Automations- und
Steuerungstechnik GmbH

Product Description

EIB Mediatechnic Gateway

Lossless coupling of the EIB to the mediatechnic in ASCII-Format

Order code: E001-H026001

IMPORTANT-READ CAREFULLY:

This b+b End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and b+b Automations- und Steuerungstechnik GmbH, for the software product identified above, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA.

SOFTWARE PRODUCT LICENSE

1. COPYRIGHT TREATIES

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

2. GRANT OF LICENSE.

This EULA grants you the following rights:

a. Software Product. b+b grants to you as an individual, a personal, nonexclusive license to make and use copies of the SOFTWARE for the sole purposes of using the SOFTWARE´s functionality.
b. Storage/Network Use. You may also store or install a copy of the SOFTWARE PRODUCT on a storage device, such as a network server, used only to install or run the SOFTWARE PRODUCT on your other computers over an internal network; however, you must acquire and dedicate a license for each separate computer on which the SOFTWARE PRODUCT is installed or run from the storage device. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.
c. Electronic Documents. Solely with respect to electronic documents included with the SOFTWARE PRODUCT, you may make an unlimited number of copies (either in hardcopy or electronic form), provided that such copies shall be used only for internal purposes and are not republished or distributed to any third party.

3. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

a. Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT.
b. Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.
c. Changing documentations. You may not make changes to the documentation of the SOFTWARE PRODUCT.
d. Termination. Without prejudice to any other rights, b+b may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

4. COPYRIGHT.

All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by b+b or its suppliers. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material except that you may install the SOFTWARE PRODUCT on a single computer provided you keep the original solely for backup or archival purposes. You may not copy the printed materials accompanying the SOFTWARE PRODUCT.

5. LIMITED WARRANTY

Except with respect to the REDISTRIBUTABLES, which are provided "as is," without warranty of any kind, b+b warrants that (a) the SOFTWARE PRODUCT will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and (b) any hardware accompanying the SOFTWARE PRODUCT will be free from defects in materials and workmanship under normal use and service for a period of one (1) year from the date of receipt.

6. CUSTOMER REMEDIES.

b+b´s entire liability and your exclusive remedy shall be, either (a) return of the price paid, or (b) repair or replacement of the SOFTWARE PRODUCT or hardware that does not meet b+b Limited Warranty. This Limited Warranty is void if failure of the SOFTWARE PRODUCT or hardware has resulted from accident, abuse, or misapplication.

NO OTHER WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, b+b DISCLAIMS ALL OTHER WARRANTIES

NO LIABILITY FOR CONSEQUENTIAL DAMAGES: TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL b+b OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR HARDWARE PRODUCT, EVEN IF b+b HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Life support:
These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. b+b customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify b+b for any damages resulting from such application.

This handbook describes also functions, which are options.
Only qualified persons are allowed to install our units.

Softwaredesign und Coding:	Volker Knapp, Hartmut Zander, Peter Bernert
Documentation:	Frank Schlaps, Peter Bernert
Editor:	Frank Schlaps

b+b Automations- und Steuerungstechnik GmbH
Klingenweg 17
64385 Reichelsheim
Tel: 06164 / 912057
e-mail: support@bb-steuerungstechnik.de
internet: http://www.bb-steuerungstechnik.de

Contents

Highlights EIB-MEDIATECHNIC-Gateway

- Simple access to the EIB
- Standard-ASCII-Code
- Supports all 32767 group addresses simultaneously
- Supports all EIS-types
- Simple setup using a terminal program
- built in German and English help texts
- Fully transparent even at 100% bus load
- Built in filter- and EIS-type tables
- All values read and writeable using plain text
- Simple integration of the EIB into own systems (PC,uC…)
- Built in bus coupling unit
- Integrated mains adapter (110-230V)
- Standard-RS232-Interface, no system drivers required
- DIN-rail mounted 9TE = 156mm for top hat rail mounting

Although the EIB-Mediatechnic-Gateway contains a large instruction set, for communication 3 commands will suffice:

Read requests of group addresses
Send values to group addresses
Listening to group address telegrams

Contents of delivery

- EIB-Mediatechnic-Gateway
- Parametrization software b+bTerminal
- Documentation

View:

Width: **9TE (156mm)**
Connectors: **EIB, 230V, RS232 over clamps or SUB-D**
Mounting: **top hat rail**

Introduction

The EIB-MEDIATECHNIC-Gateway is a serial interface to the EIB with integrated EIB-2-wire-buscoupler. The EIB-MEDIATECHNIC-GATEWAY allows an operating system independent connection of the EIB to your own system.. The communication is done in human readable text without time critical handshake signals.

Hint: The setup is easily done with the provided b+b Terminal-program.
It contains a configuration window in which all settings can be adjusted and then transferred to the EIB-MEDIENTECHNIK-GATEWAY.

The program is located in the installation folder of the EIB-Mediatechnic-Gateway and can be started with a double click on the file b+bTerminal.exe.

Functional Description (Rev 4.01)

EIB-Interface:

2-Wire twisted pair with integrated bus coupling unit
supported group addresses: 32765

Serial interface:

Host interface RS232
1 Startbit,8 Databits, 1 Stopbit, no Parity
Handshake RTS/CTS (user selectable option)
Baudrate: 38400 Baud

Connection diagram:

9-pol Mediengateway 9-pol AMX

RXD 2 ---------- 3 TXD

TXD 3 ---------- 2 RXD

GND 5 ---------- 5 GND

RTS 7 ---------- 8 CTS

CTS 8 ---------- 7 RTS

+- 6 DSR

+- 4 DTR

ASCII-Protocol description:

The communication between host and EIB-Mediatechnic-Gateway uses readable text only. The EIB-Mediatechnic-Gateway neither sends nor accepts characters below 20Hex (ASCII Blank). The only exception to this rule is the carriage return char(0d Hexadecimal), shortened called “cr” in the following text.. The cr is used to signal the end of a transmission. Characters sent from the host are not echoed by the device.
All command related characters are treated as uppercase, the means the device is “case insensitive”.

ASCII-Commands:

ASCII-Command: request help text

Command:	"?"
Purpose:	Request of the help text
Description:	The device sends a short version of this document to the host
Remark:	During this command the EIB Communication is stopped!
Reply from the Device:	Helptext

ASCII-Command: Version request

Command:	"?V"
Purpose:	Query the firmware version
Description:	The EIB-MEDIATECHNIC-GATEWAY sends its firmware revision and serialnumber.
Reply from the device:	"EIB_Terminal Vn.nn SN:xxxxxxxx" n.nn = Firmwareversion, xxxxxxxx = Serialnumber
Example reply:	EIB_Terminal V4.05 SN:3709651

ASCII-Command: Physical address request

Command:	"?P"
Purpose:	Query the physical address of the EIB-MEDIATECHNIC-GATEWAY
Description:	The EIB-MEDIATECHNIC-GATEWAY sends its physical address
Reply from the device:	"Phys. Addr=nn.nn.nnn" or "Phys. Addr=xxxx" if the hexadecimal option is activated.
Example reply:	Phys. Addr=01.01.254

ASCII-Command: Setup the physical address

Command:	"P:PA"
Purpose:	Setup the physical address of the EIB-MEDIATECHNIC-GATEWAY and store it non-volatile
Description:	PA may be given as AA.LL.DDD or Hexadecimal as "xALDD"
Telegram contents:	"P:AA.LL.DDD" or "P:xALDD", where: A=Area, L=Line, D=device
Reply from the device:	"Phys. Addr=nn.nn.nnn" or "Phys. Addr=xxxx" if the hexadecimal option is activated.
Remark:	Device must be nonzero, address 15.15.255 is invalid
Error messages:	"!Bad Command Format" if the command is badly formatted. "!Bad value" if an illegal address given.
Example:	P:1.1.250 sets the physical address of the EIB-MEDIATECHNIC-GATEWAY to 1.1.250

Hint: The setup is easily done with the provided b+b Terminal-program.
It contains a configuration window in which all settings can be adjusted and then transferred to the EIB-MEDIENTECHNIK-GATEWAY.

ASCII-Command: Get the current options

Command: "?O"

Purpose: Request the status of the various options

Reply: "OPTIONS: Dw Ew Gw Hw Nw Qw Rw Sw Vw Ww Xw"

Remark: w="+" option activated, w="-" option deactivated
During this command the EIB Communication is stopped!

Possible Options:

"OVw" Verbose mode
If Verbose mode is activated the device replies to commands with an answer, either „OK“ or an error description.

"OEw" Echo of own write telegrams
If this option is activated own write telegrams are sent back to the host after received „back“ from the bus. Since this filtering is done by comparing physical source address please pay attention to assign physical addresses only once. The echo telegram will be processed by the normal receive filter(described later).

"OQw" Report read telegrams to the host
If this option is activated received read telegrams of enabled group addresses are sent to the host. The format is similar to an value telegram, instead of „=value“ read telegrams are signaled with „*“.

"OXw" Hexadecimal mode
If this option is activated the addresses are transmitted as „XXXX“, that means as 4 hexadecimal digits.
The components are divided as follows:
Source address(physical address): "ALDD"
4 Bit Area address, 4 Bit Line address, 8 Bit Device address
Target address(group address):"0HHHHMMMUUUUUUUU"
To explain this we need to use the binary description:
The most significant Bit is always zero, followed by 4 bits for the maingroup, 3 bits for the middlegroup and 8 bits for the subgroup.
If this option is deactivated the source addresses are formatted as AA.LL.DDD and target addresses as HH/M/SSS
A=Area,L=Line,D=Device,H=Maingroup,M=Midgroup,S=Subgroup.

"ORw" Allow read telegrams to disabled group addresses
If activated the device allows to send read telegrams to disabled group addresses, else the read request is dropped.

"OSw" Send source address to the host
If enabled the source address of each telegram is inserted in front of the destination address. This causes more data transfer and requires 38400Baud!

"OHw" RTS/CTS Handshake
If activated the device sends data to the host only if the RTS line is activated. The CTS line signals that data may be transmitted by the host.

"ONw" Telegram numbering
If activated the telegrams to the host are prefixed with an 3 digit decimal or
2 digit hexadecimal number. This number is incremented by 1 after each transfer and set to zero again when reached 255/0xff .

"OGw" Language selection
OG+ selects German, OG- English

"ODw" Decimal sign selection
OD+ selects the Dot as decimal delimiter,
OD- selects comma.

“OWw” Wait message
If activated the device sends an “please wait” message to the host on lengthy operations, like setting up attribute ranges of the ERASE! Command.

ASCII-Command: Converting of group addresses

Command:	"Cga"
Purpose:	Useful utility function to convert group address formats
Remark:	Causes no read or write action to the bus
Example1:	An hexadecimal group address is given, the decimal format is wanted: Host sends: CX5F03 The device answers with X5F03=11/7/003
Example2:	The other way round: Host sends: C11/7/3 The device answers with 11/7/003=X5F03

ASCII-Command: Get group address attributes

Command:	"Gga"
Purpose:	Get the remanent attributes of group address „ga“.
Remarks:	Causes no read or write action to the bus "ga" may be specified as "HH/M/SSS” or "xNNNN". While processing this command the EIB communication is stopped!
Reply:	"Gga:n,f" n = dataformat used for this ga, 0..21 (see data formats!) f=receive enable, "E"/"D" E=Enabled, D=Disabled
Example:	G0/0/1 requests the attributes for group address 0/0/1.
Example Reply:	00/0/001:1,E that is: Format 1, receive enabled

ASCII-Command: Get group address range attributes

Command:	"Gga1-ga2"
Purpose:	Get the remanent attributes of group address „ga1“ upto “ga2”.
Remarks:	Causes no read or write action to the bus "ga" may be specified as "HH/M/SSS” or "xNNNN". While processing this command the EIB communication is stopped!
Reply:	"Gga:n,f" n = dataformat used for this ga, 0..21 (see data formats!) f=receive enable, "E"/"D" E=Enabled, D=Disabled The whole specified range is sent to the host, one line for each ga.
Possible error messages:	If option "V-" is set: none IF "V+" is set: "!Bad Command Format" "!Bad Group Address"
Example:	G0/0/1-0/0/4 requests the attributes of the group addresses 0/0/1 to 0/0/4.
Example reply:	00/0/001:1,E that is format 1, receive enabled 00/0/002:1,E that is format 1, receive enabled 00/0/003:1,E that is format 1, receive enabled 00/0/004:1,E that is format 1, receive enabled

ASCII-Command: Setup group address attributes

Commands:	"Sga:n" "Sga,f" "Sga:n,f"
Purpose:	Setup attributes for group address „ga“ and store settings nonvolatile.
Remarks:	"ga" may be specified as "HH/M/SSS” or "xNNNN". n = dataformat used for this ga, 0..21 (see data formats!) f=receive enable, "E"/"D" E=Enabled, D=Disabled While processing this command the EIB communication is stopped!
Reply:	If option "V-" is set: none If "V+" is set: "Ok".
Possible error messages:	If option "V-" is set: none IF "V+" is set: "!Bad Command Format" "!Bad Group Address"
Example:	S1/0/1:1,E defines the group address 1/0/1 as 1-6 Bit value and enables the receive This attributes are used i.e. for ON/OFF telegrams.

ASCII-Command: Setup group address range attributes

Commands:	"Sga1-ga2:n" "Sga1-ga2,f" "Sga1-ga2:n,f"
Purpose:	Setup attributes for group address „ga1“ upto “ga2” and store settings non-volatile.
Remarks:	"ga" may be specified as "HH/M/SSS” or "xNNNN". n = dataformat used for this ga, 0..21 (see data formats!) f=receive enable, "E"/"D" E=Enabled, D=Disabled While processing this command the EIB communication is stopped!
Reply:	If option "V-" is set: none If "V+" is set: "Busy...", after processing "Ok".
Possible error messages:	If option "V-" is set: none IF "V+" is set: "!Bad Command Format" "!Bad Group Address"
Example:	S1/0/1-1/0/255:1,E defines the group addresses 1/0/1 to 1/0/255 as 1-6 Bit value and enables the receive. This attributes are used i.e. for ON/OFF telegrams.
additional hint:	The device needs about 10 seconds for setting up the whole address range.

ASCII-Command: Read request to a group address

Command:	"Rga"
Purpose:	Send a value read telegram to the bus for the specified address “ga".
Remarks:	"ga" may be specified as "HH/M/SSS” or "xNNNN". The read telegram is only sent to the bus if the group address is activated or the global option “R+” has been set.
Reply:	If option "V-" is set: none If option "V-" is set: “OK” on success
Possible error messages:	If option "V-" is set: none IF "V+" is set: "!Bad Command Format" "!Bad Group Address" “!Group Address Disabled" "!EIB not connected"
Example:	R0/0/1 creates a value read telegram for GA 0/0/1.

ASCII-Command: Send value to a group address

Command:	"Wga=v"
Purpose:	Send a value write telegram to the bus for the specified address “ga" with value “v”
Remarks:	"ga" may be specified as "HH/M/SSS” or "xNNNN". "v"=Data value, dependant on the selected data format (see section data formats)
Reply:	If option "V-" is set: none If option "V-" is set: “OK” on success
Possible error messages:	If option "V-" is set: none IF "V+" is set: "!Bad Command Format" "!Bad Group Address" “!Group Address Disabled" "!EIB not connected" "!Bad value"
Examples:	W1/0/1=1 sends a „ON“ telegram on the group address 1/0/1 if this one is setup with data format 1. W1/0/1=0 sends a „OFF“ telegram on the group address 1/0/1 if this one is setup with data format 1.

Data formats:

The supported data formats are supporting all currently defined non structured "DPT" types of the EIB Specification. Those have been extended to support special host requirements.

Data formats: Format1

Data type:	Binary data, 1-6 Bit raw data length
Value range:	0..63 Decimal, x0..x3F Hexadecimal
Remarks:	Supports the DPT 1.x,2.x and 3.x On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal.

Data formats: Format2

Data type:	1 ASCII-char
Value range:	20H..0FFH
Remarks:	DPT 4.001 8 Bit ," character ASCII" On sending the char following the „=“ is sent to the bus. On receiving exactly one char is transmitted.

Data formats: Format3

Data type:	Percentage 0-100%
Value range:	0..100 Decimal, x0..x64 Hexadecimal
Remarks:	DPT 5.001 8 Bit ," percentage" On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal with 2 fractional digits.

Data formats: Format4

Data type:	Angle 0-360 Degrees
Value range:	0..360 Decimal, x0..x168 Hexadecimal
Remarks:	DPT 5.003 8 Bit ," angle degrees " On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal with 2 fractional digits.

Data formats: Format5

Data type:	1 Byte unsigned
Value range:	0..255 Decimal, x0..xff Hexadecimal
Remarks:	DPT 5.010 8 Bit ," 1 Byte unsigned value" On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal.

Data formats: Format6

Data type:	1 Byte signed
Value range:	-128..127 Decimal, x0..xff Hexadecimal
Remarks:	DPT 6.x 8 Bit ," 1 Byte signed value " On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal, negative values are prefixed with ”-“.

Data formats: Format7

Data type:	2 Byte unsigned
Value range:	0..65535 Decimal, x0..xffff Hexadecimal
Remarks:	DPT 7.x 2 Byte," 2 Byte unsigned value" On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal.

Data formats: Format8

Data type:	2 Byte signed
Value range:	-32768..32767 Decimal, x0..xffff Hexadecimal
Remarks:	DPT 8.x 2 Byte," 2 Byte signed value " On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal, negative values are prefixed with ”-“.

Data formats: Format9

Data type:	2 Byte EIB floating point
Value range:	-671088.64 to 670760.96, exponents 0..15.
Resolution:	0.01.
Data format:	"SEEEEMMMMMMMMMMM" S= sign of mantissa E= Exponent M= Mantissa
Remarks:	DPT 9.x 2 Byte," 2 Byte float value " The EIB floating point format consists of the sign bit of the mantissa, 4 bits exponent and 11 bits for the mantissa itself. Per definition the mantissa has to be multiplied with 0.01. On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. If given as hexadecimal the value must contain the floating point value already! On receiving the data format is decimal with 2 fractional digits, negative values are prefixed with ”-“.

Data formats: Format10

Data type:	3 Byte Time
Data format:	"hh:mm:ss", hh= hour 0..23, mm= minute 0..59, ss= second 0..59
Remarks:	DPT 10.x 3 Byte," Time" On sending either the format "hh:mm:ss" or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. If given as hexadecimal the value must contain the hh value in the 1st byte, the mm value in the 2nd byte and the ss value in the third byte. "xhhmmss", hh= 00..17h, mm= 00..3Bh, ss= 00..3Bh On receiving the data is formatted as "hh:mm:ss".

Data formats: Format11

Data type:	3 Byte Date
Data format:	"dd.mm.yy", dd= day 1..31, mm= month 1..12, yy= year 00..99
Remarks:	DPT 11.x 3 Byte," Date" On sending either the format "dd.mm.yy" or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. If given as hexadecimal the value must contain the dd value in the 1st byte, the mm value in the 2nd byte and the yy value in the third byte. "xddmmyy", dd= 01..1fh, mm= 01..0Ch, yy= 00..63h On receiving the data is formatted as "dd:mm:yy". The data is not completely validated, that means the date 31.02.02 will be accepted by the device.

Data formats: Format12

Data type:	4 Byte unsigned
Value range:	0...4294967295 Decimal, x0..xffffffff Hexadecimal
Remarks:	DPT 12.x 4 Byte," 4 Byte unsigned value" On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal.

Data formats: Format13

Data type:	4 Byte signed
Value range:	-2147483648..2147483647 Decimal, x0..xffffffff Hexadecimal
Remarks:	DPT 13.x 4 Byte," 4 Byte signed value " On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the data format is decimal, negative values are prefixed with ”-“.

Data formats: Format14

Data type:	4 Byte IEEE floating point according P754
Value range:	1.8446762e19, x0..xffffffff Hexadecimal
Remarks:	DPT 14.x 4 Byte," 4 Byte float value " Concerning the value range and precision please read the related literature. On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. If given as hexadecimal the value must contain the floating point value already! On receiving the data format is decimal with up to 8 fractional digits, negative values are prefixed with ”-“. On very small or big values an exponential format may be generated. There is no rounding mechanism on received values.

Data formats: Format15

Data type:	4 Byte access control
Value range:	0..4294967295 Decimal, x0..xffffffff Hexadecimal
Remarks:	DPT 12.x 4 Byte," 4 Byte unsigned value" On sending either a decimal or an hexadecimal value may be specified. Hexadecimal values must be prefixed with an “x“. On receiving the values are sent hexadecimal with 8 chars.

Data formats: Format16

Data type:	Text string
Value range:	1 to 14 ASCII chars from 20h..0ffh.
Remarks:	DPT 16.x 14 Byte," Character String " Deviating from the DPT16.x specification exactly the given count of chars are transmitted. Therefore the string must be extended with blanks to get 14 chars transmitted. If the given text exceeds 14 chars the extra chars are ignored.

Data formats: Format17

Data type: Textstring, zero terminated

Value range: 1 to 13 ASCII chars from 20h..0ffh.

Remarks: DPT 16.x 14 Byte," Character String "
Internally there is an zero byte appended to the given text.
Deviating from the DPT16.x specification exactly the given count of chars are transmitted. Therefore the string must be extended with blanks to get 13 chars and the terminating zero transmitted. If the given text exceeds 13 chars the extra chars are ignored.

Data formats: Format18

Data type: 1 to 14 Byte Decimal format, Semicolon separated

Value range: Range per Byte 0..255

Remarks: The single byte values have to be separated by a semicolon char(„;“)
If the source address option is activated more than 64 chars may result on receiving. These will be limited to 64 chars.

Example: Send a 1 Byte Telegram: "Wga=127"
Send a 2 Byte Telegram: "Wga=127;0"
Send a 3 Byte Telegram: "Wga=127;0;100" …

On receiving the data is formatted decimal, the values are semicolon separated.

Data formats: Format19

Data type: 1 to 14 Byte Hexadecimal format, Semicolon separated

Value range: range per Byte 0..ffh

Remarks: The single byte values have to be separated by a semicolon char („;“).
Hexadecimal prefix neither needed nor allowed.
If the source address option is activated more than 64 chars may result on receiving. These will be limited to 64 chars.

Example: Send a 1 Byte Telegram: "Wga=7f"
Send a 2 Byte Telegram: "Wga=7f;0"
Send a 3 Byte Telegram: "Wga=7f;0;64" …

On receiving the data is formatted hexadecimal, the values are semicolon separated.

Data formats: Format20

Data type: 1 to 14 Byte Hexadecimal format, unseparated

Value range: range per byte 0..ffh

Remarks: The different byte values must not be separated.
Hexadecimal prefix neither needed nor allowed.
For each byte value 2 chars must be transmitted.

Example: Send a 1 Byte Telegram: "Wga=7f"
Send a 2 Byte Telegram: "Wga=7f00"
Send a 3 Byte Telegram: "Wga=7f0064" …

On receiving the data is formatted hexadecimal, the values are not separated,
for each byte 2 chars are generated.

Data formats: Format21

Data type: 1 to 6 Bit Hexadecimal format

Value range: 0..3fh.

Remarks: Hexadecimal prefix neither needed nor allowed.

Example: Send a 1-6 Bit Telegram: "Wga=01"

On receiving the data is formatted hexadecimal, 2 chars are generated.

Receive telegrams

Received telegrams containing group address data are first checked for „Echo“ (see option „OE“).
Then the device checks whether the ratget group address is enabled (see “Sga” command).
If the address is enabled the received datalength is compared with the setup data len.
If the length is different the Value field gets “?” assigned as content. Now the data is formatted as follows:

"N PH>GA=Value"

N= Telegram number, only if "ON+"
PH= source address, only if "OS+"
GA= target address

Dependant on the Option „X“ those datas are formatted as

"017 12.03.127>0/3/49=..." or as "11 B37F>0331=..."

The data value itself is formatted depending on the selected data format.

Example telegram hexadecimal: 13 000A>0003=0

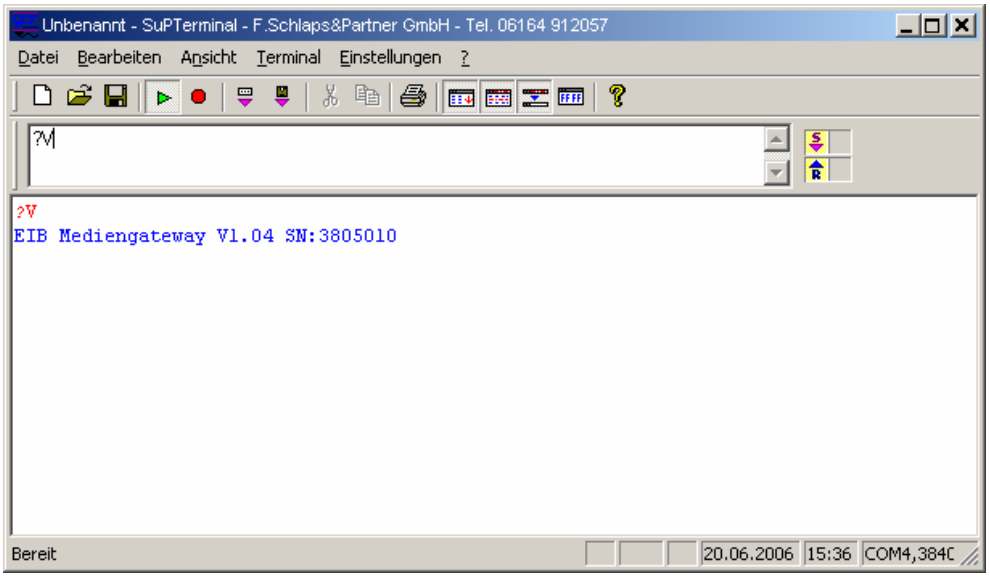
Example telegram decimal: 026 00.00.010>00/0/003=0

Basic parameterization with the b+b Terminal-Program

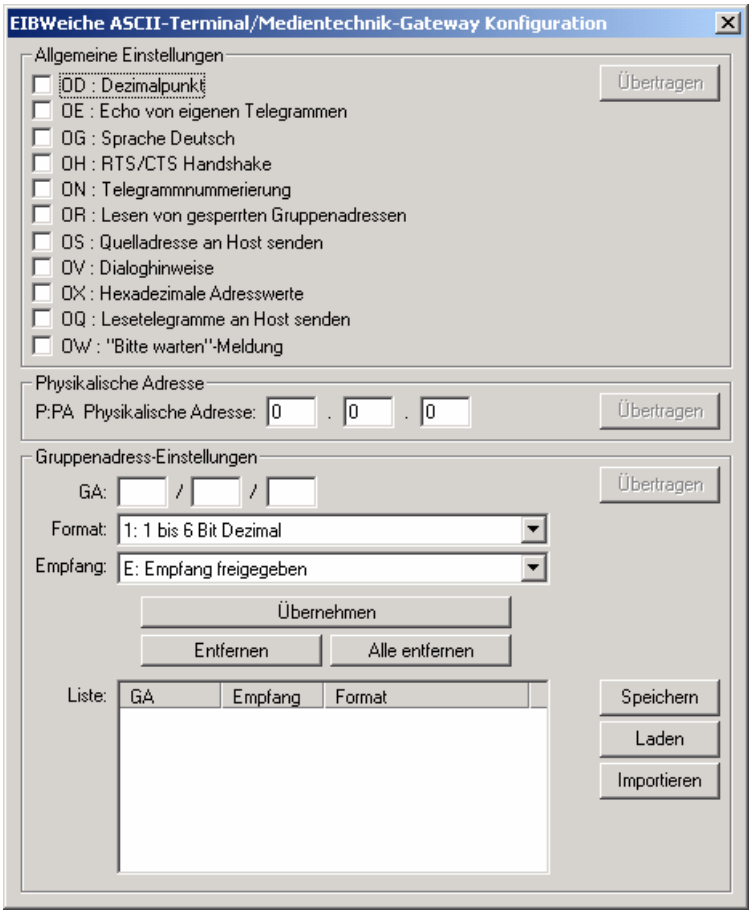
The basic parameterization of the EIB-Mediatchnic-Gateway can be easily done with the b+b Terminal-Program.
This terminal-program is located in the installation folder of the Mediatechnic-Gateway and can be started with a double click on the file b+bTerminal.exe.
First you have to configure the settings for the serial interface in the program by clicking in the menu on „Settings“ and then on the item „Schnittstelle / Allgemeine Einstellungen“. In the appearing dialog you have to apply the following settings:
Interface: COM port, where the gateway is connected to
Baudrate: 38400
Databits: 8
Stopbits: 1
Parity: N (keine)
Protocol: Hardware (RTS/CTS)

The dialog must be colsed with the button „OK“.

Now the connection to the EIB-Mediatechnic-Gateway can be opened, while clicking in the toolbar on Start (green arrow). To check the connection you now should request the version of the EIB-Mediatechnic-Gateway with the command „?V“. If the gateway doesn't answer, you should check if your connection settings are correct.



The dialog to parameterize the gateway is opened, when you click in the menu „Settings“ on “EIB-Weiche ASCII-Terminal/Medientechnik-Gateway“. The following dialog appears:



In chapter „Allgemeine Einstellungen“ the options of the EIB-Mediatechnic-Gateway can be parameterized, like described above. We recommend to activate the following options. These options are also used in the AMX-example at the end of the documentation: D, E, G, H, Q, R, V.
The options N, S, W, X should be deactivated.

After you have applied the settings, it will be transferred to the Mediatechnic-Gateway if you click on the button „Übertragen“.

In chapter „Physical Address“ you can setup the physical address of the EIB-Mediatechnic-Gateway.

In chapter „Gruppenadress-Einstellungen“ you can configure the goup addresses for the Mediatechnic-Gateway. Over the button „Save“ the actual group address are save in a text file. The content of this text file can be used, to configure the Mediatechnic-Gateway with the address definitions. The configuration is done directly when you click on the button „Transfer“.
The saved settings can be loaded again with the button „Load“. With the button „Import“ you can load the complete data of ein EIB project, that has been imported with EIB Explorer or the FIAVisManager. To do so you have to create a EIB Explorer or FIAVis Manager project in the according software and to import the EIB-addresses from an existing ETS-project.
Then you have to transfer the data into a directory. In the EIB Explorer software you have to select the menu item „Transfer data“ or in the FIAVis Manager the menu item „Transfer data to application“. In the following window have have to select the option „EIB.VB directory“, configure the destination directory and execute the transfer with the button „Start...“. The created GA-file contains all address informations and can be imported in the b+b Terminal-Program.

Example for an AMX Axcent control

```
PROGRAM_NAME='EMTGW_Beispiel'
(***********************************************************)
(*  FILE CREATED ON: 05/05/2006  AT: 11:53:23               *)
(***********************************************************)
(*  FILE_LAST_MODIFIED_ON: 05/05/2006  AT: 13:17:06         *)
(***********************************************************)
(***********************************************************)
(* System Type :                                           *)
(***********************************************************)

(* ********************************************************************** *)
(*          DEVICE NUMBER DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_DEVICE
RS232_1         = 1                     (* RS232 channels                 *)
TP              = 128                   (* touchpanel                     *)

(* ********************************************************************** *)
(*               CONSTANT DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_CONSTANT
EIB_MTG         = RS232_1               (* b+b EIB-MTG on serial 1        *)

(* ********************************************************************** *)
(*               VARIABLE DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_VARIABLE
MASTER_MSG_BUFFER[100]                  (* BUFFER FOR 'MSG.LIB'           *)
EMTG_BUFFER[255]                        (* Receive Buffer for EIB-MTG     *)
EMTG_INIT                               (* EIB-MTG init state             *)
EMTG_LAST_CMD[255]                      (* EIB-MTG last command sent      *)
EMTG_RC                                 (* EIB-MTG receive return code    *)
EMTG_ANS[32]                            (* EIB-MTG last answer            *)
tmp                                     (* some temporary var             *)

(* ********************************************************************** *)
(*               LATCHING DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_LATCHING

(* ********************************************************************** *)
(*       MUTUALLY EXCLUSIVE DEFINITIONS GO BELOW                          *)
(* ********************************************************************** *)
DEFINE_MUTUALLY_EXCLUSIVE
```

```
(* ********************************************************************** *)
(*           SUBROUTINE DEFINITIONS GO BELOW                              *)
(* ********************************************************************** *)

(************************************************************************)
(* Send EIB-MTG Command, Answer is checked                              *)
(************************************************************************)
DEFINE_CALL 'EMTG_SENDCOMMAND_ANS' (DEV,SENDSTR[255],REPLY[255],ANSWER[255],RC)
LOCAL_VAR x,substr[255]
{
 OFF [RC]                                  (* no complete frame got          *)
 ANSWER = ''                               (* cleanup answer                 *)
 IF(EMTG_LAST_CMD <> SENDSTR)              (* do not send twice              *)
  {
   SEND_STRING DEV,"SENDSTR,13"            (* send the request               *)
   EMTG_LAST_CMD = "SENDSTR"               (* save sended string             *)
  }

 while(LENGTH_STRING(EMTG_BUFFER) AND NOT [RC])  (* receive?                *)
  {
   x = Get_buffer_char(EMTG_BUFFER)        (* get char from buffer           *)
   IF (x > 32)                             (* good char                      *)
    {
     ANSWER = "ANSWER,x"                   (* add char to answer             *)
    }
   ELSE IF ((x = 13) AND LENGTH_STRING(ANSWER))    (* answer end char found  *)
    {                                      (* YES:                           *)
     ON [RC]                               (* signal completed answer got    *)
    }
  }

 substr = LEFT_STRING(ANSWER,LENGTH_STRING(REPLY))    (* check answer        *)
 IF(substr = REPLY) ON [RC]                (* answer expected?               *)
 ELSE               OFF[RC]                (* bad answer                     *)
}                                          (* EMTG_SENDCOMMAND_ANS()         *)

(************************************************************************)
(* Send a value to a GA                                                 *)
(************************************************************************)
DEFINE_CALL 'EMTG_SEND_GA' (DEV,GA[32],VAL[32])
LOCAL_VAR sendstr[64]
{
 sendstr = "'W',GA,'=',VAL"                (* create command                 *)
 IF(EMTG_LAST_CMD <> sendstr)              (* stop sending twice             *)
  {
   SEND_STRING DEV,"sendstr,13"            (* send the request               *)
   EMTG_LAST_CMD = "sendstr"               (* save sended string             *)
  }
}                                          (* EMTG_SEND_GA()                 *)
```

```
(* ********************************************************************** *)
(*                STARTUP CODE GOES BELOW                                 *)
(* ********************************************************************** *)
DEFINE_START
CREATE_BUFFER 0,MASTER_MSG_BUFFER
CREATE_BUFFER EIB_MTG,EMTG_BUFFER
EMTG_INIT     = 0
EMTG_RC       = 0
EMTG_LAST_CMD = ''
EMTG_ANS      = ''

(* ********************************************************************** *)
(*            THE ACTUAL PROGRAM GOES BELOW                               *)
(* ********************************************************************** *)
DEFINE_PROGRAM

If(EMTG_INIT < 100)                         (* Init running?               *)
 {                                          (* YES:                        *)
  SELECT                                    (* checkout what to do         *)
   {
    ACTIVE (EMTG_INIT = 0):                 (* Init EIB-MTG RS232          *)
     {
      SEND_COMMAND EIB_MTG,'SET BAUD 38400,N,8,1'    (* Baud...           *)
      SEND_COMMAND EIB_MTG,'HSON'                    (* Hardware handshake *)
      EMTG_INIT = EMTG_INIT + 1             (* switch to next state        *)
     }

    ACTIVE (EMTG_INIT = 1):                 (* Init EIB-MTG options        *)
     {
      WAIT 10                               (* Wait 1 second for AMX port  *)
       {
        CALL 'EMTG_SENDCOMMAND_ANS'(EIB_MTG,"'OV+D+E+G+H+N-Q+R+S-W-X-',13" ,'OK',
              EMTG_ANS,EMTG_RC)
        IF(EMTG_RC) EMTG_INIT = EMTG_INIT + 1
       }
     }

    ACTIVE (EMTG_INIT = 2):                 (* we are done                 *)
     EMTG_INIT =  100
   }
 }
ELSE                                        (* init done ...               *)
 {
  PUSH [TP,1]                               (* simple toggle               *)
   {
    if([TP,1])  tmp = 0                     (* currently on ==> off        *)
    else        tmp = 1                     (* currently off ==> on        *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/0',itoa(tmp))    (* send new value   *)
   }

  PUSH [TP,2]                               (* rel. Dimm up                *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/10',itoa(9))

  PUSH [TP,3]                               (* rel. Dimm stop              *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/10',itoa(0))

  PUSH [TP,4]                               (* rel. Dimm down              *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/10',itoa(1))

  PUSH [TP,5]                               (* abs. Dimm to 100%           *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/11',itoa(100))

  PUSH [TP,6]                               (* abs. Dimm to 50%            *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/11',itoa(50))

  PUSH [TP,7]                               (* abs. Dimm to 0%             *)
    CALL 'EMTG_SEND_GA' (EIB_MTG,'1/1/11',itoa(0))
```

```
  CLEAR_BUFFER EMTG_BUFFER                (* ensure no buffer overrun       *)
 }

(***********************************************************)
(*                      END OF PROGRAM                     *)
(*         DO NOT PUT ANY CODE BELOW THIS COMMENT          *)
(***********************************************************)
```

Description

The program was created and translated with the Netlinx Studio 2.1 Build 2.1.0.64.

In chapter DEFINE_DEVICE the used devices are defined. Because the EIB-Mediatechnic-Gateway should be used at the first device, device 1 is used. To make interactive outputs additionally a touch-panel device is added on device number 127.

In chapter DEFINE_CONSTANT an alias is added which refers to the used interface. If the second interface should be used, you only have to change the constant and translate / transfer the program again.

In chapter DEFINE_VARIABLE some variables are defined, which are necessary for the program flow.

Two calls are used, to keep the source code readable.

EMTG_SENDCOMMAND_ANS

CALL 'EMTG_SENDCOMMAND_ANS' (DEV,SENDSTR[255],REPLY[255],ANSWER[255],RC)

This function is used to send a command to the EIB-Mediatechnic-Gateway. The function sends the command and then reads the received answer. The answer is compared to a default to be able to give a return over success or failure.

Parameters:

DEV	DEVICE	device number where the EIB-Mediatechnic-Gateway is connected to
SENDSTR	STRING	command, that should be sent
REPLY	STRING	expected reply of the EIB-Mediatechnic-Gateway
ANSWER	STRING	really received answer
RC	INTEGER	0=error, 1=OK

EMTG_SEND_GA

CALL 'EMTG_SEND_GA' (DEV,GA[32],VAL[32])

This function is used to send a value to an EIB group address over the EIB-Mediatechnic-Gateway.

Parameters:

DEV	DEVICE	device number where the EIB-Mediatechnic-Gateway is connected to
GA	STRING	group address where to value should be sent to e. g. 1/1/0
VAL	STRING	value that should be sent e.g. 1 or 1.05

In chapter DEFINE_START the variables are initialized and the buffers are assigned to the devices for the data exchange.

In chapter DEFINE_PROGRAM the program is located. It is divided into two parts:

- Initialization
- Execution

The two parts are identified by the IF statement. As long as the value of the init status (EMTG_INIT) is smaller than 100, the program is in the initialization phase. If the value is >= 100 the initialization is finished.

Initialization

The initialization is done in several steps, because the control can't handle too much commands at the same time. So the initialization is done in several cycles.

In the first cycle the interface is configured. According to the AMX description the interface is configured to 38400 Baud, 8 databits, no parity and 1 stoppbit. Additionally the handshake is activated to prevent buffer overflows.

In the next cycle one second is waited and then the sequence with the options is sent to the EIB-Mediatechnic-Gateway. The cycle is finished as soon as the answer ***OK*** was received.

The following options are set:

- **V+** Verbose mode, that ***OK*** is sent
- **D+** Decimal sign selection, use decimal point
- **E+** activated echo of own write telegrams
- **G+** Language selection: German.
- **H+** activate RTS/CTS Handshake
- **N-** Telegram numbering
- **Q+** Report read telegrams to the host
- **R+** Allow read telegrams to disabled group addresses
- **S-** don't send source address to the host
- **W-** Deactivate wait message.
- **X-** Hexadecimal mode off

In the next cycle the group addresses could be written to the filter table. But if they have already been configured like in our example, initialization can be finished.

Execution

In the execution we react to the sent pushchannels of the touchpanel. Channel 1 switches the group address 1/1/0 (bitvalue).

The pushchannels 2,3,4 execute a relative dimming command (4-Bit) on group address 1/1/10.

The pushchannels 5,6,7 execute an absolute dimming command (8-Bit) on group address 1/1/11.

Example for advanced users with reception of EIB-Telegrams

```
PROGRAM_NAME='EIB-MG'
(*********************************************************)
(*  FILE CREATED ON: 04/18/2006  AT: 13:21:26             *)
(*********************************************************)
(*  FILE_LAST_MODIFIED_ON: 04/21/2006  AT: 12:29:23       *)
(*********************************************************)

(* ********************************************************************** *)
(*               CONSTANT DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_CONSTANT
EMG_ADDR1     =    1                    (* addresspart 1                   *)
EMG_ADDR2     =    2                    (* addresspart 2                   *)
EMG_ADDR3     =    3                    (* addresspart 3                   *)
EMG_TP_TYP    =    4                    (* TP Item Typ 1=button,2=Text,3=Level*)
EMG_TP_FBI    =    5                    (* TP Feedback Item No             *)
EMG_TP_VTN    =    6                    (* TP Variable Text No             *)
EMG_TP_LVL    =    7                    (* TP Level No                     *)
EMG_EIB_TYP   =    8                    (* EIB-MG Typ                      *)
EMG_DSP_FMT   =    9                    (* 0=Value,1=Name,2=UNIT           *)
(*-------------------------------------------------------------------------*)
EMG_MAX_GA    =   64
EMG_MAX_PQ    =   32

(* ********************************************************************** *)
(*               VARIABLE DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_VARIABLE
EMG_BUFFER       [MAX_EMGBUFFER   ]    (* Buffer for EIB-MG device        *)
(*-------------------------------------------------------------------------*)
EMG_LAST_CMD[255]                      (* receive helper                  *)
EMG_MESSAGE [255]                      (* EIB-MG receiver                 *)
EMG_INIT_STATE                         (* Init status                     *)
EMG_RC                                 (* EIB-MG receive return code      *)
EMG_ANS[32]
EMG_DEFINED
(*-------------------------------------------------------------------------*)
EMG_GALIST[EMG_MAX_GA][ 8]
EMG_GAPROP[EMG_MAX_GA][ 9]
EMG_GANAME[EMG_MAX_GA][16]
EMG_GAUNIT[EMG_MAX_GA][ 8]
EMG_PANELC[EMG_MAX_PQ][48]
EMG_IPOS
EMG_SPOS

(* ********************************************************************** *)
(*               LATCHING DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_LATCHING

(* ********************************************************************** *)
(*       MUTUALLY EXCLUSIVE DEFINITIONS GO BELOW                          *)
(* ********************************************************************** *)
DEFINE_MUTUALLY_EXCLUSIVE

(* ********************************************************************** *)
(*           SUBROUTINE DEFINITIONS GO BELOW                              *)
(* ********************************************************************** *)

(*************************************************************************)
(* Add GA's                                                              *)
(*************************************************************************)
DEFINE_CALL 'EMG_ADD_GA'(ADDR1,ADDR2,ADDR3,EIB_TYP,TP_TYP,TP_ITEM,TP_TEXT,TP_LEVEL
                         NAME[16],UNIT[8],DISP_FORMAT)
LOCAL_VAR GA[20],idx
{
```

```
 idx = EMG_DEFINED+1
 GA  =
"RIGHT_STRING("'00',itoa(ADDR1)",2),'/',itoa(ADDR2),'/',RIGHT_STRING("'000',itoa(A
DDR3)",3)"
 EMG_GALIST[idx]               = GA
 EMG_GANAME[idx]               = NAME
 EMG_GAUNIT[idx]               = UNIT
 EMG_GAPROP[idx][EMG_ADDR1]    = ADDR1
 EMG_GAPROP[idx][EMG_ADDR2]    = ADDR2
 EMG_GAPROP[idx][EMG_ADDR3]    = ADDR3
 EMG_GAPROP[idx][EMG_TP_TYP]   = TP_TYP
 EMG_GAPROP[idx][EMG_TP_FBI]   = TP_ITEM
 EMG_GAPROP[idx][EMG_TP_VTN]   = TP_TEXT
 EMG_GAPROP[idx][EMG_TP_LVL]   = TP_LEVEL
 EMG_GAPROP[idx][EMG_EIB_TYP] = EIB_TYP
 EMG_GAPROP[idx][EMG_DSP_FMT] = DISP_FORMAT
 EMG_DEFINED = EMG_DEFINED + 1
}

(**************************************************************************)
(* Find GA index                                                          *)
(**************************************************************************)
DEFINE_CALL 'EMG_GA_GETINDEX'(GA[16],index)
LOCAL_VAR x
{
 x     = 1
 index = 0
 WHILE((x <= EMG_DEFINED) AND (index = 0))
  {
   if(EMG_GALIST[x] = GA) index = x
   x = x + 1
  }
}

(**************************************************************************)
(* Find GA index                                                          *)
(**************************************************************************)
DEFINE_CALL 'EMG_NAME_GETINDEX'(NAME[16],index)
LOCAL_VAR x
{
 x     = 1
 index = 0
 WHILE((x <= EMG_DEFINED) AND (index = 0))
  {
   if(EMG_GANAME[x] = NAME) index = x
   x = x + 1
  }
}

(**************************************************************************)
(* Send EIB-MG Command, Answer don't care                                 *)
(**************************************************************************)
DEFINE_CALL 'EMG_SENDCOMMAND' (DEV,SENDSTR[255],ANSWER[255],RC)
LOCAL_VAR x
{
 off [RC]                                  (* no complete frame got       *)
 ANSWER = ''                               (* cleanup answer              *)
 IF(EMG_LAST_CMD <> SENDSTR)
  {
   SEND_STRING DEV,"SENDSTR,13"            (* send the request            *)
   EMG_LAST_CMD = "SENDSTR"                (* save sended string          *)
  }

 while(LENGTH_STRING(EMG_BUFFER) AND NOT [RC])   (* receive?             *)
  {
   x = Get_buffer_char(EMG_BUFFER)         (* get char from buffer        *)
   IF (x > 32)
    {
     ANSWER = "ANSWER,x"                   (* add char to answer          *)
```

```
    }
   ELSE IF ((x = 13) && LENGTH_STRING(ANSWER))
    {
     ON [RC]                                  (* complete ans                *)
    }
  }

 IF(NOT [RC]) ANSWER = 'TIMEOUT'              (* no answer got               *)
}                                             (* EMG_SENDCOMMAND()           *)

(*****************************************************************************)
(* Send EIB-MG Command, Answer is checked                                    *)
(*****************************************************************************)
DEFINE_CALL 'EMG_SENDCOMMANDEX' (DEV,SENDSTR[255],REPLY[255],ANSWER[255],RC)
LOCAL_VAR x,substr[255]
{
 off [RC]                                     (* no complete frame got       *)
 ANSWER = ''                                  (* cleanup answer              *)
 IF(EMG_LAST_CMD <> SENDSTR)
  {
   SEND_STRING DEV,"SENDSTR,13"               (* send the request            *)
   EMG_LAST_CMD = "SENDSTR"                   (* save sended string          *)
  }

 while(LENGTH_STRING(EMG_BUFFER) AND NOT [RC])   (* receive?                 *)
  {
   x = Get_buffer_char(EMG_BUFFER)            (* get char from buffer        *)
   IF (x > 32)
    {
     ANSWER = "ANSWER,x"                      (* add char to answer          *)
    }
   ELSE IF ((x = 13) AND LENGTH_STRING(ANSWER))
    {
     ON [RC]                                  (* complete ans                *)
    }
  }

 substr = LEFT_STRING(ANSWER,LENGTH_STRING(REPLY))
 IF(substr = REPLY) ON [RC]
 ELSE               OFF[RC]
}                                              (* EMG_SENDCOMMANDEX()        *)

(*****************************************************************************)
(* Get reply from EIB-MG                                                     *)
(*****************************************************************************)
DEFINE_CALL 'EMG_RECEIVE_MESSAGE' (RC)
LOCAL_VAR c
{
 OFF[RC]
 while(LENGTH_STRING(EMG_BUFFER) AND NOT [RC])  (* receive?                 *)
  {
   c = GET_BUFFER_CHAR(EMG_BUFFER)            (* get char from buffer        *)
   IF (c > 32)
    {
     EMG_MESSAGE = "EMG_MESSAGE,c"            (* add char to answer          *)
    }
   ELSE IF ((c = 13) AND LENGTH_STRING(EMG_MESSAGE))
    {
     ON [RC]                                  (* complete ans                *)
    }
  }
}                                             (* EMG_RECEIVE_MESSAGE()       *)

(*****************************************************************************)
(* Send a value to a GA                                                      *)
(*****************************************************************************)
DEFINE_CALL 'EMG_SEND_GA' (DEV,GA[32],VAL[32],RC)
LOCAL_VAR sendstr[64]
{
```

```
 sendstr = "'W',GA,'=',VAL"                  (* create command              *)
 IF(EMG_LAST_CMD <> sendstr)
  {
   SEND_STRING DEV,"sendstr,13"              (* send the request            *)
   EMG_LAST_CMD = "sendstr"                  (* save sended string          *)
  }
 ON[RC]
}                                            (* EMG_SEND_VALUE()            *)

(*****************************************************************************)
(* Send a value to a user supplied name                                      *)
(*****************************************************************************)
DEFINE_CALL 'EMG_SEND_NAME'(NAME[16],VALUE[16])
LOCAL_VAR idx,rc
{
 CALL 'EMG_NAME_GETINDEX'(NAME,idx)
 if((idx > 0) AND (idx <= EMG_DEFINED))
  CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[idx],VALUE,rc)
}

(*****************************************************************************)
(* Show EIB values on panel                                                  *)
(*****************************************************************************)
DEFINE_CALL 'EMG_ENCODE_MESSAGE'(MSG[60])
LOCAL_VAR GA[16],VALUE[32],idx,ps,pval,FB,VTXT,LVL,PTY
{
 IF((MSG[1] = '>') AND (LENGTH_STRING(MSG) > 10))    (* valid message        *)
  {
   GA    = MID_STRING(MSG,2,8)                     (* GA                   *)
   VALUE = RIGHT_STRING(MSG,LENGTH_STRING(MSG)-10) (* value                *)
   ps    = FIND_STRING(VALUE,'>',1)                (* demaged data         *)

   if(ps = 0)                                      (* data okay            *)
    {
     CALL 'EMG_GA_GETINDEX'(GA,idx)
     if((idx > 0) AND (idx <= EMG_DEFINED))
      {
       FB   = EMG_GAPROP[idx][EMG_TP_FBI]          (* binary feedback ID    *)
       VTXT = EMG_GAPROP[idx][EMG_TP_VTN]          (* variable text ID      *)
       LVL  = EMG_GAPROP[idx][EMG_TP_LVL]          (* Level ID              *)
       PTY  = EMG_GAPROP[idx][EMG_TP_TYP]          (* amx action            *)

       if(PTY = 0)                                 (* internal type        *)
        {
         if(idx = 1)                               (* special action       *)
          {
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[2],itoa([RELAY,1]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[3],itoa([RELAY,2]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[4],itoa([RELAY,3]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[5],itoa([RELAY,4]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[6],itoa([RELAY,5]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[7],itoa([RELAY,6]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[8],itoa([RELAY,7]),pval)
           CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[9],itoa([RELAY,8]),pval)
          }
         else                                      (* handle relay        *)
          {
           pval = atoi(VALUE)
           [RELAY,idx-1] = pval
          }
        }
       else                                        (* panel action        *)
        {
         if(FB <> 0)
          {
           EMG_PANELC[EMG_IPOS] = "1,FB,VALUE"
           EMG_IPOS = EMG_IPOS + 1
           IF(EMG_IPOS > EMG_MAX_PQ) EMG_IPOS = 1
          }
```

```
          if(VTXT <> 0)
           {
            if(LVL <> 0)
             {
              pval = (atoi(VALUE) * 100) / 255
              EMG_PANELC[EMG_IPOS] = "2,VTXT,itoa(pval),' ',EMG_GAUNIT[idx]"
             }
            else
             EMG_PANELC[EMG_IPOS] = "2,VTXT,VALUE,' ',EMG_GAUNIT[idx]"

            EMG_IPOS = EMG_IPOS + 1
            IF(EMG_IPOS > EMG_MAX_PQ) EMG_IPOS = 1
           }
          if(LVL <> 0)
           {
            EMG_PANELC[EMG_IPOS] = "3,LVL,VALUE"
            EMG_IPOS = EMG_IPOS + 1
            IF(EMG_IPOS > EMG_MAX_PQ) EMG_IPOS = 1
           }
         }
       }
     }
   }
}                                                    (* EMG_ENCODE_MESSAGE()    *)

(*****************************************************************************)
(* Show values on panel                                                      *)
(*****************************************************************************)
DEFINE_CALL 'EMG_HANDLE_PANEL'
LOCAL_VAR action,VALUE[32],id
{
 action = Get_buffer_char(EMG_PANELC[EMG_SPOS]) (*EMG_PANELC[EMG_SPOS][1]*)
 id     = Get_buffer_char(EMG_PANELC[EMG_SPOS]) (*EMG_PANELC[EMG_SPOS][2]*)
 VALUE  = EMG_PANELC[EMG_SPOS]
(*MID_STRING(EMG_PANELC[EMG_SPOS],2,LENGTH_STRING(EMG_PANELC[EMG_SPOS])-2)*)

 SELECT
  {
   ACTIVE (action = 1):
    CALL 'PANEL_SEND_BUTTON_FB'(id,atoi(VALUE))

   ACTIVE (action = 2):
    CALL 'PANEL_SEND_TEXT'(id,VALUE)

   ACTIVE (action = 3):
    CALL 'PANEL_SEND_LEVEL'(id,atoi(VALUE))
  }

 EMG_SPOS = EMG_SPOS + 1
 IF(EMG_SPOS > EMG_MAX_PQ) EMG_SPOS = 1
}

(*****************************************************************************)
(*                                                                           *)
(*****************************************************************************)
DEFINE_CALL 'EMG_HANDLE_RELAY'(idx)
LOCAL_VAR rc,v,fbid
{
 v           = [RELAY,idx]
 OUT_PI[idx] = v
 fbid        = 50 + idx

 EMG_PANELC[EMG_IPOS] = "1,fbid,itoa(v)"
 EMG_IPOS = EMG_IPOS + 1
 IF(EMG_IPOS > EMG_MAX_PQ) EMG_IPOS = 1
 CALL 'EMG_SEND_GA' (EIB_MG,EMG_GALIST[idx+1],itoa(v),rc)
}

(* ************************************************************************* *)
```

```
(*                STARTUP CODE GOES BELOW                              *)
(* ********************************************************************* *)
DEFINE_START
CREATE_BUFFER EIB_MG,EMG_BUFFER         (* Buffer for EIB-MG device         *)

EMG_LAST_CMD    = ''                              (* EIB-MG last command
               *)
EMG_MESSAGE     = ''                              (* EIB-MG last message
               *)
EMG_INIT_STATE  = 0
EMG_DEFINED     = 0
EMG_IPOS        = 1
EMG_SPOS        = 1

(* ********************************************************************* *)
(*            THE ACTUAL PROGRAM GOES BELOW                              *)
(* ********************************************************************* *)
DEFINE_PROGRAM
If(EMG_INIT_STATE  < 100)
 {
  SELECT                                (* checkout what to do             *)
   {
    ACTIVE (EMG_INIT_STATE = 0):        (* Init EIB-MG RS232               *)
     {
      SEND_COMMAND EIB_MG,'SET BAUD 38400,N,8,1'
      SEND_COMMAND EIB_MG,'HSON'
      EMG_INIT_STATE = EMG_INIT_STATE + 1
     }

    ACTIVE (EMG_INIT_STATE = 1):        (* Init EIB-MG options             *)
     {
      WAIT 10
       {
        CALL 'EMG_SENDCOMMANDEX' (EIB_MG,"'OV+D+E+G+H+N-Q+R+S-W-X-
',13",'OK',EMG_ANS,EMG_RC)
        IF(EMG_RC) EMG_INIT_STATE = EMG_INIT_STATE + 1
       }
     }

    ACTIVE (EMG_INIT_STATE = 2):       (* Group addresses                 *)
     {
      EMG_INIT_STATE =  100
     }
   }
 }
ELSE
 {
  CALL 'EMG_RECEIVE_MESSAGE' (EMG_RC)   (* any EIB message income?          *)
  IF(EMG_RC)                                    (* YES: message in EMG_MESSAGE
            *)
   {
    CALL 'EMG_ENCODE_MESSAGE'(EMG_MESSAGE)          (* update vars        *)
    EMG_MESSAGE = ''                            (* cleanup message
                 *)
   }

  WAIT_UNTIL (EMG_SPOS <> EMG_IPOS)
   CALL 'EMG_HANDLE_PANEL'

  WAIT_UNTIL (OUT_PI[1] <> [RELAY,1])
   CALL 'EMG_HANDLE_RELAY'(1)

  WAIT_UNTIL (OUT_PI[2] <> [RELAY,2])
   CALL 'EMG_HANDLE_RELAY'(2)

  WAIT_UNTIL (OUT_PI[3] <> [RELAY,3])
   CALL 'EMG_HANDLE_RELAY'(3)

  WAIT_UNTIL (OUT_PI[4] <> [RELAY,4])
```

```
   CALL 'EMG_HANDLE_RELAY'(4)

  WAIT_UNTIL (OUT_PI[5] <> [RELAY,5])
   CALL 'EMG_HANDLE_RELAY'(5)

  WAIT_UNTIL (OUT_PI[6] <> [RELAY,6])
   CALL 'EMG_HANDLE_RELAY'(6)

  WAIT_UNTIL (OUT_PI[7] <> [RELAY,7])
   CALL 'EMG_HANDLE_RELAY'(7)

  WAIT_UNTIL (OUT_PI[8] <> [RELAY,8])
   CALL 'EMG_HANDLE_RELAY'(8)
 }

(* ********************************************************************** *)
(*                      END OF PROGRAM                                    *)
(*        DO NOT PUT ANY CODE BELOW THIS COMMENT                          *)
(* ********************************************************************** *)




PROGRAM_NAME='PANEL'
(***********************************************************)
(*  FILE CREATED ON: 04/18/2006  AT: 13:24:37               *)
(***********************************************************)
(*  FILE_LAST_MODIFIED_ON: 04/21/2006  AT: 12:29:23        *)
(***********************************************************)

(* ********************************************************************** *)
(*               VARIABLE DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_VARIABLE
 PANEL_BUFFER     [MAX_PANELBUFFER ]    (* BUFFER FOR RESPONSES FROM PANEL  *)
 PANEL_LAST_TEXT  [255]

(* ********************************************************************** *)
(*               LATCHING DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_LATCHING

(* ********************************************************************** *)
(*       MUTUALLY EXCLUSIVE DEFINITIONS GO BELOW                          *)
(* ********************************************************************** *)
DEFINE_MUTUALLY_EXCLUSIVE

(* ********************************************************************** *)
(*           SUBROUTINE DEFINITIONS GO BELOW                              *)
(* ********************************************************************** *)

DEFINE_CALL 'PANEL_SEND_COMMAND'(CMD_STR[128])
{
 SEND_COMMAND TP,"CMD_STR"
}

DEFINE_CALL 'PANEL_SEND_TEXT'(CH,MSG[60])
{
 CALL 'PANEL_SEND_COMMAND'("'TEXT',itoa(CH),'-',MSG")
 PANEL_LAST_TEXT = MSG
}

DEFINE_CALL 'PANEL_UPDATE_STATUS_TEXT'(MSG[60])
{
 CALL 'PANEL_SEND_TEXT'(200,MSG)
```

```
}

(***********************************************************)
(* Send Button Feedback Command to panel                   *)
(***********************************************************)
DEFINE_CALL 'PANEL_SEND_BUTTON_FB'(CH,VAL)
{
 IF (VAL>0)  [TP,CH] = 1
 ELSE        [TP,CH] = 0
}

(***********************************************************)
(* Send Level Command to all panels                        *)
(***********************************************************)
DEFINE_CALL 'PANEL_SEND_LEVEL'(CH,VAL)
{
 SEND_LEVEL TP,CH,VAL
}

(* ********************************************************************** *)
(*                STARTUP CODE GOES BELOW                                 *)
(* ********************************************************************** *)
DEFINE_START
 CREATE_BUFFER TP,PANEL_BUFFER          (* Panel Buffer from TP            *)
 PANEL_LAST_TEXT = ''

(* ********************************************************************** *)
(*            THE ACTUAL PROGRAM GOES BELOW                               *)
(* ********************************************************************** *)
DEFINE_PROGRAM

(* ********************************************************************** *)
(*                      END OF PROGRAM                                    *)
(*        DO NOT PUT ANY CODE BELOW THIS COMMENT                          *)
(* ********************************************************************** *)
```

```
PROGRAM_NAME='LB06_MAIN'
(***********************************************************)
(*  FILE CREATED ON: 04/18/2006  AT: 11:34:19               *)
(***********************************************************)
(*  FILE_LAST_MODIFIED_ON: 04/21/2006  AT: 12:29:23         *)
(***********************************************************)

(***********************************************************)
(* System Type :                                           *)
(***********************************************************)

(* ********************************************************************** *)
(*          DEVICE NUMBER DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_DEVICE
RS232_1     = 1                         (* RS232 channels  *)
RS232_2     = 2
RS232_3     = 3
RS232_4     = 4
RS232_5     = 5
RS232_6     = 6
RELAY       = 7                         (* channel 1...8   *)
```

```
IR1          = 8
IR2          = 9
IR3          = 10
IR4          = 11
IR5          = 12
IR6          = 13
IO           = 14                        (* channel 1...6   *)
TP           = 170                       (* touchpanel      *)
TP_2         = 171
TP_3         = 172
TP_4         = 173

(* ********************************************************************** *)
(*               CONSTANT DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_CONSTANT
NO_DEVICE         = 65535                      (* Not a device_id
*)
EIB_MG            = RS232_1              (* b+b EIB-MG                     *)
(* ---------------------------------------------------------------------- *)
MAXIR             =   6                  (* Max IR channels               *)
MAXIRQDEPTH       =  30                  (* IR Queue depth                *)
MAXINP            =   6                  (* Max. Digital In               *)
MAXOUT            =   8                  (* Max Relays                    *)
MAX_MASTERBUFFER  = 100
MAX_EMGBUFFER     = 255
MAX_PANELBUFFER   =  64                  (* Max Panelbuffer               *)

(* ********************************************************************** *)
(*               VARIABLE DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_VARIABLE
MASTER_MSG_BUFFER[MAX_MASTERBUFFER]      (* BUFFER 'MSG.LIB'              *)
(*------------------------------------------------------------------------*)
IR_CMDQUEUE[MAXIR][MAXIRQDEPTH]          (* 6 x int.                      *)
IR_ADDPOS  [MAXIR]                       (* position where to add next    *)
IR_SENDPOS [MAXIR]                       (* position where to send next   *)
IR_MAP     [MAXIR]                       (* linear device map             *)
IR_PTIME   [MAXIR]                       (* pulsetime per device          *)
(*------------------------------------------------------------------------*)
INP_PI     [MAXINP]                      (* last inp state                *)
OUT_PI     [MAXOUT]                      (* last relay state              *)
(*------------------------------------------------------------------------*)
PS                                       (* current push channel          *)
RS                                       (* current release channel       *)
PD                                       (* current push device           *)
RD                                       (* current release device        *)
(*------------------------------------------------------------------------*)
MAIN_INIT_STATE                          (* main init state               *)
CONTROLLER_RESET                                 (* controller reset detected
            *)
(*------------------------------------------------------------------------*)
tmp

(* ********************************************************************** *)
(*               LATCHING DEFINITIONS GO BELOW                            *)
(* ********************************************************************** *)
DEFINE_LATCHING

(* ********************************************************************** *)
(*       MUTUALLY EXCLUSIVE DEFINITIONS GO BELOW                          *)
(* ********************************************************************** *)
DEFINE_MUTUALLY_EXCLUSIVE

(* ********************************************************************** *)
(*           SUBROUTINE DEFINITIONS GO BELOW                              *)
(* ********************************************************************** *)
INCLUDE 'PANEL.AXI'
INCLUDE 'EIB-MG.AXI'
```

```
(* ********************************************************************** *)
(*                 STARTUP CODE GOES BELOW                                *)
(* ********************************************************************** *)
DEFINE_START
CREATE_BUFFER 0,MASTER_MSG_BUFFER       (* BUFFER FOR 'MSG.LIB'           *)
ON[CONTROLLER_RESET]                          (* controller was reseted
            *)
MAIN_INIT_STATE = 0                          (* main init
            *)

(*CALL
'EMG_ADD_GA'(ADDR1,ADDR2,ADDR3,EIB_TYP,TP_TYP,TP_ITEM,TP_TEXT,TP_LEVEL,NAME[16],UN
IT[8],DISP_FORMAT)*)

(* Relays *)
CALL 'EMG_ADD_GA'(15,0,0,1,0,0,0,0,'','',0) (* 15/0/0 *)
CALL 'EMG_ADD_GA'(15,0,1,1,0,0,0,0,'','',0) (* 15/0/1 *)
CALL 'EMG_ADD_GA'(15,0,2,1,0,0,0,0,'','',0) (* 15/0/2 *)
CALL 'EMG_ADD_GA'(15,0,3,1,0,0,0,0,'','',0) (* 15/0/3 *)
CALL 'EMG_ADD_GA'(15,0,4,1,0,0,0,0,'','',0) (* 15/0/4 *)
CALL 'EMG_ADD_GA'(15,0,5,1,0,0,0,0,'','',0) (* 15/0/5 *)
CALL 'EMG_ADD_GA'(15,0,6,1,0,0,0,0,'','',0) (* 15/0/6 *)
CALL 'EMG_ADD_GA'(15,0,7,1,0,0,0,0,'','',0) (* 15/0/7 *)
CALL 'EMG_ADD_GA'(15,0,8,1,0,0,0,0,'','',0) (* 15/0/8 *)

(* Messewand *)
CALL 'EMG_ADD_GA'(2,2,0,5,1,110,100,1,'','%',0) (* 2/2/0 *)
CALL 'EMG_ADD_GA'(2,2,1,5,1,120,101,2,'','%',0) (* 2/2/1 *)
CALL 'EMG_ADD_GA'(2,2,2,5,1,130,102,3,'','%',0) (* 2/2/2 *)

CALL 'EMG_ADD_GA'(2,1,0,1,1,100,0,0,'L1','',0)  (* 2/1/0 *)
CALL 'EMG_ADD_GA'(2,1,1,1,1,101,0,0,'L2','',0)  (* 2/1/1 *)
CALL 'EMG_ADD_GA'(2,1,2,1,1,102,0,0,'L3','',0)  (* 2/1/2 *)

CALL 'EMG_ADD_GA'(2,3,0,1,1,111,0,0,'L4','',0)  (* 2/3/0 *)
CALL 'EMG_ADD_GA'(2,0,0,1,1,112,0,0,'','',0)    (* 2/0/0 *)
CALL 'EMG_ADD_GA'(2,0,2,1,1,113,0,0,'','',0)    (* 2/0/2 *)
CALL 'EMG_ADD_GA'(2,0,1,1,1,114,0,0,'','',0)    (* 2/0/1 *)
CALL 'EMG_ADD_GA'(2,3,1,1,1,121,0,0,'L5','',0)  (* 2/3/1 *)
CALL 'EMG_ADD_GA'(2,0,3,1,1,122,0,0,'','',0)    (* 2/0/3 *)
CALL 'EMG_ADD_GA'(2,0,5,1,1,123,0,0,'','',0)    (* 2/0/5 *)
CALL 'EMG_ADD_GA'(2,0,4,1,1,124,0,0,'','',0)    (* 2/0/4 *)
CALL 'EMG_ADD_GA'(2,3,2,1,1,131,0,0,'L6','',0)  (* 2/3/2 *)
CALL 'EMG_ADD_GA'(2,0,6,1,1,132,0,0,'','',0)    (* 2/0/6 *)
CALL 'EMG_ADD_GA'(2,0,8,1,1,133,0,0,'','',0)    (* 2/0/8 *)
CALL 'EMG_ADD_GA'(2,0,7,1,1,134,0,0,'','',0)    (* 2/0/7 *)

(* Wand Control *)
CALL 'EMG_ADD_GA'(2,2,3,1,1,135,0,0,'','',0)    (* 2/2/3 *)
CALL 'EMG_ADD_GA'(2,2,4,1,1,136,0,0,'','',0)    (* 2/2/4 *)

(* Dimm Control *)
CALL 'EMG_ADD_GA'(2,2,5,1,1,0,0,0,'','',0)      (* 2/2/5 *)
CALL 'EMG_ADD_GA'(2,2,6,1,1,0,0,0,'','',0)      (* 2/2/6 *)

(* M-BUS Minol *)
CALL 'EMG_ADD_GA'(5,1,1,12,1,0,110,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,2,14,1,0,111,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,3,14,1,0,112,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,4,14,1,0,113,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,5,14,1,0,114,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,6,14,1,0,115,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,7,14,1,0,116,0,'','',0)

(* M-BUS ABB *)
CALL 'EMG_ADD_GA'(5,1,10,14,1,0,120,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,11,12,1,0,121,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,12,14,1,0,122,0,'','',0)
```

```
CALL 'EMG_ADD_GA'(5,1,13,14,1,0,123,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,14,14,1,0,124,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,15,14,1,0,125,0,'','',0)
CALL 'EMG_ADD_GA'(5,1,16,14,1,0,126,0,'','',0)

(* ********************************************************************** *)
(*            THE ACTUAL PROGRAM GOES BELOW                               *)
(* ********************************************************************** *)
DEFINE_PROGRAM

PS = PUSH_CHANNEL                       (* current push channel            *)
RS = RELEASE_CHANNEL                    (* current release channel         *)
PD = PUSH_DEVICE                        (* current push device             *)
RD = RELEASE_DEVICE                     (* current release device          *)

IF(CONTROLLER_RESET)                    (* Controller restart              *)
 {
  SELECT                                (* checkout what to do             *)
   {
    ACTIVE (MAIN_INIT_STATE = 0):       (* Init EIB-MG RS232               *)
     {
      WAIT 20
       {
        CALL 'PANEL_SEND_COMMAND'("'PAGE-Main Page'")    (* goto Intro Page   *)
        MAIN_INIT_STATE = MAIN_INIT_STATE + 1
       }
     }

    ACTIVE (MAIN_INIT_STATE = 1):
     {
      CALL 'PANEL_UPDATE_STATUS_TEXT'('Bitte warten, Initialisierung ...')
      MAIN_INIT_STATE = MAIN_INIT_STATE + 1
     }


    ACTIVE (MAIN_INIT_STATE = 2):       (* Init EIB-MG RS232               *)
     {
      IF(EMG_INIT_STATE = 100)
       {
        MAIN_INIT_STATE = MAIN_INIT_STATE + 1
       }
     }

    ACTIVE (MAIN_INIT_STATE = 3):       (* Other init                      *)
     {
      CALL 'PANEL_UPDATE_STATUS_TEXT'('Initialisierung abgeschlossen.')
      WAIT 10
       MAIN_INIT_STATE = 100            (* Init Done                       *)
     }

    ACTIVE (MAIN_INIT_STATE = 100):     (* Init done                       *)
     {
      CALL 'PANEL_UPDATE_STATUS_TEXT'('')
      OFF[CONTROLLER_RESET]             (* init finished                   *)
     }
   }
 }
ELSE                                    (* normal operation                *)
 {
  PUSH [TP,100]
   {
    if([TP,100])  tmp = 0
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L1',itoa(tmp))
   }

  PUSH [TP,111]
   {
    if([TP,111])  tmp = 0
```

```
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L4',itoa(tmp))
   }

  PUSH [TP,101]
   {
    if([TP,101])  tmp = 0
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L2',itoa(tmp))
   }

  PUSH [TP,121]
   {
    if([TP,121])  tmp = 0
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L5',itoa(tmp))
   }

  PUSH [TP,102]
   {
    if([TP,102])  tmp = 0
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L3',itoa(tmp))
   }

  PUSH [TP,131]
   {
    if([TP,131])  tmp = 0
    else          tmp = 1
    CALL 'EMG_SEND_NAME'('L6',itoa(tmp))
   }

  PUSH [TP,51]
  PUSH [TP,52]
  PUSH [TP,53]
  PUSH [TP,54]
  PUSH [TP,55]
  PUSH [TP,56]
  PUSH [TP,57]
  PUSH [TP,58]
    [RELAY,PS-50] = NOT [RELAY,PS-50]
 }
(* ********************************************************************** *)
(*                       END OF PROGRAM                                   *)
(*         DO NOT PUT ANY CODE BELOW THIS COMMENT                         *)
(* ********************************************************************** *)
```

Description

The program was created and compiled with the Netlinx Studio 2.1 Build 2.1.0.64. It is based on the first example in this document.

For receiving EIB-telegrams the following calls have been added:

EMG_RECEIVE_MESSAGE:

In this function the data received over the serial interface are saved in the variable EMG_MESSAGE.

EMG_ENCODE_MESSAGE:

If a telegram has been received with EMG_RECEIVE_MESSAGE, in this function the contents of the received telegram is read and the according to the receivied group address e.g. the display of the touchpanel is updated.