the output data length of the application. The normal value is 32 so that each device can be connected. If the application doesn't support 32 byte, it can reduce this. If the device needs more data, the connection will fail

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pConfig	pointer to the data structure containing the data

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.4.3.2 IOL_GetPortConfig()

```
LONG __stdcall IOL_GetPortConfig (
            LONG Handle,
            DWORD Port,
            TPortConfiguration * pConfig )

  \brief    Reads out the actual port configuration
```

This function reads out the actual Port configuration for a given Port

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pConfig	pointer to the data structure containing the data

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.4.3.3 IOL_GetMode()

```
LONG __stdcall IOL_GetMode (
              LONG Handle,
              DWORD Port,
              TInfo * pInfo )

  \brief     Gets the current Mode
```

This function gets the current state and Mode information of the Port on the USB IO-Link Master Gateway. The result will be stored in the data structure pointed to by the parameter pInfo.

- COM contains the Device name of the IO-Link Master (such as "COM3")
- DeviceID contains the device ID of the connected device
- VendorID contains the Vendor ID of the connected device
- FunctionID contains the Function ID of the connected device
- ActualMode is the actual running mode of the port. the values are a subset of the values used by SetPortConfig:
  - SM_MODE_RESET Port is deactivated
  - SM_MODE_IOLINK_PREOP Port is in IO-Link mode
  - SM_MODE_SIO_INPUT Port is in SIO Input mode
  - SM_MODE_SIO_OUTPUT Port is in SIO Output mode
- SensorState defines the actual state of the sensor:
- MasterCycle defines the actual cycle time which is used in the connection
- CurrentBaudrate defines the actual used baud rate of the connection

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pInfo	Pointer to TInfo structure

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_DELAYED	the answer will be delayed because a callback is defined
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

Warning
This function is depreciated and should not be used anymore. Please use IOL_GetStatus and IOL_GetModeEx instead because they have advantages.

2.4.3.4 IOL_SetCommand()

```
LONG __stdcall IOL_SetCommand (
             LONG Handle,
             DWORD Port,
             DWORD Command )

  \brief     Send Command to the IO-Link Master
```

This function sends a command out of a predefined list of commands. These commands are transmitted to the sensor. Possible values for the Command are:

- SM_COMMAND_FALLBACK switch Device from IO-Link mode back to SIO
- SM_COMMAND_PD_OUT_VALID send outputs_valid to device
- SM_COMMAND_PD_OUT_INVALID send outputs_invalid to device
- SM_COMMAND_OPERATE switch from preoperate to operate state

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
Command	Pointer to TMasterinfo structure

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.4.3.5 IOL_GetSensorStatus()

```
LONG __stdcall IOL_GetSensorStatus (
            LONG Handle,
            DWORD Port,
            DWORD * Status )

  \brief     Return the current Sensor Status
```

This function will return the current Sensorstatus. It will write the same bits to the Variable Status as it is written in Processdata Exchange.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
Status	Status information (Events, Processdata Valid, ...)

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.4.3.6 IOL_GetModeEx()

```
LONG __stdcall IOL_GetModeEx (
            LONG Handle,
            DWORD Port,
            TInfoEx * pInfoEx,
            BOOL OnlyStatus )

  \brief     Gets the current Mode
```

This function gets the current state and Mode information of the Port on the USB IO-Link Master Gateway. The result will be stored in the data structure pointed to by the parameter pInfoEx.

- COM contains the Device name of the IO-Link Master (such as "COM3")
- DirectParameterPage contains the complete DPP1 of the device if OnlyStatus was false
- ActualMode is the actual running mode of the port. the values are a subset of the values used by SetPortConfig:
  - SM_MODE_RESET Port is deactivated
  - SM_MODE_IOLINK_PREOP Port is in IO-Link mode

  - SM_MODE_SIO_INPUT Port is in SIO Input mode
  - SM_MODE_SIO_OUTPUT Port is in SIO Output mode

- SensorState defines the actual state of the sensor:
- CurrentBaudrate defines the actual used baud rate of the connection

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pInfoEx	Pointer to TInfo structure
OnlyStatus	

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_DELAYED	the answer will be delayed because a callback is defined
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.5 Process Data Handling

Functions

- LONG __stdcall IOL_ReadOutputs (LONG Handle, DWORD Port, BYTE *ProcessData, DWORD *Length, DWORD *Status)
- LONG __stdcall IOL_ReadInputs (LONG Handle, DWORD Port, BYTE *ProcessData, DWORD *Length, DWORD *Status)
- LONG __stdcall IOL_WriteOutputs (LONG Handle, DWORD Port, BYTE *ProcessData, DWORD Length)
- LONG __stdcall IOL_TransferProcessData (LONG Handle, DWORD Port, BYTE *ProcessDataOut, DWORD LengthOut, BYTE *ProcessDataIn, DWORD *LengthIn, DWORD *Status)

2.5.1 Detailed Description

These functions are used to get and set process data. In addition the data loggin can be activated and deactivated.

2.5.2 Function Documentation

2.5.2.1 IOL_ReadOutputs()

```
LONG __stdcall IOL_ReadOutputs (
            LONG Handle,
```

```
            DWORD Port,
            BYTE * ProcessData,
            DWORD * Length,
            DWORD * Status )

  \brief    Read-back the Output Process Data written
```

This function reads-back the Process Data written to the Process-Data- Output-Buffer previously with IOL_WriteOutputs.

Parameters

Handle	Handle to work on/with
Port	Port from which to read, 0xFF = ALL Ports
ProcessData	Pointer to write the Process Data to
Length	Length of written Process Data
Status	Status information (Events, Processdata Valid, ...)

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.5.2.2 IOL_ReadInputs()

```
LONG __stdcall IOL_ReadInputs (
            LONG Handle,
            DWORD Port,
            BYTE * ProcessData,
            DWORD * Length,
            DWORD * Status )

  \brief    Read the Input Process Data from the Sensor connected
```

This function reads the Process Data from the USB IO-Link Master Gateway, which was received from the Sensor. for specific port numbers, the structure contains the Length, the data, and a valid information for port 0xFF, which means ALL Ports, first byte is the number of entries. Then the above structure follows Length, data, valid

Parameters

Handle	Handle to work on/with
Port	Port from which to read, 0xFF = ALL Ports
ProcessData	Pointer to write the Process Data to
Length	Length of written Process Data
Status	Status information (Events, Processdata Valid, ...)

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.5.2.3 IOL_WriteOutputs()

```
LONG __stdcall IOL_WriteOutputs (
              LONG Handle,
              DWORD Port,
              BYTE * ProcessData,
              DWORD Length )

  \brief      Write Output Process Data to the USB IO-Link Master
```

This function writes the Process Data refered by ProcessData to the USB IO-Link Master. The data is then transferred to the connected Sensor.

Parameters

Handle	Handle to work on/with
Port	Port from which to read, 0xFF = ALL Ports
ProcessData	Pointer to the Process Data to be written
Length	Length of Process Data

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.5.2.4 IOL_TransferProcessData()

```
LONG __stdcall IOL_TransferProcessData (
              LONG Handle,
              DWORD Port,
              BYTE * ProcessDataOut,
              DWORD LengthOut,
              BYTE * ProcessDataIn,
              DWORD * LengthIn,
              DWORD * Status )

  \brief      Transfers Process Data in both directions
```

This function transfers Process Data in both directions. It first sends out the Processdata referenced by ProcessDataOut. And then receives the response and writes it's content to ProcessDataIn.

Parameters

Handle	Handle to work on/with
Port	Port from which to read and write, 0xFF = ALL Ports
ProcessDataOut	Pointer to read the Process Data from
LengthOut	Length of Process Data to be output
ProcessDataIn	Pointer to write the Process Data to
LengthIn	Length of written Process Data
Status	Status information (Events, Processdata Valid, ...)

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.6 ISDU handling

Data Structures

- struct TParameter

Functions

- LONG __stdcall IOL_ReadReq (LONG Handle, DWORD Port, TParameter *pParameter)
- LONG __stdcall IOL_WriteReq (LONG Handle, DWORD Port, TParameter *pParameter)

2.6.1 Detailed Description

These functions are used to get and set parameter data via ISDU requests.

2.6.2 Function Documentation

2.6.2.1 IOL_ReadReq()

```
LONG __stdcall IOL_ReadReq (
            LONG Handle,
            DWORD Port,
            TParameter * pParameter )

  \brief     Request read on SPDU from the Sensor
```

This function sends a Read Request to the USB IO-Link Master, which passes it to the Device connected. The pParameter struct is used to set the Index and Subindex, that is requested via the SPDU-Channel.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pParameter	Pointer to TParameter struct

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_DELAYED	the answer will be delayed because a callback is defined
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.6.2.2 IOL_WriteReq()

```
LONG __stdcall IOL_WriteReq (
            LONG Handle,
            DWORD Port,
            TParameter * pParameter )

  \brief     Request to write on SPDU to the Sensor
```

This function sends a Write Request to the USB IO-Link Master, which passes it to the Device connected. The pParameter struct is used to set the Index and Subindex, that is requested to be written via the SPDU-Channel. The pParameter struct also contains the Data that will be written.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pParameter	Pointer to TParameter struct

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_DELAYED	the answer will be delayed because a callback is defined

Return values

RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.7 Event handling

Data Structures

- struct TEvent

Functions

- LONG __stdcall IOL_ReadEvent (LONG Handle, TEvent *pEvent, DWORD *Status)

Event definitions

These values define the content of the event buffer

- #define EVNT_INST_UNKNOWN
- #define EVNT_INST_PHL
- #define EVNT_INST_DL
- #define EVNT_INST_AL
- #define EVNT_INST_APPL
- #define EVNT_TYPE_ERROR
- #define EVNT_TYPE_WARNING
- #define EVNT_TYPE_MESSAGE
- #define EVNT_MODE_SINGLE
- #define EVNT_MODE_COMING
- #define EVNT_MODE_GOING
- #define EVNT_CODE_M_PDU_CHECK
- #define EVNT_CODE_S_DEVICELOST
- #define EVNT_CODE_S_WRONGSENSOR
- #define EVNT_CODE_S_RETRY
- #define EVNT_CODE_P_SHORT
- #define EVNT_CODE_P_SENSOR
- #define EVNT_CODE_P_ACTOR
- #define EVNT_CODE_P_POWER
- #define EVNT_CODE_P_RESET
- #define EVNT_CODE_S_FALLBACK
- #define EVNT_CODE_M_PREOPERATE
- #define EVNT_CODE_DSREADY_NOACTION
- #define DS_FAULT_IDENT
- #define DS_FAULT_SIZE
- #define DS_FAULT_UPLOAD
- #define DS_FAULT_DOWNLOAD
- #define DS_FAULT_DEVICE_LOCKED
- #define EVNT_CODE_DSREADY_DOWNLOAD
- #define EVNT_CODE_DSREADY_UPLOAD

- #define EVNT_CODE_S_WRONG_PDINLENGTH
- #define EVNT_CODE_S_WRONG_PDOUTLENGTH
- #define EVNT_CODE_S_WRONG_REVISION
- #define EVNT_CODE_S_WRONG_COMP_VENDORID
- #define EVNT_CODE_S_WRONG_COMP_DEVICEID
- #define EVNT_CODE_S_WRONG_COMP10_VENDORID
- #define EVNT_CODE_S_WRONG_COMP10_DEVICEID
- #define EVNT_CODE_S_WRONG_SERNUM
- #define EVNT_CODE_S_WRONG_CYCLE

2.7.1 Detailed Description

These functions are used to handle the device events.

2.7.2 Macro Definition Documentation

2.7.2.1 EVNT_INST_UNKNOWN

`#define EVNT_INST_UNKNOWN`
instance is unknown

2.7.2.2 EVNT_INST_PHL

`#define EVNT_INST_PHL`
instance physical layer

2.7.2.3 EVNT_INST_DL

`#define EVNT_INST_DL`
instance data layer

2.7.2.4 EVNT_INST_AL

`#define EVNT_INST_AL`
instance Application Layer

2.7.2.5 EVNT_INST_APPL

`#define EVNT_INST_APPL`
instance Application

2.7.2.6 EVNT_TYPE_ERROR

`#define EVNT_TYPE_ERROR`
event shows an error

2.7.2.7 EVNT_TYPE_WARNING

`#define EVNT_TYPE_WARNING`
event shows a warning

2.7.2.8 EVNT_TYPE_MESSAGE

`#define EVNT_TYPE_MESSAGE`
event shows a Message

2.7.2.9 EVNT_MODE_SINGLE

`#define EVNT_MODE_SINGLE`
event shows a single message or warning

2.7.2.10 EVNT_MODE_COMING

`#define EVNT_MODE_COMING`
event shows that an error has appeared

2.7.2.11 EVNT_MODE_GOING

`#define EVNT_MODE_GOING`
event shows that an error has disappeared

2.7.2.12 EVNT_CODE_M_PDU_CHECK

`#define EVNT_CODE_M_PDU_CHECK`
a frame with a CRC error has been received

2.7.2.13 EVNT_CODE_S_DEVICELOST

`#define EVNT_CODE_S_DEVICELOST`
Device has been disconnected: coming: line break going: device is in operate

2.7.2.14 EVNT_CODE_S_WRONGSENSOR

`#define EVNT_CODE_S_WRONGSENSOR`
a wrong sensor has been detected. Unspecific error. The normal case is code 64-72

2.7.2.15 EVNT_CODE_S_RETRY

`#define EVNT_CODE_S_RETRY`
Retries have been detected

2.7.2.16 EVNT_CODE_P_SHORT

`#define EVNT_CODE_P_SHORT`
a short circuit has been detected on the C/Q line

2.7.2.17 EVNT_CODE_P_SENSOR

`#define EVNT_CODE_P_SENSOR`
there is an error in the Sensor supply

2.7.2.18 EVNT_CODE_P_ACTOR

`#define EVNT_CODE_P_ACTOR`
there is an error in the Actor supply

2.7.2.19 EVNT_CODE_P_POWER

```
#define EVNT_CODE_P_POWER
```

there is an error in the Power Supply of the IO-Link master

2.7.2.20 EVNT_CODE_P_RESET

```
#define EVNT_CODE_P_RESET
```

an event is send if a port has been resetted

2.7.2.21 EVNT_CODE_S_FALLBACK

```
#define EVNT_CODE_S_FALLBACK
```

fallback has been done successful, device is back in SIO state

2.7.2.22 EVNT_CODE_M_PREOPERATE

```
#define EVNT_CODE_M_PREOPERATE
```

device has reached the preoperate state

2.7.2.23 EVNT_CODE_DSREADY_NOACTION

```
#define EVNT_CODE_DSREADY_NOACTION
```

data storage come to the end, but there os no action, because the CRC was correct

2.7.2.24 DS_FAULT_IDENT

```
#define DS_FAULT_IDENT
```

the sensor doesn't match the content in the data storage

2.7.2.25 DS_FAULT_SIZE

```
#define DS_FAULT_SIZE
```

the sensor parameters doesn't fit in the memory of the data storage

2.7.2.26 DS_FAULT_UPLOAD

```
#define DS_FAULT_UPLOAD
```

error in uploading the data storage

2.7.2.27 DS_FAULT_DOWNLOAD

```
#define DS_FAULT_DOWNLOAD
```

error in downloading the data storage

2.7.2.28 DS_FAULT_DEVICE_LOCKED

```
#define DS_FAULT_DEVICE_LOCKED
```

error in data storage function because the device is locked

2.7.2.29 EVNT_CODE_DSREADY_DOWNLOAD

```
#define EVNT_CODE_DSREADY_DOWNLOAD
```

the parameter download has come to the end

2.7.2.30 EVNT_CODE_DSREADY_UPLOAD

```
#define EVNT_CODE_DSREADY_UPLOAD
```

the parameter upload has come to the end

2.7.2.31 EVNT_CODE_S_WRONG_PDINLENGTH

```
#define EVNT_CODE_S_WRONG_PDINLENGTH
```

process data input length don't match

2.7.2.32 EVNT_CODE_S_WRONG_PDOUTLENGTH

```
#define EVNT_CODE_S_WRONG_PDOUTLENGTH
```

process data output length don't match

2.7.2.33 EVNT_CODE_S_WRONG_REVISION

```
#define EVNT_CODE_S_WRONG_REVISION
```

device revision doesn't match

2.7.2.34 EVNT_CODE_S_WRONG_COMP_VENDORID

```
#define EVNT_CODE_S_WRONG_COMP_VENDORID
```

vendor id is wrong V1.1 sensor

2.7.2.35 EVNT_CODE_S_WRONG_COMP_DEVICEID

```
#define EVNT_CODE_S_WRONG_COMP_DEVICEID
```

device id is wrong V1.1 sensor

2.7.2.36 EVNT_CODE_S_WRONG_COMP10_VENDORID

```
#define EVNT_CODE_S_WRONG_COMP10_VENDORID
```

vendor id is wrong V1.0 sensor

2.7.2.37 EVNT_CODE_S_WRONG_COMP10_DEVICEID

```
#define EVNT_CODE_S_WRONG_COMP10_DEVICEID
```

device id is wrong V1.0 sensor

2.7.2.38 EVNT_CODE_S_WRONG_SERNUM

```
#define EVNT_CODE_S_WRONG_SERNUM
```

serial number is wrong

2.7.2.39 EVNT_CODE_S_WRONG_CYCLE

```
#define EVNT_CODE_S_WRONG_CYCLE
```

cycle time not matching

2.7.3 Function Documentation

2.7.3.1 IOL_ReadEvent()

```
LONG __stdcall IOL_ReadEvent (
              LONG Handle,
              TEvent * pEvent,
              DWORD * Status )

  \brief     Get the last event out of the Event Buffer
```

This function gets the most next Event out of the internal FIFO Buffer. The DLL stores occuring events in an internal FIFO Buffer with enough Space for 10 Events. If this function doesn't get called after 10 Events the last Event will be overridden.

Parameters

Handle	Handle to work on/with
pEvent	Pointer to a TEvent struct
Status	Status information (Events, Processdata Valid, ...)

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.8 Data Storage

Functions

- LONG __stdcall IOL_DS_Command (LONG Handle, DWORD Port, DWORD DSCommand)
- LONG __stdcall IOL_DS_ContentGet (LONG Handle, DWORD Port, BYTE *pDSContentData, DWORD *pDSContentLength)
- LONG __stdcall IOL_DS_ContentSet (LONG Handle, DWORD Port, BYTE *pDSContentData, DWORD DSContentLength)

Commands which are used in IOL_DS_Command.

These commands are used to activate data storage commands.

- #define DS_CMD_UPLOAD
- #define DS_CMD_DOWNLOAD
- #define DS_CMD_CLEAR

2.8.1 Detailed Description

These functions are used to handle the data storage commands.

2.8.2 Macro Definition Documentation

2.8.2.1 DS_CMD_UPLOAD

```
#define DS_CMD_UPLOAD
```

upload parameter-set

2.8.2.2 DS_CMD_DOWNLOAD

```
#define DS_CMD_DOWNLOAD
```

download current parameter-set

2.8.2.3 DS_CMD_CLEAR

```
#define DS_CMD_CLEAR
```

clear stored parameter set

2.8.3 Function Documentation

2.8.3.1 IOL_DS_Command()

```
LONG __stdcall IOL_DS_Command (
             LONG Handle,
             DWORD Port,
             DWORD DSCommand )

  \brief     sends a data storage command
```

This function sends a data storage command to the data storage for a given port. The command is set in the parameter DSCommand and can contain the following values:
DS_CMD_UPLOAD starts an upload from the device DS_CMD_DOWNLOAD starts a download to the device DS_CMD_CLEAR clears the content of the data storage

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
DSCommand	Command which shall be sent to the data storage

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.8.3.2 IOL_DS_ContentGet()

```
LONG __stdcall IOL_DS_ContentGet (
              LONG Handle,
              DWORD Port,
              BYTE * pDSContentData,
              DWORD * pDSContentLength )

  \brief     Reads out the content of the data storage
```

This function reads the data storage buffer of the IO-Link master for a given port.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pDSContentData	pointer to a buffer for the content of the data storage
pDSContentLength	pointer to the length. Must be initialized with the size of the buffer

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.8.3.3 IOL_DS_ContentSet()

```
LONG __stdcall IOL_DS_ContentSet (
              LONG Handle,
              DWORD Port,
              BYTE * pDSContentData,
              DWORD DSContentLength )

  \brief     Writes the content of a data storage to the IO-Link master
```

This function writes a buffer to the data storage of the IO-Link master.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pDSContentData	pointer to a buffer for the content of the data storage
DSContentLength	length of the buffer which shall be written

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid

Return values

RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.9 Firmware Update Functions

Data Structures

- struct TFWUpdateState
- struct TFwUpdateInfo

Functions

- LONG __stdcall IOL_FwUpdateAbort (LONG Handle, DWORD Port, TFWUpdateState *pUpdateState)
- LONG __stdcall IOL_FwUpdateStart (LONG Handle, DWORD Port, TFwUpdateInfo *pFwUpdateInfo, TFWUpdateState *pUpdateState)
- LONG __stdcall IOL_FwUpdateStartByMetafile (LONG Handle, DWORD Port, char *pFileName, TFwUpdateInfo *pFwUpdateInfo, TFWUpdateState *pUpdateState)
- LONG __stdcall IOL_FwUpdateContinue (LONG Handle, DWORD Port, char *pPassword, TFWUpdateState *pUpdateState)

Return codes which are used in the library functions.

These return codes define the reaction of the firmware update library functions. Codes less than zero are reported from the DLL. please see the codes RETURN_xxx in TMGIOLUSBIF20.h

- #define FWUPDATE_RET_OK
- #define FWUPDATE_RET_ERROR_BUSY
- #define FWUPDATE_ID_WRONG_VENDORID
- #define FWUPDATE_ID_WRONG_REVISION
- #define FWUPDATE_ID_WRONG_HWKEY
- #define FWUPDATE_ID_WRONG_BOOTSTATUS
- #define FWUPDATE_BOOT_MODE_NOT_REACHED
- #define FWUPDATE_RET_ACTIVATION_FAILED
- #define FWUPDATE_RET_BLOB_ERROR
- #define FWUPDATE_RET_XML_ERROR

Firmware update state

these codes define the actual state of the firmware update state machine

- #define FWUPDATE_STATE_IDLE
- #define FWUPDATE_STATE_IDENTIFICATION
- #define FWUPDATE_STATE_VERIFICATION
- #define FWUPDATE_STATE_PASSWORD
- #define FWUPDATE_STATE_SWITCHTOBOOTLOADER

- #define FWUPDATE_STATE_WAITREBOOT
- #define FWUPDATE_STATE_STARTDOWNLOAD
- #define FWUPDATE_STATE_DOWNLOADFIRMWARE
- #define FWUPDATE_STATE_ACTIVATENEWFIRMWARE
- #define FWUPDATE_STATE_WAITACTIVATE
- #define FWUPDATE_STATE_CHECKNEWFIRMWARE
- #define FWUPDATE_STATE_ERROR

2.9.1 Detailed Description

These definitions and functions implement the firmware update functions.

2.9.2 Macro Definition Documentation

2.9.2.1 FWUPDATE_RET_OK

`#define FWUPDATE_RET_OK`
function has been executed successfully

2.9.2.2 FWUPDATE_RET_ERROR_BUSY

`#define FWUPDATE_RET_ERROR_BUSY`
function could not be executed because the state machine is busy

2.9.2.3 FWUPDATE_ID_WRONG_VENDORID

`#define FWUPDATE_ID_WRONG_VENDORID`
the vendor ID doesn't fit to the vendor id in the device

2.9.2.4 FWUPDATE_ID_WRONG_REVISION

`#define FWUPDATE_ID_WRONG_REVISION`
the revision doesn't fit to the revision in the device

2.9.2.5 FWUPDATE_ID_WRONG_HWKEY

`#define FWUPDATE_ID_WRONG_HWKEY`
the hardware key doesn't fit to the hardware key in the device

2.9.2.6 FWUPDATE_ID_WRONG_BOOTSTATUS

`#define FWUPDATE_ID_WRONG_BOOTSTATUS`
the state after booting is not correct

2.9.2.7 FWUPDATE_BOOT_MODE_NOT_REACHED

`#define FWUPDATE_BOOT_MODE_NOT_REACHED`
the boot mode could not be reached

2.9.2.8 FWUPDATE_RET_ACTIVATION_FAILED

`#define FWUPDATE_RET_ACTIVATION_FAILED`
the activation of the new firmware failed

2.9.2.9 FWUPDATE_RET_BLOB_ERROR

```
#define FWUPDATE_RET_BLOB_ERROR
```

there was an error during the download of the firmware

2.9.2.10 FWUPDATE_RET_XML_ERROR

```
#define FWUPDATE_RET_XML_ERROR
```

the xml file is incorrect

2.9.2.11 FWUPDATE_STATE_IDLE

```
#define FWUPDATE_STATE_IDLE
```

before starting or after downloading, the state changes to IDLE

2.9.2.12 FWUPDATE_STATE_IDENTIFICATION

```
#define FWUPDATE_STATE_IDENTIFICATION
```

read our information from device (vendor id, device id, hw id)

2.9.2.13 FWUPDATE_STATE_VERIFICATION

```
#define FWUPDATE_STATE_VERIFICATION
```

verify the data against the metafile information

2.9.2.14 FWUPDATE_STATE_PASSWORD

```
#define FWUPDATE_STATE_PASSWORD
```

optional password step. Must be implemented by calling application

2.9.2.15 FWUPDATE_STATE_SWITCHTOBOOTLOADER

```
#define FWUPDATE_STATE_SWITCHTOBOOTLOADER
```

after verification and password protection we switch the device to boot loader with use of system commands

2.9.2.16 FWUPDATE_STATE_WAITREBOOT

```
#define FWUPDATE_STATE_WAITREBOOT
```

the device shall restart with another device ID. After reconnect, a new verification will be done

2.9.2.17 FWUPDATE_STATE_STARTDOWNLOAD

```
#define FWUPDATE_STATE_STARTDOWNLOAD
```

start the blob download

2.9.2.18 FWUPDATE_STATE_DOWNLOADFIRMWARE

```
#define FWUPDATE_STATE_DOWNLOADFIRMWARE
```

the firmware binary will be downloaded to the device with the BLOB mechanism

2.9.2.19 FWUPDATE_STATE_ACTIVATENEWFIRMWARE

```
#define FWUPDATE_STATE_ACTIVATENEWFIRMWARE
```

last step in firmware update. write system command BM_ACTIVATE to device

2.9.2.20 FWUPDATE_STATE_WAITACTIVATE

```
#define FWUPDATE_STATE_WAITACTIVATE
```

the device shall restart with another device ID. After reconnect, a new verification will be done

2.9.2.21 FWUPDATE_STATE_CHECKNEWFIRMWARE

```
#define FWUPDATE_STATE_CHECKNEWFIRMWARE
```

the device has been restartet, check if a new device id (either the old one or a new due to function differences) is set, and the boot loader status has been changed

2.9.2.22 FWUPDATE_STATE_ERROR

```
#define FWUPDATE_STATE_ERROR
```

state which will be entered in case of any error. can only be left with Abort

2.9.3 Function Documentation

2.9.3.1 IOL_FwUpdateAbort()

```
LONG __stdcall IOL_FwUpdateAbort (
            LONG Handle,
            DWORD Port,
            TFWUpdateState * pUpdateState )

  \brief     aborts a firmware update
```

This function aborts the BLOB-transmission using TMGDLL. BLOB_ID of device will be zero after that.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pUpdateState	pointer to a struct which will be used to return the actual status

Returns

Errorcode

2.9.3.2 IOL_FwUpdateStart()

```
LONG __stdcall IOL_FwUpdateStart (
            LONG Handle,
            DWORD Port,
            TFwUpdateInfo * pFwUpdateInfo,
            TFWUpdateState * pUpdateState )

  \brief     starts a firmware update, parameters are raw data parameters
```

This function reads a BLOB from the device using TMGDLL. Read data is stored in given buffer.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pFwUpdateInfo	pointer to a struct with the information about the update
pUpdateState	pointer to a struct which will be used to return the actual status

2.9.3.3 IOL_FwUpdateStartByMetafile()

```
LONG __stdcall IOL_FwUpdateStartByMetafile (
              LONG Handle,
              DWORD Port,
              char * pFileName,
              TFwUpdateInfo * pFwUpdateInfo,
              TFWUpdateState * pUpdateState )

  \brief     starts a firmware update, parameters are raw data parameters
```

This function starts the firmware update by using the metafile as a parameter.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pFileName	file name of the package file
pFwUpdateInfo	Pointer to a firmware update info struct. will be filled by this function
pUpdateState	pointer to a struct which will be used to return the actual status

2.9.3.4 IOL_FwUpdateContinue()

```
LONG __stdcall IOL_FwUpdateContinue (
              LONG Handle,
              DWORD Port,
              char * pPassword,
              TFWUpdateState * pUpdateState )

  \brief     continues the fw update protocol
```

This function is used to execute the next firmware update step. this approach is used so that the calling application can show progress bar etc, or can abort the update.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pPassword	password which shall be used if the fwRequired is TRUE. this parameter is only evaluated and used in State FWUPDATE_STATE_PASSWORD
pUpdateState	pointer to a struct which will be used to return the actual status

2.10 BLOB functions

Data Structures

- struct TBLOBStatus

Functions

- LONG __stdcall BLOB_Abort (LONG Handle, DWORD Port, TBLOBStatus *pBlobStatus)
- LONG __stdcall BLOB_uploadBLOB (LONG Handle, DWORD Port, LONG target-BLOB_ID, DWORD bufferSize, BYTE *BLOB_buffer, DWORD *lengthRead, TBLOBStatus *pBlobStatus)
- LONG __stdcall BLOB_downloadBLOB (LONG Handle, DWORD Port, LONG target-BLOB_ID, DWORD target_BLOB_size, BYTE *BLOB_data, TBLOBStatus *pBlobStatus)
- LONG __stdcall BLOB_ReadBlobID (LONG Handle, DWORD Port, LONG *blob_id, TBLOBStatus *pBlobStatus)
- LONG __stdcall BLOB_Continue (LONG Handle, DWORD Port, TBLOBStatus *pBlobStatus)

Return values which are used in the BLOB-functions

These return codes define the reaction of the firmware update library functions. positive values incl. 0 come from the blob protocol negative values are results from the DLL. Please see RETURN_xxx from TMGIOLUSBIF20.h for negative values

- #define BLOB_RET_OK
- #define BLOB_RET_ERROR_BUSY
- #define BLOB_RET_ERROR_ISDU_READ
- #define BLOB_RET_ERROR_ISDU_WRITE
- #define BLOB_RET_ERROR_STATECONFLICT
- #define BLOB_RET_ERROR_CHECKBLOBINFO_FAILED
- #define BLOB_RET_ERROR_WRONGCRC
- #define BLOB_RET_ERROR_SIZEOVERRUN
- #define BLOB_RET_ERROR_STOPPED

BLOB statemachine state definition

These codes define the actual state of the BLOB state machine

- #define BLOB_STATE_IDLE
- #define BLOB_STATE_PREPARE_DOWNLOAD
- #define BLOB_STATE_DOWNLOAD
- #define BLOB_STATE_FINALIZE_DOWNLOAD
- #define BLOB_STATE_PREPARE_UPLOAD
- #define BLOB_STATE_UPLOAD
- #define BLOB_STATE_FINALIZE_UPLOAD
- #define BLOB_STATE_ERROR

2.10.1 Detailed Description

These definitions and functions implement the IO-Link BLOB functionality.

2.10.2 Macro Definition Documentation

2.10.2.1 BLOB_RET_OK

`#define BLOB_RET_OK`
successful execution of the command

2.10.2.2 BLOB_RET_ERROR_BUSY

`#define BLOB_RET_ERROR_BUSY`
there is a service pending. it should be aborted or ended before starting a new one

2.10.2.3 BLOB_RET_ERROR_ISDU_READ

`#define BLOB_RET_ERROR_ISDU_READ`
error in ISDU read

2.10.2.4 BLOB_RET_ERROR_ISDU_WRITE

`#define BLOB_RET_ERROR_ISDU_WRITE`
error in ISDU write

2.10.2.5 BLOB_RET_ERROR_STATECONFLICT

`#define BLOB_RET_ERROR_STATECONFLICT`
the function cannot be called in the actual state

2.10.2.6 BLOB_RET_ERROR_CHECKBLOBINFO_FAILED

`#define BLOB_RET_ERROR_CHECKBLOBINFO_FAILED`
there was an error during checking of the BLOB info

2.10.2.7 BLOB_RET_ERROR_WRONGCRC

`#define BLOB_RET_ERROR_WRONGCRC`
the CRC was wrong

2.10.2.8 BLOB_RET_ERROR_SIZEOVERRUN

```
#define BLOB_RET_ERROR_SIZEOVERRUN
```
the size of the BLOB content was too large

2.10.2.9 BLOB_RET_ERROR_STOPPED

```
#define BLOB_RET_ERROR_STOPPED
```
the BLOB has stopped

2.10.2.10 BLOB_STATE_IDLE

```
#define BLOB_STATE_IDLE
```
n BLOB service activated

2.10.2.11 BLOB_STATE_PREPARE_DOWNLOAD

```
#define BLOB_STATE_PREPARE_DOWNLOAD
```
preparation of Download

2.10.2.12 BLOB_STATE_DOWNLOAD

```
#define BLOB_STATE_DOWNLOAD
```
download of the buffer

2.10.2.13 BLOB_STATE_FINALIZE_DOWNLOAD

```
#define BLOB_STATE_FINALIZE_DOWNLOAD
```
finalize the download

2.10.2.14 BLOB_STATE_PREPARE_UPLOAD

```
#define BLOB_STATE_PREPARE_UPLOAD
```
preparation of Upload

2.10.2.15 BLOB_STATE_UPLOAD

```
#define BLOB_STATE_UPLOAD
```
Upload of the buffer

2.10.2.16 BLOB_STATE_FINALIZE_UPLOAD

```
#define BLOB_STATE_FINALIZE_UPLOAD
```
finalize the Upload

2.10.2.17 BLOB_STATE_ERROR

```
#define BLOB_STATE_ERROR
```
error state, can only be left with Abort

2.10.3 Function Documentation

2.10.3.1 BLOB_Abort()

```
LONG __stdcall BLOB_Abort (
              LONG Handle,
              DWORD Port,
              TBLOBStatus * pBlobStatus )

  \brief      aborts the BLOB-transmission
```

This function aborts the BLOB-transmission using TMGDLL. BLOB_ID of device will be zero after that.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pBlobStatus	Pointer to the structure where detailed error-information is put in

2.10.3.2 BLOB_uploadBLOB()

```
LONG __stdcall BLOB_uploadBLOB (
              LONG Handle,
              DWORD Port,
              LONG targetBLOB_ID,
              DWORD bufferSize,
              BYTE * BLOB_buffer,
              DWORD * lengthRead,
              TBLOBStatus * pBlobStatus )

  \brief      reads a BLOB from device
```

This function reads a BLOB from the device using TMGDLL. Read data is stored in given buffer.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
targetBLOB_ID	BLOB-ID to read from
bufferSize	Size of the given read-buffer
BLOB_buffer	Pointer to the read buffer
lengthRead	Pointer to a variable the read length to put
pBlobStatus	Pointer to the structure where detailed error-information is put in

2.10.3.3 BLOB_downloadBLOB()

```
LONG __stdcall BLOB_downloadBLOB (
              LONG Handle,
              DWORD Port,
              LONG targetBLOB_ID,
              DWORD target_BLOB_size,
              BYTE * BLOB_data,
              TBLOBStatus * pBlobStatus )

  \brief     writes a BLOB to device
```

This function writes data to the device using BLOB-mechanism and the TMGDLL.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
targetBLOB_ID	BLOB-ID to write to
target_BLOB_size	Size of the given data
BLOB_data	Pointer to the data to write to BLOB
pBlobStatus	Pointer to the structure where detailed error-information is put in

2.10.3.4 BLOB_ReadBlobID()

```
LONG __stdcall BLOB_ReadBlobID (
              LONG Handle,
              DWORD Port,
              LONG * blob_id,
              TBLOBStatus * pBlobStatus )

  \brief     reads the current BLOB-ID from the device
```

This function reads the current BLOB-ID(Index 49) from the device using TMGDLL.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
blob_id	Pointer to the variable the BLOB-ID to put
pBlobStatus	Pointer to the structure where detailed error-information is put in

2.10.3.5 BLOB_Continue()

```
LONG __stdcall BLOB_Continue (
            LONG Handle,
            DWORD Port,
            TBLOBStatus * pBlobStatus )

  \brief     continues the BLOB protocol
```

This function is used to execute the next BLOB step. this approach is used so that the calling application can show progress bar etc, or can abort the update.

Returns

Errorcode

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pBlobStatus	Pointer to the structure where detailed error-information is put in

2.11 Statistic Functions

Data Structures

- struct TStatisticCounter

Functions

- LONG __stdcall IOL_GetStatisticCounter (LONG Handle, DWORD Port, TStatisticCounter *pStatisticCounter, BOOL bResetCounter)

2.11.1 Detailed Description

These functions are used to get access to the statistic counter.

2.11.2 Function Documentation

2.11.2.1 IOL_GetStatisticCounter()

```
LONG __stdcall IOL_GetStatisticCounter (
            LONG Handle,
            DWORD Port,
            TStatisticCounter * pStatisticCounter,
            BOOL bResetCounter )

  \brief     reads out the statistic counter
```

This function reads out the actual statistic counters from the USB master which are clculated during the whole time. they can be reset via the parameter bResetCounter

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pStatisticCounter	pointer to the target memory where the statistic counter shall be written to
bResetCounter	boolean. if set to TRUE the counter will be reset on read.

Return values

RETURN_OK	Everything worked out allright
RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_FUNCTION_NOT_IMPLEMENTED	function is not implemented by this device
RESULT_STATE_CONFLICT	the command is not applicable in the actual state
RESULT_NOT_SUPPORTED	the command is not supported on this device

2.12 Transparent Mode

Data Structures

- struct TTransparentParameters

Functions

- LONG __stdcall IOL_SetTransparentMode (LONG Handle, TTransparentParameters *pTransparentParameters)
- LONG __stdcall IOL_SetTransparentModeExt (LONG Handle, DWORD Port, TTransparentParameters *pTransparentParameters)

Transmission Flags

These Definitions are used for the parameter TransmissionFlags. They can be combined by logical OR of the values

- #define TRANSFLAGS_7BIT
- #define TRANSFLAGS_8BIT
- #define TRANSFLAGS_NOPARITY
- #define TRANSFLAGS_ODDPARITY
- #define TRANSFLAGS_EVENPARITY
- #define TRANSFLAGS_MSBFIRST
- #define TRANSFLAGS_LSBFIRST
- #define TRANSFLAGS_SENDRETURN
- #define TRANSFLAGS_DONTSENDRETURN
- #define TRANSFLAGS_ECHO

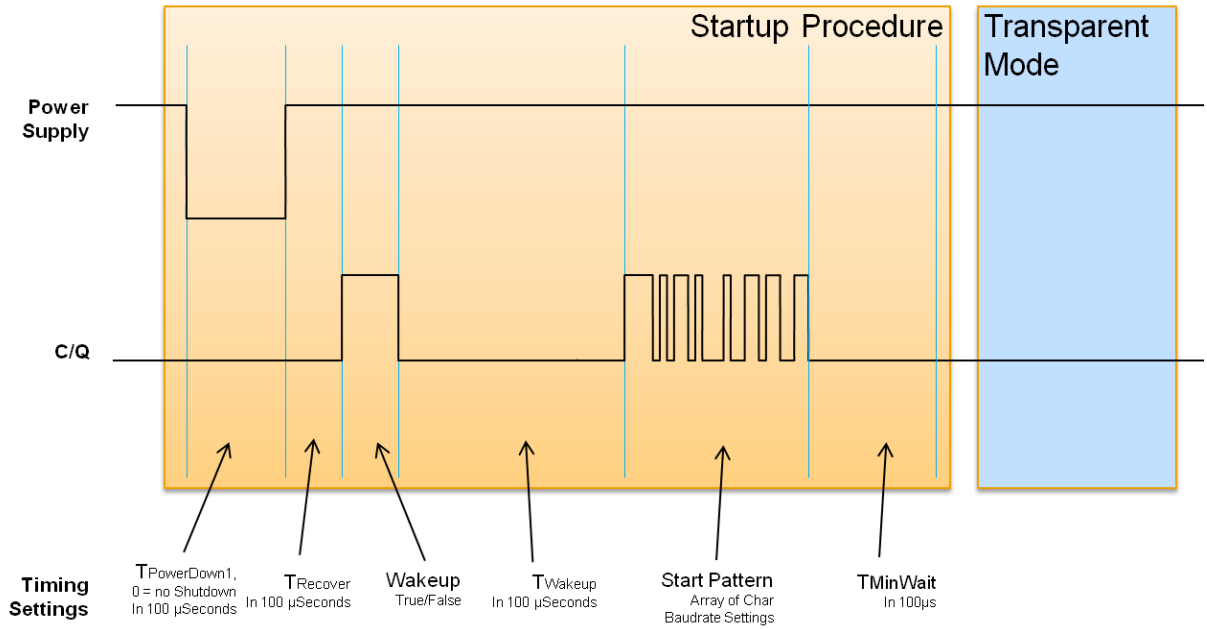
- #define TRANSFLAGS_NOPOWERATEND
- #define TRANSFLAGS_FULLDUPLEX

2.12.1 Detailed Description

These functions are used to set the Interface to transparent mode. In transparent mode, the device behaves like a UART with IO-Link PHY attached. The communication parameters can be defined, as well as the time values for the state machine of the transparent mode.

Remark
The transparent functions are an optional package and not available by default. By contract these functions can be enabled for a vendor specific version. If the functions are not allowed, the related functions will return with an error code. The same applies for the Full duplex mode which is only available for some vendor specific versions. The TMG versions have full support for the transparent mode.

Not that the transmission parameters must be defined in during the switch to transparent mode and cannot be changed after the starting procedure. The parameters which are set during COM-Port-Connection have no influence on the transmission. For this reason, it's not possible to run with changing baud rates or transmission parameters. To set the device back to normal IO-Link mode, a pattern can be defined which will cause the device to switch back.
The following picture shows the timing diagram for the activation of the transparent mode:



For all timing parameters the following rules apply:

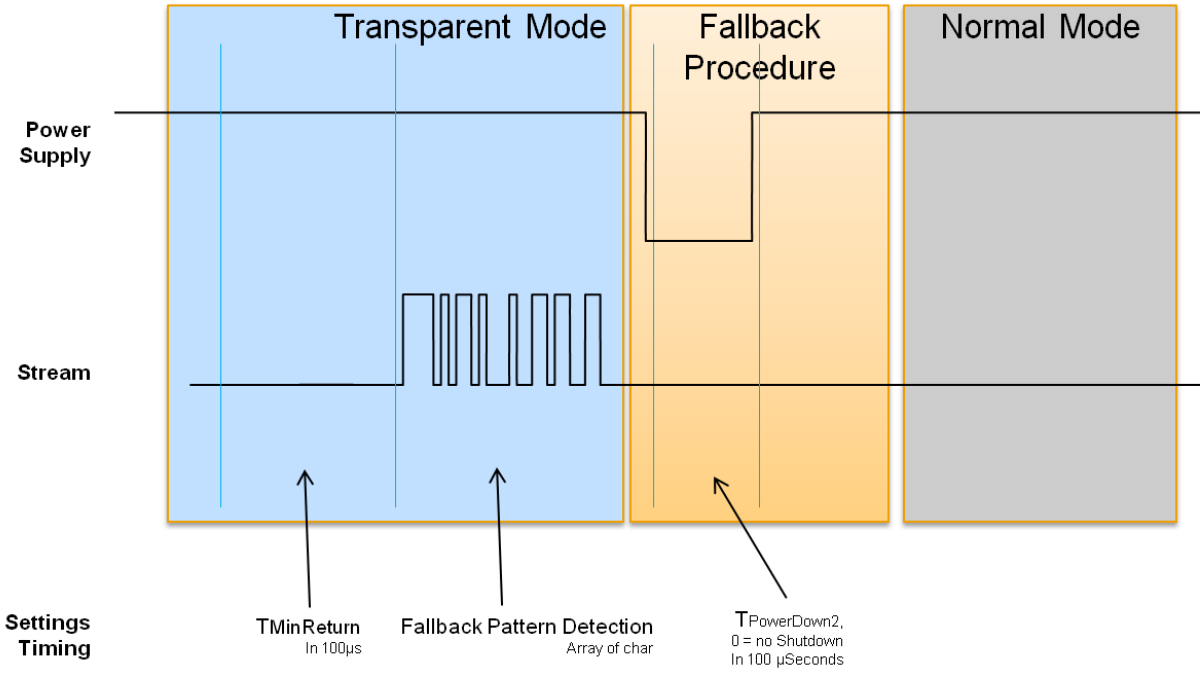
- the unit of the value is 100 microseconds
- due to technical reasons, the value can run from 0 to 0xED0A (60682), which means that the max. time is about 6,07seconds.
- the time accuracy is about 50 microseconds, but due to interrupt latencies it cannot be guaranteed that the real accuracy is below 100 microseconds. For this reason, avoid special very fast timing constraints in the sensors.

- if the value is set to zero, the parameter will be ignored, and the next step begins immediately.

During the switch to transparent mode the following phases occur:

- **Phase 1: Power Down Phase 1** some of the devices require that a firmware download can only be started during a specific time after reboot. To support this, the transparent mode is able to switch off the power supply of the device for a given time. The time will be set with the Parameter TPowerDown1. Choose this time to be sure that the device is completely powered down. If the device doesn't need a power down, set this value to zero.
- **Phase 2: Power Up Phase 2** after Powering up the device will need some time to restart properly. The USB master will wait the time TRecover1 until it will proceed with the next phase. If this waiting is not necessary (e.g. because there was no power down), set this value to zero.
- **Phase 3: Wakeup** If the device needs a wakeup, the USB Master will initiate one. This wakeup is exactly the same as the IO-Link. The USB master will read the input value, will calculate the invers value and will initiate a pulse output with about 80 microseconds. If the device doesn't need such a wakeup, set the Parameter InitiateWakeup to zero. Otherwise the parameter must be non zero (the real value doesn't care.
- **Phase 4: Wakeup reaction time** after initiation of the wakeup the device needs some time to switch to passive mode. This time can be defined with the parameter TWakeup.
- **Phase 5: Starting Pattern transmission** after waiting for the reaction of the device a start pattern can be defined which is sent to the device at the end of the switching. This should be done here if the device needs such a pattern during a specific time after starting. If the device is not dependend on such a time, this start sequence can be omitted. In this case set the length to zero. After this phase the device is in transparent mode. During this mode, it will behave like a UART. All characters which will be received by the master will be send without change to the device. The character parameters for the transmissions are given by the parameters Baudrate and TransmissionFlags. The Baudrate parameter supports the following values:
  - 1200 baud
  - 2400 baud
  - 4800 baud
  - 9600 baud
  - 19600 baud
  - 38400 baud
  - 57600 baud
  - 115200 baud
  - 230400 baud The TransmissionFlags Parameter supports the usual transmission parameters and some additional flags for the startup and return. Please see below for additional info.
- **Phase 6: minimum wait time before first transmission of the data stream** This time (TMinWait) defines how long the USB master will wait after transmission of the starting pattern until it will transmit the first byte of the data stream.

To switch the USB master back to normal mode, a return pattern will be send. After detecting the return pattern the USB Interface will start the "Fallback" procedure. Before the master will check for the return pattern, a quiet phase on the line (Send and receive) must be expired. The following image is showing the phases of the fallback procedure:



During the switch back to normal mode the following phases occur:

- **Phase 7: Timeout for activating the return pattern recognition** To avoid errornous returning to normal mode (e.g. if the pattern is contained in the some binary), a timeout (TWaitReturn) should occur after the last sent byte before the return pattern is been given to the IO-Link Master.
- **Phase 8: Detection of return pattern** The USB interface will detect the return pattern. Dependend of the TransmissionFlags the pattern will be sent to the device or not.
- **Phase 9: Power Down Phase 2** Some device require a reboot at the end of a firmware download to activate the new firmware. The parameter "TPowerDown2" will define the time how long this will be. A zero value defines that there is no Power down. Non zero values are at a base of 100 microseconds. The time can be 0x0-0xFFFF
- **Phase 10: Restart to normal mode** the last step is a restart of the whole device. Note that the device is rebooting, so the virtual Com Port on the PC side will temporary not available. So the best way is to close the handle on the PC-Side after sending the return pattern, wait some time (dependend on the parameters), and then the USB master can be used as normal.

2.12.2 Macro Definition Documentation

2.12.2.1 TRANSFLAGS_7BIT

`#define TRANSFLAGS_7BIT`

only 7 bits will be transmitted. The Most significant bit of each byte will be ignored

2.12.2.2 TRANSFLAGS_8BIT

```
#define TRANSFLAGS_8BIT
```
all 8 bits of each byte will be transmitted

2.12.2.3 TRANSFLAGS_NOPARITY

```
#define TRANSFLAGS_NOPARITY
```
no parity transmission

2.12.2.4 TRANSFLAGS_ODDPARITY

```
#define TRANSFLAGS_ODDPARITY
```
odd parity setting

2.12.2.5 TRANSFLAGS_EVENPARITY

```
#define TRANSFLAGS_EVENPARITY
```
even parity setting

2.12.2.6 TRANSFLAGS_MSBFIRST

```
#define TRANSFLAGS_MSBFIRST
```
characters will be transmitted with MSB first

2.12.2.7 TRANSFLAGS_LSBFIRST

```
#define TRANSFLAGS_LSBFIRST
```
characters will be transmitted with LSB first

2.12.2.8 TRANSFLAGS_SENDRETURN

```
#define TRANSFLAGS_SENDRETURN
```
return pattern will be send to the device

2.12.2.9 TRANSFLAGS_DONTSENDRETURN

```
#define TRANSFLAGS_DONTSENDRETURN
```
return pattern will not be send to the device

2.12.2.10 TRANSFLAGS_ECHO

```
#define TRANSFLAGS_ECHO
```
echo all of the sent data bytes

2.12.2.11 TRANSFLAGS_NOPOWERATEND

```
#define TRANSFLAGS_NOPOWERATEND
```
if set the power will not go active after TPowerDown2, because for some devices there would be a double power impulse which is bad if the device writes firmware after first repowering

2.12.2.12 TRANSFLAGS_FULLDUPLEX

```
#define TRANSFLAGS_FULLDUPLEX
```
if set, the Master will send/receive in Full-Duplex using Pin 2 and 4

2.12.3 Function Documentation

2.12.3.1 IOL_SetTransparentMode()

```
LONG __stdcall IOL_SetTransparentMode (
            LONG Handle,
            TTransparentParameters * pTransparentParameters )

  \brief     activates the transparent mode
```

This function activates the transparent mode for the USB master. It will return up on end of the starting sequence. No other function can be called after the call to this function. The function will close and destroy the handle which has been used for the communication. The application / or another program must connect to the virtual COM Port as usual.

Parameters

Handle	Handle to work on/with
pTransparentParameters	Pointer to the Transparent Mode Parameters

Return values

RETURN_OK	Everything worked out allright
RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_FUNCTION_NOT_IMPLEMENTED	function is not implemented by this device
RETURN_FUNCTION_CALLEDFROMCALLBACK	calling a DLL function from inside a callback is not allowed
RESULT_STATE_CONFLICT	the command is not applicable in the actual state
RESULT_NOT_SUPPORTED	the command is not supported on this device

2.12.3.2 IOL_SetTransparentModeExt()

```
LONG __stdcall IOL_SetTransparentModeExt (
            LONG Handle,
            DWORD Port,
            TTransparentParameters * pTransparentParameters )

  \brief     activates the transparent mode
```

This function activates the transparent mode for the USB master. It will return up on end of the starting sequence. No other function can be called after the call to this function. The function will close and destroy the handle which has been used for the communication. The application / or another program must connect to the virtual COM Port as usual. Note: this function is not available on all firmware versions of the USB Master V2. if not supported, an error message will be schown.

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
pTransparentParameters	Pointer to the Transparent Mode Parameters

Return values

RETURN_OK	Everything worked out allright
RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_FUNCTION_NOT_IMPLEMENTED	function is not implemented by this device
RETURN_FUNCTION_CALLEDFROMCALLBACK	calling a DLL function from inside a callback is not allowed
RESULT_STATE_CONFLICT	the command is not applicable in the actual state
RESULT_NOT_SUPPORTED	the command is not supported on this device

2.13 Process Data Logging

Functions

- LONG __stdcall IOL_StartDataLogging (LONG Handle, DWORD Port, char *FileName, DWORD *pSampleTimeMs)
- LONG __stdcall IOL_StartDataLoggingInBuffer (LONG Handle, DWORD Port, LONG MemorySize, DWORD LoggingMode, DWORD *pSampleTime)
- LONG __stdcall IOL_ReadLoggingBuffer (LONG Handle, LONG *pBufferSize, BYTE *pData, DWORD *pStatus)
- LONG __stdcall IOL_StopDataLogging (LONG Handle)

LoggingMode Operation mode of the data logging. There are two modes: time driven and cycle synchron. In time driven mode,

the USB Master will send the process data each invall given by the settings. the unit is microseconds, but it will be rounded to a multiple of 5ms, because there is no advantage if it would be faster, because it is not synchronized with the IO-Link cycle. In cycle synchron mode, the setting will be used as a counter for the cycles. On elapsing of the counter the samples will be sent to the PC. The real cycle depends on the cycle time of the master and the counter

- #define LOGGING_MODE_TIME
- #define LOGGING_MODE_CYCLES

LoggingStatus Bit Codings used in the status of the function IOL_ReadLoggingBuffer

- #define LOGGING_STATUS_RUNNING
- #define LOGGING_STATUS_AVAILABLE
- #define LOGGING_STATUS_OVERRUN

InputValidity coding for validity of inputs, see IOL_ReadLoggingBuffer

- #define LOGGING_INPUTS_VALID
- #define LOGGING_INPUTS_INVALID

2.13.1 Detailed Description

These functions are used to log the process data to a file or into a buffer.

2.13.2 Macro Definition Documentation

2.13.2.1 LOGGING_MODE_TIME

`#define LOGGING_MODE_TIME`
time driven logging mode

2.13.2.2 LOGGING_MODE_CYCLES

`#define LOGGING_MODE_CYCLES`
cycle synchron logging mode

2.13.2.3 LOGGING_STATUS_RUNNING

`#define LOGGING_STATUS_RUNNING`
1 if Logging is started, 0 if stopped

2.13.2.4 LOGGING_STATUS_AVAILABLE

`#define LOGGING_STATUS_AVAILABLE`
1 if there are more data available in the read buffer

2.13.2.5 LOGGING_STATUS_OVERRUN

`#define LOGGING_STATUS_OVERRUN`
The application has not read out the results fast enough. for this reason the logging has stopped.
The bit is reset on call of the function see IOL_StopDataLogging or see IOL_StartDataLoggingInBuffer

2.13.2.6 LOGGING_INPUTS_VALID

`#define LOGGING_INPUTS_VALID`
inputs are valid and can be used

2.13.2.7 LOGGING_INPUTS_INVALID

`#define LOGGING_INPUTS_INVALID`
the inputs are invalid, and the use of the data should not be done because the content is not guaranteed

2.13.3 Function Documentation

2.13.3.1 IOL_StartDataLogging()

```
LONG __stdcall IOL_StartDataLogging (
              LONG Handle,
              DWORD Port,
              char * FileName,
              DWORD * pSampleTimeMs )

  \brief     Starts the process data logging into a file
```

This function informs the master to send cyclically the process input data to the DLL. The DLL will store them together with the output data into the file which has been defined by the FileName. The logging will only occur if the mode of the port is not in Deactivated. The Sample Time will be given in ms, however the master interface will round this time to a value it can provide

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
FileName	Pointer to the full filename
pSampleTimeMs	time interval between the process data samples

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_WRONG_PARAMETER	One of the paramters was wrong.
RETURN_OK	Everything worked out allright
RETURN_STATE_CONFLICT	interface is in the wrong state
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

2.13.3.2 IOL_StartDataLoggingInBuffer()

```
LONG __stdcall IOL_StartDataLoggingInBuffer (
              LONG Handle,
              DWORD Port,
              LONG MemorySize,
              DWORD LoggingMode,
              DWORD * pSampleTime )

  \brief     Starts the process data logging into a buffer
```

This function informs the master to send cyclically the process input data to the DLL. The DLL will store them together with the output data into a buffer. The logging will only occur if the mode of the port is not in Deactivated. The Sample Time will be given in microseconds, however the master interface will round this time to a value it can provide. The data can be read by the function IOL_ReadLoggingBuffer

Parameters

Handle	Handle to work on/with
Port	Port number of the used port
LoggingMode	Mode which defines the trigger for the transmission of process data. See LoggingMode for the different modes.
MemorySize	size of the ring buffer. the memory will be allocated by the DLL.
pSampleTime	dependend on the mode parameter this parameter defines either the time interval between the process data samples in micro seconds (Mode = 0) or the number of cycles to elapse between two samples in mode 1

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_WRONG_PARAMETER	One of the paramters was wrong.
RETURN_OK	Everything worked out allright
RETURN_STATE_CONFLICT	interface is in the wrong state
RETURN_FUNCTION-CALLEDFROMCALLBACK	calling a DLL function from inside a callback is not allowed

2.13.3.3 IOL_ReadLoggingBuffer()

```
LONG __stdcall IOL_ReadLoggingBuffer (
              LONG Handle,
              LONG * pBufferSize,
              BYTE * pData,
              DWORD * pStatus )

  \brief     Ready out the logged data from the buffer
```

This function is used to read out a part of the logging buffer. The pointer and the size of the buffer will part of the parameter. The function will read out as much as logging data as possible. The data will not be segmented. If there is no more place in the buffer, the size will be reduced. The data is an array of logging entries. A logging Entry is structured as seen below (all members are byte or byte arrays): Port Port number of the logged port InLength Length of the logged input data including an additional byte for the validity which is the last byte InputData Array of the inputs. The length is InLength-1 InValidity validity of the inputs OutLength Length of the output data OutputData Array of the outputs. The length is OutLength

Parameters

Handle	Handle to work on/with
pBufferSize	pointer to the size of the ring buffer. On call, it shall contain the maximum length of the buffer. This function will change it to the real length.

Note
The buffer Size shall be at least the maximum of an IO-Link frame + some bytes of structure information. for this reason the length should be bigger than 128 byte. Otherwise it might be possible that you cannot read out the buffer.

Parameters

pData	pointer to a buffer where the logging data shall be stored.
pStatus	Pointer to a DWORD buffer where a bit coded status will be stored. The possible bit values for the status are coded in LoggingStatus.

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_WRONG_PARAMETER	One of the paramters was wrong.
RETURN_OK	Everything worked out allright
RETURN_STATE_CONFLICT	interface is in the wrong state
RETURN_FUNCTION-CALLEDFROMCALLBACK	calling a DLL function from inside a callback is not allowed

2.13.3.4 IOL_StopDataLogging()

```
LONG __stdcall IOL_StopDataLogging (
            LONG Handle )

  \brief     Stops the data logging
```

This function informs the master to stop the logging of the process data.

Parameters

Handle	Handle to work on/with

Return values

RETURN_UNKNOWN_HANDLE	Handle is not valid
RETURN_INTERNAL_ERROR	Error that should not occur.
RETURN_WRONG_PARAMETER	One of the paramters was wrong.
RETURN_STATE_CONFLICT	interface is in the wrong state
RETURN_OK	Everything worked out allright
RETURN_FUNCTION_CALLEDFROM-CALLBACK	calling a DLL function from inside a callback is not allowed

Data Structure Documentation

3.1 TBLOBStatus Struct Reference

Data Fields

- BYTE errorCode
- BYTE additionalCode
- LONG dllReturnValue
- DWORD Position
- BYTE PercentComplete
- BYTE nextState

3.1.1 Detailed Description

TBLOBStatus_Struct contains the status information about BLOB service.

3.1.2 Field Documentation

3.1.2.1 errorCode

`BYTE TBLOBStatus::errorCode`
State which was executed during the call of the continue function error code for the result of the service

3.1.2.2 additionalCode

`BYTE TBLOBStatus::additionalCode`
additional error code of the result

3.1.2.3 dllReturnValue

`LONG TBLOBStatus::dllReturnValue`
return value from IOL - function

3.1.2.4 Position

`DWORD TBLOBStatus::Position`
actual position

3.1.2.5 PercentComplete

`BYTE TBLOBStatus::PercentComplete`
percentage of download will be computed

3.1.2.6 nextState

`BYTE TBLOBStatus::nextState`
next step which will be executed or has been entered (in case of error or idle)

3.2 TDeviceIdentification Struct Reference

Data Fields

- char Name [8]
- char ProductCode [16]
- char ViewName [100]

3.2.1 Detailed Description

TDeviceIdentification contains the information about an USB IO-Link master.

3.2.2 Field Documentation

3.2.2.1 Name

`char TDeviceIdentification::Name[8]`
contains the device name which should be used for the driver

3.2.2.2 ProductCode

`char TDeviceIdentification::ProductCode[16]`
product identification

3.2.2.3 ViewName

`char TDeviceIdentification::ViewName[100]`
name which is shown in the device manager

3.3 TDLLCallbacks Struct Reference

Data Fields

- DWORD TParameter * pParameter
- DWORD TEvent * pEvent

3.3.1 Detailed Description

TDLLCallbacks contains the list of pointer to different callbacks which are used to make the functions asynchronous which have access to the sensor variables. These are IOL_ReadRequest, IOL_WriteRequest and the IOL_GetMode and IOL_GetModeEx functions, because they are reading Device parameters. The callbacks must be given to the DLL with the function IOL_SetCallbacks and are valid per Master Interface. After each IOL_Create the list of callbacks is empty, so after connection establishing the list must be set. The List entries may be empty (NULL), which means that the callback will not be called. If the list entries are not empty, the Request function will return RETURN_FUNCTION_DELAYED to show that the result will be delayed. There are some rules for the usage of the callbacks:

- The data structure which are given with the request must be valid until the callback is done.

- the callback will block the reception of the USB master interface, so the code in the callback must be very short
- no DLL function may be called from the callbacks.
- The list of callbacks may be changed anytime if there is no function pending.

The following callbacks are supported:

- IOL_CallbackReadConfirmation is the callback which is called if an IOL_ReadRequest has been called before. The result of the service is contained in the data structure which has been given to the service on Request
  - Handle Handle to work on/with
  - Port Port number of the used port
  - pParameter Pointer to the structure containing the result and data of the service
- IOL_CallbackWriteConfirmation is the callback which is called if an IOL_WriteRequest has been called before. The result of the service is contained in the data structure which has been given to the service on Request
  - Handle Handle to work on/with
  - Port Port number of the used port
  - pParameter Pointer to the structure containing the result of the service
- IOL_CallbackEventInd is called if an Event has been received. The memory of the data structure of the event will be part of the DLL, so after returning from the callback the data will be lost. The event will not be stored in the event list, the callback has to implement it's own event list if this is necessary.
  - Handle Handle to work on/with
  - Port Port number of the used port
  - pEvent Pointer to the data structure containing the new event

3.3.2 Field Documentation

3.3.2.1 pParameter

`DWORD TParameter * TDLLCallbacks::pParameter`
callback which is called if a Parameter Read has been terminated
callback which is called if a Parameter Write has been terminated

3.3.2.2 pEvent

`DWORD TEvent* TDLLCallbacks::pEvent`
callback which is called if an Event has been received

3.4 TDllInfo Struct Reference

Data Fields

- char Build [20]
- char Datum [20]
- char Version [20]

3.4.1 Detailed Description

TDllInfo contains the DLL version information

3.4.2 Field Documentation

3.4.2.1 Build

`char TDllInfo::Build[20]`
Build revision of the DLL

3.4.2.2 Datum

`char TDllInfo::Datum[20]`
build date of the DLL

3.4.2.3 Version

`char TDllInfo::Version[20]`
major revision of the DLL

3.5 TEvent Struct Reference

Data Fields

- WORD Number
- WORD Port
- WORD EventCode
- BYTE Instance
- BYTE Mode
- BYTE Type
- BYTE PDValid
- BYTE LocalGenerated

3.5.1 Detailed Description

TEvent contains the data of an occured event

3.5.2 Field Documentation

3.5.2.1 Number

`WORD TEvent::Number`
number of the event, is incremented by the DLL

3.5.2.2 Port

`WORD TEvent::Port`
port on which the event occured

3.5.2.3 EventCode

`WORD TEvent::EventCode`
event code

3.5.2.4 Instance

`BYTE TEvent::Instance`
instance of the event

3.5.2.5 Mode

`BYTE TEvent::Mode`
event mode

3.5.2.6 Type

`BYTE TEvent::Type`
event type

3.5.2.7 PDValid

`BYTE TEvent::PDValid`
event mode

3.5.2.8 LocalGenerated

`BYTE TEvent::LocalGenerated`
TRUE if the event was generated by the IO-Link master

3.6 TFwUpdateInfo Struct Reference

Data Fields

- WORD vendorID
- BYTE fwPasswordRequired
- BYTE hwKey [64+1]
- BYTE * pFirmware
- DWORD fwLength

3.6.1 Detailed Description

TFwUpdateInfo contains the information about an Update request.

3.6.2 Field Documentation

3.6.2.1 vendorID

`WORD TFwUpdateInfo::vendorID`
vendor ID of the attached device. Must match

3.6.2.2 fwPasswordRequired

`BYTE TFwUpdateInfo::fwPasswordRequired`
from meta file. not used at this moment

3.6.2.3 hwKey

`BYTE TFwUpdateInfo::hwKey[64+1]`
the correct hardware key which shall be used. the meta file must support more than one, but in this case the parser of the meta file can look for the correct one.

3.6.2.4 pFirmware

`BYTE* TFwUpdateInfo::pFirmware`
pointer to the firmware object. Must be consecutive memory

3.6.2.5 fwLength

`DWORD TFwUpdateInfo::fwLength`
length of the firmware image

3.7 TFWUpdateState Struct Reference

Data Fields

- BYTE errorCode
- BYTE additionalCode
- LONG dllReturnValue
- LONG blobReturnValue
- BYTE nextState
- TBLOBStatus BlobStatus

3.7.1 Detailed Description

TFWUpdateState_Struct contains the status information about an Update request.

3.7.2 Field Documentation

3.7.2.1 errorCode

`BYTE TFWUpdateState::errorCode`
State which was executed during the call of the continue function error code for the result of the last executed service

3.7.2.2 additionalCode

`BYTE TFWUpdateState::additionalCode`
additional error code of the result of the last executed service

3.7.2.3 dllReturnValue

`LONG TFWUpdateState::dllReturnValue`
return value from IOL - function for the result of the last executed service

3.7.2.4 blobReturnValue

`LONG TFWUpdateState::blobReturnValue`
return value of the BLOB state machine during download

3.7.2.5 nextState

`BYTE TFWUpdateState::nextState`
next step which will be executed or has been entered (in case of error or idle)

3.7.2.6 BlobStatus

`TBLOBStatus TFWUpdateState::BlobStatus`
in download states we copy the status from the blob state machine

3.8 THardwareInfo Struct Reference

Data Fields

- DWORD InfoVersion
- DWORD PowerSource
- DWORD PowerLevel

3.8.1 Detailed Description

THardwareInfo contains actual hardware information about the connected master

3.8.2 Field Documentation

3.8.2.1 InfoVersion

`DWORD THardwareInfo::InfoVersion`
version of the structure. 0: only PowerSource and PowerLevel

3.8.2.2 PowerSource

`DWORD THardwareInfo::PowerSource`
actual source of the power. 0 = internal power, all other values = external Power

3.8.2.3 PowerLevel

`DWORD THardwareInfo::PowerLevel`
actual Power level in units of 100mV

3.9 TInfo Struct Reference

Data Fields

- char COM [10]
- BYTE DeviceID [3]
- BYTE VendorID [2]
- BYTE FunctionID [2]
- BYTE ActualMode
- BYTE SensorState
- BYTE MasterCycle
- BYTE CurrentBaudrate

3.9.1 Detailed Description

TInfo contans the information about a connected sensor and the state of a port

3.9.2 Field Documentation

3.9.2.1 COM

`char TInfo::COM[10]`
device interface name

3.9.2.2 DeviceID

`BYTE TInfo::DeviceID[3]`
Device ID

3.9.2.3 VendorID

`BYTE TInfo::VendorID[2]`
Vendor ID

3.9.2.4 FunctionID

`BYTE TInfo::FunctionID[2]`
Function ID

3.9.2.5 ActualMode

`BYTE TInfo::ActualMode`
Actual Mode of the Port, Deactivated, IO-Link or SIO

3.9.2.6 SensorState

`BYTE TInfo::SensorState`
state of the sensor see

See also

SensorStateDefinitions

3.9.2.7 MasterCycle

`BYTE TInfo::MasterCycle`
used cycle time if sensor is connected

3.9.2.8 CurrentBaudrate

`BYTE TInfo::CurrentBaudrate`
current baud rate

3.10 TInfoEx Struct Reference

Data Fields

- char COM [10]
- BYTE DirectParameterPage [16]
- BYTE ActualMode
- BYTE SensorStatus
- BYTE CurrentBaudrate

3.10.1 Detailed Description

TInfoEx contains the extended information about a connected sensor

3.10.2 Field Documentation

3.10.2.1 COM

`char TInfoEx::COM[10]`
device interface name

3.10.2.2 DirectParameterPage

`BYTE TInfoEx::DirectParameterPage[16]`
information from direct parameter page (Index 0)

3.10.2.3 ActualMode

`BYTE TInfoEx::ActualMode`
actual master port state

3.10.2.4 SensorStatus

`BYTE TInfoEx::SensorStatus`
actual connection state of the sensor

3.10.2.5 CurrentBaudrate

`BYTE TInfoEx::CurrentBaudrate`
actual baud rate

3.11 TMasterInfo Struct Reference

Data Fields

- char Version [13]
- BYTE Major
- BYTE Minor
- BYTE Build
- BYTE MajorRevisionIOLStack
- BYTE MinorRevisionIOLStack
- BYTE BuildRevisionIOLStack

3.11.1 Detailed Description

TMasterInfo contains revision information from the connected master

3.11.2 Field Documentation

3.11.2.1 Version

`char TMasterInfo::Version[13]`
string which was build from the following parameters

3.11.2.2 Major

`BYTE TMasterInfo::Major`
major firmware revision

3.11.2.3 Minor

`BYTE TMasterInfo::Minor`
minor firmware revision

3.11.2.4 Build

`BYTE TMasterInfo::Build`
build revision of the firmware

3.11.2.5 MajorRevisionIOLStack

`BYTE TMasterInfo::MajorRevisionIOLStack`
major revision of the IO-Link stack used by the master

3.11.2.6 MinorRevisionIOLStack

`BYTE TMasterInfo::MinorRevisionIOLStack`
minor revision of the IO-Link stack used by the master

3.11.2.7 BuildRevisionIOLStack

`BYTE TMasterInfo::BuildRevisionIOLStack`
build revision of the IO-Link stack used by the master

3.12 TParameter Struct Reference

Data Fields

- BYTE Result [256]
- WORD Index
- BYTE SubIndex
- BYTE Length
- BYTE ErrorCode
- BYTE AdditionalCode

3.12.1 Detailed Description

TParameter contains the information which are used for ISDU read and write

3.12.2 Field Documentation

3.12.2.1 Result

`BYTE TParameter::Result[256]`
buffer for data bytes (read and write)

3.12.2.2 Index

`WORD TParameter::Index`
index of the variable to be read or written

3.12.2.3 SubIndex

`BYTE TParameter::SubIndex`
subindex of the variable to be read or written

3.12.2.4 Length

`BYTE TParameter::Length`
length of the parameter data

3.12.2.5 ErrorCode

`BYTE TParameter::ErrorCode`
error code for the result of the service

3.12.2.6 AdditionalCode

`BYTE TParameter::AdditionalCode`
additional error code of the result

3.13 TPortConfiguration Struct Reference

Data Fields

- BYTE PortModeDetails
- BYTE TargetMode
- BYTE CRID
- BYTE DSConfigure
- BYTE Synchronisation
- BYTE FunctionID [2]
- BYTE InspectionLevel
- BYTE VendorID [2]
- BYTE DeviceID [3]
- BYTE SerialNumber [16]
- BYTE InputLength
- BYTE OutputLength

3.13.1 Detailed Description

TPortConfiguration contains the port configuration information

3.13.2 Field Documentation

3.13.2.1 PortModeDetails

`BYTE TPortConfiguration::PortModeDetails`
additional info for the port

3.13.2.2 TargetMode

`BYTE TPortConfiguration::TargetMode`
Mode in which the port shall be run

3.13.2.3 CRID

`BYTE TPortConfiguration::CRID`
configured revision ID

3.13.2.4 DSConfigure

`BYTE TPortConfiguration::DSConfigure`
Data Storage configuration

3.13.2.5 Synchronisation

`BYTE TPortConfiguration::Synchronisation`
Synchronisation, not used

3.13.2.6 FunctionID

`BYTE TPortConfiguration::FunctionID[2]`
Function ID, not used

3.13.2.7 InspectionLevel

`BYTE TPortConfiguration::InspectionLevel`
NO_CHECK, TYPE_COMP, IDENTICAL

3.13.2.8 VendorID

`BYTE TPortConfiguration::VendorID[2]`
validation: Vendor ID of the device

3.13.2.9 DeviceID

`BYTE TPortConfiguration::DeviceID[3]`
validation: Device ID of the device

3.13.2.10 SerialNumber

`BYTE TPortConfiguration::SerialNumber[16]`
NULL-terminated string with the serial number

3.13.2.11 InputLength

`BYTE TPortConfiguration::InputLength`
configured input length

3.13.2.12 OutputLength

`BYTE TPortConfiguration::OutputLength`
configured Output length

3.14 TStatisticCounter Struct Reference

Data Fields

- DWORD CycleCounter
- DWORD RetryCounter
- DWORD AbortCounter

3.14.1 Detailed Description

TStatisticCounter contain the statistic counter

3.14.2 Field Documentation

3.14.2.1 CycleCounter

`DWORD TStatisticCounter::CycleCounter`
counts the number of frame cycles

3.14.2.2 RetryCounter

`DWORD TStatisticCounter::RetryCounter`
counts the number of retries

3.14.2.3 AbortCounter

`DWORD TStatisticCounter::AbortCounter`
counts the number of connection aborts

3.15 TTransparentParameters Struct Reference

Data Fields

- DWORD TPowerDown1
- DWORD TRecover1
- DWORD InitiateWakeup
- DWORD TWakeup
- DWORD TransmissionFlags
- DWORD Baudrate
- DWORD StartPatternLength
- DWORD TMinWait
- DWORD TWaitReturn
- DWORD ReturnPatternLength
- DWORD TPowerDown2
- BYTE StartPattern [16]
- BYTE ReturnPattern [32]

3.15.1 Detailed Description

TTransparentParameters contain the Parameters for the Transparent Mode

3.15.2 Field Documentation

3.15.2.1 TPowerDown1

`DWORD TTransparentParameters::TPowerDown1`
time which defines how long the power will be put down (unit is 100 microseconds)

3.15.2.2 TRecover1

`DWORD TTransparentParameters::TRecover1`
time which the device needs for recovering after Power Up (unit is 100 microseconds)

3.15.2.3 InitiateWakeup

`DWORD TTransparentParameters::InitiateWakeup`
boolean. non zero if the Master shall initiate a wakeup pulse

3.15.2.4 TWakeup

`DWORD TTransparentParameters::TWakeup`
wakeup time which the device needs to react on the wakeup

3.15.2.5 TransmissionFlags

`DWORD TTransparentParameters::TransmissionFlags`
contain the different flags for the UART transmission

3.15.2.6 Baudrate

`DWORD TTransparentParameters::Baudrate`
baudrate of the transmission in bits per seconds. However, only the specified values are allowed

3.15.2.7 StartPatternLength

`DWORD TTransparentParameters::StartPatternLength`
length of the starting pattern 0..16

3.15.2.8 TMinWait

`DWORD TTransparentParameters::TMinWait`
minimum waiting time after start pattern

3.15.2.9 TWaitReturn

`DWORD TTransparentParameters::TWaitReturn`
timeout to activate the pattern recognition

3.15.2.10 ReturnPatternLength

`DWORD TTransparentParameters::ReturnPatternLength`
Length of the returning pattern 0..32

3.15.2.11 TPowerDown2

`DWORD TTransparentParameters::TPowerDown2`
time which defines how long the power will be put down at the end of the transparent mode(unit is 100 microseconds)

3.15.2.12 StartPattern

`BYTE TTransparentParameters::StartPattern[16]`
starting pattern

3.15.2.13 ReturnPattern

`BYTE TTransparentParameters::ReturnPattern[32]`
returning pattern

Index