

# **CTEIB3**

## **AMX goes Instabus®**



## **Manual for the Comm-Tec EIB-Gateway AXG-EIB and the CTEIB3 control software**

Revision 3.01  
**Date: 30. October 2001**

©2000 COMM-TEC  
Siemensstr. 14, 73066 Uhingen  
Phone: +49/7161/3000-0  
Fax: +49/7161/3000-400

# Table of contents

<b>TABLE OF CONTENTS.....</b>	<b>2</b>
<b>PREFACE.....</b>	<b>3</b>
WHAT IS IT FOR? .....	3
ROLE OF THE EIB INSTALLER.....	3
EIB FROM THE AMX PROGRAMMER'S POINT OF VIEW .....	3
EIB DETAILS WHEN USING THE GATEWAY .....	4
<b>SYSTEM DESCRIPTION .....</b>	<b>6</b>
HARDWARE COMPONENTS .....	6
SOFTWARE COMPONENTS.....	7
GROUPS, ADDRESSES AND FORMATS .....	9
TRIGGER.....	10
GATEWAY INSTALLATION.....	11
INSTALLATION OF THE RS232 BOX/CARD.....	11
CONNECTION GATEWAY-BOX.....	12
CONNECTION BOX-MASTER .....	12
TRANSFERRING THE BOX PROGRAM.....	12
ENTERING GROUP ADDRESSES IN THE BOX .....	13
SUPPLYING THE BOX FROM MAIN PROGRAM .....	13
<b>COMMUNICATING WITH THE BOX.....</b>	<b>14</b>
SYSTEM_CALLS .....	14
<i>Setting values</i> .....	15
<i>Feedback</i> .....	15
<i>Active requests (Polling)</i> .....	15
<i>Commands to the box</i> .....	16
<i>Data conversion</i> .....	16
TERMINAL MODE.....	17
CHANNELS AND LEVELS .....	17
<b>COMMAND SET OF THE BOX .....</b>	<b>18</b>
ADD – ENTER A GROUP ADDRESS .....	18
ADR – SELECT OUTPUT ADDRESS FORMATTING .....	18
BOXCH – LINK BETWEEN LOGIC CHANNELS OF 232++ AND 1-BIT ACTUATORS.....	18
BURST – LIMIT FEEDBACK SENDING SPEED .....	19
DEL – DELETE A GROUP ADDRESS .....	19
DELTA – DIFFERENTIAL TRANSMISSION ON/OFF.....	19
LIST – DISPLAY EXISTING DEFINITIONS .....	19
NOPOLL – DELETE A POLL TRIGGER.....	20
POLL - ACTIVE VALUE REQUEST.....	20
RESEND – SEND ALL VALUES AGAIN .....	20
RESET – PERFORM A GATEWAY RESET.....	20
SET – WRITE TO GROUP ADDRESS .....	20
START – START THE UPDATE PROCESS .....	20
STATUS – SHOW STATUS INFORMATION .....	21
STOP – STOP THE UPDATE PROCESS.....	21
UPLOAD – LOAD GATEWAY FILTER TABLE .....	21
WATCH – OBSERVE A GROUP ADDRESS .....	21
WHEN .. POLL – DEFINING A POLL TRIGGER .....	22
<b>APPENDIX .....</b>	<b>23</b>
COMPARISON „OLD“ CALLS – NEW CALLS.....	23
VALUES FOR DIM4 ACTUATORS.....	24
EXAMPLE: ENTERING THE GROUPS.....	25
EXAMPLE: MAIN PROGRAM .....	25
LOCAL CALLS TO REPLACE „OLD“ INITIALIZATION.....	28
FAQ'S .....	29
ERROR MESSAGES .....	30

## Preface

### What is it for?

The CTEIB3 software package is used to connect AMX control systems to the "European Installation Bus" EIB ("Instabus"). It provides an easy to use interface for developers to comfortably access the bus.

### Role of the EIB installer

It can not be expressed strongly enough: when connecting to an EIB system, solid knowledge of the EIB and close contact to experienced EIB installers is strongly recommended. A faulty set Reading flag in an actuator, or a restrictively programmed line coupler can be really hard to find without good analysis tools.

In no way can CTEIB3 configure an EIB system. The package is used to connect to a working EIB, and can access only those bus elements whose usage is permitted. Because of that, it should be planned together with the EIB installers if all desired functions are *allowed* to the bus components.

### EIB from the AMX programmer's point of view

Analog to AMX systems is the European Installation Bus – as the name implies – a bus system: all components are in principle connected to the same wire and share the available bandwidth.

The bus itself is a 2-wire cable, used to supply 24VDC to the devices, as well as transferring data between them.

Unlike AMX, the EIB system is a decentralized structure – there is no special master controlling the communication, but any device can send data to any other device.

A sophisticated protocol ensures that only one device is sending at any time, and collisions are avoided as much as possible.

All communication happens in the form of telegrams. A telegram is a data packet, that consists of the following parts:

- Sender address – Hardware address of the sending device
- Receiver address – Group address of the receiving devices
- Payload

Telegrams can be sent to several target devices, i.e. to switch off all lamps in a room simultaneously.

So there is a fundamental difference between sender and receiver addresses:

A source address is a hardware address, describing the *device* that is sending the telegram.

A destination address is a group address, describing the *function* to be influenced.

Every device on the EIB has exactly one hardware address, but can react to several group addresses. Also, it is possible, that several devices react to the same group address.

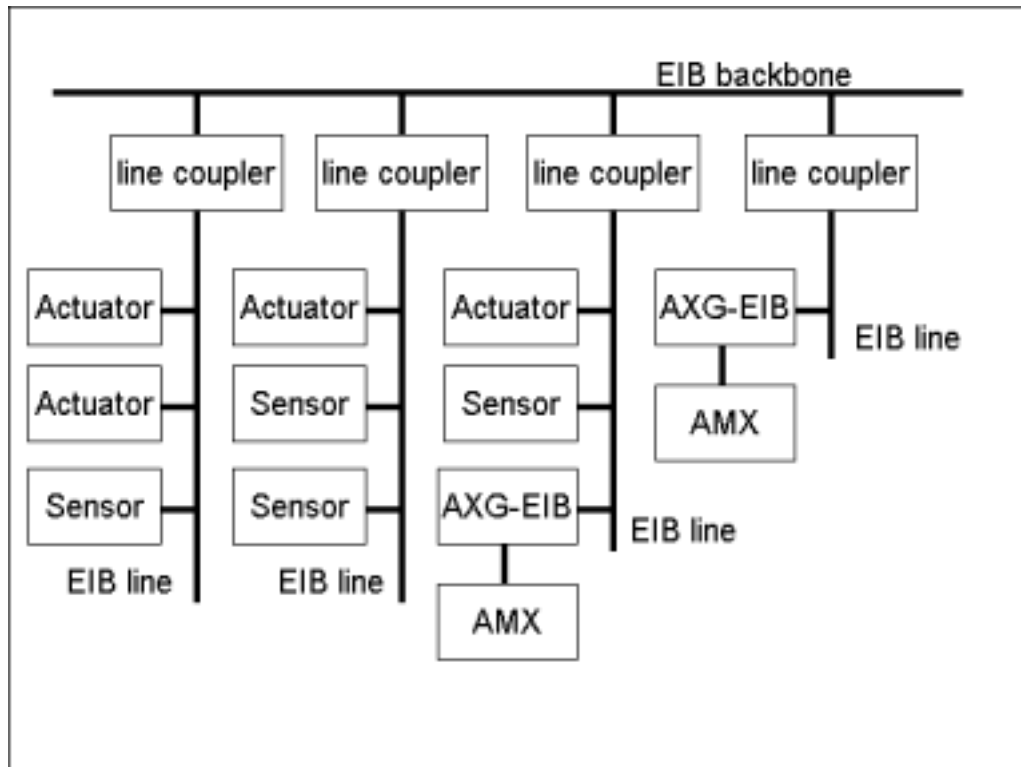
Both kinds of addresses are defined by the EIB installer.

Hardware addresses describe the kind and amount of used devices and are allocated during planning and installation. The gateway is not interested in hardware addresses at all.

The much more important addresses for AMX are the group addresses.

They define the functions an EIB installation can perform. So functions are simply used by sending specified values to group addresses.

## EIB details when using the Gateway



In principle, the EIB gateway AXG-EIB is a normal EIB device, and can therefore be connected at any location to the EI bus. Unlike simple actuators or sensors it can be responsible for a large amount (up to 1530) of group addresses – a normal dimmer for example, just reacts to four addresses.

So one must be very careful to ensure that the gateway has a real chance to react to all bus telegrams of interest. Especially when using line couplers, solid planning is absolutely necessary. The following points should be seriously considered:

On the one hand all telegrams should *reach* the gateway. If line couplers are inserted between gateway and controlled components, their filter tables are to be programmed in such a way that all relevant telegrams are really passed through.

On the other hand the EI bus couplers are sluggish – upon receipt of a telegram, an EIB device needs some time until the next telegram can be accepted. Especially scene controllers can generate a flood of telegrams, which are sent to all actuators participating in the scene. Because these are usually *different* devices, the „dead time“ of the bus couplers does not count much – each coupler has enough time to recreate before the next telegram gets received.

Not in the case of the gateway: usually *every* group address of a scene is of interest – the gateway should have the possibility to read in *all* the scene telegrams.

Problem: the first telegram acknowledgement is enough, but this does not necessarily come from the gateway! In fact, it can happen that an actuator which is participating in the scene acknowledges a telegram, but the gateway does not notify it, because it is still busy processing the previous telegram. If this should ever happen, there are two possible tactics:

1. The scene controller's parameters are changed, that it does not use the full bus bandwidth (approx. 50 telegrams/second), but just sends around 10 telegrams/second.
2. The gateway gets its own line, by isolating it from the „targets“ by its own line coupler. Then the line coupler can assert that all telegrams *to* the gateway are acknowledged *by* the gateway (as long, as the line couplers bus coupler is fast enough...).

## Foreword to Audit 3.01

The adaption of the software for the AXB-232++ (AXC-232++) has been a consequence of an elementary technical revision of the Gateways.

The main criterions for the new software have been

- a smoothly switch to the new product, under consideration of the knowledge already gained dealing with earlier revisions
- the possibility to equip already existing installations with the new Gateway
- the compatibility in combination with the NetLinx-Generation

Your AMX-Connections to an EIB-System can still be realized as it used to be. The internal software of the 232++ is automatically recognizing the linked-up Gateway – you don't have to take care of altered memory-administrations or something like that.

**Only innovation:** the data fields with the current Switch-, Dim4-, 1Byte-, 2Byte-, 3Byte-, 4Byte-rates must be declared as Integer-Arrays (also see the example on Page 28). The functions of the SYSTEM\_CALLs enclosed have not been changed. In case of an operation with a Gateway of the latest generation, certainly the latest revision has to be used.

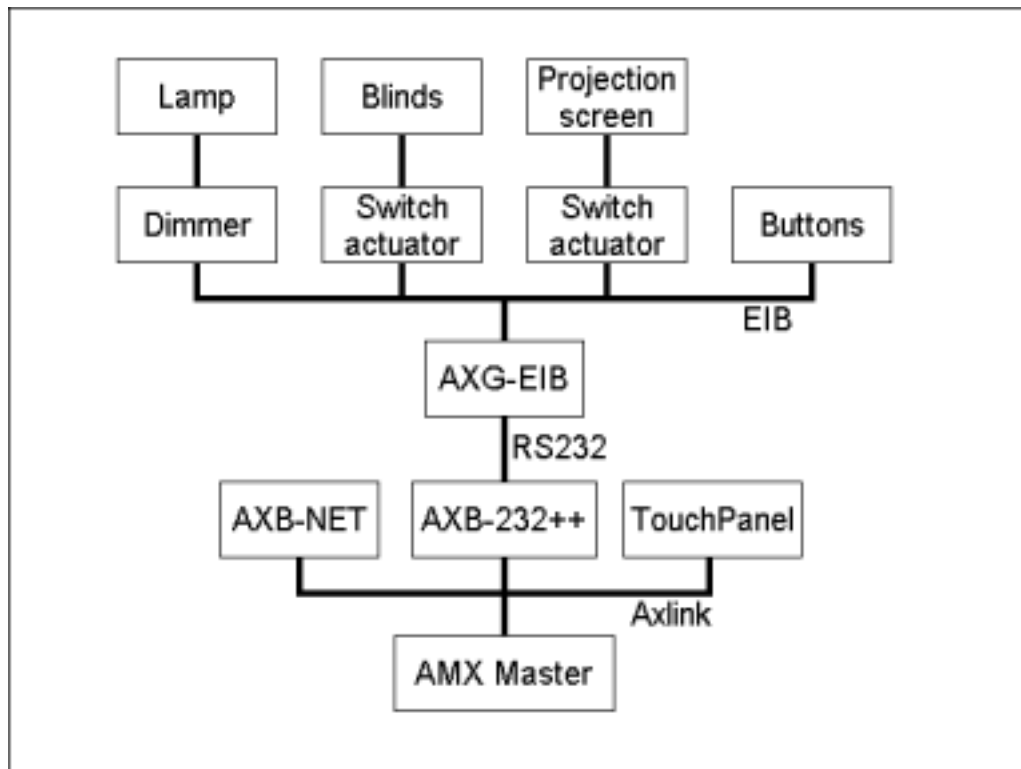
Further, it is not possible to change an operating Gateway without backing up the Master (the 232++, of course) first.

The Software-Version 3.01 including the SYSTEM\_CALLs is working together with all Gateways.

Uhingen, October 2001

## System description

### Hardware components



To connect an AMX system to EIB, several hardware components are necessary:

- An AMX-Master, processing the main program, which generates control commands and receives feedback from the EIB (called „Master“ in the following)
- An EIB-Gateway AXG-EIB to physically connect to the EIB, containing the bus coupler and a configurable packet filter („Gateway“)
- A smart RS232 interface AXB-232++ (bus device) or AXC-232++ (slot card) to manage EIB messages and commands („Box“).

The complete configuration data resides in the box, not necessarily in the main program. Though it is possible to transfer all addresses to the box upon every system restart (via `system_call`), there is no need to do so. It is sufficient if the main program knows the meaning of the „channels“ (type/number).

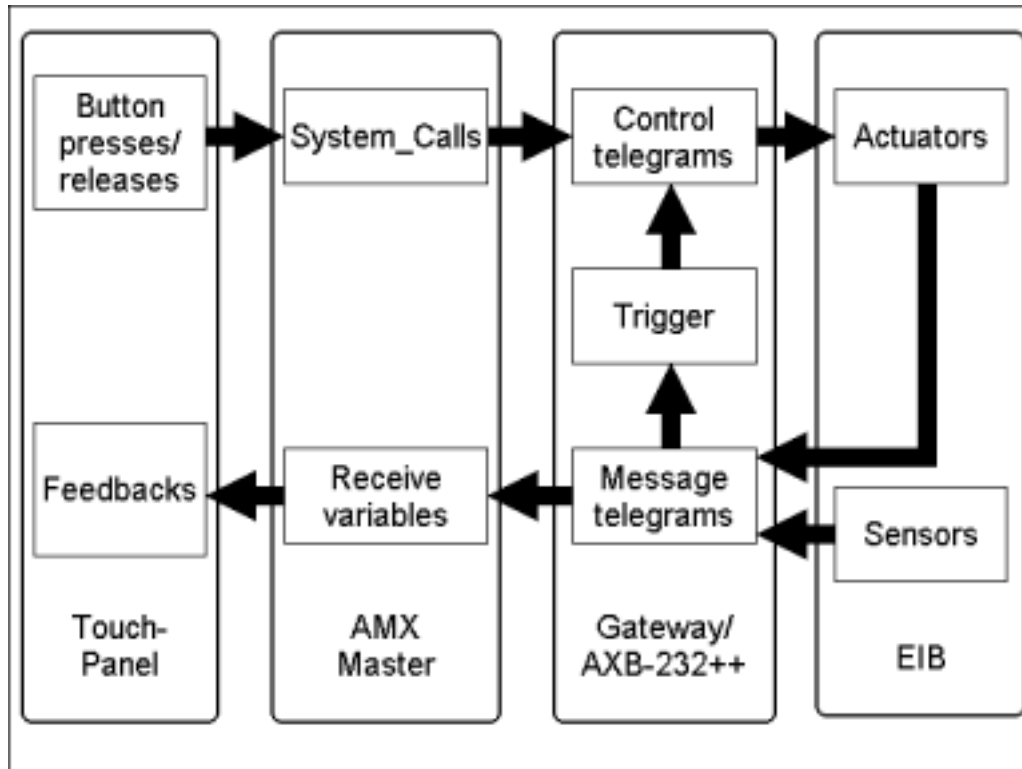
When addresses and/or types are changed, the box automatically updates the packet filter data in the gateway AXG-EIB. This ensures that all used messages are passed through (as long as they are generated on EIB side...), and that no „data junk“ from unused groups loads the system.

On every connection to the gateway, a checksum (generated by the gateway internally) gets checked against a stored version in the box. If there is deviation the gateway will be reconfigured.

On the AMX side, the handling is similar to that for Touch Panels: every action is handled by channel numbers, level numbers and text numbers; the appearance on the display plays no role for the program.

Also similar to Touch Panels: all application data is stored in memory which, even though it is backed up by a battery, is nevertheless volatile. It is recommended to keep a machine-readable version of the configuration after every change to quickly reestablish functions after an emergency case.

## Software components



The CTEIB3 software package consists of a control program for the RS232 box AXB-232++ and several program libraries (system\_calls), that can be included in your own (main) programs.

The control program gets loaded into the box (or slot card), and the box' interface is connected with the gateway AXG-EIB. The program is already compiled, and has only to be transferred for installation. All parameters are set by the AMX master.

On the AMX side, the gateway behaves - due to the usage of an RS232 box - like a „normal“ bus device; one can communicate with it, using „normal“ send\_command/send\_string statements. Because of the telegram structure of the EIB and the variety of possible messages and commands, the demands/requirements for the send and receive logic including packet processing are relatively high. So for performance reasons (response time), the included system\_calls should be used.

They define a set of functions that can be easily – by system\_call statements – included in your own programs. Due to their modular structure, only the really necessary program parts are included to save master's resources (memory, processing time, bus load).

## Data model

EIB defines several data types (normed by EIS), to fulfill different tasks. Apart from simple On/Off switching types, there are special data types for unsigned numbers in different widths, signed numbers, floating point values, percentages, date and time, wind directions and many more.

No one needs all of them in the AMX world, and CTEIB3 uses a simplified data model to cover the most common issues, but is also absolutely transparent to allow random access.

All in all are six different data types available:

- **Switch** Binary switch, On/Off
- **Dim4** 4-Bit dimming actuator, often used control type for dimmers
- **1Byte** All EIB data types with one Byte (8 Bit) length
- **2Byte** 2-Byte-EIB-types (16 Bit)
- **3Byte** 3-Byte-EIB-types (24 Bit)
- **4Byte** 4-Byte-EIB-types (32 Bit)

The first two of them are defined EIS types, which should cover at least 50% of the most installations. All other EIB types are mapped to the types 1Byte..4Byte – depending on their length. An element of the type 1Byte can therefore be an 8Bit integer number (-128..127), but also can be a percentage (0..100%).

All values are passed 1:1; if you ever want to get an 32Bit IEEE float value (EIS9) from the bus – feel free. But what the content of the four delivered „raw“ bytes means, is a problem of your application software. CTEIB3 is a pure transport vehicle, no interpretation of data happens.

The only data type that can be used in the AMX without a „pre wash“ is an unsigned 8Bit integer in the range 0..255 – a 1Byte type. Handy – because this is the type the most dimmers use for setting and reporting absolute values.

To use any other EIB data type you have to build some methods to convert this type into strings of according length – if possible in both directions and with checking for useful values...



## Groups, addresses and formats

Every component of an EIB system, that shall be accessible by AMX, is on the EIB side defined by an address, the group address. This 15Bit address signals on the bus which actuator(s) are being spoken to.

The gateway uses on the EIB side only the group address, a complete and actual list of all relevant group addresses is therefore absolutely necessary to install the system.

The ETS1 format displays group addresses by a (decimal) pair of numbers (Maingroup/Subgroup), where Maingroup HG must be in the range 0..15, the Subgroup UG in the range 0..2047.

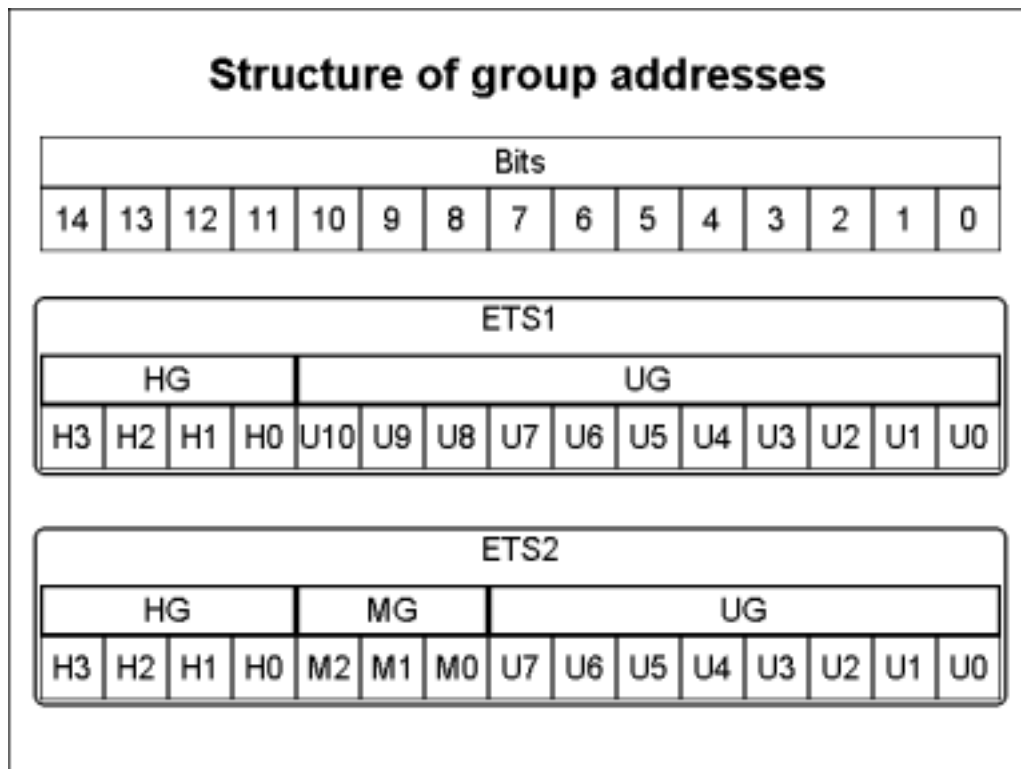
Example: 1/1027

With the introduction of ETS2 software, a new address format was specified, which uses a number triple HG/MG/UG (HG: Maingroup 0..15, MG: Middlegroup 0..7, UG: Subgroup 0..255).

Example: 1/4/3

**Whether group addresses are noted in ETS1 form (2 groups) or in the new ETS2 form (3 groups) plays no role – the box processes both. The difference is only in the style of writing.**

The following graphic displays the structure of the Group Addresses:



Both forms can easily be converted in the other form:

The Maingroup stays the same:

$$HG_{ETS1} = HG_{ETS2}$$

The 11 UG bits of ETS1 format divide in a 3:8 relation to MG and UG in ETS2 format:

$$MG_{ETS2} = UG_{ETS1} \text{ div } 256$$

$$UG_{ETS2} = UG_{ETS1} \text{ mod } 256$$

and

$$UG_{ETS1} = 256 \cdot MG_{ETS2} + UG_{ETS2}$$

**When specifying an address, the box automatically detects the format** and switches its output formatting to the last used form. Should a list in the other form be desired, the output form can be selected by using the "ADR 2" or „ADR 3" command (see below).

## Trigger

When having to do with EIB, it can sometimes occur, that value changes of actuators are not automatically notified over the bus. A typical example are light control actuators, which usually have four group addresses:

- 1 Bit switch function, On/Off
- 4 Bit relative dimming
- 1 Byte absolute dimming (setting values)
- 1 Byte value report

Remember: all four group addresses belong to the same lighting circuit!

If now the dimmer is accessed by one of the first three group addresses, the brightness of the connected lamp changes, and with that the value of the fourth group address.

This means, the 1Byte absolute dimming address does not necessarily contain the actual brightness value, though it is used to set the brightness to absolute values.

The only way to get the actual brightness, is to ask the fourth address for its actual value, to poll the address.

If one now always needs the actual brightness values of the lamp – i.e. to drive a bargraph on a touch panel – every change of one of the first three groups values must start a request to the fourth address.

This can be programmed „by hand“ by issuing a system\_call ‘EIB Poll ... ‘ upon detection of any change. But this can cause real work: for every interesting group address, an old compare value must be stored and this value must be compared in every main loop cycle.

This problem can be solved much more elegantly by using a trigger.

The box stores a reference for each group address, which address should be polled upon value changes. During definition of addresses (or later) it is specified which other address (in this case: the second 1Byte address) has to be requested for its value.

By including three ‘WHEN...POLL’ statements (see below) it is asserted that always the actual brightness value is available.

Independent of that, a value request can be started at any time by using the system\_call.

Constructions like that, where several actuators influence a „sum“ whose value has to be asked for explicitly, seem to occur often in EIB systems, for example in scene controllers.

So if you ever have to struggle with feedback that are „stuck“ or are just wrong, a well-placed poll trigger can sometimes really help.

But please hold back with using active value requests – the EIB bandwidth is used by *all* connected devices *commonly*.

There is no problem with polling some values from time to time, but polling all available groups every second is wasting resources and is a sure way to get trouble.

## Installation and startup

### Gateway installation

The EIB gateway AXG-EIB is mounted on a standard rail.  
The red/black clamps "Bus+" and "Bus-" are connected to the EIB.  
The clamps "230V~" are connected to mains power.



If the gateway shall have a hardware address assigned on EIB side, this can be done with the „Prog-Taste“ button and the ETS software. On delivery, the gateway has no hardware address assigned, and is therefore not „visible“ on the bus. In most cases, this is not necessary, but when accessing devices, that are mounted „behind“ a line coupler, this can become necessary. The assignment of a hardware address should always be done by an EIB installer!

### Installation of the RS232 box/card

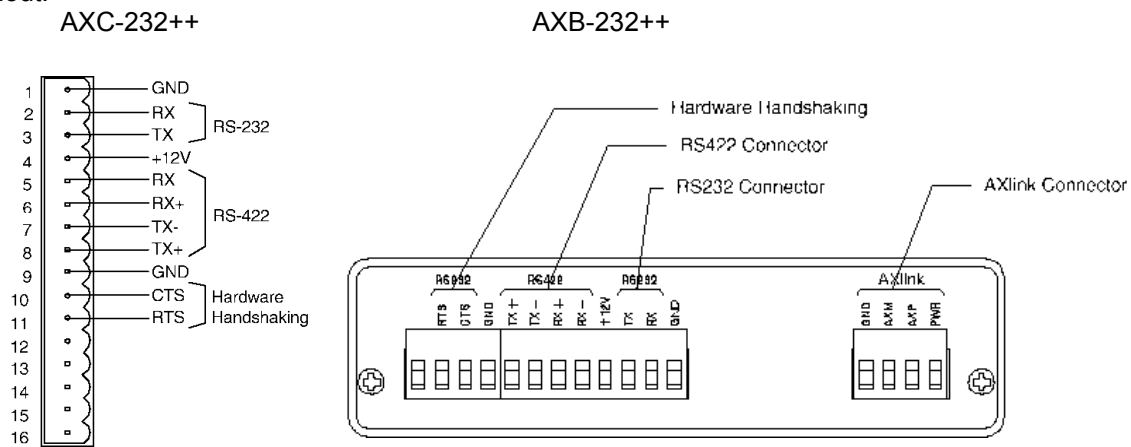
When using an AXB-232++, it gets mounted beneath (max. 30m away) the gateways (i.e. using the rack mount kit AC-RK), and assigned a unique device ID by the DIP switches on the front panel. When using a slot card AXC-232++, the server card's address and the number of the used slot define the card's device ID.

In both cases the RS232 parameters must be set to fit those of the gateway.  
The transmission from/to gateway is always done with 9600bps, 8 data bits, 1 stop bit, no parity (9600/8/N/1), so all switches except #7 are to be ON.

## Connection gateway-box

For this connection, a 5-wire cable has to be used, because the gateway needs hardware handshake (RTS/CTS).

Pinout:



### Gateway (SubD9)

Pin 2  
Pin 3  
Pin 4  
Pin 5  
Pin 6  
Pin 7  
Pin 8

### AXB-232++/AXC-232++ (Phoenix)

Rx  
Tx  
  
GND  
  
RTS  
CTS

## Connection box-master

The connection to the AMX master is done via Axlink wire with a maximum length of 900m.

## Transferring the box program

The control program for the RS232 box comes not as Access source code, but as a binary file that can be directly loaded into the box. For this transfer TOKEN is used, the command-line version of the Access compiler.

All necessary files can be found in the \BOXLOAD subdirectory of the delivered diskette.

The following files can be found in the \BOXLOAD subdirectory:

TOKEN.EXE	AMX token compiler
RTM.EXE	run-time library for token
SYSTEM	Binary file of the box program
BOXLOAD.BAT	A example batch file to directly load the box
EIBLOAD.EXE	A Windows program to load the box
VBRUN300.DLL	run-time library for eibload.exe

For loading the program, either the Windows program EIBLOAD can be used, which allows the comfortable selection of the used Com port, the master's baud rate and the device ID of the box, or TOKEN is started directly with the needed parameters (see batch file).

Function check: If the transfer was successful and the RS232 connection to the gateway is cabled completely (RTS/CTS?), then both LEDs on the front side of the box „Tx“ and „Rx“ should blink hecticly and after a few seconds stay lit permanently.

Tip: On some fast (!, >= 400MHz Pentium II) PCs seem to occur sometimes problems with the program transfer. Though the connection to the master is established, the transfer itself is aborted with a „Communications Error“ message.

This means: cable and PC's com port are OK, and the box's device ID is correct. Please check this first.

But if all of this is *really* set correctly, it could be a timing problem.

In fact there seems to be some relation between timing behavior of TOKEN and CPU's speed – for fast CPUs, the AMX master just reacts too slow on requests, token aborts the transfer.

The work-around sounds ridiculous, but did already work:

Copy the complete \BOXLOAD subdirectory to a diskette and start the transfer from there.

Seems to be slow enough...

## Entering group addresses in the box

Every group address that shall be accessed must be „introduced“ to the box.

After having found out which group addresses and data types the different functions have, all functions are divided into the six supported data types and then enumerated.

For each of the six types, the program can handle 255, so maximum 1530 group addresses can be contacted.

The RS232 box/card now gets a list of all addresses, types and numbers and stores it.

All commands from the master to the box, and all messages from the box to the master are from now on handled by just specifying type and number.

This has the advantage that the address does not have to be passed on every access, and it saves precious memory in the master.

The transmission can either be done from main program – by using the system\_call 'EIB Send Command', or directly in the terminal mode of the box (see below).

After having entered all addresses, the box „programs“ the gateway automatically; the formerly necessary step to load the gateway filter table with an external Windows software is no longer necessary.

A sample program to load addresses in the box can be found in the appendix.

## Supplying the box from main program

By including the delivered system\_calls in the main program, control functions to EIB can be sent and messages about value changes are sorted in the corresponding variables.

To enable the Axxess compiler to include those system\_calls, they first have to be copied in a directory on your hard drive. In *which* directory depends on how the compiler was installed.

On a standard installation from the AMX CD under Win95 or NT, all system\_calls reside in the subdirectory „c:\programs\shared files\AMXShare\SYCs“.

Please copy all \*.lib files from the \SYSCALL subdirectory of the delivered diskette there.

If you are using version 3.07 or higher of the Axxess compiler, please take a look at the version display screen (F1 File\Version...). If there exists an entry „AMXLIBS=...“, please copy all LIBs in the directory which is specified after the =.

If there is no such entry, the compiler will search the system\_calls in the *actual* directory only. In this case, please copy the LIB files to the same location as your source code files.

## Communicating with the box

### System\_Calls

Included in the CTEIB3 package are 16 system\_calls, that can be included in your own main programs.

To read in the most common data types (and to ease conversion of existing programs) the system\_call 'EIB Read Standard' can be used. It reads in all Switch, Dim4 and 1Byte values from the receive buffer.

To read in all six data types, the system\_call 'EIB Read All' is used.

For each of the six types exist a call to set an address, i.e. system\_call 'EIB Set Switch', and a call to actively request (poll) an address, i.e. system\_call 'EIB Poll Switch'

The fifteenth call - system\_call 'EIB Send Command' – is used to pass commands to the box, like defining group addresses.

The system\_call 'EIS5 to String' converts the data type EIS5 – a 2-Byte floating point value – in its decimal digit representation.

You can find the following files on the diskette in the \SYSCALL subdirectory:

File name	system_call name	meaning
CTEIB301.LIB	'EIB Read Standard'	read in Switch, Dim4 and 1Byte values
CTEIB302.LIB	'EIB Read All'	read in all values
CTEIB303.LIB	'EIB Set Switch'	set Switch values
CTEIB304.LIB	'EIB Set Dim4'	set Dim4 values
CTEIB305.LIB	'EIB Set 1Byte'	set 1Byte values
CTEIB306.LIB	'EIB Set 2Byte'	set 2Byte values
CTEIB307.LIB	'EIB Set 3Byte'	set 3Byte values
CTEIB308.LIB	'EIB Set 4Byte'	set 4Byte values
CTEIB309.LIB	'EIB Poll Switch'	poll Switch values
CTEIB310.LIB	'EIB Poll Dim4'	poll Dim4 values
CTEIB311.LIB	'EIB Poll 1Byte'	poll 1Byte values
CTEIB312.LIB	'EIB Poll 2Byte'	poll 2Byte values
CTEIB313.LIB	'EIB Poll 3Byte'	poll 3Byte values
CTEIB314.LIB	'EIB Poll 4Byte'	poll 4Byte values
CTEIB315.LIB	'EIB Send Command'	send commands to the box
CTEIB316.LIB	'EIS5 to String'	convert 2-Byte float to string

## Setting values

To set values on the EIB, the following six calls are used:

```
System_Call 'EIB Set Switch' (Dev, Val, Nr)
System_Call 'EIB Set Dim4' (Dev, Val, Nr)
System_Call 'EIB Set 1Byte' (Dev, Val, Nr)
System_Call 'EIB Set 2Byte' (Dev, Val[2], Nr)
System_Call 'EIB Set 3Byte' (Dev, Val[3], Nr)
System_Call 'EIB Set 4Byte' (Dev, Val[4], Nr)
```

Parameter:

<i>Dev</i>	Device ID of the Box
<i>Val</i>	Value to be set
<i>Nr</i>	Number of the actuator in the box

They can be used at any place in the program (like send\_string).

## Feedback

To read in value changes from the box into main program, the following calls can be used:

```
System_Call 'EIB Read Standard' (Buffer[255], SW[255], D4[255], B1[255])
System_Call 'EIB Read All' (Buffer[255], SW[255], D4[255], B1[255], B2[255][2],
                             B3[255][3], B4[255][4])
```

Parameter:

<i>Buffer[]</i>	Receive buffer of the RS232 box/card
<i>SW[]</i>	Table of actual Switch values
<i>D4[]</i>	Table of actual Dim4 values
<i>B1[]</i>	Table of actual 1Byte values
<i>B2[][2]</i>	Table of actual 2Byte values
<i>B3[][3]</i>	Table of actual 3Byte values
<i>B4[][4]</i>	Table of actual 4Byte values

It is strongly recommended to directly use one of the receive calls in DEFINE\_PROGRAM to immediately process all waiting packets in the receive buffer. Additionally, the two receive calls take care of text messages from the box and reroute them into Axxcess' terminal window. So even if your application does not need feedback it is useful to include one of them.

## Active requests (Polling)

Active request can be inserted at any place in the program.

```
System_Call 'EIB Poll Switch' (Dev, Nr)
System_Call 'EIB Poll Dim4' (Dev, Nr)
System_Call 'EIB Poll 1Byte' (Dev, Nr)
System_Call 'EIB Poll 2Byte' (Dev, Nr)
System_Call 'EIB Poll 3Byte' (Dev, Nr)
System_Call 'EIB Poll 4Byte' (Dev, Nr)
```

Parameter:

<i>Dev</i>	Device ID of the box
<i>Nr</i>	Number of the actuator

After submitting a request, the program continues running; after receiving the value report, it is sorted in the main program variables by the receive call.

## Commands to the box

To pass commands to the box, the following call is used:

System\_Call 'EIB Send Command' (Dev, Command[50])

Parameter:

<i>Dev</i>	Device ID of the box
<i>Command</i>	Character string to be interpreted by the box

All commands can be used either by the system\_Call 'EIB Send Command' or directly in the terminal mode (see below).

## Data conversion

2-Byte EIB types are often used to display analog values, like temperatures. Usually, the EIS data type EIS5 is used for that, which contains a floating point value in 16 Bit. The 16 Bit is divided like this:

1 Bit	Sign	(S, 0..1)
4 Bit	Exponent	(E, 0..15)
11 Bit	Mantissa	(M, 0..2047)

The value itself computes like  $-1^S * M * 0.01 * 2^E$

⇒ value range: +/- 670760,96

Because the AMX-AXCESS does not know float types at all, but the values often have to be displayed anyway, (i.e. in a text button on a touch panel), the system\_Call 'EIS5 to String' offers the possibility to convert a 2-Byte EIS5 type in the according digit representation.

SYSTEM\_CALL 'EIS5 to String' (Bytes[2], String[10])

Parameter:

<i>Bytes[2]</i>	Binary value to be converted
<i>String[10]</i>	Decimal digit string, i.e. '-670760,96'

The opposite way, to convert an AMX variable (16 Bit cardinal, 0..65535) into an EIS5 type is not supported because rounding losses can occur, and only a limited value range is available.



## Terminal mode

During installation of the EIB gateway it is probable that some address changes happen. But it would be just nasty to include all necessary commands in a test program every time. On the other hand, the AXB-232++ - apart from its RS232 interface (which is already used by the gateway)- has only the Axlink port to transmit configuration data.

But you can not randomly access the master over the Axlink connection because packets could be mistakenly exchanged with the main program. So a possibility is missing to talk with the box „bypassing“ the master.

Luckily enough, however, every AMX master offers such a mechanism.

In Access' terminal window, every keyboard command from the connected PC can be rerouted directly to a bus device, and every output from a bus device can be rerouted to terminal window by issuing a simple command. For this, the connection between the main program and the box gets cut temporarily. The command for that is PASS, followed by the box' device ID, i.e. „PASS 85“.

Since the box does not know about this it keeps sending (binary coded) values to the main program, so you have to draw the box' „attention“ to not get flooded with junk strings.

For this reason, the box permanently waits for the occurrence of the word „JUHU“ in the data stream (juhu is German for „yoo-hoo“), and then temporarily interrupts sending feedback.

In the other way, EIB values could change during „Smalltalk“ with the box. To send these changes to main program, the box has to know when it has direct connection to the master, and when rerouting is active.

This step back is done automatically 40 seconds after the last command, upon the next setting of a value or at the next value request.

To remove the rerouting (reconnect the master direct to the box), a special key sequence must be sent to the master:

**+ + Esc Esc** (twice Plus, twice Escape)

After that, the master prompt ">" should appear.

The general procedure looks like this:

- I. Jump to terminal window (Ctrl-T)
- II. Enable the master's echo function (enter „ECHO ON“)
- III. Reroute the box onto the terminal (PASS <Id>)
- IV. Draw the box' attention (JUHU)
- V. Talk to the box (List, Add, etc.)
- VI. Stop rerouting (+ + Esc Esc)
- VII. Exit terminal window (F10)

## Channels and levels

For two data types, there are additional access mechanisms:

All switch types (Switch) are mapped to the box' channels; when switching on, a PUSH[EIB,Nr] is generated from the box, and a RELEASE[EIB,Nr] is generated upon switching off.

Also, switching actuators can be accessed with the ON[],OFF[], TO[] and PULSE[] keywords.

The first(!) eight(!) 1Byte addresses are also available through levels (levels 1..8 of the box); upon change of one of these values, the actual value is passed as level to the main program, for example to drive a bargraph display.

The opposite way, to access actuators via SEND\_LEVEL is not supported.

## Command set of the box

To not having to carry around a huge set of system\_calls to initialize the box, all commands to the box and all error and status reports from there are done in plain text format.

Messages from the box are automatically redirected to the terminal window by the receive calls, (text) commands to the box are submitted by the system\_Call 'EIB Send Command'.

Commands consist of one or more words, separated by blanks (Space, \$20); capitalizing is ignored. Only the characters {'0'..'9','A'..'Z','a'..'z',' ','/','\$'} are allowed.

The following abbreviations are valid for data types:

SW	Switch
D4	Dim4
1B	1Byte
2B	2Byte
3B	3Byte
4B	4Byte

The box supports the following commands:

### ADD – Enter a group address

The ADD command adds a (new) group address, thus creating a connection between EIB group address and the type/number pair in the AMX. If group address or type/number are already used, the old entry will be replaced.

Syntax:        ADD <Type> <Nr> <Address>

Example:	ADD SW 1 08/15	Switch #1 has address 08/15
	ADD 4B 255 15/1234	4-Byte actuator 255 has address 15/1234
	ADD SW 2 1/2/3	also ETS2 notation is possible

### ADR – Select output address formatting

ADR switches the display formatting for group addresses between the old ETS1 form (HG/UG) and the new ETS2 form (HG/MG/UG). This selection is only needed for LIST outputs; when entering new addresses, the box automatically switches to the last form used.

Syntax:        ADR <Format>

Example:	ADR 2	Output in ETS1 form HH/UUUU
	ADR 3	Output in ETS2 form HH/M/UUU

### BOXCH – link between logic channels of 232++ and 1-Bit actuators

Short circuits occurring on the bus (AXLink) or a temporary loss of power can cause the master to reset the channels in the 232++ box. This, in turn, turns off all 1-bit actuators.

The BOXCH command disengages the direct link between the logic channels of the 232++ box and the 1-bit actuators, yet still allows the 1-bit actuators to be switched as before with the standard SYSTEM\_CALL 'EIB Set Switch'(<Parameter>).

**PLEASE NOTE:** This command only treats the symptoms of an improperly installed system! Finding the source of the error is the responsibility of the system integrator or programmer.

Syntax:        BOXCH <Status>

Example:	BOXCH ON	(default) Direct link between logic channels of 232++ and actuators
	BOXCH OFF	No direct link

## BURST – Limit feedback sending speed

Normally, the box sends all pending data as fast as possible to main program. In very large main programs it can happen that the main loop cycle can be too long until the next processing of the receive buffer, resulting in buffer overflow.

With BURST, the sending speed of the box to main program can be limited to 10 packets/second, leaving the master enough time to process all of the buffer content.

Disadvantage: when receiving many short packets the reaction time increases...

Syntax: BURST <State>

Example: BURST ON (default) Send as fast as possible  
BURST OFF Send maximum ten packets per second

## DEL – Delete a group address

With the DEL command, group addresses can be removed from memory.

Syntax: DEL <Type> <Nr>

Example: DEL SW 4 Remove switch #4  
DEL 3B 123 Remove 3-Byte number 123

## DELTA – Differential transmission on/off

With the DELTA command, it can be specified if the box sends a packet to the main program only if the value of an actuator has changed, or if it sends a packet to the main program every time it receives one from the EIB – whether the value has changed or not.

Usually, only value changes are reported in order to keep traffic as low as possible. In some special(!) cases, it can be useful to transmit messages about every received value telegram to main program.

The RESEND logic (see below) is not influenced by this.

Syntax: DELTA <State>

Example: DELTA ON (default) Send only value changes to main program  
DELTA OFF Pass all received values

## LIST – Display existing definitions

LIST generates a plain text list of used group addresses and actual values per type.

Syntax: LIST <Type> <StartNr> [<EndNr>]

Example: LIST SW 1 How is switch Nr. 1 defined?  
=> Output:

:l sw 1  
Switch #1: 1/0/1 Val: \$0

LIST D4 1 255 Show definitions of all 4Byte types  
=> Output:

:list d4 1 255  
Dim4 #1: 1/0/112 Val: \$0 => polls 1Byte #2  
Dim4 #2: 1/0/122 Val: \$0 => polls 1Byte #4

## **NO POLL – Delete a poll trigger**

NO POLL deletes all automatic value requests, triggered by an address.

Syntax: NO POLL <Type> <Nr>

Example: NO POLL SW 13  
NO POLL 1B 217

## **POLL - Active value request**

The POLL command sends a telegram on the EIB bus to inquire the actual value of a group address (whose „reading“ flag has to be set for that). This command should be used in terminal mode only; for active requests from main program please use the appropriate system\_calls.

Syntax: POLL <Type> <Nr>

Example: POLL SW 17 Request actual value of switch 17  
POLL 2B 128 What value has 2Byte actuator number 128?

## **RESEND – Send all values again**

After a RESEND command, the box sends the actual values of all known groups to the main program to synchronize memory. This function is started automatically after exiting the terminal mode.

Syntax: RESEND

## **RESET – Perform a gateway reset**

By performing a gateway reset, all pending packets in the gateway are discarded, the bus coupler is initialized, and all marked group addresses are polled (see WHEN...POLL).

Syntax: RESET

## **SET – Write to group address**

With the SET command, each address can be accessed to check its function in terminal mode. Depending on the type, up to four (byte) values are to be specified, either as decimal or - with preceding \$ sign – as hexadecimal values. For write access from main program, please use the appropriate system\_calls.

Syntax: SET <Typ> <Nr> Value [<Val2> [<Val3> [<Val4>]]]

Example: SET SW 1 0 Switch off switch number 1  
SET D4 3 \$E Set dimmer 3 to 14  
SET 4B 217 1 2 3 4 Set 4-Byte actuator number 217 to {1,2,3,4}

## **START – Start the update process**

A START command restarts the transmission of values of the type previously stopped with the STOP command. By default, all six types are sent.

Syntax: START <Type>

Example: START SW From now on, send switch values again  
START 3B Send 3-Byte values

## STATUS – Show status information

STATUS generates a report with the most important gateway state parameters.

Syntax:        STATUS  
              => Output:  
                  :status  
                  Gateway Version: 2.15  
                  EEPROM Check: \$E922  
                  ADR mode: 3 groups (H/MM/UUU)  
                  DELTA mode: ON (report changes)  
                  BURST mode: ON (send full speed)  
                  WATCH mode: OFF  
                  BOXCH mode: ON  
                  No errors reported.

## STOP – Stop the update process

By issuing a STOP command, the transmission of value reports to main program for a special type can be disabled, for example to study received variable's contents without being disturbed by new packets.

Syntax:        STOP <Type>

Example:        STOP SW                    Stop sending switch values  
                  STOP 4B                  No more 4-Byte values, please

## UPLOAD – Load gateway filter table

With the UPLOAD command, a refresh of the gateway's internal filter table can be forced. This is not normally necessary because the box automatically detects if group addresses have changed, or a new (not yet programmed) gateway has been connected. But you never know.

Syntax:        UPLOAD

## WATCH – Observe a group address

With the WATCH function, the state of a group address can be monitored in order to check the function of an actuator or sensor during installation. Whenever a telegram of a watched address is received, the box generates a plain text message with the old and new value.

Syntax:        WATCH <Type> <Nr>  
                  or  
                  WATCH OFF

Example:        WATCH SW 4                Watch switch number 4  
                  WATCH 2B 23              Watch 2-Byte number 23  
                  WATCH OFF                No more watching

Output:  
                  :watch sw 4  
                  OK, watching SW #4  
                  Change: SW #4: \$0 => \$1  
                  Change: SW #4: \$1 => \$0  
                  Change: SW #4: \$0 => \$1  
                  Change: SW #4: \$1 => \$0  
                  :watch off  
                  WATCH mode set off

A WHEN...POLL statement can be used to automatically request (poll) an actuator when a value of another actuator has been changed. This is helpful if write access to some addresses do influence other groups which do not automatically report their value changes (like dimmers or scene controllers).

Syntax: WHEN <Type> <Nr> POLL <Type> <Nr>  
or  
WHEN START POLL <Type> <Nr>

22

## Appendix

### Comparison „old“ Calls – new Calls

Since the introduction of CTEIB3, names and tasks of the system\_calls have changed. To migrate existing projects to CTEIB3 without problems, it is necessary to replace the „old“ system\_calls 'CTCT9xxx' with the new system\_calls.

The most frequent calls – setting of values and reading in the feedback – follow the same conventions for parameter passing; in most cases it is sufficient to replace the system\_call names with the search/replace function (<Alt>-R in Axxcess).

Please find in following an overview of all system\_calls of the EIB software version 1.411 with their corresponding calls in CTEIB3.

<b>Old Calls (Version 1.411 ff.)</b>	<b>New Call (CTEIB3)</b>
<b>Gateway-Reset</b>	
'CTCT9000' (DEV)	'EIB Send Command' (DEV,'RESET')
<b>Switch processing</b>	
'CTCT9010' (DEV,ADR[7],NR)	'EIB Send Command' (DEV, ""ADD SW ',itoa(NR),' ',ADR[]"
'CTCT9011' (DEV,VALUE,NR)	'EIB Set Switch' (DEV,VALUE, NR)
'CTCT9019' (DEV,NR)	'EIB Poll Switch' (DEV,NR)
<b>Dim4 processing</b>	
'CTCT9020' (DEV,ADR[7],NR)	'EIB Send Command' (DEV, ""ADD D4 ',itoa(NR),' ',ADR[]"
'CTCT9021' (DEV,VALUE,NR)	'EIB Set Dim4' (DEV, VALUE, NR)
'CTCT9022' (DEV,DIR,STEP,NR)	--
'CTCT9023' (DEV,PANEL,BTN_ON, BTN_OFF,SW_NR,D4_NR)	--
'CTCT9029' (DEV,NR)	'EIB Poll Dim4' (DEV, NR)
<b>1Byte processing</b>	
'CTCT9030' (DEV,ADR[7],NR)	'EIB Send Command' (DEV, ""ADD 1B ',itoa(NR),' ',ADR[]"
'CTCT9031' (DEV,VALUE,NR)	'EIB Set 1Byte' (DEV,VALUE, NR)
'CTCT9039' (DEV,NR)	'EIB Poll 1Byte' (DEV,NR)
<b>Feedback</b>	
'CTCT9100' (PUFFER[255], SW[255],D4[255],D8[255])	'EIB Read Standard' (PUFFER[255], SW[255],D4[255],D8[255])

Please take special notice of the 4Bit dim actuators (Dim4), which can only be set directly to a value; the old calls CTCT9021..CTCT9023 are therefore all mapped to the new call 'EIB Set Dim4'.

To enter the group addresses (CTCT90x0), local calls can be used to translate the parameters, instead of using the ugly statement above. Examples can be found as block files on the diskette and below.

## Values for Dim4 actuators

The call 'EIB Set Dim4' passes the specified values directly to the actuator – the values sent back now really equal the actual values (in the range 0..15), that were sent.

The meaning of the 16 possible values are hereby defined in the EIB data type EIS2:

Bits 0..2 define the dimming speed (0: Stop, 7: Max)

Bit 3 defines the dimming direction (0: down, 1: up)

This results in the following table:

0	Stop	8	Stop
1	Slowly darker	9	Slowly brighter
2	...	10 (\$A)	...
3	...	11 (\$B)	...
4	darker	12 (\$C)	brighter
5	...	13 (\$D)	...
6	...	14 (\$E)	...
7	Fast darker	15 (\$F)	Fast brighter



## Example: Entering the groups

The following example program shows the transmission of group definitions to the box. In a loop, every 200ms a command is set to the box via system\_call 'EIB Send Command', to either connect a group address with a type/number pair, or to define an automatic value request (poll trigger). It is sufficient to do this loading once during system installation, because the box stores all data in its local memory. The box has to be brought to the latest state only after changes in the configuration. For this purpose a small independent loader program that is temporarily loaded into the master is just more handy. We strongly advise against integrating the initialization in the DEFINE\_START portion of the main program. SYSTEM\_CALLS are already working in the mainline while the program is still trying to add address groups in the Gateway.

```
PROGRAM_NAME='E - boxload demo to enter the addresses'
(*   DATE:01/11/00   TIME:11:44:53   *)
DEFINE_DEVICE
    EIB = 17  (* Device ID AXB-232++ / AXC-232++ *)

DEFINE_VARIABLE
    X          (* Counter *)
    Cmd[50]    (* Actual Command *)

DEFINE_START
    X = 0      (* Stop on reset *)
    WAIT 20    (* wait 2 seconds *)
    X = 1      (* enter first group *)

DEFINE_PROGRAM
WAIT 2        (* check every 200ms *)
IF(X)        (* something to enter ? *)
{
    SELECT
    {
        ACTIVE(X = 1): Cmd = 'ADD SW 1 1/1'  (* Switch 1: address 1/1 *)
        ACTIVE(X = 2): Cmd = 'ADD SW 2 1/2'
        ACTIVE(X = 3): Cmd = 'ADD SW 3 1/0/3'
                        (* Switch 3: address equals 1/3 in ETS1 notation *)
        ACTIVE(X = 4): Cmd = 'ADD SW 4 1/22' (* On/Off dimmer *)
        ACTIVE(X = 5): Cmd = 'ADD D4 1 1/23' (* Dimming relative *)
        ACTIVE(X = 6): Cmd = 'ADD 1B 1 1/24' (* Dimming absolute *)
        ACTIVE(X = 7): Cmd = 'ADD 1B 2 1/25' (* Actual dimmer value *)

        ACTIVE(X = 8): Cmd = 'WHEN START POLL 1B 2' (* read after reset... *)
        ACTIVE(X = 9): Cmd = 'WHEN SW 4 POLL 1B 2'  (* ...when switches... *)
        ACTIVE(X = 10): Cmd = 'WHEN D4 1 POLL 1B 2' (* ...when dimming... *)
        ACTIVE(X = 11): Cmd = 'WHEN 1B 1 POLL 1B 2' (* ...and when set *)

        ACTIVE(1): Cmd = "" (* The End *)
    }

    IF(Cmd <> "") (* something to send ? *)
    {
        SYSTEM_CALL 'EIB Send Command' (EIB,Cmd) (* send to box *)
        X = X + 1                                (* next one *)
    }
    ELSE                                          (* end reached *)
        X = 0                                  (* disable counter *)
    }
}
```

## Example: Main program

```
PROGRAM_NAME='E - Demo main program for EIB access'
(*   DATE:01/11/00   TIME:11:39:59   *)
```

```

DEFINE_DEVICE
    EIB = 17 (* Device ID of AXB-232++/AXC-232++ *)
    TP  = 128 (* Any control device, i.e. AXT-EL+ *)

DEFINE_CONSTANT
    Sw_Light1 = 1 (* Switch number light 1 *)
    Sw_Light2 = 2 (* Switch number light 2 *)
    Sw_Light3 = 3 (* Switch number light 3 *)
    Sw_Dimmer = 4 (* Switch dimmer on/off *)

    D4_Dimmer = 1 (* Dimming relative *)

    B1_Dim_Set = 1 (* Dimming absolute *)

DEFINE_CONNECT_LEVEL
    (EIB,2,TP,1) (* Display brightness on bargraph #1 *)

DEFINE_VARIABLE
    EIB_Buffer[255] (* Receive buffer of the box *)
    INTEGER EIB_Switch[255] (* Actual Switch values *)
    INTEGER EIB_Dim4[255] (* Actual Dim4 values *)
    INTEGER EIB_1Byte[255] (* Actual 1Byte values *)

DEFINE_START
    CREATE_BUFFER EIB,EIB_Buffer (* Connect receive buffer *)

DEFINE_PROGRAM
    (* Read in feedbacks *)
    SYSTEM_CALL 'EIB Read Standard' (EIB_Buffer,EIB_Switch,EIB_Dim4,EIB_1Byte)

    (* Switch light1, one button each for 'On' and 'Off' *)
    PUSH[TP,1] (* Light1 on *)
        SYSTEM_CALL 'EIB Set Switch' (EIB,1,Sw_Light1)

    PUSH[TP,2] (* Light1 off *)
        SYSTEM_CALL 'EIB Set Switch' (EIB,0,Sw_Light1)

    [TP,1] = [EIB,Sw_Light1] (* Feedback 'on' *)
    [TP,2] = not [EIB,Sw_Light1] (* Feedback 'off' *)

    (* Switch light2, one button toggles between 'on' and 'off' *)
    PUSH[TP,3] (* Light2 on/off *)
        SYSTEM_CALL 'EIB Set Switch' (EIB,not [EIB,Sw_Light2],Sw_Light2)

    [TP,3] = [EIB,Sw_Light2] (* Feedback 'on' *)

    (* Switch light3, button switches on for 10 seconds *)
    PUSH[TP,4] (* Light3 on for 10 secs *)
        {
            CANCEL_WAIT 'Light3'
            SYSTEM_CALL 'EIB Set Switch' (EIB,1,Sw_Light3)
            WAIT 100 'Light3'
            SYSTEM_CALL 'EIB Set Switch' (EIB,0,Sw_Light3)
        }

    [TP,4] = [EIB,Sw_Light3] (* Feedback 'on' *)

```

```

(* Dimmer control *)
PUSH[TP,5] (* Dimmer off *)
    SYSTEM_CALL 'EIB Set Switch' (EIB,0,Sw_Dimmer)

PUSH[TP,6] (* Dimmer to 25% *)
    SYSTEM_CALL 'EIB Set 1Byte' (EIB,64,B1_Dim_Set)

PUSH[TP,7] (* Dimmer to 50% *)
    SYSTEM_CALL 'EIB Set 1Byte' (EIB,128,B1_Dim_Set)

PUSH[TP,8] (* Dimmer to 75% *)
    SYSTEM_CALL 'EIB Set 1Byte' (EIB,192,B1_Dim_Set)

PUSH[TP,9] (* Dimmer to 100% *)
    SYSTEM_CALL 'EIB Set Switch' (EIB,1,Sw_Dimmer)

[TP,5] = (EIB_1Byte[B1_Dim_Val] = 0)    (* Feedback 0% *)
[TP,6] = (EIB_1Byte[B1_Dim_Val] = 64)   (* Feedback 25% *)
[TP,7] = (EIB_1Byte[B1_Dim_Val] = 128) (* Feedback 50% *)
[TP,8] = (EIB_1Byte[B1_Dim_Val] = 192) (* Feedback 75% *)
[TP,9] = (EIB_1Byte[B1_Dim_Val] = 255) (* Feedback 100% *)

PUSH[TP,10] (* Brighter *)
    SYSTEM_CALL 'EIB Set Dim4' (EIB,10,D4_Dimmer)

PUSH[TP,11] (* Darker *)
    SYSTEM_CALL 'EIB Set Dim4' (EIB,2,D4_Dimmer)

RELEASE[TP,10] (* Bright enough *)
RELEASE[TP,11] (* Dark enough *)
    SYSTEM_CALL 'EIB Set Dim4' (EIB,0,D4_Dimmer)

[TP,10] = ((EIB_Dim4[D4_Dimmer] >= 9) and (EIB_Dim4[D4_Dimmer] <= 15))
[TP,11] = ((EIB_Dim4[D4_Dimmer] >= 1) and (EIB_Dim4[D4_Dimmer] <= 7))

```

## Local calls to replace „old“ initialization

In many existing programs, all group addresses are sent to the box on every system restart, usually done by the system\_calls CTCT9010, CTCT9020 and CTCT9030.

They each enter one group address for a Switch, Dim4 or 1Byte actuator, doing in principle the same as an „ADD“ command (in terminal mode or via „EIB Send Command“).

For example, a

```
SYSTEM_CALL 'CTCT9010' (EIB, '0/815', 1)
```

would change to

```
SYSTEM_CALL 'EIB Send Command' (EIB, 'ADD SW 1 0/815')
```

It came up that the amount of work for this conversion is relative high, because the necessary parameters differ strongly. When having just a few groups, this is no serious problem, but when moving the definitions of several hundreds of group addresses to the new calls, the search/replace function of the Axxess compiler (<Alt>-R) does not help too much.

To minimize this effort, it is easier to include some DEFINE\_CALLs in the main program with the names CTCT9010..CTCT9030 that convert the „old“ parameters:

```
DEFINE_CALL 'CTCT9010' (DEVICE, ADDRESS[7], COUNTER)
  IF(DEVICE AND (DEVICE <= $FF)) (* Device OK ? *)
    IF(COUNTER AND (COUNTER <= $FF)) (* Counter OK ? *)
      SYSTEM_CALL 'EIB Send Command'
        (DEVICE, "'ADD SW ', ITOA(COUNTER), ' ', ADDRESS")

DEFINE_CALL 'CTCT9020' (DEVICE, ADDRESS[7], COUNTER)
  IF(DEVICE AND (DEVICE <= $FF)) (* Device OK ? *)
    IF(COUNTER AND (COUNTER <= $FF)) (* Counter OK ? *)
      SYSTEM_CALL 'EIB Send Command'
        (DEVICE, "'ADD D4 ', ITOA(COUNTER), ' ', ADDRESS")

DEFINE_CALL 'CTCT9030' (DEVICE, ADDRESS[7], COUNTER)
  IF(DEVICE AND (DEVICE <= $FF)) (* Device OK ? *)
    IF(COUNTER AND (COUNTER <= $FF)) (* Counter OK ? *)
      SYSTEM_CALL 'EIB Send Command'
        (DEVICE, "'ADD 1B ', ITOA(COUNTER), ' ', ADDRESS")
```

Then the only change is to remove the „system\_“ out of the keyword „system\_call“, like:

```
CALL 'CTCT9010' (EIB, '0/815', 1)
```

## FAQ's

### Question:

After having loaded the software in the box and having connected it, The Tx LED flashes, but the Rx LED stays dark.

### Answer:

The RS232 parameters are probably incorrect. Set the DIP switches on the box/card to 9600bps, 8 data bits, 1 stop bit, no parity => all switches on, except number 7.

### Question:

Neither Rx nor Tx are on – the software is in the box, and I even have connected the hardware handshake wires. But nothing happens on my AXB-232++.

### Answer:

Caution! The leftmost Phoenix connector pin of the AXB-232++ is not used. Maybe the handshakes (RTS and CTS) are off one pin to the left...

### Question:

Dimming of a lighting circuit works properly, but I do not get any feedback from EIB about it.

### Answer:

Perhaps it is an actuator that has a special group address to read out its value (ref. „Trigger“). Basically, it can be necessary for any EIB address to explicitly ask it for its actual value – not every sensor/actuator automatically generates a telegram upon value changes.

### Question:

When resetting the gateway, several functions are started without the AMX trying to do so.

### Answer:

This is probably caused by a value request (poll) of an address that defines a scene. Whenever a scene controller sends its actual state („scene 27 active“) on the bus, all actuators participating in the scene interpret this telegram as a command to activate this scene...  
Work-around: do not poll scene controllers.

### Question:

The 1-bit actuators suddenly turn off after I plug in my touch panel

### Answer:

Power has been lost either because the wrong cable is being used, or the installation was improperly carried out. The logic channels in the box have been turned off, and the link between the channels and the 1-bit actuators turns the actuators off. Check the wiring used and the installation details for permanent remedy, and use the BOXCH commands for a temporary fix.

## Error messages

In some situations the box generates messages that are displayed in the terminal window by the receiver calls (EIB Read...). They are preceded with the string „EIB:“, and contain plain text messages which are listed alphabetically below:

- **"Address <Address> not used"** The gateway has reported the value change of an unused address
- **"BA Busy: No connection to EIB"** The gateway has no connection to EIB
- **"BA Error: Internal Error"** Internal error of the gateway's bus coupler
- **"BA-Layer: EIB problem"** Problems accessing EIB
- **"Bad ADD address: <Address>"** The address parameter in an ADD is invalid
- **"Bad ADD number: <Nr>"** The number parameter in an ADD is invalid (0 or >255)
- **"Bad ADD parm count"** The ADD command is incomplete or has too many arguments
- **"Bad ADD type: <Type>"** The type argument in an ADD is invalid (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad ADR parm count"** The ADR command has none or more than one argument
- **"Bad DEL number: <Nr>"** The number parameter in a DEL is invalid (0 or >255)
- **"Bad DEL parm count"** The DEL command is incomplete or has too many arguments
- **"Bad DEL type: <Type>"** The type argument in a DEL is invalid (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad HELP parm count"** The HELP command has more than one argument
- **"Bad LIST parm count"** The LIST command has the wrong count of parameters
- **"Bad LIST start <Nr>"** Invalid start number in a LIST (0 or >255)
- **"Bad LIST type: <Type>"** The type argument in LIST is invalid (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad NOPOLL parm count"** Wrong parameter count in NOPOLL
- **"Bad POLL number: <Nr>"** Invalid number argument in POLL (0 or >255)
- **"Bad POLL parm count"** Wrong parameter count in POLL
- **"Bad POLL trigger - <Type> <Nr> unused"** The specified trigger in WHEN...POLL is unused
- **"Bad POLL type: <Type>"** Invalid type parameter in POLL (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad Poll <Type> #<Nr> unused"** The specified target is unused
- **"Bad SET - <Type> <Nr> unused"** The specified SET target is unused
- **"Bad SET <Type> - illegal value"** The specified values are not allowed for this type to SET
- **"Bad SET <Type> - wrong parm len"** SET value length does not fit the referenced type
- **"Bad SET number <Nr>"** Invalid number argument in SET (0 or >255)
- **"Bad SET parm count"** Wrong parameter count in SET
- **"Bad SET type <Type>"** Invalid type parameter in SET (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad WHEN - POLL expected"**
- **"Bad WHEN - START expected"** Invalid syntax of WHEN...POLL command
- **"Bad WHEN dest number: <Nr>"**
- **"Bad WHEN number: <Nr>"** Invalid number parameter in WHEN...POLL (0 or >255)
- **"Bad WHEN dest type: <Type>"**
- **"Bad WHEN type: <Type>"** Invalid type parameter in WHEN...POLL (SW, D4, 1B, 2B, 3B, 4B)
- **"Bad WHEN parm count"** Wrong parameter count in WHEN...POLL
- **"CRC: Gateway EEPROM CRC-Error"** Internal checksum error of the gateway
- **"Gateway Timeout"** The gateway has not answered for two seconds (cabling?)
- **"NAK: No EIB Device at <Address>"** No reaction to this group address on EIB
- **"Poll queue overflow"** The box has more poll requests than the gateway can handle
- **"Send buffer overflow"** The box' sending queue is full
- **"Term Timeout"** In terminal mode, no command has been entered for 40 seconds
- **"Term overflow... Bye"** The actual (terminal) command is longer than 80 characters
- **"Tx: Transmit buffer overflow"** The gateway's sending queue is full
- **"Unknown: <Cmd>"** The box does not know the specified command