

Programmable Systems
System *HiMatrix*

***COM USER TASK
(CUT)***

Manual



HIMA Paul Hildebrandt GmbH
Industrial Automation

HI 800 329 CEA

Important Notes

All HIMA products mentioned in this manual are protected under the HIMA trademark. Unless not explicitly noted, this may apply for other referenced manufacturers and their respective products.

All technical statements and data in this manual have been written with great care and effective quality measures have been taken to ensure their validity; however this manual may contain flaws or typesetting errors.

For this reason, HIMA does not offer any warranties nor assume legal responsibility nor any liability for possible consequences of any errors in this manual. HIMA appreciates any correspondence noting potential errors.

Technical modifications reserved.

For more information see the documentation on CD-ROM and on our web site www.hima.com.

More information can be requested from:

HIMA Paul Hildebrandt GmbH
Postfach 1261
68777 Brühl

Tel: +49(6202)709 0
Fax: +49(6202)709 107

e-mail: info@hima.com

Contents

1	COM User Task (CUT)	5
1.1	Preconditions	5
1.1.1	ELOP II Factory	5
1.1.2	Development Environment	5
1.1.3	Usable Controllers	5
1.2	Abbreviations	6
2	CUT Interface in ELOP II Factory	7
2.1	Schedule Interval [ms]	7
2.2	Scheduling Preprocessing	7
2.3	Scheduling Postprocessing	7
2.4	STOP_INVALID_CONFIG	8
2.5	Signals of the CUT Interface (CPU<->CUT)	8
2.5.1	State	9
2.5.2	Control	9
2.5.3	Input Records (COM->CPU)	10
2.5.4	Output Records (CPU->COM)	11
3	CUT Functions	12
3.1	COM User Callback Functions	12
3.2	COM User Library Functions	12
3.3	Header Files	12
3.4	Code/Data Area and Stack for the CUT	12
3.5	Start Function CUCB_TaskLoop	13
3.6	Serial Interface RS485 / RS232 IF	13
3.6.1	CUL_AscOpen	13
3.6.2	CUL_AscClose	15
3.6.3	CUL_AscRcv	15
3.6.4	CUCB_AscRcvReady	17
3.6.5	CUL_AscSend	18
3.6.6	CUCB_AscSendReady	19
3.7	UDP/TCP-Socket-IF	20
3.7.1	CUL_SocketOpenUdpBind	20
3.7.2	CUL_SocketOpenUdp	21
3.7.3	CUL_NetMessageAlloc	21
3.7.4	CUL_SocketSendTo	22
3.7.5	CUCB_SocketUdpRcv	23
3.7.6	CUL_NetMessageFree	23
3.7.7	CUL_SocketOpenTcpServer	24
3.7.8	CUCB_SocketTryAccept	24
3.7.9	CUL_SocketAccept	25
3.7.10	CUL_SocketOpenTcpClient	26
3.7.11	CUCB_SocketConnected	26
3.7.12	CUL_SocketSend	27
3.7.13	CUCB_SocketTcpRcv	28
3.7.14	CUL_SocketClose	28
3.8	Timer-IF	29
3.8.1	CUL_GetTimeStampMS	29
3.8.2	CUL_GetDateAndTime	29
3.9	COM User IRQ Task (only at MHS31A)	30
3.9.1	CUCB_IrqService	30
3.9.2	CUL_IrqServiceEnable	30
3.9.3	CUL_IrqServiceDisable	31
3.9.4	CUL_DeviceBaseAddr	31
3.10	NVRam IF (only at MHS31A)	32
3.10.1	CUL_NVRamWrite	32
3.10.2	CUL_NVRamRead	32
3.11	Semaphore IF (only at MHS31A)	33
3.11.1	CUL_SemaRequest	33
3.11.2	CUL_SemaRelease	33

3.11.3	CUL_SemaTry.....	34
3.12	COM IO IF (only at MHS31A)	35
3.12.1	CUL_IORead.....	35
3.12.2	CUL_IOWrite	35
3.12.3	CUL_IOConfigure.....	36
3.13	Diagnosis	36
4	Installation of the Development Environment.....	37
4.1	Installation of the Cygwin Environment.....	37
4.2	Installation of the GNU Compiler	40
5	Create New CUT Project	41
5.1	CUT Makefiles.....	42
5.1.1	Makefile with the appendix „mke“.....	42
5.1.2	Makefile	43
5.1.3	Makefile with the Extension „makeinc.inc.app“	44
5.2	Adapting C Source Code	45
5.2.1	Configure Input and Output Signals	45
5.2.2	Start Function in the CUT	45
5.2.3	Example Code „example_cut.c“	45
5.2.4	Creation of Executable Code (ldf file)	47
5.3	Implement the COM User Task in the Project	48
5.3.1	Create COM User Task.....	48
5.3.2	Loading Program Code into the Project.....	48
5.3.3	Connect Signals to the CUT.....	49
5.3.3.1	Create Signals in the Signal Editor	49
5.3.3.2	CUT Dialog „Signal Connections “	49
5.3.3.3	Configuring the Input Signals (COM->CPU)	49
5.3.3.4	Configuring the Output Signals (CPU->COM)	50
5.3.4	Creation of the ELOP II Factory Application Program	50
5.3.5	Configuring the Schedule Interval [ms]	51
5.3.6	Creation and Loading of the Code into the Controller.....	51
5.3.7	Test the COM User Task.....	51
5.4	Tips and Tricks.....	52
5.4.1	Failure during Loading of a Configuration with the CUT	52

1 COM User Task (CUT)

Beside of the application program created in ELOP II Factory the user has the additional possibility to run a C program in the controller.

This not safe C program runs as COM User Task in a non-interacting way to the safe CPU in the COM module of the controller.

The COM User Task has its own cycle time independent from the cycle of the CPU.

By this any application can be programmed in C and implemented as COM User Task e.g.:

- Communication interfaces for special protocols (TCP, UDP etc.).
- Gateway function between TCP/UDP and serial communication.

1.1 Preconditions

For programming a COM User Task an additional library to the normal C commands is used with defined functions (see chapter 2).

1.1.1 ELOP II Factory

The COM User Task function is implemented since version 8.36 of the ELOP II Factory Hardware Management.

The loading of the configuration with COM User Task requires a corresponding license.

1.1.2 Development Environment

The development environment comprises the GNU C Compiler and Cygwin which are available on a separate installation CD (not included in ELOP II Factory) and are subject to the conditions of the GNU General Public License (see www.gnu.org).

The newest versions and documents for the development environment can be downloaded from the corresponding sites in the internet www.cygwin.com and www.gnu.org.

1.1.3 Usable Controllers

A COM User Task can be created in the application program of the following resources:

Controller	CPU OS	COM OS
HIMatrix F20 HIMatrix (HW-Rev. 02) F30, F31, F35, F60	from version 6.44	from version 11.24
MHS31A	from version 6.44	from version 11.24

Table 1: Usable controllers for the COM User Task

In the *HIMatrix* controllers the COM User Task has no access to the safe hardware inputs and outputs. If you want to have access to the safe hardware inputs and outputs then an application program of the CPU is necessary for the connection of the signals (see 2.5).

At the controller MHS31A the COM User Task has no access to the safe hardware inputs and outputs.

The MHS31A has additional not safety-related inputs and outputs to which the COM User Task has a direct access. An application program of the CPU for connection of the signals (see 2.5) is not necessary for the not safety-related inputs and outputs.

1.2 Abbreviations

Abbreviations	Meaning
CPU	Central processor of the controller
COM	Communication processor of the controller
CUCB	COM User Callback (CUCB_ Functions invoked by the COM)
CUIT	COM User IRQ Task (only for MHS31A)
CUL	COM User Library (CUL_ Functions are invoked in the CUT)
CUT	COM User Task
GNU	GNU project
IF	Interface
FB	Field bus interface of the controller
FIFO	First In First Out (Data memory)
NVRam	Non Volatile Random Access Memory

Table 2: Abbreviations

2 CUT Interface in ELOP II Factory

The process data communication of the COM User Task runs between COM and CPU.



The loaded COM User Task must not use privileged commands of the COM processor.

The code of the CUT runs in the COM in a non-interacting way to the CPU. Hereby the safe CPU is protected against the code of the CUT.

It has to be regarded that errors in the CUT code can disturb or stop the function of the COM in total and by this way also the function of the controller. Nevertheless the safety functions of the CPU are not affected.

2.1 Schedule Interval [ms]

The COM User Task is invoked in a parameterizable Schedule Interval [ms] during the states RUN and STOP_VALID_CONFIG of the controller (COM processor).

The Schedule Interval [ms] can be parameterized by the user in ELOP II Factory within the properties of the COM User Task.

Schedule Interval [ms]	
Range of value:	10 .. 255 ms
Default value:	15 ms

Table 3: Parameter in ELOP II Factory dialog „Properties“ of the COM User Task

Note The time of the COM processor available to the CUT is dependent to the other parameterizable functions of the COM as e.g. SafeEthernet, Modbus-TCP etc.

If the CUT has not finished within the schedule interval, then each call for the re-start of the CUT will be ignored, until the CUT has been processed.

2.2 Scheduling Preprocessing

In the state RUN of the controller:

Before each call of the CUT the COM provides the process data of the CPU to the CUT in a memory range defined by the CUT.

In the state STOP of the controller:

There is no process data exchange from the safe CPU to the COM.

2.3 Scheduling Postprocessing

In the state RUN of the controller:

After each call of the CUT the COM provides the process data of the CUT to the safe CPU.

In the state STOP of the controller:

There is no process data exchange from the COM to the safe CPU.

2.4 STOP_INVALID_CONFIG

If the COM is in the state STOP_INVALID_CONFIG then the CUT is not executed.
If the COM changes to the state STOP_INVALID_CONFIG and executes the CUT or the CUIT these functions will be terminated.

2.5 Signals of the CUT Interface (CPU<->CUT)

The user can define a not safety-related process data communication between the safe CPU and the COM (CUT).

Send direction	Max. size of the process data
COM->CPU	16375 bytes data (16384 bytes – 9 status bytes)
CPU->COM	16382 bytes data (16384 bytes – 2 control bytes)

Note: Beside the process data communication at the same time additional protocols (e.g. Modbus, Profibus-DP and Ethernet I/P) can run on the controller.
In total 16384 bytes of data can be sent and 16384 bytes of data can be received.
The size of the send and receive data can be arbitrarily divided between the protocols.

Process data exchange with COM User Task (CUT)

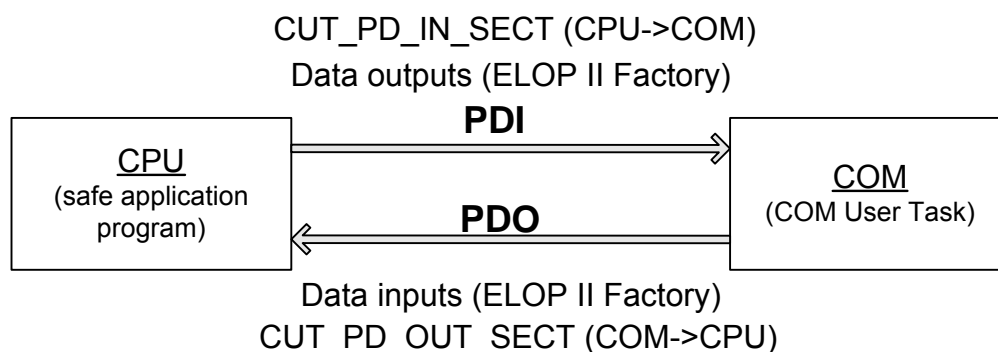


Figure 1: Process data exchange between CPU and COM (CUT)

All data types can be exchanged which are configurable as signal in ELOP II Factory.

The structure of the data has to be parameterized in ELOP II Factory.
The size of the data structures CUT_PDI and CUT_PDO (in the compiled C Code of the CUT) must fit to the same size of the configured data structure in ELOP II Factory.

Note If the data structures CUT_PDI and CUT_PDO are not available in the compiled c code or do not have the same size as the data structure of the parameterized process data in ELOP II Factory then the configuration is invalid. The COM gets in the state STOP_INVALID_CONFIG.
The process data communication is only running in the state RUN.

In the dialog *Signal Connections* of the COM User Tasks there are the tabs for the process data communication. These tabs are described in the following chapters.

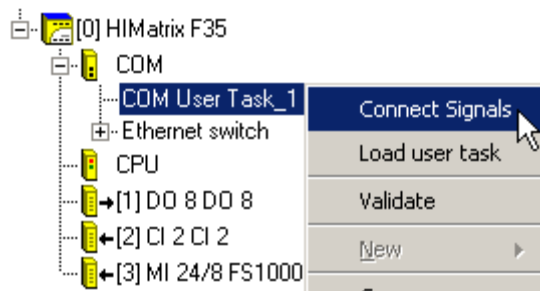


Figure 2: Context menu of the COM User Task

2.5.1 State

The tab **State** contains the following system parameters for state monitoring of the COM (CUT):

Name	Type	Function
Execution time	DWORD	Execution time of the COM User Task in μ s
Real schedule interval	DWORD	Time between two COM User Task cycles in ms
State of the User Task	BYTE	1 = RUNNING (CUT runs) 0 = ERROR (CUT does not run because of an error)

Table 4: Signals in the tab „State“ of the COM User Task

2.5.2 Control

The tab **Control** contains the system parameter *Control of user task State*.

The following table shows the variants how the user can control the COM User Task with the system parameter *Control of user task State*:

Control the user task State (WORD)		
Function	Value	Description
DISABLED	0x8000	The application program locks the CUT (CUT is not started).
AUTOSTART (Default)	0	After termination a new start of the CUT is automatically allowed if the error is eliminated
TOGGLE_MODE_0	0x0100	After termination of the CUT a new start of the CUT is only allowed after writing TOGGLE_MODE_1.
TOGGLE_MODE_1	0x0101	After termination of the CUT a new start of the CUT is only allowed after writing TOGGLE_MODE_0.

Table 5: Signals in tab „Control“ of the COM User Task

2.5.3 Input Records (COM->CPU)

In the tab **Input Records** those signals are enlisted which shall be transferred from the COM (CUT) to the CPU (input range of the CPU). There the data types and the size of the data structure (of the input range of the CPU) are configured.



Please regard that the not safety-related signals of the COM User Task must not disturb the safety functions of the CPU application program.

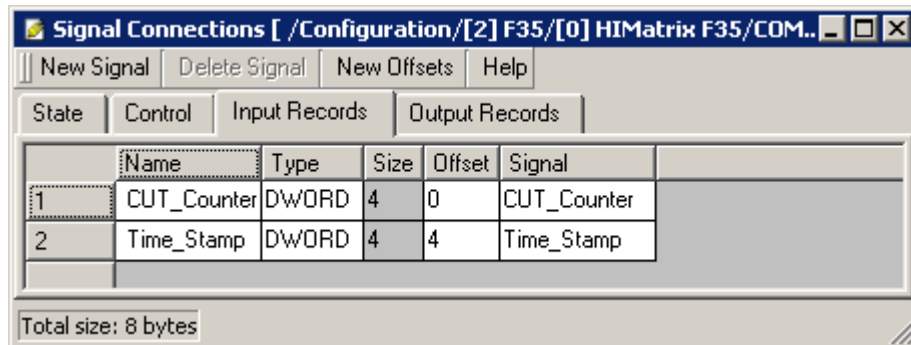


Figure 3: Tab „Input Records“ of the COM User Task

Required entry in the c code

The C code of the COM User Tasks must have the following data structure CUT_PDO for the outputs of the COM (input range of the CPU):

```
/* ELOP II Factory Input Records (COM->CPU) */
udword CUT_PDO[2] __attribute__((section("CUT_PD_OUT_SECT"), aligned(1)));
```

The size of the data structure CUT_PDO has to fit to the size of the configured data inputs in ELOP II Factory.

2.5.4 Output Records (CPU->COM)

In the tab **Output Records** those signals are enlisted which shall be transferred from the CPU (output range of the CPU) to the COM (CUT). There the data types and the size of the data structure (of the output range of the CPU) are configured.

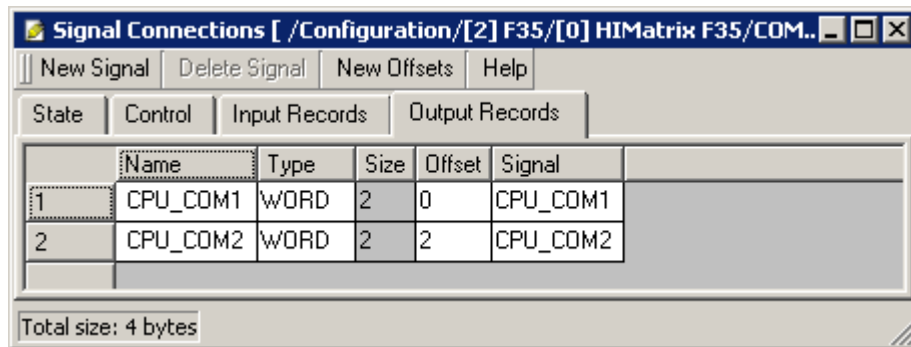


Figure 4: Tab „Output Records“ of the COM User Task

Required entry in the C code

The C code of the COM User Tasks must have the following data structure CUT_PDI for the inputs of the COM (output range of the CPU):

```
/* ELOP II Factory Output Records (CPU->COM) */
uword CUT_PDI[2] __attribute__((section("CUT_PD_IN_SECT"), aligned(1)));
```

The size of the data structure CUT_PDI has to fit to the size of the configured data outputs in ELOP II Factory.

3 CUT Functions

3.1 COM User Callback Functions

The COM user callback functions have all the Prefix "**CUCB _**" and are invoked directly from the COM when an event occurs.

Note All COM user callback functions must be defined in the C code of the user!

The COM user callback (CUCB) and the COM user library (CUL) functions share the same code and data memories and the stack. These functions guarantee mutually the consistency of the common used data (variables). The only exception represents the function `CUCB_IrqService()`, which possesses its own stack. The data (variable) to be used together by the CUCB/CUL functions and the function `CUCB_IrqService()` must be protected with semaphores.

3.2 COM User Library Functions

All COM user library functions and variables have the Prefix "**CUL _**" and are invoked in the CUT. This CUL functions are available all over the object file **libcut.a**.

3.3 Header Files

The two header files **cut.h** and **cut_types.h** contain all function prototypes for CUL/CUCB and the associated data types and constants.

The following short cuts for the data types are defined in the header file **cut_types.h**:

```
typedef unsigned long  udword;
typedef unsigned short uword;
typedef unsigned char  ubyte;
typedef signed long    dword;
typedef signed short   word;
typedef signed char    sbyte;
#ifndef HAS_BOOL
typedef unsigned char  bool; // with 0=FALSE, otherwise TRUE
#endif
```

3.4 Code/Data Area and Stack for the CUT

The code/data area is a coherent storage area which begins with the code segment and the initial data segment and continued with the data segments.

In the HIMA linker files (**makeinc.inc.app** and **section.dld**) the described sequence of the segments and the memory capacity are fixed.

The COM User Task optimally shares with the assistance of the HIMA linker files the available storage area between the code and the data.

Start address 0x790000

Length 448 kByte

The stack is located in a reserved storage area which is specified at runtime of the COM operating system.

End address Dynamically (by the view of the CUT)

Length 64 kByte

3.5 Start Function CUCB_TaskLoop

The function `CUCB_TaskLoop()` is the starting function to the COM User Task. The program execution of the COM User Task begins with the call of this function (see `Schedule Interval[ms]` chapter 2.1).

Function prototype:

```
void CUCB_TaskLoop(udword mode)
```

Parameter:

The function has the following parameter:

Parameter	Description
mode	1 = MODE_STOP corresponds to the mode STOP_VALID_CONFIG 2 = MODE_RUN normal operating of the controller

3.6 Serial Interface RS485 / RS232 IF

The used fieldbus interfaces must be equipped with the corresponding plug-in modules (hardware).

Note	System documentation is available for each <i>HIMatrix</i> controller. (See system manual HI 800 141 and data sheets of the <i>HIMatrix</i> controllers).
-------------	---

3.6.1 CUL_AscOpen

The function `CUL_AscOpen()` initializes the entered serial interface (*comId*) with the passed parameters.

After the call of the function `CUL_AscOpen()` the COM immediately begins with the reception of data via this interface.

The received data is stored in a software FIFO with a size of 1kByte for **each** initialized serial interface.

The data is stored until the data is read out with the function `CUL_AscRcv()` of the CUT.

Note	If the read out of the data from the FIFO is slower than the reception of new data, then the new received data is discarded.
-------------	--

Function prototype:

```
udword CUL_AscOpen( Udword comId,
                    Ubyte duplex,
                    udword baudRate,
                    ubyte parity,
                    ubyte stopBits)
```

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
duplex	0 = Full-Duplex (only for FB4 if RS232 permitted) 1 = Half-Duplex
baudRate	1 = 1200 Bit 2 = 2400 Bit 3 = 4800 Bit 4 = 9600 Bit 5 = 19200 Bit 6 = 38400 Bit 7 = 57600 Bi 8 = 115000 Bit
The data bit length is firmly adjusted to 8 data bits. To these 8 data bits the start-, parity- and stop bits must be added.	
parity	0 = NONE 1 = EVEN 2 = ODD
stopBits	1 = 1 Bit 2 = 2 Bits

Return value:

An error code (udword) will be returned. The error codes are defined in the header file cut.h.

Error code	Description
CUL_OKAY	The interface was successfully initialized.
CUL_ALREADY_IN_USE	The interface is used by other functions of the COM or is already open.
CUL_INVALID_PARAM	Illegal parameters or parameter combinations were passed.
CUL_DEVICE_ERROR	Other errors

3.6.2 CUL_AscClose

The function `CUL_AscClose()` closes the serial interface entered in *comId*.
Thereby the already received data in the FIFO is deleted, when this data was not readout with the function `CUL_AscRcv()`.

Function prototype:

```
Udword CUL_AscClose(udword comId)
```

Parameter:

The function has the following parameter:

Parameter	Description
comId	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

Return value:

An error code (udword) will be returned. The error codes are defined in the header file *cut.h*.

Error code	Description
CUL_OKAY	The interface was successfully closed.
CUL_NOT_OPENED	The interface was not opened (by the CUT).
CUL_INVALID_PARAM	Illegal parameters or parameter combinations were passed.
CUL_DEVICE_ERROR	Other errors

3.6.3 CUL_AscRcv

The function `CUL_AscRcv()` instruct the COM to make available a defined data size from the FIFO.

As soon as the requested data is available (and the CUL and/or the scheduling permits), the COM calls the function `CUCB_AscRcvReady()`.

If there is not enough data in the FIFO then the function `CUL_AscRcv()` returns immediately.

The order for the data reception is stored until:

- The order was completely processed or
- the function `CUL_AscClose()` is invoked or
- redefined by a new order.

Note

Until the order is finished, the contents of **pBuf* may only be changed via the function `CUCB_AscRcvReady()`.

Function prototype:

```
Udword CUL_AscRcv(udword comId, CUCB_ASC_BUFFER *pBuf)
```

```
typedef struct CUCB_AscBuffer {

    bool  bAscState;    // for using by CUT/CUCB
    bool  bError;       // for using by CUT/CUCB
    uword align;        // COM is 4 aligned, long's are higher-performance
    udword mDataIdx;     // Byte offset in aData from there on the data are loacted
    udword mDataMax;     // max. Byte offset: (mDataMax-mDataIdx) indicates,
                        // how many bytes in aData have to be sent or received

    udword aData[1];    // Start point of the data copy range
}CUCB_ASC_BUFFER;
```

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
pBuf	Defines the requested amount of data and the location, to which the data should be copied, before CUCB_AscReady() is invoked. When already sufficient data is received in the FIFO, the function CUCB_AscRcvReady() is invoked during CUL_AscRcv().

Return value:

An error code (udword) will be returned. The error codes are defined in the header file cut.h.

Error code	Description
CUL_OKAY	If the order was successful, otherwise error code.
CUL_NOT_OPENED	If the interface was not opened by the CUT.
CUL_INVALID_PARAM	Illegal parameters or parameter combinations were passed.
CUL_DEVICE_ERROR	Other error

Restrictions:

- If the storage area (defined by CUCB_ASC_BUFFER) is not allocated in the data segment of the CUT, then the CUT and the CUIT are terminated.
- At maximum 1024 byte data can be requested.

3.6.4 CUCB_AscRcvReady

If the COM calls the function `CUCB_AscRcvReady()`, then the requested amount of data is ready in the FIFO (data from the serial interface defined in the parameter `comId`).

The data was requested with the function `CUL_AscRcv()`.

The call of the function `CUCB_AscRcvReady()` can be done from outside and during the invoke of the function `CUL_AscRcv()`. The task context is always the task of the CUT.

The function `CUCB_AscRcvReady()` may invoke all CUT library functions.

Also permitted is:

- the increase of `mDataMax`, and/or
- the new parameterization of `mDataIdx` and `mDataMax` with `*.pBuf` data assigned by the `comId` (to reading on)

The structure element of `CUCB_ASC_BUFFER.mDataIdx` has the value of `CUCB_ASC_BUFFER.mDataMax`.

Function prototype:

```
void CUCB_AscRcvReady(udword comId)
```

Parameter:

The function has the following parameter:

Parameter	Description
<code>comId</code>	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

Restrictions:

If the storage area (defined by `CUCB_ASC_BUFFER`) is not located in the data segment of the CUT, then the CUIT and CUT are terminated.

3.6.5 CUL_AscSend

The function `CUCB_AscSend` sends the data set defined by the parameter `pBuf` over the serial interface `comId`.

The defined data set must be ≥ 1 byte and ≤ 1 kByte.

After sending the function `CUCB_AscSendReady()` is invoked.

In the event of an error

- it will not be sent
- the function `CUCB_AscSendReady()` will not be invoked.

Function prototype:

```
Udword CUL_AscSend( udword comId, CUCB_ASC_BUFFER *pBuf )
```

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
pBuf	Defines the data amount to be sent

Return value:

An error code (udword) will be returned. The error codes are defined in the header file `cut.h`.

Error code	Description
CUL_OKAY	If the sending was successfully
CUL_WOULDBLOCK	If a message sent before was not sent yet
CUL_NOT_OPENED	If the interface was not opened by the CUT
CUL_INVALID_PARAM	Illegal parameters or parameter combinations were passed.
CUL_DEVICE_ERROR	Other errors

Restrictions:

If the storage area (defined by `CUCB_ASC_BUFFER`) is not located in the data segment of the CUT, then the CUIT and CUT are terminated.

3.6.6 CUCB_AscSendReady

If the COM invokes the function `CUCB_AscSendReady()`, then the transmitting of the data with the function `CUCB_AscSend()` over the serial interface is completed.

The task context is always the task of the CUT. The function `CUCB_AscSendReady()` may invoke all CUT library functions.

Function prototype:

```
void CUCB_AscSendReady(udword comId)
```

Parameter:

The function has the following parameter:

Parameter	Description
comId	Field bus Interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

3.7 UDP/TCP-Socket-IF

In maximum 8 sockets are available simultaneously independent from the used protocol.
The physical connection runs over the 10/100BaseT Ethernet interfaces of the controller.

3.7.1 CUL_SocketOpenUdpBind

The function `CUL_SocketOpenUdpBind()` creates a socket of the type UDP and binds the socket to the selected port.

The address for the binding is always `INADDR_ANY`, i.e. all messages for UDP/port addressed to the COM are received. Sockets run always in the non-blocking mode, i.e. this function will not block.

Function prototype:

```
dword CUL_SocketOpenUdpBind( uword port, uword *assigned_port_ptr )
```

Parameter:

The function has the following parameters:

Parameter	Description
port	A free port number not reserved by the COM ≥ 0 . If the parameter port = 0, then the socket is bound to the first free port.
assigned_port_ptr	Address to which the bounded port number shall be copied, if the parameter port = 0 or NULL if not

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
Socket number	Assigned socket number for UDP if > 0 ; Error codes are < 0
CUL_ALREADY_BOUND	Binding to a port for UDP not possible
CUL_NO_MORE_SOCKETS	No more resources for the socket available
CUL_SOCK_ERROR	Other socket errors

Restrictions:

If the CUT does not possess `assigned_port_ptr` then CUT/CUIT are terminated.

3.7.2 CUL_SocketOpenUdp

The function `CUL_SocketOpenUdp()` creates a socket of the type UDP without binding to a port. Afterwards messages can only be sent over the socket, but not received.

Function prototype:

```
dword CUL_SocketOpenUdp ( void )
```

Parameter:

no

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
Socket number	Assigned socket number for UDP if > 0; Error codes are < 0
CUL_NO_MORE_SOCKETS	No more resources for the socket available
CUL SOCK_ERROR	Other socket errors

3.7.3 CULNetMessageAlloc

The function `CULNetMessageAlloc()` allocates message memory for the using of

- `CUL_SocketSendTo()` at UDP and
- `CUL_SocketSend()` at TCP

In maximum 10 messages can be used simultaneously in the CUT.

Function prototype:

```
void *CULNetMessageAlloc(udword size, ubyte proto)
```

Parameter:

The function has the following parameters:

Parameter	Description
size	Needed storage size in bytes, must be ≥ 1 byte and ≤ 1400 byte
proto	0 = TCP 1 = UDP

Return value:

Buffer address to which the data to send must be copied. Memory ranges outside of the allocated area must never be written. There are no ranges for the used transport protocols (Ethernet/IP/UDP or TCP).

Restrictions:

If there are no more memory resources or if the parameter size is too big or `proto > 1`, then the CUT and the CUIT are terminated.

3.7.4 CUL_SocketSendTo

The function `CUL_SocketSendTo()` sends the message allocated and filled with the function `CULNetMessageAlloc()` as UDP package to the target address `destIp/destPort`. After sending the message memory `pMsg` is released automatically. Before each sending the message memory must be allocated with the function `CULMessageAlloc()`.

Function prototype:

```
dword CUL_SocketSendTo( dword socket,
                        void *pMsg,
                        udword size,
                        udword destIp,
                        uword destPort)
```

Parameter:

The function has the following parameters:

Parameter	Description
Socket	Created socket built with <code>CUL_SocketOpenUdp()</code>
pMsg	Allocated memory of the UDP user data created with <code>CULNetMessageAlloc()</code>
Size	Memory size in bytes, must be \leq allocated memory
destIp	Target address $\neq 0$, also <code>0xffffffff</code> is allowed as broadcast
destPort	Target port $\neq 0$

Return value:

An error code (udword) will be returned.
The error codes are defined in the header file `cut.h`.

Error code	Description
CUL_OKAY	Message successfully sent
CUL_NO_ROUTE	No routing available to obtain <code>destIp</code>
CUL_WRONG SOCK	Wrong socket type or socket not available
CUL_SOCKET_ERROR	Other socket error

Restrictions:

If the CUT does not possess the message `pMsg` or if the size for `pMsg` is too big, then the CUT and the CUIT are terminated.

3.7.5 CUCB_SocketUdpRcv

The COM invokes the function CUCB_SocketUdpRcv() if data from the socket is available. In the callback the data must be copied on demand from *pMsg to CUT data. After the return of the function an access to *pMsg is not more allowed.

Function prototype:

```
void CUCB_SocketUdpRcv( dword socket,
                        void *pMsg,
                        udword packetLength,
                        udword dataLength)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket created with CUL_SocketOpenUdp()
pMsg	pMsg points to the begin of the UDP package incl. the Ethernet header. Via the Ethernet header the transmitter of the message can be identified.
packetLength	The length of the package is stored in packetLength, incl. the length of the header.
dataLength	The length of the UDP user data part is stored in dataLength.

3.7.6 CUL_NetMessageFree

The function CUL_NetMessageFree() releases the message allocated before with CUL_NetMessageAlloc().

Normally this function is not necessary because by the call of the function CUL_SocketSendTo() there is an automatic release.

Function prototype:

```
void CUL_NetMessageFree(void *pMsg)
```

Parameter:

The function has the following parameter:

Parameter	Description
pMsg	Memory reserved by CUL_NetMessageAlloc()

Restrictions:

If the CUT does not possess the message pMsg the CUT/CUIT are terminated.

3.7.7 CUL_SocketOpenTcpServer

The function `CUL_SocketOpenServer()` creates a socket of type TCP and binds the socket to the selected port.

The address for binding always is `INADDR_ANY`. Additionally the COM has to make a listen on the stream socket. Sockets are always running in the non-blocking mode, e.g. this function does not block.

For further using of the socket see `CUCB_SocketTryAccept()` and `CUL_SocketAccept()`.

Function prototype:

```
dword CUL_SocketOpenTcpServer(udword port, udword backlog)
```

Parameter:

The function has the following parameters:

Parameter	Description
port	Not used port number by the COM > 0
backlog	Max. number of waiting connections for socket

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
Socket number	Already allocated socket number for UDP if > 0 Error codes are < 0
CUL_ALREADY_BOUND	Binding to a port/proto not possible
CUL_NO_MORE_SOCKETS	No more resources for socket available
CUL SOCK_ERROR	Other socket errors

Restrictions:

If successfully one socket is used.

3.7.8 CUCB_SocketTryAccept

The COM invokes the function `CUL_SocketTryAccept()` if a TCP connection demand is pending.

With this request a socket can be created with the function `CUL_SocketAccept()`.

Function prototype:

```
void CUCB_SocketTryAccept(dword serverSocket)
```

Parameter:

The function has the following parameters:

Parameter	Description
serverSocket	Socket created by <code>CUL_SocketOpenTcpServer()</code> .

3.7.9 CUL_SocketAccept

The function `CUL_SocketAccept ()` creates a new socket, that was signalized with the connection demand `CUCB_SocketTryAccept()` before.

Function prototype:

```
dword CUL_SocketAccept( dword serverSocket,  
                        udword *pIpAddr,  
                        uword *pTcpPort )
```

Parameter:

The function has the following parameters:

Parameter	Description
serverSocket	with <code>CUCB_SocketTryAccept()</code> signalized serverSocket
pIpAddr	Address to which the IP address of the peer shall be copied or 0 if not
pTcpPort	Address to which the TCP port number of the peer shall be copied or 0 if not

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
socket	if > 0, new created Socket
CUL_WRONG_SOCKET	Wrong socket type or socket not available
CUL_NO_MORE_SOCKETS	There are no more socket resources available
CUL_SOCKET_ERROR	Other socket errors

Restrictions:

If the CUT does not possess the messages `pIpAddr` and `pTcpPort` then the CUT/CUIT are terminated.

3.7.10 CUL_SocketOpenTcpClient

The function `CUL_SocketOpenTcpClient()` creates a socket of the type TCP with free local port and assigns a connection to `destIp` and `destPort`. Sockets are always running in the non-blocking mode, e.g. this function does not block. If the connection has been established the function `CUCB_SocketConnected()` is invoked.

Function prototype:

```
dword CUL_SocketOpenTcpClient(udword destIp, uword destPort)
```

Parameter:

The function has the following parameters:

Parameter	Description
destIp	IP address of the communication partners
destPort	Port number of the communication partners

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
Socket number	if > 0; error codes are < 0
CUL_NO_MORE_SOCKETS	no more resources available for socket
CUL_NO_ROUTE	no routing available to reach destIp
CUL SOCK_ERROR	Other socket errors

3.7.11 CUCB_SocketConnected

The function `CUCB_SocketConnected()` is invoked by the COM if a TCP connection was established with the function `CUL_SocketOpenTcpClient()`.

Function prototype:

```
void CUCB_SocketConnected(dword socket, bool successfully )
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Created and assigned socket by <code>CUL_SocketOpenTcpClient()</code>
successfully	TRUE if the connection attempt was successfully, otherwise FALSE

3.7.12 CUL_SocketSend

The function `CUL_SocketSend()` sends the message filled and allocated by `CULNetMessageAlloc()` as TCP package.
After sending the message memory `pMsg` is released automatically.
With each sending first message memory has to be allocated with the function `CULMessageAlloc()`.

Function prototype:

```
DWORD CUL_SocketSend(  DWORD socket,
                      void *pMsg,
                      UDWORD size)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket created by <code>CUL_SocketAccept()</code> / <code>CUL_SocketOpenTcpClient()</code>
pMsg	memory for the TCP user data reserved by <code>CULNetMessageAlloc()</code>
size	Memory size in bytes, must be \leq the allocated memory

Return value:

An error code (DWORD) will be returned.
The error codes are defined in the header file `cut.h`.

Error code	Description
CUL_OKAY	Message was sent successfully
CUL_WRONG_SOCKET	Wrong socket type or socket not available
CUL_WOULD_BLOCK	Message cannot be sent because socket would be blocked otherwise
CUL_SOCKET_ERROR	Other socket error

Restrictions:

If the CUT does not possess the message `pMsg` or if the size for `pMsg` is too big then the CUT/CUIT are terminated.

3.7.13 CUCB_SocketTcpRcv

The function `CUCB_SocketTcpRcv ()` is invoked by the COM if the user data of the socket are placed.

After quitting the function `CUCB_SocketTcpRcv ()` an access to `*pMsg` is not allowed anymore.

If the user data are needed also outside the function `CUCB_SocketTcpRcv ()` the user data has to be copied from `*pMsg` in a prepared range.

Note If the TCP connection is cut asynchronous (after an error or due to a request from the other side) the function `CUCB_SocketTcpRcv ()` with `dataLength = 0` is selected.

 The CUT is invoked by the `CUCB_SocketTcpRcv ()` function with the purpose to newly synchronize the socket for the communication.

Function prototype:

```
void CUCB_SocketTcpRcv( dword socket,
                        void *pMsg,
                        udword dataLength)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket via which the user data was received.
pMsg	The parameter <code>pMsg</code> points to the start section of the user data without Ethernet /IP /TCP header.
dataLength	The length of the user data in bytes

3.7.14 CUL_SocketClose

The function `CUL_SocketClose ()` closes a socket created before.

Function prototype:

```
dword CUL_SocketClose(dword socket)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Created socket

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
CUL_OKAY	Socket closed and one socket resource free again.
CUL_WRONG_SOCKET	Socket not available

3.8 Timer-IF

3.8.1 CUL_GetTimeStampMS

The function `CUL_GetTimeStampMS()` provides a millisecond tick. This tick is suitable to implement own timer in the CUT/CUIT. The counter is derived from the quartz of the COM processor and therefore has the same precision.

Function prototype:

```
udword CUL_GetTimeStampMS(void)
```

3.8.2 CUL_GetDateAndTime

The function `CUL_GetDateAndTime()` provides the seconds since 1970 to the memory `*pSec` and the milliseconds to the memory `*pMsec`. The values are compared by the safe CPU and depending on the parameterization can be synchronized externally via SNTP¹. The values for `CUL_GetDateAndTime()` should **not** be used for time measurement, timer or something else because the values can be provided by the synchronization and/or by the user during operation.

Function prototype:

```
void CUL_GetDateAndTime(udword *pSec, udword *pMsec)
```

Restrictions:

If the memory of `pSec` or `pMsec` are not allocated in the CUT data segment then the CUT/CUIT are terminated.

¹ HIMatrix standard function

3.9 COM User IRQ Task (only at MHS31A)

The COM User IRQ Task (CUIT) shares the code and data memory with the CUT. It has an own stack and is configured as well as the CUT stack by the COM OS (regarded by the CUIT dynamically). The stack has a size of 8kByte. There is only one CUIT in the COM. Initially the IrqServices are disabled, e.g. after power on or after loading the configuration.

Restrictions:

From the CUIT only the CUL functions for the semaphore handling are invoked.

3.9.1 CUCB_IrqService

The function `CUCB_IrqService()` is invoked by the COM after triggering of one of the two possible CAN IRQs.

This function is responsible for the operation of the IRQ source `devNo` of the corresponding CAN chip and must take care that the CAN chip cancels its IRQ request.

Function prototype:

```
void CUCB_IrqService(udword devNo)
```

Restrictions:

The IRQ management of the COM processor is carried out by the COM OS and must not carried out by the function `CUCB_IrqService()`.

Note

The function `CUCB_IrqService()` must be implemented very efficiently to avoid unneeded latencies of the COM processor functions.

Otherwise it is possible that at high load of the COM processor functions cannot be carried out anymore. Thereby the safe communication of the safe CPU may be disturbed.

The CUT library enables the unlocking and switching off of the COM IRQ channel to which the CAN controller is connected.

3.9.2 CUL_IrqServiceEnable

The function `CUL_IrqServiceEnable()` enables the COM IRQ channel for the selected CAN Controller `devNo`. From now on CAN IRQs trigger the call of the CUT IRQ task.

Function prototype:

```
void CUL_IrqServiceEnable(udword devNo)
```

Parameter:

The function has the following parameter:

Parameter	Description
devNo	1 = CAN Controller A 2 = CAN Controller B

Restrictions:

If `devNo` values uneven to 1 or 2 are used then the CUIT/CUT are terminated.

3.9.3 CUL_IrqServiceDisable

The function `CUL_IrqServiceDisable()` locks the COM IRQ channel for the CAN controller *devNo*. From now on the CAN IRQs do not trigger the call for the CUT IRQ task. Not yet processed IRQ treatments are nevertheless carried out.

Function prototype:

```
void CUL_IrqServiceDisable(udword devNo)
```

Parameter:

The function has the following parameter:

Parameter	Description
devNo	1 = CAN Controller A 2 = CAN Controller B

Restrictions:

If *devNo* values uneven to 1 or 2 are used then the CUIT/CUT are terminated.

3.9.4 CUL_DeviceBaseAddr

The function `CUL_DeviceBaseAddr()` provides the 32 bit basic address of the CAN controllers.

Function prototype:

```
void* CUL_DeviceBaseAddr(udword devNo)
```

Parameter:

The function has the following parameter:

Parameter	Description
devNo	1 = CAN Controller A 2 = CAN Controller B

Restrictions:

If *devNo* values uneven to 1 or 2 are used then the CUIT/CUT are terminated.

3.10 NVRam IF (only at MHS31A)

The CUT and the CUIT can read and write the available range.
The NVRam reserved for this function has a size of 9kBytes.

Note	The COM does not take care for the consistency of the data in an access during a drop out of the operation voltage and also not during a simultaneous access from two tasks.
-------------	---

3.10.1 CUL_NVRamWrite

The function `CUL_NVRamWrite()` writes data in the NVRam.

Function prototype:

```
void CUL_NVRamWrite(udword offset, void *source, udword size)
```

Parameter:

The function has the following parameters:

Parameter	Description
offset	In the range of the NVRam, valid values are 0 – 9215
source	Memory range in CUT data segment which shall be copied into the NVRam.
size	Number of bytes which shall be copied

Restrictions:

At invalid parameters the CUT and the CUIT are terminated, e.g. at:

- offset ≥ 9216
- offset+size > 9216
- source not within CUT data segment
- source+size not in CUT data segment

3.10.2 CUL_NVRamRead

The function `CUL_NVRamRead()` reads data from the NVRam.

Function prototype:

```
void CUL_NVRamRead(udword offset, void *destination, udword size)
```

Parameter:

The function has the following parameters:

Parameter	Description
offset	In the range of the NVRam, valid values are 0 – 9215
destination	Memory range in CUT data segment which shall be copied into the NVRam.
size	Number of bytes which shall be copied

Restrictions:

At invalid parameters the CUT and the CUIT are terminated, e.g. at:

- offset ≥ 9216
- offset+size > 9216
- destination not within CUT data segment
- destination+size not in CUT data segment

3.11 Semaphore IF (only at MHS31A)

The CUT and the CUIT have **one** common semaphore for process synchronization. The data of the CUCB and CUL functions which are used in common with the function `CUCB_IrqService()` must be protected via a semaphore. Thereby the consistency of the data used in common with the function `CUCB_IrqService()` is guaranteed.

3.11.1 CUL_SemaRequest

The function `CUL_SemaRequest()` demands the semaphore of the CUT/CUIT.

If the semaphore

- is free the function returns with the value `pContext`.
- is not free the invoked task is blocked until the semaphore is released by another task and returns with the value `pContext`.

The context which is referenced by the parameter `pContext` is only used for the invoked task by the CUL functions and must not be changed between request and release.

Function prototype:

```
void CUL_SemaRequest(udword *pContext)
```

Parameter:

The function has the following parameter:

Parameter	Description
<code>pContext</code>	Only used by the function CUL within the invoking task. The context is returned by <code>pContext</code> and must be noted at the function <code>CUL_SemaRelease()</code> .

Restrictions:

If the number of permitted recursions is exceeded then the CUT/CUIT are terminated.

If the CUT is blocked by a semaphore, no more CUCB_'s are executed with the exception of `CUCB_IrqService()`.

3.11.2 CUL_SemaRelease

The function `CUL_SemaRelease()` releases again the semaphore defined by `*pContext`.

Function prototype:

```
void CUL_SemaRelease(udword *pContext)
```

Parameter:

The function has the following parameter:

Parameter	Description
<code>pContext</code>	With the same value for <code>*pContext</code> as written by <code>CUL_SemaRequest</code> or <code>CUL_SemaTry</code> .

Restrictions:

If Release is invoked more times than Request/Try, then the CUT/CUIT are terminated.

3.11.3 CUL_SemaTry

The function `CUL_SemaTry()` tries to request the semaphore of the CUT/CUIT.

If the semaphore

- is free the function returns with the value `TRUE` and reserves the semaphore.
- is not free the function returns with the value `FALSE` and does not reserve the semaphore.

The udword referenced by `pContext` is only used by CUL for the invoked task and must not be changed between Request/Release.

Function prototype:

```
bool CUL_SemaTry(udword *pContext)
```

Parameter:

The function has the following parameter:

Parameter	Description
<code>pContext</code>	Is only used by CUL within the invoking task

Return value:

An error code (udword) will be returned.

The error codes are defined in the header file `cut.h`.

Error code	Description
<code>TRUE</code>	Semaphore could be reserved
<code>FALSE</code>	Semaphore could not be reserved

Context is returned via `pContext` and must be noted by the function `CUL_SemaRelease`.

Restrictions:

If the number of admissible recursions is exceeded then the CUT/CUIT are terminated.

If the CUT is blocked by a semaphore no more CUCB_'s are executed with the exception of the function `CUCB_IrqService()`.

Note	The functions <code>CUL_SemaTry()</code> and <code>CUL_SemaRequest()</code> can be invoked without blockade if the invoking task already has reserved the semaphore; the same number of invokes of the function <code>CUL_SemaRelease</code> have to be carried out until the semaphore is free again. The recursion allows 32000 steps in minimum. Whether more steps are possible depends on the corresponding version of the COM.
-------------	--

3.12 COM IO IF (only at MHS31A)

Interface for operation of the standard IO of the COM. The filtering of the DI channels is not parameterizable; in fact there are always filtered and not filtered values available.

```
struct CUL_IoBufferIn {
    uword mDIRaw;          //unfiltered DI values,
                           //with Bit[0]=DI[1]..Bit[11]=DI[12]
                           //Bit[12]=DI[13]/DO[1]..Bit[15]=DI[16]/DO[4]

    uword mDIFilter;       //filtered DI values,
                           //with Bit[0]=DI[1]..Bit[15]=DI[16]
}CUL_IO_BUFFER_IN;

struct CUL_IoBufferOut {
    uword mDO;             // DO value 1=on, 0=off,
                           // with Bit[0]=DO[1] .. Bit[3]=DO[4]
}CUL_IO_BUFFER_OUT; // fitting to mDIDODir and Bit[4]..Bit[15]=0

struct CUL_IoBufferCfg {
    uword mDIDODir;        // value 1=output, 0=input,
                           // with Bit[0]=DI/DO[1]..Bit[3]=DI/DO[4]
                           // Bit[4..15] undefined, must be 0
} CUL_IO_BUFFER_CFG;
```

The channels 1 to 4 are the parameterizable DI/DO channels. The channel numbers correspond to the DI as well as to the DO in the structures above. If a DI/DO channel is configured as output so its mDIRaw and mDIFilter is not valid.

3.12.1 CUL_IORead

Reads the inputs. The properties of the DI corresponding to the filter-/raw values are defined by the hardware specifications.

Function prototype:

```
void CUL_IORead(CUL_IO_BUFFER_IN *pIN)
```

Restrictions:

If the CUT does not possess the assigning pointer the CUT/CUIT are terminated.

3.12.2 CUL_IOWrite

Writes the outputs. Before CUL_IOWrite() is invoked for the first time 0 values are written at the outputs.

Function prototype:

```
void CUL_IOWrite(CUL_IO_BUFFER_OUT *pOUT)
```

Restrictions:

If the CUT does not possess the assigning pointer the CUT/CUIT are terminated.

3.12.3 CUL_IOConfigure

The function `CUL_IOConfigure()` configures the inputs/outputs. Before the first call of `CUL_IOConfigure()` the DI/DO channels are configured as input. `CUL_IOConfigure()` can be invoked several times to reconfigure the DI/DO channels. DI/DO channels whose configuration changes from

- Output to input don't put any energy outwards,
- Input to output put out the value defined with `CUL_IOWrite()`.

The COM_IO functions may be invoked several times by CUIT/CUT during a CUCB call.

Function prototype:

```
void CUL_IOConfigure(CUL_IO_BUFFER_CFG *pCFG)
```

Restrictions:

If the CUT does not possess the assigning pointer the CUT/UIT are terminated.

3.13 Diagnosis

The function `CUL_DiagEntry()` puts an entry for an event into the COM short time diagnosis. The event entry can be read out via the PADT.

Function prototype:

```
void CUL_DiagEntry( udword severity,
                   udword code,
                   udword param1,
                   udword param2)
```

Parameter:

The function has the following parameters:

Parameter	Description
severity	severity serves for the classification of events 0x45 ('E') == error, 0x57 ('W') == warning, 0x49 ('I') == information
code	The user defines the parameter code with an arbitrary number for the corresponding events. If the event occurs the number is displayed in the diagnosis.
param1, param2	Additional information about the event

4 Installation of the Development Environment

In this chapter the installation of the development environment and the creation of a COM User Task is described.

The development environment is implemented on the installation CD (see also chapter 1.1.2).

4.1 Installation of the Cygwin Environment

The Cygwin environment is necessary because the GNU C Compiler Tools only runs under the Cygwin environment. Cygwin environment must be installed under Windows 2000/XP.

Note Regard the preconditions for the installation in chapter 1.1. **Deactivate the virus scanner** on your PC, on which Cygwin shall be installed, to prevent problems at the installation of Cygwin.

Execute the following steps to install the Cygwin environment:

Step 1: Start the setup program for the installation of Cygwin:

- ☐ Copy the Cygwin installation archive `cygwin_2005-03-11` from the installation CD to your local hard disk (e. g. Drive C:\).
- ☐ Open the Cygwin index in the Windows Explorer
C:\ \ cygwin_2005-03-11.
- ☐ Start the installation of Cygwin with a double-click on the file **setup.exe**.
- ☐ Click on the button **Next** to start the setup.

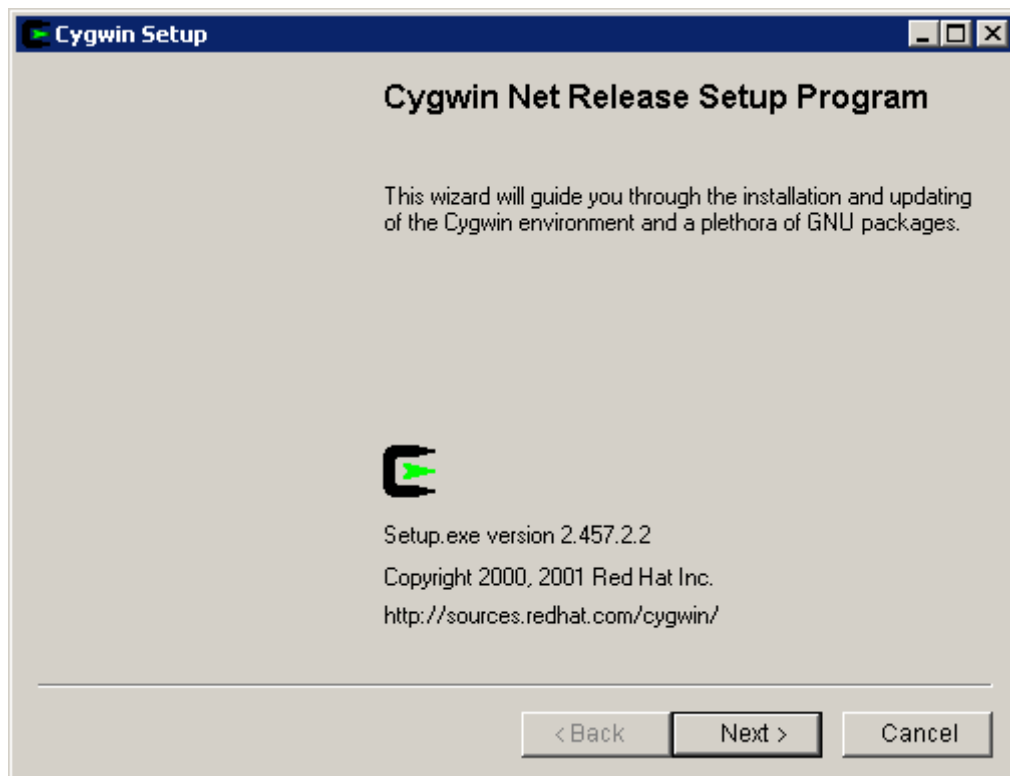


Figure 5: Cygwin setup dialog *Cygwin Setup*

Step 2:

The dialog *Disable Virus Scanner* appears if the virus scanner was **not** deactivated.

Follow these steps to deactivate the virus scanner during the installation of Cygwin.

Note	Deactivate the virus scanner before the Cygwin installation because dependent from the used virus scanner it can happen that the warning dialog does not appear although the virus scanner is running.
-------------	--

- ☐ Select **Disable Virus scanner** to prevent any problems caused by the virus scanner during the installation.
- ☐ Click on the button **Next** to confirm the input.

Step 3:

Select the installation source in the dialog *Choose Installation Type*:

- ☐ Select as installation source **Install from Local Directory**.
- ☐ Click on the button **Next** to confirm the input.

Step 4:

Select the installation target in the dialog *Choose Installation Directory* of Cygwin:

- ☐ Enter the directory in which Cygwin shall be installed.
- ☐ Take over all presettings of the dialog.
- ☐ Click on the button **Next** to confirm the input.

Step 5:

Select the Cygwin installation archive in the dialog *Select Local Package Directory*:

- ☐ Enter the name for the Cygwin installation archive in the field *Local Package Directory*. Select the archive with the installation files.
- ☐ Click on the button **Next** to confirm the input.

Step 6: Select all packages for the installation in the dialog *Select Packages*:

- ❑ Select the radio button **Curr**.
- ❑ Click slowly several times on the installation option beside **All** in the display field until **Install** is displayed for a complete installation of all packages (approx. 1.86 GB memory requirement).

Note Regard that behind each package **Install** is selected.
If the packages are not fully installed then important functions are missing for the compiling of the C code of the CUT!

- ❑ Click on the button **Next** to confirm the input.

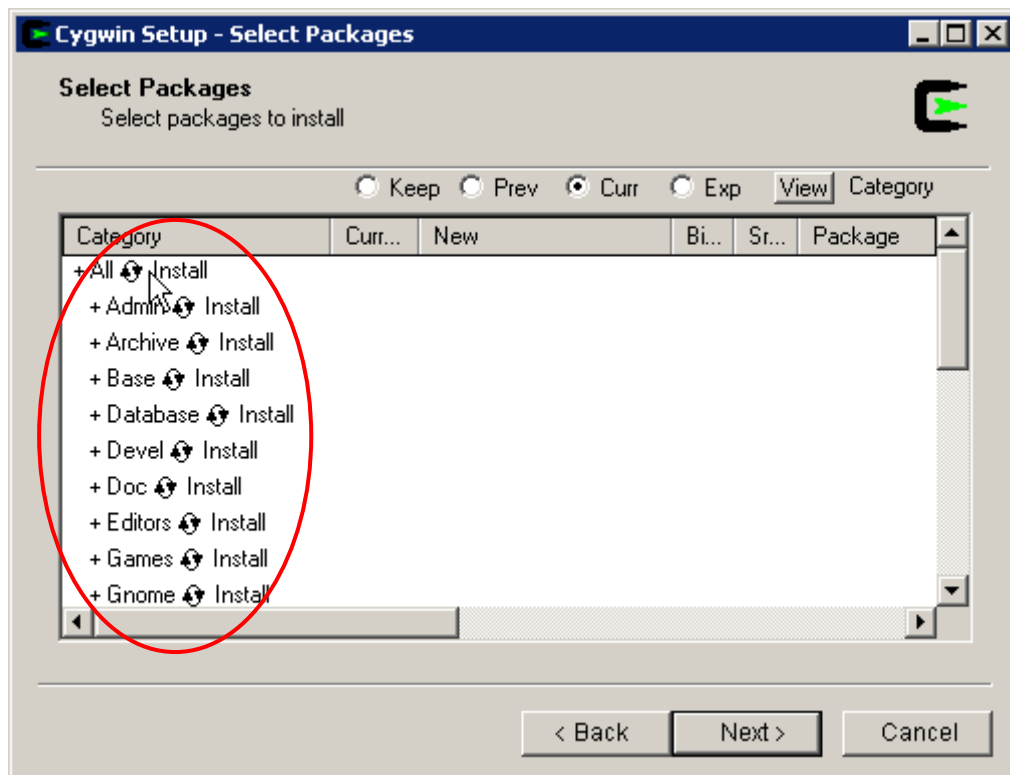


Figure 6: Cygwin setup dialog *Select Packages*

Step 7: Finish the Cygwin installation with the following entries:

- ❑ Select the entry into the **start menu**.
- ❑ Select the entry **desktop icon**.
- ❑ Click on the button **Finish** to finish the installation of Cygwin.

Cygwin commands	Description
cd (name of directory)	Change directory
cd ..	Change in higher level directory
ls -l	Display all files of a directory
help	Overview over Bash Shell commands

Table 6: Commands in Cygwin (Bash Shell)

4.2 Installation of the GNU Compiler

Execute the following steps to install the GNU Compiler:

- Open the directory of the installation CD in the Windows Explorer.
- Double-click on the zip-File `gcc-ppc-v3.3.2_binutils-v2.15.zip`.
- Extract all files in the Cygwin directory (e.g. `C:\cygwin\...`).
The GNU Compiler is extracted in the Cygwin directory in the folder **gcc-ppc**.
- Set the environment variables in the system control:
 - Open the system properties via the Windows start menu
Settings->System control->System.
 - Select the tab **Advanced**.
 - Click on the button **Environment variables**.
 - Select the system variable **Path** in the field *System variables* and extend the system variable with the entry: `C:\cygwin\gcc-ppc\bin`.

Copy the folder **cut_src** from the installation CD to the home directory.

The folder **cut_src** includes all necessary "include" and "lib" directories for the creation of a COM User Task. The additional source file **cut_cbf.c** is located in the directory

`..\cut_src\cutapp\`.

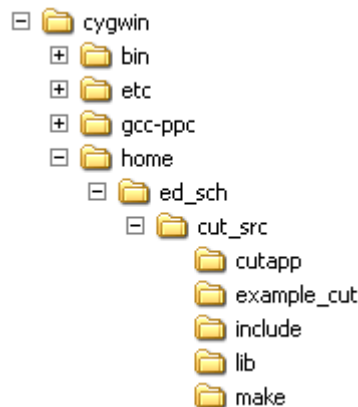


Figure 7: Structure tree of cygwin

Note If the home directory has not been created automatically then create the home directory with the Windows Explorer (e.g. `C:\cygwin\home\User1`).

If you want to create another home directory for the Cygwin Bash Shell, then you have to put the command `set Home` in the batch file `cygwin.bat`.

```
@echo off
```

```
C:
chdir C:\cygwin\bin
set Home=C:\User1
bash --login -i
```

Figure 8: Batch file *Cygwin.bat*

5 Create New CUT Project

This chapter shows how to create a new CUT project and which files hereby have to be adapted.

Note The CUT project **example cut** is fully adapted and located on the installation CD.

For creating new CUT projects it is recommended to create a new directory below the directory ...*cut*'s for each CUT project.

Example:

As a test the directory *example cut* is created, the C source is named **example_cut.c**, the created ldb file in the directory *make* is named **example_cut.ldb**.

Create the folder *example_cut* for the new COM User Task.

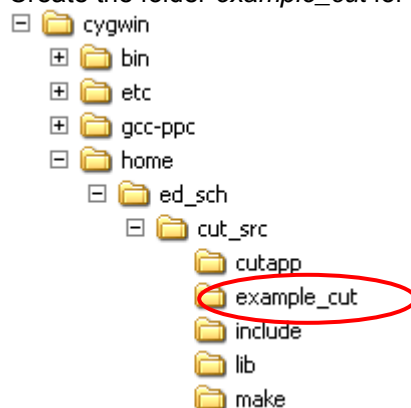


Figure 9: Structure tree of cygwin

- Copy the files
 - cutapp.c
 - cutapp.mke
 - makefile
 from the directory *cutapp* in the directory *example_cut*.
- The file names must be renamed into the loadable names (e.g. **cutapp** in **example_cut**).

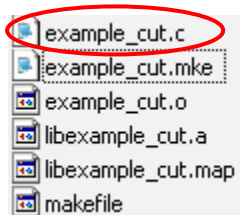


Figure 10: C code file in the folder *example_cut*

The changes in the mke file and the makefile must be executed for a new project, as described in the following chapters.

5.1 CUT Makefiles

Configuration of the CUT makefiles for different source files and ldb files.
In total three makefiles must be adapted as described in the following chapters.

5.1.1 Makefile with the appendix „.mke“

The mke file is located in the corresponding source code directory,
e.g. *cut_src\example_cut\example_cut.mke*.

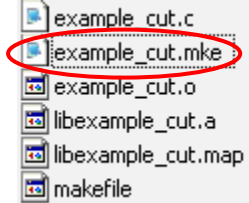


Figure 11: mke file in the folder example_cut

Make the following changes in the mke file:

- The variable **module** must have the same loadable name as the corresponding .mke file (e.g. example_cut).
- One or several c files which are needed for the creation of the target code (loadable file) are assigned to the variable **c_sources**.

```
#####
#
# make file (DOS/NT)
# $Id: cutapp.mke 58869 2005-10-11 12:35:46Z es_fp $
#####
#
# assign name of module here (e.g. nl for NetworkLayer)
module= example_cut
#
# assign module sources here
sources=

c_sources= $(module).c
asm_sources=
```

Figure 12: mke file from line 1 on

5.1.2 Makefile

The makefile file is located in the corresponding source code directory, e.g.
cut_src\example_cut\makefile

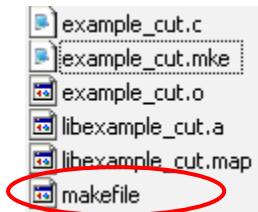


Figure 13: makefile in the folder *example_cut*

Make the following changes in the mke file:

- Put the include line for the mke file to the top and rename the .mke file to the present name.
- Expand the make call with the two variables **SUBMOD_DIRS** and **CUT_NAME**.

all_objects:

include example_cut.mke

cut:

\$(MAKE) -C ../make elf **SUBMOD_DIRS=cut_src/\$(module)** **CUT_NAME=\$(module)**

all_objects: \$(c_objects) \$(asm_objects) \$(objects)

Figure 14: makefile from line 34 on

5.1.3 Makefile with the Extension „makeinc.inc.app“

As a unique change for this and all following CUT projects the name of the CUT loadable will be made changeable via a make variable.

The makeinc.inc.app file is located in the cut_src directory
e.g. *cut_src\makeinc.inc.app*.

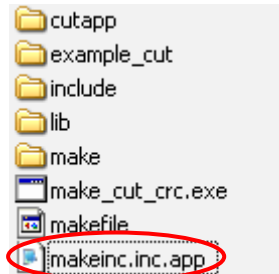


Figure 15: makeinc.inc.app file in the folder *example_cut*

Make the following changes in the makeinc.inc.app file:

- Expand the file with the variable **CUT_NAME**.

```
all: lib$(module).$(LIBEXT)
@echo 'did make for module ['lib$(module).$(LIBEXT)']'

lib$(module).$(LIBEXT): $(objects) $(c_objects) $(asm_objects) $(libraries)

SUBMOD2_LIBS=$(foreach lib,$(SUBMOD_LIBS),../$(lib))
```

CUT_NAME=cut

```
makeAllLibs:
$(MAKE) -C ../../cut_src cut_src
```

```
makeLoadable:
@echo; \
BGTYPE=" $(CUT_NAME)"; \
if [ ! -f $$BGTYPE.map ] ; then \
echo "Error: MAP-Datei $$BGTYPE.map existiert nicht"; \
exit 1; \
fi; \
OS_LENGTH=$(gawk '/__OS_LENGTH/ {print substr($$1,3,8)}' $$BGTYPE.map); \
echo; \
$(OBJCOPY) --strip-all --strip-debug -O binary $$BGTYPE.elf $$BGTYPE.bin; \
echo; \
echo "Building C3-Loadable-Binary ..."; \
$(MCRG) $$BGTYPE.bin 0 $$OS_LENGTH $$OS_LENGTH $$BGTYPE.ldb; \
echo; \

$(CUT_NAME).elf: makeAllLibs $(SUBMOD2_LIBS)

elf:
@echo; test -f section.dld && $(MAKE) $(CUT_NAME).elf && $(MAKE) makeLoadable \
|| { echo "ERROR: Wrong subdir. Please invoke elf target only from make/ subdirectory." && \
echo && false ; } ;
```

end of file: makeinc.inc

Figure 16: makeinc.inc.app from line 247 on

5.2 Adapting C Source Code

Execute the following steps to open the source code file:

- Open the project directory *cut_src/example_cut* which you have created and configured in the preceding steps.
- Open the C source code file with the extension *.c* with an editor (e.g. notepad).

5.2.1 Configure Input and Output Signals

Execute the following steps to configure the input and output signals in the source code file:

- The data size of the signals which shall be created in the tab **Data Outputs** in the ELOP II Factory Hardware Management must be set in the array **CUT_PDI[X]** in the source code file.
- The data size of the signals which shall be created in the tab **Data Inputs** in the ELOP II Factory Hardware Management must be set in the array **CUT_PDO[X]** in the source code file.

5.2.2 Start Function in the CUT

The C function `void CUCB_TaskLoop (udword mode)` is the start function and will be invoked cyclically by the application program.

5.2.3 Example Code „example_cut.c“

The following C code copies the value from the input **CUT_PDI[0]** to the output **CUT_PDO[0]** and returns the value unchanged to the ELOP II Factory application program.

Note The C code **example_cut.c** is located on the installation CD.

```
/* Example for the CUT implemetation */
#include "include/cut_types.h"
#include "include/cut.h"
#ifdef __cplusplus
extern "C" {
#endif
/*****
/* ELOP II Factory Output Records (CPU->COM) */
udword CUT_PDI[1] __attribute__ ((section("CUT_PD_IN_SECT"), aligned(1)));
/* ELOP II Factory Input Records (COM->CPU) */
udword CUT_PDO[1] __attribute__ ((section("CUT_PD_OUT_SECT"), aligned(1)));
*****/

/* Callback function for starting the CUT */
void CUCB_TaskLoop(udword mode)
{
    if (CUT_PDI[0] > CUT_PDO[0]) /*This is executed only, if the          */
    {                             /*ELOP II Factory application program    */
                                /*was processed.                  */
                                /*The ELOP II Factory application program*/
                                /*adds the value 1 to CUT_PDO[0] and        */
                                /*writes the result into CUT_PDI[0]      */
                                */

        CUT_PDO[0] = CUT_PDI[0]; /*Copies the value from input CUT_PDI[0] */
                                /*into output CUT_PDO[0] of the SPS        */
                                */
        if (CUT_PDO[0] == 65535)
            { CUT_PDO[0] = 0; }
    }
}
/*****
```

```

/*****
void CUCB_AscRcvReady(udword comId)
{
    CUL_DiagEntry(0x49, 1, comId, 0);
}
*****/
/*****
void CUCB_AscSendReady(udword comId)
{
    CUL_DiagEntry(0x49, 2, comId, 0);
}
*****/
/*****
void CUCB_SocketTryAccept(dword serverSocket)
{
    CUL_DiagEntry(0x49, 3, serverSocket, 0);
}
*****/
/*****
void CUCB_SocketConnected(dword socket, bool Okay)
{
    CUL_DiagEntry(0x49, 4, socket, Okay);
}
*****/
/*****
void CUCB_SocketTcpRcv(dword socket, void *pMsg, udword dataLength)
{
    CUL_DiagEntry(0x49, 5, socket, dataLength);
}
*****/
/*****
void CUCB_SocketUdpRcv(dword socket, void *pMsg, udword packetLength,
                      udword dataLength)
{
    CUL_DiagEntry(0x49, 6, socket, dataLength);
}
*****/
/*****
void CUCB_IrqService(udword devNo)
{
    CUL_DiagEntry(0x49, 7, devNo, 0);
}
*****/

#ifdef __cplusplus
} /* end extern "C" */
#endif

/* end of file */

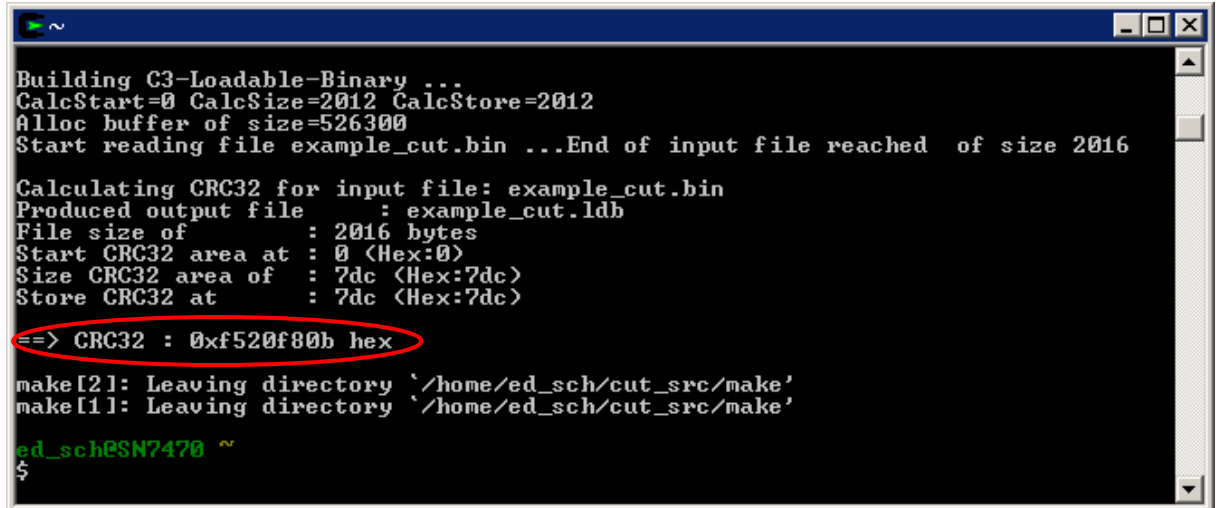
```

Figure 17: C code example_cut.c

5.2.4 Creation of Executable Code (ldb file)

Execute the following steps for creation of executable code (ldb file):

- Start the **Cygwin Bash Shell**.
- Change to the directory `/cut_src/example_cut/`.
- Start the code generation with the command `make cut`.
The binary file **cut.ldb** in the directory `/cut_src/make/` will be generated automatically.
- If the CRC32 was generated then also executable code was generated (see red marking Figure 18).



```
Building C3-Loadable-Binary ...
CalcStart=0 CalcSize=2012 CalcStore=2012
Alloc buffer of size=526300
Start reading file example_cut.bin ...End of input file reached of size 2016

Calculating CRC32 for input file: example_cut.bin
Produced output file      : example_cut.ldb
File size of              : 2016 bytes
Start CRC32 area at      : 0 (Hex:0)
Size CRC32 area of       : 7dc (Hex:7dc)
Store CRC32 at           : 7dc (Hex:7dc)
==> CRC32 : 0xf520f80b hex
make[2]: Leaving directory `/home/ed_sch/cut_src/make'
make[1]: Leaving directory `/home/ed_sch/cut_src/make'
ed_sch@SN7470 ~
$
```

Figure 18: Cygwin Bash Shell

This executable code (ldb file) will be loaded in the COM User Task (see chapter 5.3).

5.3 Implement the COM User Task in the Project

Execute the following steps in ELOP II Factory to implement the COM User Task in the project:

5.3.1 Create COM User Task

- Select the resource in which the CUT shall be created within ELOP II Factory Hardware Management.
- Select **New->COM User Task** in the context menu to create the COM User Task.

Note Always only one COM User Task per resource can be created.

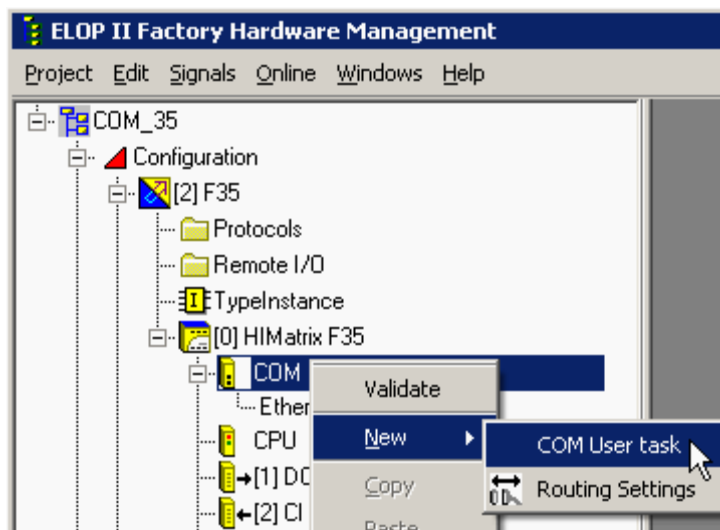


Figure 19: Creation of a COM User Task

5.3.2 Loading Program Code into the Project

- Select the menu item **Load User Task** within the menu context. A standard dialog opens for loading files.
- Open the directory `.../cut_src/make/`
- Select the ldb file which shall be executed in the COM User Task.

Note By loading the executable code (ldb file) anew different versions of the ldb file can be integrated. The contents of the ldb file is not proved on correctness during loading. The ldb file is afterwards compiled together with the resource configuration in the project and can then be loaded into the controller. If the ldb file is changed the project has to be compiled and loaded anew.

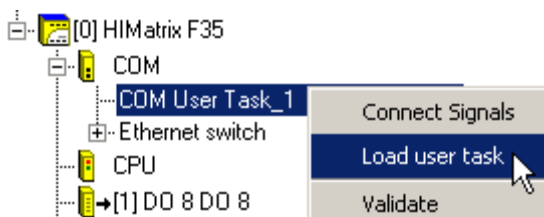


Figure 20: Loading a COM User Task

5.3.3 Connect Signals to the CUT

The user can define a not safety-related process communication between the safe CPU and the not safe COM (CUT). Hereby in maximum 16kByte data per direction can be exchanged.

5.3.3.1 Create Signals in the Signal Editor

Create the following two signals in the signal editor:

Signal	Type
COM_CPU	UINT
CPU_COM	UINT

5.3.3.2 CUT Dialog „Signal Connections“

Select the menu item **Connect Signals** in the context menu of the **COM User Task** to open the dialog *Signal Connections*.

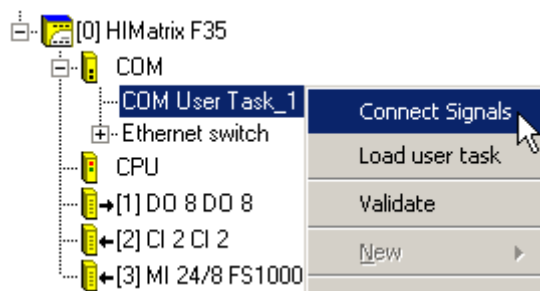


Figure 21: Open the dialog window *Signal Connections*

5.3.3.3 Configuring the Input Signals (COM->CPU)

In the tab **Input Records** signals are entered which are transferred from the COM to the CPU (Input Records).

- Select the tab **Input Records**.
- Select the signal from the signal editor and drag the signal per Drag & Drop in the tab **Input Records** of the dialog window *Signal Connections*.
- Click on the button **New Offsets** within the dialog window „Signal Connections“.
- Click on the button **Numbering** in the pop-up window *Numbering Offsets*.
- Close the dialog window.

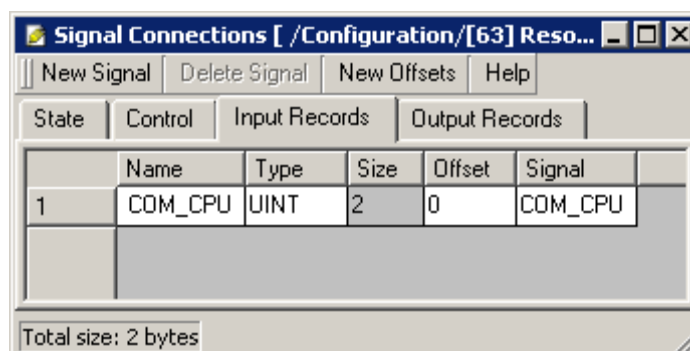


Figure 22: Tab *Input Records* of the COM User Task

5.3.3.4 Configuring the Output Signals (CPU->COM)

In the tab **Output Records** signals are entered which are transferred from the CPU to the COM (Output Records).

- Select the tab **Output Records**.
- Select the signal from the signal editor and drag the signal per Drag & Drop in the tab **Output Records** of the dialog window *Signal Connections*.
- Click on the button **New Offsets** within the dialog window „Signal Connections“.
- Click on the button **Numbering** in the pop-up window *Numbering Offsets*.
- Close the dialog window.

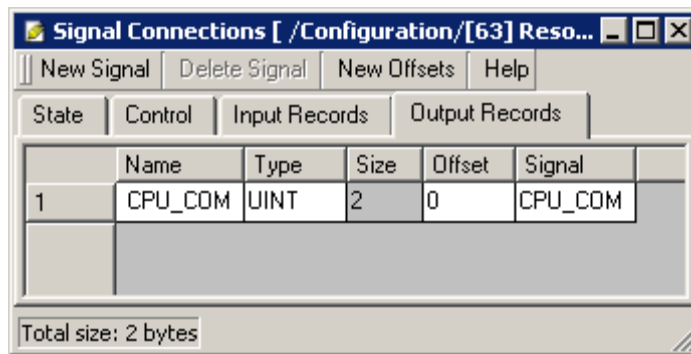


Figure 23: Tab *Output Records* of the COM User Task

5.3.4 Creation of the ELOP II Factory Application Program

Create the following application program in the project management:

- Open the *Program Instance* in the project management.
- Drag the signals **COM_CPU** and **CPU_COM** per Drag & Drop from the signal editor into the application program.
- Create the application program as displayed in the following figure.
- Close the *program instance*.

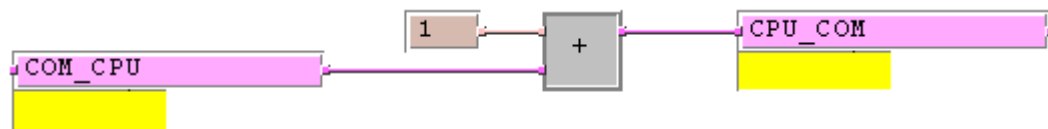


Figure 24: ELOP II Factory application program

Function of the ELOP II Factory application program:

The ELOP II Factory application program adds the value **1** to the signal **COM_CPU** (Input Records) and passes the result to the signal **CPU_COM** (Output Records).

With the next CUT call (see chapter 5.3.5) the signal **CPU_COM** will be passed to the CUT function (see chapter 5.2.3).

The COM User Task receives the signal **CPU_COM** and returns the value unchanged with the signal **COM_CPU**.

5.3.5 Configuring the Schedule Interval [ms]

- Select the menu item **Properties** in the context menu of the **COM User Task** to open the dialog *Properties*.
- The value in which intervals the COM User Task shall be invoked can be entered in the input field *Schedule Interval [ms]*.

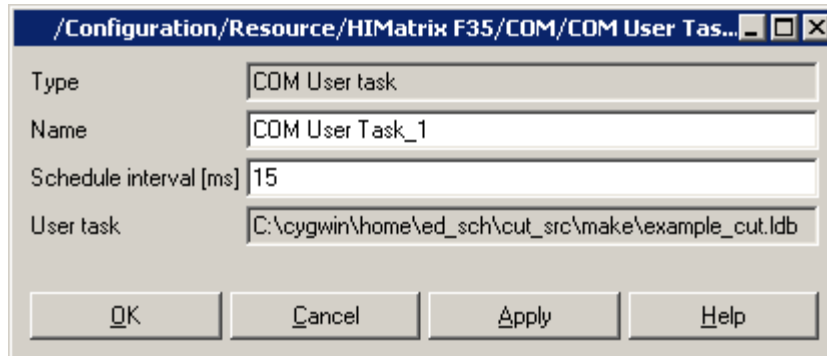


Figure 25: Dialog *Properties* of the COM User Task

5.3.6 Creation and Loading of the Code into the Controller

- Start the code generator.
- Ensure that the code was generated error-free (see Error State Viewer).
- Open the Control Panel.
- Load the code into the controller.
- Start the controller.

5.3.7 Test the COM User Task

Test the COM User Task with the Online Test:

- Change into the ELOP II Factory Project Management.
- Open the context menu of the resource and select **Online Test**.
- Open the TypeInstance with a double-click on the symbol **TypeInstance**.

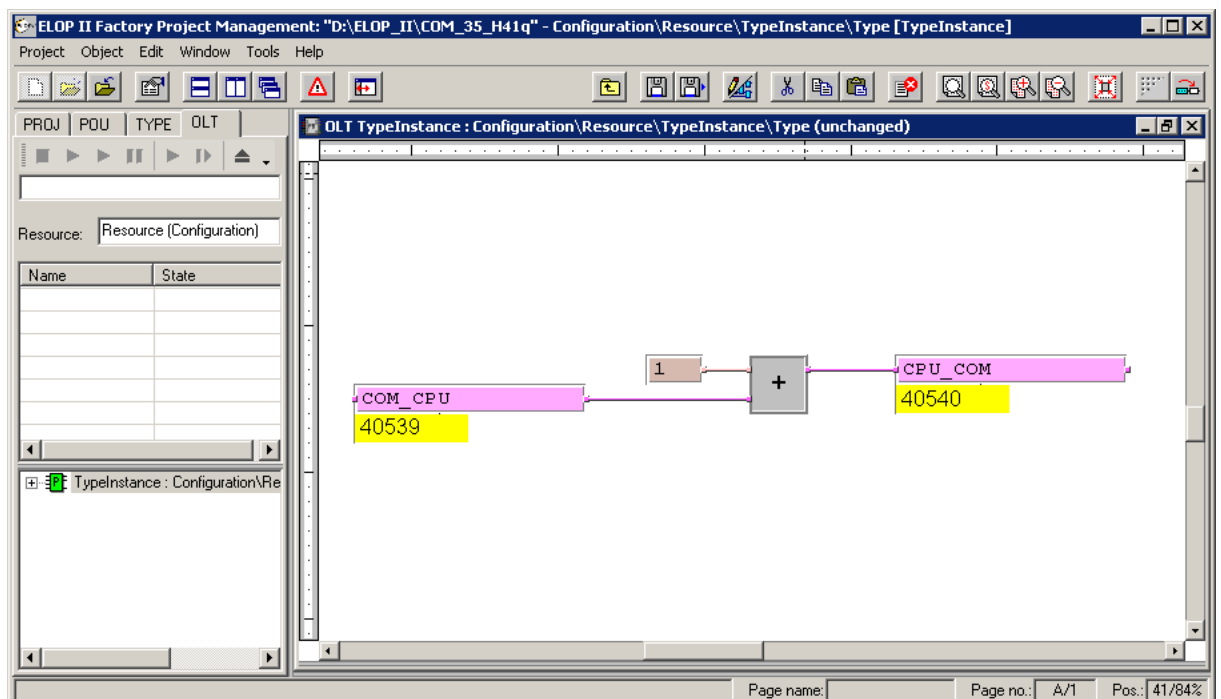


Figure 26: Online Test

5.4 Tips and Tricks

5.4.1 Failure during Loading of a Configuration with the CUT

Run time problems (e.g. COM User Task in endless loop):

Reason for run time problems:

If in the corresponding CUT source code a loop was programmed which runs very long this can lead to a “deadlock” of the COM processor.

As a consequence a connection to the controller cannot be established anymore and the clearing of the resource configuration is not possible anymore.

Solution: Reset of the controller:

- Make a reset of the controller (see data sheet of the controller).
- Clear the resource configuration via
Control Panel->Extra->Clear Resource Configuration
- Create a new CUT (without run time errors, endless loops).
- Load the CUT (ldb file) into the project.
- Generate the code.
- Load the code into the controller.
If the controller cannot be loaded then switch off and switch on the controller. Then load the code once more anew.

HIMA
...the safe decision.



HIMA Paul Hildebrandt GmbH
Industrial Automation
Postfach 1261 • D - 68777 Bruehl
Phone: (+49) 6202 709-0 • Fax: (+49) 6202 709-107
E-mail: info@hima.com • Internet: www.hima.com

(0637)