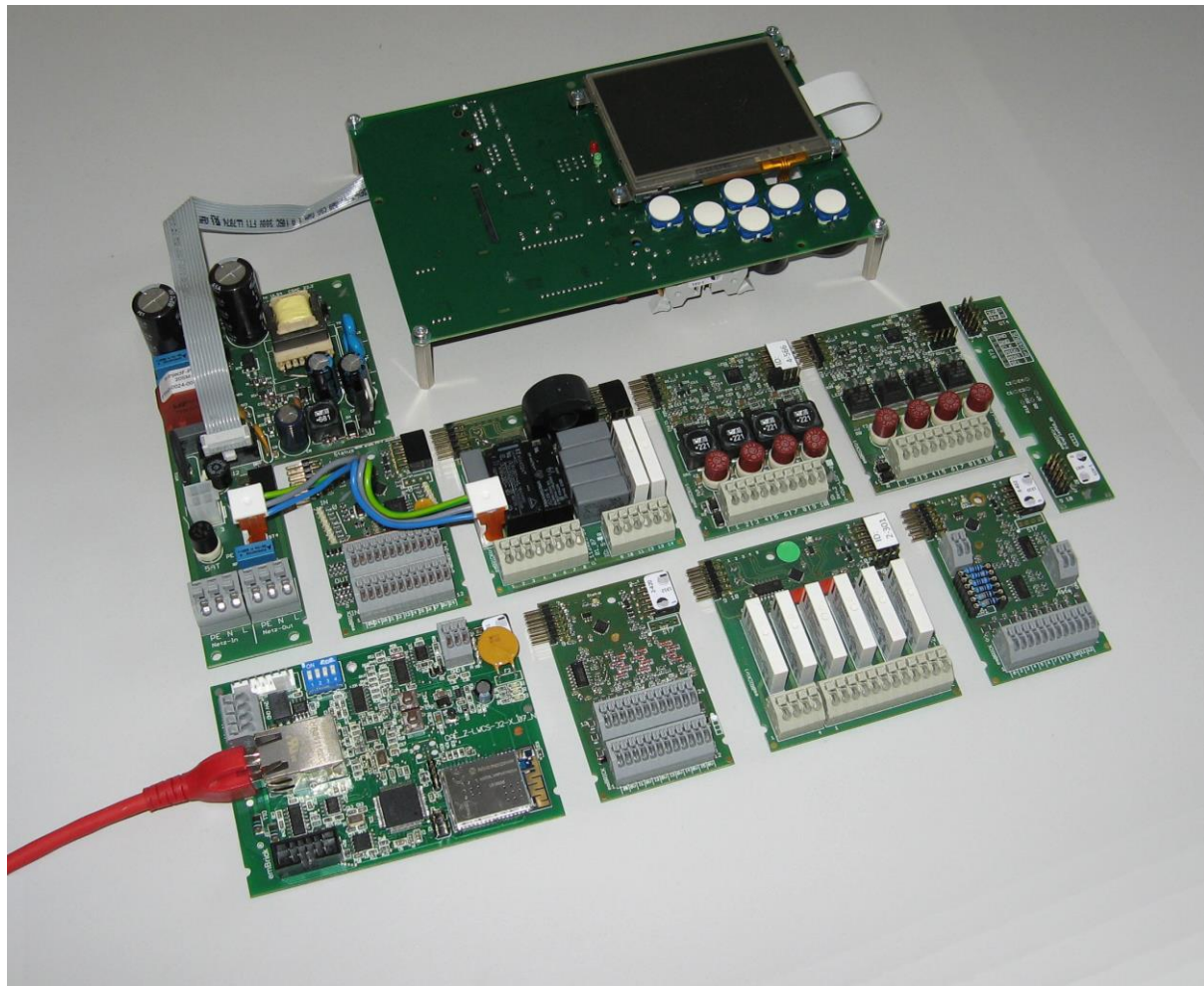


# ***emBRICK®* - EPC**

Embedded Patch-board Controller



# Programmers Manual

Rev. 12

emBRICK® is developed and supported by



IMACS GmbH  
Alfred-Nobel-Straße 2  
D – 55411 Bingen am Rhein  
[www.imacs-gmbh.com](http://www.imacs-gmbh.com)  
[www.embrick.de](http://www.embrick.de)

[support@embrick.de](mailto:support@embrick.de)  
Hotline: +49 (0) 7154 80 83 - 15

IMACS GmbH reserves the right to make changes without further notice to any products herein. IMACS GmbH makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does IMACS GmbH assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in IMACS GmbH data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. IMACS GmbH does not convey any license under its patent rights nor the rights of others.

copyright © IMACS GmbH 2022. All rights reserved.

Reproduction, in part or whole, without the prior written consent of IMACS GmbH is prohibited.

# Content

1. The emBRICK® Mission .....	5
1.1 Typical Applications.....	5
1.2 Basic Characteristics.....	6
1.3 Available Hardware Products.....	6
1.4 Available Host Platforms, Connectivity.....	6
1.5 Available Programming Platforms .....	6
2. Introduction .....	7
2.1 About this Manual.....	7
2.2 References / Manual Overview .....	7
3. Overview.....	8
3.1 Currently supported Programming Methodes/Languages.....	8
3.2 Currently supported Targets / Plattformen.....	9
4. Basic Information.....	10
4.1 Ways of Adaption.....	10
4.2 Ways of Programming.....	10
4.3 I/O-Addressing .....	11
4.4 Bus Protocol - Versions and Compatibility .....	11
4.4.1 History – Local Bus Protocol .....	11
4.4.2 How to get the protocol version of my Local Bus?.....	11
4.4.3 Remote Bus Protocol.....	11
5. Local-Bus Communication/Access .....	12
6. Remote-Bus Communication/Access .....	13
6.1.....	13
6.2 Features of Coupling Masters.....	13
6.2.1 LEDs, Status Indication.....	13
6.2.2 Errors .....	13
6.2.3 Timing.....	14
6.2.4 Synchronous vs. Asynchronous brickBUS .....	14
6.2.5 PC-Visualization.....	14
6.2.6 Plugin "LAN" .....	20
6.2.7 Plugin "CAN".....	20
6.2.8 Plugin "Modbus Large Block" .....	20
6.2.9 Plugin "Modbus Nativ" (planned) .....	21
6.2.10 Plugin "Local Application" (on demand) .....	21
6.2.11 Software History and internal repository .....	21
6.3 LAN/RSxxx Communications.....	22
6.3.1 Basics .....	22
6.3.2 Remote Bus Protocol Definition V4 .....	23
6.3.3 Data Transportation.....	28
6.3.4 Error detection and handling.....	29
6.3.5 Protocol History .....	30
6.4 CAN based remote communication (planned) .....	32
6.4.1 Basics .....	32
6.4.2 Mode of Operation .....	33
6.4.3 Description of Operation .....	33

6.4.4 Example of a Configuration Message .....	36
6.4.5 Examples of a Data Message .....	37
6.4.6 Examples of a Command/Status Message .....	39
6.4.7 Error detection and handling .....	39
6.4.8 Protocol History .....	39
6.5 ModBUS Large Block .....	40
6.5.1 Data representation .....	40
6.5.2 Supported commands .....	41
6.5.3 Restrictions .....	42
6.5.4 Data Transportation .....	42
6.5.5 Error detection and handling .....	43
6.5.6 Protocol History .....	45
6.6 Modbus Nativ (further feature) .....	46
6.6.1 .....	46
6.6.2 .....	46
6.6.3 .....	46
6.6.4 .....	46
6.6.5 .....	46
6.6.6 .....	46
6.6.7 .....	46
6.6.8 .....	46
6.6.9 .....	46
7. Local Bus Access Programming .....	47
7.1 Using C/C++ via Raspberry Pi (Raspian, Geany) .....	47
7.2 Using C/C++ via BeagleBoneBlack (Angstrom) .....	47
7.3 Using Model-based/C/C++ via radCASE .....	48
7.3.1 radCASE-Assign-Strings .....	48
7.3.2 Setting Outputs .....	49
7.3.3 Flowmeter .....	49
7.4 Using IEC61131 via CODESYS .....	50
7.5 Using Middlware based via Gamma .....	51
8. Remote Bus Access Programming .....	52
8.1 Using native C/C++ Programing (i.e. via MSVC) .....	52
8.2 Model-based/C/C++ with radCASE (IMACS GmbH) .....	52
8.2.1 Basics .....	52
8.2.2 Handling .....	52
8.3 Middleware Gamma (RST GmbH) .....	55
8.4 Modelbased Programing with eTrice (PROTOS GmbH) .....	55
8.5 IEC61131 Soft-PLC with logi.CAD3 (logi.cals) .....	55
8.6 IEC61499 Soft PLC with 4diac (fortiss GmbH) .....	55
8.7 IEC61131Soft-PLC with CODESYS (3S GmbH) .....	55
8.7.1 Create your own brick description .....	55
8.8 Phytion .....	58
8.9 Node-RED .....	58
8.10 Labview .....	58
8.11 .....	58
8.11.1 .....	58
8.11.2 .....	58
8.11.3 .....	58

9. Troubleshooting .....	59
9.1 Slavemodul state LED.....	59
9.2 Local Mode Operation .....	60
9.2.1 Check-List.....	60
9.2.2 radCASE Project.....	60
9.2.3 BeagleboneBrick/RaspberryBrick Projects .....	60
9.3 Remote Mode Operation .....	61
9.3.1 Log file.....	61

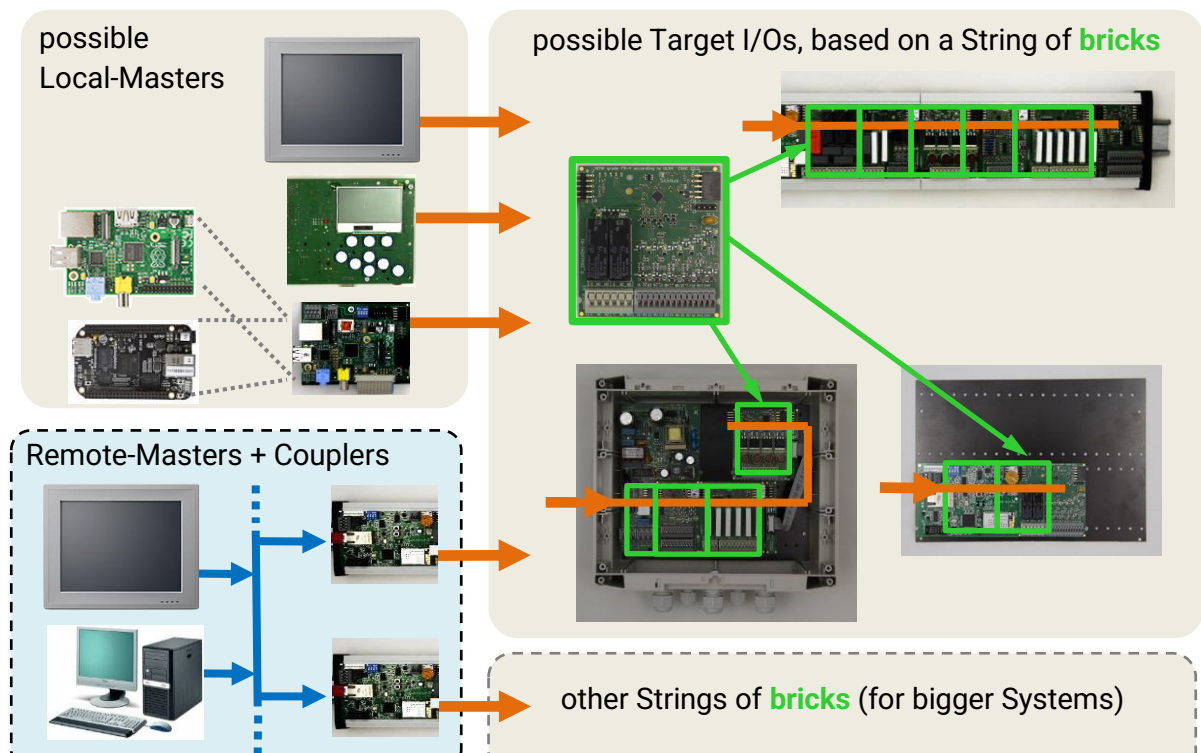
# 1. The emBRICK® Mission

The mission of emBRICK® is an **open** and **free** I/O system to ...

build **compact** and **industrial suited** electronic **control systems**  
by **assembling** small **existing/own** embedded **boards (bricks)** ...

... via a SPI-based **local interface** and optional **remote buses** (LAN, WLAN, CAN, RSxxx, ...).

We call this new class of controllers simple **EPC** (= Embedded Patch-board Controller).



emBRICK® combines in a perfect way the **cost-efficient** and **tailored** characteristics of a dedicated embedded system with the **ready to use** and **flexibility** of a PLC system.

To ensure a high acceptance, it is an open and free system. I.e. besides buying existing devices, everyone can develop his own components to realize easily his individually tailored, cost-efficient and industrial-suited measure and control system.

## 1.1 Typical Applications

- Small, medium and large size **measure and control systems**
- **Sectoral purpose**, with direct sensor/actor interface
- **Autonomous single box** control solutions i.e. with HMI and communication interfaces
- **Rapid hardware prototyping** system for control and measuring applications
- **PLC replacement** (i.e. with a Soft-PLC, IPC or an embedded controller)
- Medium and large size **distributed IO-systems** (i.e. building automation)
- Physical front-end for **IoT** (Internet of Things)

For more details see *Product\_Catalogue* and *Application\_Manual*.

## 1.2 Basic Characteristics

- **free** - also for commercial use in own appliances (for pure EMS with a license fee)
- **open** - supplying reference schematics, protocol source code, samples and starter kits
- **adaptable** to all systems, using common, low cost standard  $\mu$ Cs/components
- **half ... third price** compared to common control systems (complete system view)
- scalable local and remote topologies, **1 ... >1000 I/Os**, up to **1ms update**, deterministic
- **low own power** consumption, average 50mW/slave module in operation (outputs inactive)
- global and sector specific modules for **direct connection** of various **sensors and actors**
- **easy installation**, no configuration necessary, simple plug modules together and use
- works with / programmable by **various established**, well known **platforms / languages**

## 1.3 Available Hardware Products

Beside own developments, currently the following components are available from IMACS:

Slave-Modules ..... > 50 different modules for the sectors: General Purpose, Building Automation, Process Control (Safety, Medical/Analytics planed)

Master boards ..... Core: Cortex-M3/4, ARM9/11, PIC24/32; HMI: 128x64 ... WVGA

Adaption boards ..... for LAN, WLAN, CAN, RSxxx, Raspberry Pi, Beaglebone Black

Appliances / Enclosures ..... ready Single Box Controller for and top-hat rail and wall mounting

Starterkits ..... for MSVC, CODESYS, Raspberry Pi, Beaglebone Black

## 1.4 Available Host Platforms, Connectivity

*emBRICK®* can be adapted to all platforms with almost every footprint/performance. For master units, currently the following system implementations are available (others planed):

Computer platforms ..... PC, Embedded-PC, Module-PC, Raspberry Pi, Beaglebone Black

$\mu$ Controller platforms ..... ARM-Ax, ARM-Cortex-Mx, Microchip PIC24 / PIC32

Host Interfaces ..... Ethernet, CAN, RS232, RS485

Wireless Interfaces ..... WLAN

## 1.5 Available Programming Platforms

*emBRICK®* can be programmed by various systems, languages and IDEs (integrated development interface). Currently for master units the following systems are available (others planed):

OS / RTOS ..... Windows, Linux, FreeRTOS, proprietary

Programming languages ..... C, C++, IEC61131, Model-based (by implementing UML)

Model-based / Soft-PLC ..... CODESYS, radCASE, Enterprise Architect

C/C++ IDEs ..... MSVC, Cocox (GCC), MPLab (Microchip), Geany (Raspberry Pi), every other C/C++ IDE



## 2. Introduction

### 2.1 About this Manual

This manual describes the current available programming methods/languages and their host platforms to having access to *emBRICK®* components. It is addressed to application developers, who want to use existing development environments or will access the strings direct via a Coupler and the Remote-Bus.

For developers who wants to create own components or low-level driver (i.e. master adaption) please refer to the developers manual.

### 2.2 References / Manual Overview

For *emBRICK®* and *brickBUS®* the following documents are available. Before reading this document, it is recommended to read them in the given order:







- [System Manual](#).....(*embrick\_System-Manual\_#.pdf*) ... the basic system manual that contains the idea, the intention and the basic technical concept of *emBRICK®/ brickBUS®* like mechanics, electronics and communication protocol. It includes the glossary for all other documents.
- [Application Examples](#) ..... (*emBRICK\_Application-Examples\_#.pdf*) ... overview of typical *emBRICK®* device configurations and sample constellations for different industrial applications. It gives an idea how to use *emBRICK®* as an alternative to a normal PLC or an individual PCB / embedded system.
- [Product Catalogue](#) ..... (*emBRICK\_Product-Catalogue\_#.pdf*) ... contains the overviews and detailed datasheets of all IMACS-available *emBRICK®* components and products. This includes electrical and mechanical characteristics, terminal assignment and notes about their usage.
- [Programmers Manual](#)..... (*emBRICK\_Programmers-Manual\_#.pdf*) ... is the manual for application software programmers when using established programing systems like Embedded-IDEs, Soft-PLCs, CASE-Tools but also native C/C++-coding.
- [FAQ Manual](#) ..... (*emBRICK\_FAQ-Manual\_#.pdf*) ... contains answers to the most frequently asked questions about *emBRICK®* and its usage.
- Developers Manual ..... is the manual for system developers, who like to create their own slave modules or master adaptations. It includes all technical details specifications of *brickBUS®* and also sample schematics and code samples of the software stacks. This document is only available on request from IMACS GmbH and needs the agreement on the *emBRICK®* free license conditions. Please contact [support@embrick.de](mailto:support@embrick.de).



## 3. Overview

### 3.1 Currently supported Programming Methodes/Languages

Gray = planed / on request (please contact us)



	Product (Company)	Language	OS	Platt- form	IDE
	VisualStudio (Microsoft)	C, C++, C#	Windows	(E)PC	
	radCASE (IMACS)	UML, C/C++ Signal-Chart	all	all	
	Enterprise Architect (Sparxs Systems)	UML	all	all	
	eTRICE (Protos)	ROOM	all	all	
	CODESYS (3S)	IEC61131	Windows	(E)PC	
	4DIAC	IEC61499	all	all	
	logi.cad (logi.cals)		all	all	
	Labview		Windows	(E)PC	

## 3.2 Currently supported Targets / Platforms

Gray = planed / on request (please contact us)

Coupling: L = Local, direct via brickBUS

R = Remote, by coupling device via Ethernet, CAN, Modbus, RSxxx, etc.

	Product (Company)		OS	Cou- pling	IDE
	Raspberry Pi		Linux	L, R	
	Beaglebone (TI)		Linux	L, R	
	Arduino				
	ARM Cortex M3/M4/M7		FreeRTOS	L	
	ARM 9/11		LINUX	R	
	Mircochip PIC24/32		none, FreeRTOS	L	

## 4. Basic Information

**Note:** For understanding the following content it is necessary to be familiar with the principles of *emBRICK®* and the different adaption methods "local" and "remote". Please refer to the System Manual.

### 4.1 Ways of Adaption

The access to the *emBRICK®* components can be executed (according the individual project size/requirements) by two possible **adaption methods**: Either via the...

- 1) **Local-Bus** (= *brickBUS®*) direct, i.e. an enhanced SPI or by bit-banging or the controller
- 2) **Remote-Bus** by using a bus-coupler (currently the CAE\_Z-LWCS-M32-0#) and the (currently available) interfaces LAN, WLAN, CAN or RS485

### 4.2 Ways of Programming

For both *adaption methods* (Local/Remote) the programing can be done...

- a) **Native**, by using the protocol description and develop own low-level drivers/hosts (see chapter 5, 6) or
- b) **Template-based**, by using an existing implementation with drivers, prepared for different platforms and development environments (see chapter 7, 8).

For *template-based* working, the following platforms with corresponding templates and drivers are available (grey = under development / planed):

Adapt.	Language	Hardware, Platform	OS	IDE
Local	C, C++, UML	PIC24/32, ARMx Cortex-Mx, AX	proprietary Free-RTOS Linux	div. GNU (coocox, ...) MPLAB X radCASE, Enterprise Architect
Local	C, C++	Raspberry PI	Linux	Geany
Local	IEC61131-3	Raspberry PI	Linux	CODESYS
Local	Python, Java	Raspberry PI	Linux	
Local	C, C++	Beagle Bone Black	Linux, Gamma	Geany
Remote	C, C++	PC, IPC, emb. PC	Windows, Gamma	MSVC
Remote	C, C++, UML	PC, IPC, emb. PC	Windows	radCASE, Enterprise Architect
Remote	IEC61131-3	PC, IPC, emb. PC	Windows	CODESYS
Remote	proprietary	Measuring Systems		Labview
Remote	proprietary	Building Automation Systems, DDC-GA		FHEM, myGEKKO, SYMCON

## 4.3 I/O-Addressing

To address an I/O on the different strings/modules, the following information is necessary:

- node-ID (string-ID), if multiple strings      1...x      (only for remote bus systems)
- module-number (inside the string)      1...32
- byte-offset (inside the module)      0...x      (see module data sheet)
- bit-number (inside the Byte), if Bit-I/O      0...7      (see module data sheet)

The discrete implementation depends on the chosen adaption methode or programming language / developing environment.

## 4.4 Bus Protocol - Versions and Compatibility

### 4.4.1 History – Local Bus Protocol

Version	Description	Features/Bugfixes
11	Basic Release	first official serial version
2	Enchantment	Enhanced checksum

Currently Local Bus Protocol version 2 is the actual version.

We are currently delivering emBrick Products with the enhancement 2

### 4.4.2 How to get the protocol version of my Local Bus?

The brick modules transfer their protocol version to the local master during initialization. Most application programs show this protocol version number in their brick analysis screen. Contact the supplier of your application or the supplier of the brickBUS stack used for further details. For bridge modules, the analysis screen shows the protocol version of every slave module.

The standard brickBUS Stack implementation (used e.a. in the bridge modules) checks the protocol version of every brick during initialization. In case the brickBUS stack does not support the protocol version of one brick, the brickBUS is halted and an error notification is generated.

### 4.4.3 Remote Bus Protocol

For details see chapter 6 below to learn about the protocol version history and how the Remote Bus protocol version is transmitted to the remote master. Contact the supplier of the Remote Master how this version can be seen.

## 5. Local-Bus Communication/Access

For details, refer to the *Developer Manual*

This can be requested for free at [support@embrick.de](mailto:support@embrick.de)

## 6. Remote-Bus Communication/Access

Remote bus access offers the possibility to connect emBRICK® components (i.e. I/O-modules) to a master, that does not support the brickBUS® (= the emBRICK® Local-Bus based on SPI) directly.

To do this, a **coupling master** hardware is used to "convert" standardized communications (TCP/IP, Modbus, CAN, ...) to the local brickBUS®.

Ready available coupling master device are:

- CAE\_Z-CouplingBrick\_# (EOL, not for new projects)
- CAE\_Z-PatBridgeMX-11 (open frame)
- CAE\_Z-UniBridgeMX-1#-RB (DIN Rail enclosure)

These devices connect to the remote brickBUS master device via LAN, WLAN, CAN, RSxxx etc to the local brickBUS.

### 6.2 Features of Coupling Masters

Note: The following description refers to the newest software version (see 6.2.11).

#### 6.2.1 LEDs, Status Indication

##### LED2 green:

Local Bus Indicator ..... Flashing (approx.. 5Hz) : Local Bus is configuring, at start or after failiure.  
 ..... Flashing (approx.. 1Hz) : Local Bus is operating.

##### LED1 orange:

Network configuration ..... Off : Not in network configuration mode and no error pending  
 ..... Flashing (approx.. 1Hz) : Network configuration, searching address from DHCP.  
 ..... Permanently on : in network configuration mode, connected.

Error notification..... Off : Not in network configuration mode and no error pending  
 ..... Morse code (5 bit, 0.5 sec = 1 / 0.2 sec = 0, Pause 5 sec) : an error is pending (see below).

##### LED3 yellow:

Remote Bus indicator..... Off : No connection to remote master  
 ..... On : Remote master connected and sending  
 ..... Flashing (approx. 1Hz) Connection to remote master is established, but no data is sent (timeout).

#### 6.2.2 Errors

Some errors make the LED1 (orange) send an error code repeated every 5 seconds. These errors will occur immediately after power on. The codes mean the following:

0x000: remanent data storage corrupteds or defective  
 0x001 .. 0x004: remanent parameter area corrupted

0x005 .. 0x008: remanent system parameter area corrupted  
0x009 .. 0x00C: remanent calibration data area (most times unused) corrupted  
0x00D .. 0x010: remanent process data area corrupted  
0x011 .. 0x014: REMA data area corrupted  
0x01F: internal program error which can not be repaired

The different numbers of certain errors are for use by embrick support to identify the error more specifically.

If such an error occurs, the program is halted. To continue, press the Config button.

CAUTION: Pressing the Config Button in this case sets the corrupted area back to factory default values! Therefore, the device will then enter the Configuration mode and the configuration must be checked and eventually updated / corrected.

If the error persists even after the configuration has been updated or in case of error code 0x1F please contact [support@embrick.com](mailto:support@embrick.com).

## 6.2.3 Timing

The timing depends on several conditions. Generally, the time between 2 consecutive send/receive cycles consists of 3 parts:

- The time required for transmission of the data via the physical data layer
- The Latency of the LWCS software
- The Latency of the Host software

The data transmission via RS485 runs with typically 1 M Baud (further possible: 9600 ... 1M Baud), therefore the transmission time of a 250 Byte (max. length) data block is 2.5 ms and approx. 0.1 ms for a simple request. The transmission time via Ethernet is highly dependant on the load of the Ethernet connection, but typically it is < 1ms.

The LWCS Software has a latency of up to 2 ms for LAN communication and up to 10 ms for ModBUS Large Block communication.

Totally, you can calculate with a time of 3ms + Latency of the host in case of the LAN communication via Ethernet and <= 13 ms + Latency of the host in case of the ModBUS Large Block RTU communication.

## 6.2.4 Synchronous vs. Asynchronous brickBUS

The local bus (= *brickBUS*®) can be operated in asynchronous mode (the bus is permanently updated) or in synchronous mode (the update is triggered by an event).

In synchronous mode every time a data exchange request (0x10h) is received a brick-BUS-update cycle is triggered. The update is also triggered if for 40 ms no data exchange request is received.

If the localmaster application software itself manipulates the IO's on the bricks, the brickBUS must be operated in asynchronous mode in order to make these local updates effective.

For programming details see documentation in the brickBUS stack code.

## 6.2.5 PC-Visualization

The coupling master application is used to start a new project with the LWCS, uniBridge or Pat-Bridge. This means you can connect your coupling master (LWCS, uniBridge or PatBridge) via



Ethernet with your computer and start the visualization. There are two types of visualization, the first one is a webpage, which opens in any browser on your computer under the IP address of the coupling master set with the DIP switches, see below. The other version is a standalone version which is coupled with the firmware version installed on the coupling brick. If you use the web visualization, you will see all necessary information of the coupling brick and you can set e.g. the communication channel and change the IP address. If you use the standalone version, you additionally will see all information about the brickBUS which can the coupling master read from it.

### 6.2.5.1 Switching IP Address

On all coupling master exists two options of changing the current IP-Address.

- Changing the IP-Address with the PC-visualization (see 6.2.5.4.1 or 6.2.5.3.2)
- Changing the IP-Address with the on-board DIP-switches (see 6.2.5.2)

You must change the IP-Address of your coupling master, if you will use more than one coupling master in your network.

### 6.2.5.2 Switching IP Address with DIP-switch

The DIP-switch position determines a value between 0...15 as:  $Sw1 + Sw2 \times 2 + Sw3 \times 4 + Sw4 \times 8$ .

The standard position of the onboard DIP-switch is "1000" (DIP Switch value 8). In this case the IP 192.168.3.10 is set to your coupling master.

#### Usage during power on:

DIP switch value 0: ..... use DHCP

DIP switch value 1: ..... IP-address set in the Visualisation

DIP switch value 2 - 7: ..... unused

DIP switch value 8 ... 15: ..... determines fix IP-address (192.168.3.10 ... 17)

Here is a list of the switch positions and the resulting IP-addresses:

Switch-positions	DIP-switches value in the VIS	IP-Address
0000	0	DHCP
0001	1	Software set Address
1000	8	192.168.3.10
1001	9	192.168.3.11
1010	10	192.168.3.12
1011	11	192.168.3.13
1100	12	192.168.3.14
1101	13	192.168.3.15
1110	14	192.168.3.16
1111	15	192.168.3.17

Please note :

1. When setting the DIP-Switches to "DHCP" you need a possibility to get the actual IP address of the coupling master. Tools for this purpose are available from different suppliers.
2. When setting the DIP-Switches to "Software set Adress", the shipped value of the IP address is 192.168.1.0

### 6.2.5.3 Web visualization

Our newest coupling bricks (patBridge, uniBridge) with preinstalled firmware versions V0.53 and later have an integrated web visualization where you can set main configuration settings of the communication between coupling brick and remote master.

To open the web visualization, you must connect your coupling brick via an ethernet cable with your computer (your ethernet port (TCP / IPv4 protocol) must be set to 192.168.3.1). After connecting your coupling brick with your computer and correctly setting the TCP / IPv4 protocol, you can open the visualization on any browser under the IP address 192.168.3.10 (standard IP address after shipping, also with any IP addresses you'll set). When the visualization is loaded, you will see an overview of your coupling brick with the following informations:

- Hardware-Version
- Firmware-Version
- Master communication channel
- brickBUS synchronous to Master (see **Fehler! Verweisquelle konnte nicht gefunden werden.**)
- and remote timeout

#### 6.2.5.3.1 Overview



Localmaster emBRICK®

Overview

Network Settings

### Overview

Hardware-Version: 1.0

Firmware-Version: 0.54

Master comm. chan.: Inactive ▼

brickBUS synch. to Master: No ▼

Remote timeout [s]: 1,00

Save settings

Copyright © 2019 IMACS GmbH, Mittelfeldstraße 25, D-70806 Kornwestheim

In this overview you can set the communication channel, which will be used for your application. You can choose between LAN/WLAN or ModbusLB (Modbus Largeblock). If the master communication channel is set to "Inactive", you won't communicate with any remote master, so you must set this value either to LAN/WLAN or ModbusLB.

With the value “brickBUS synch. To Master” you can set how the brickBUS will be updated with incoming data. “No” means, the brickBUS will be updated continuously and “Yes” means, the Bricks will wait for a startsignal from the RemoteMaster and start after this signal the local communication. This can be helpful if you will send big sized packages

With the remote timeout you will set the time which the coupling master will wait till it sends a request to the connected devices on the brickBUS. In this case, all Outputs are set to 0.

### 6.2.5.3.2 Network settings



**Localmaster emBRICK®**

---

Overview

**Network Settings**

### Network Settings

MAC Address: 04916251bd41

IP Address:

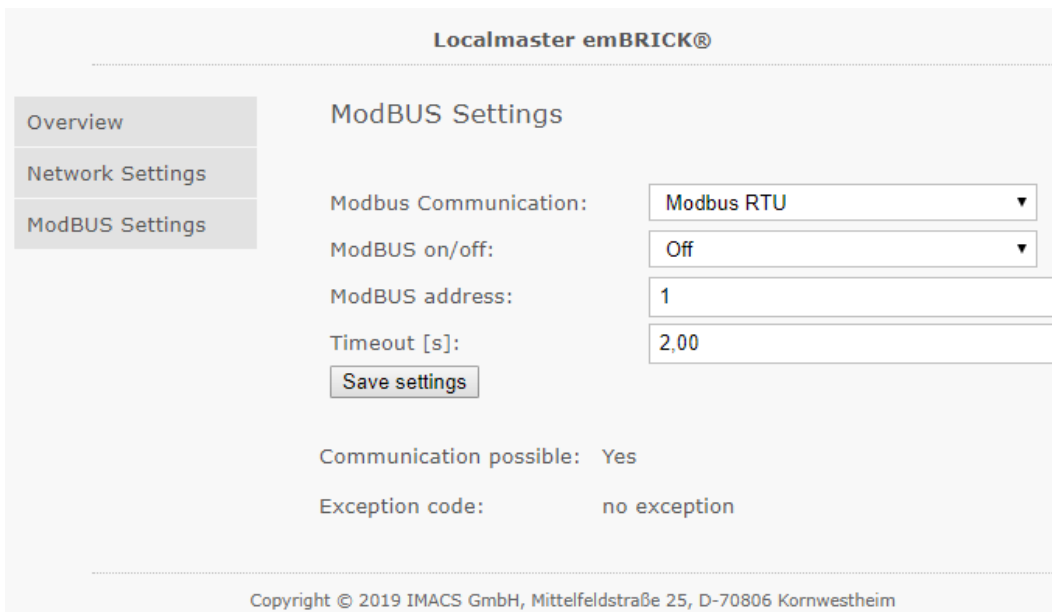
Subnet:

Gateway:

☐ Netzwerkeinstellungen automatisch beziehen (DHCP)

In the Network settings you will see the used MAC-address and you can change the IP settings of your coupling brick. Before doing this, set the DIP-switches to “0001”. If you don’t set them to the position “0001”, the IP Address of the DIP-switch is set. You can reboot the system by clicking on the onboard reset button.

### 6.2.5.3.3 ModBUS Settings



**Localmaster emBRICK®**

---

Overview

Network Settings

**ModBUS Settings**

### ModBUS Settings

Modbus Communication:

ModBUS on/off:

ModBUS address:

Timeout [s]:

Communication possible: Yes

Exception code: no exception

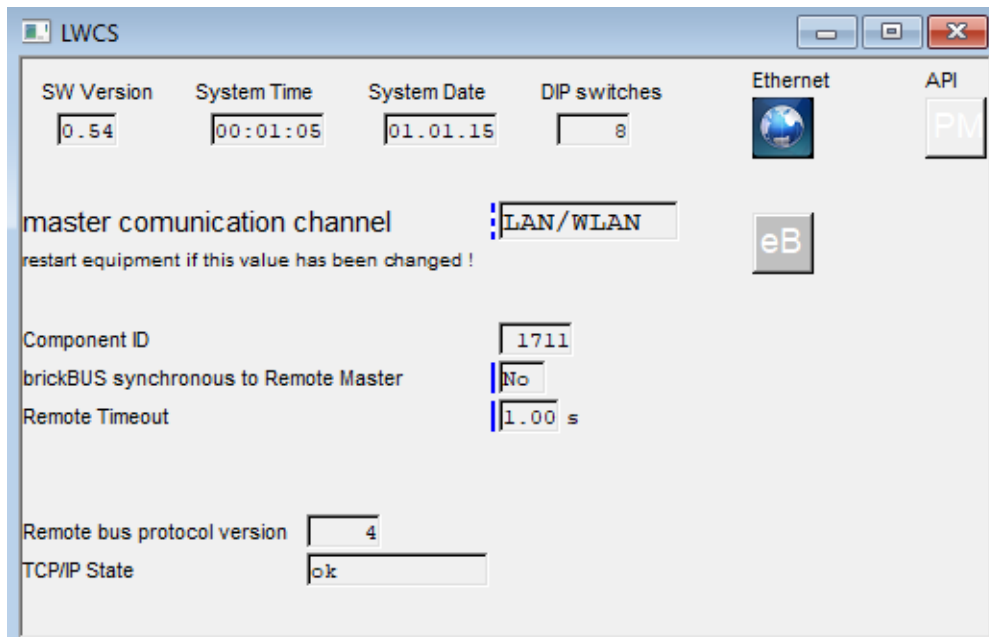
Copyright © 2019 IMACS GmbH, Mittelfeldstraße 25, D-70806 Kornwestheim

In the ModBUS settings you will see all necessary settings used for ModBUS. You can change the ModBUS communication between “Modbus RTU” and “Modbus TCP/IP”. You can turn ModBUS on or off, set your ModBUS address and the ModBUS timeout.

#### 6.2.5.3.4 Future planning

- Configuration of:
  - Baudrate-RSxxx, Com-Parameter-RSxxx
  - Baudrate-CAN, Address-CAN
- eB\_Overview
- setting of behavior by failure of the remote bus

#### 6.2.5.4 Visualization Overview (standalone version)

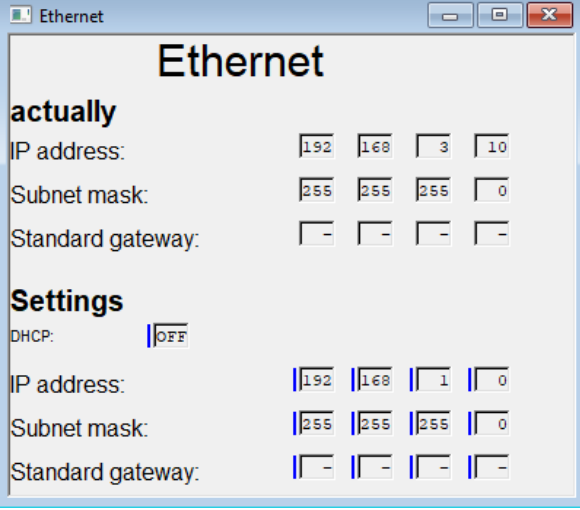


This is the start window of your visualisation. It shows:

- The installed software version on your coupling master
- The current System Time and Date
- The Value set with the DIP switches
- The state of the remote access. You can turn on / off the remote access between the coupling master and a RemoteMaster (PC)
- You can switch the synchronization between the RemoteMaster and the coupling master
  - Normal synchronization of the local BrickBUS is asynchronous
  - If the synchronization is set to synchronous, the Bricks will wait for a startsignal from the RemoteMaster and starts after this signal the local communication. This can be helpful if you will send big sized packages
- With the remote timeout you will set the time which the coupling master will wait till it sends a request to the connected devices on the brickBUS. In this case, all Outputs are set to 0.

##### 6.2.5.4.1 Switching IP Address with PC-visualization

When you connect the coupling master with the PC-visualization, you can click on the Button "Ethernet". The following window will open (window varies slightly depending on settings)



**Ethernet**

**actually**

IP address: 192 168 3 10

Subnet mask: 255 255 255 0

Standard gateway: - - - -

**Settings**

DHCP: ☐ OFF

IP address: 192 168 1 0

Subnet mask: 255 255 255 0

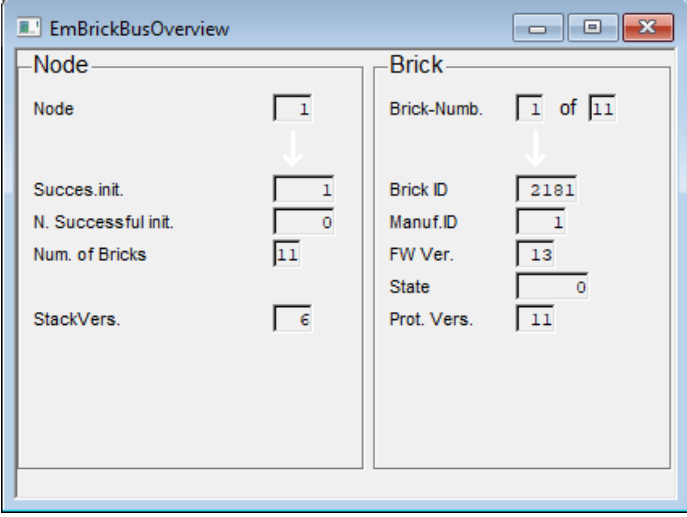
Standard gateway: - - - -

The upper area shows the current settings.

You can change the IP Address of your coupling master in the lower area. The new IP Address is valid after a restart of the coupling master. Before you do this, set the DIP-switches to "0001". If you don't set them to the position "0001", the IP Address is set according to the DIP switches. You can reboot the system by clicking on the onboard reset button.

#### 6.2.5.4.2 eB\_Overview

The next point in the main window is the "eB Overview". Here you will get all the information of the connectet slaveBRICKs.



**EmBrickBusOverview**

**Node**

Node 1

Succes.init. 1

N. Successful init. 0

Num. of Bricks 11

StackVers. 6

**Brick**

Brick-Num. 1 of 11

Brick ID 2181

Manuf.ID 1

FW Ver. 13

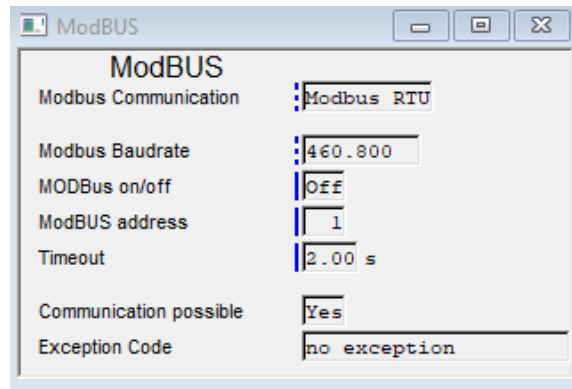
State 0

Prot. Vers. 11

- You can choose the Node and see
  - o The total number of initialization attempts (counts successfull and non successfull initialisations)
  - o the number of connectet Bricks on the bus
  - o and the used StackVersion
- on the right side you can switch between all connectet Bricks and you get the following informations
  - o Brick-ID

- Manufacture-ID
- Hardware-Version (FW-Ver.)
- State of the Brick
- Protocol-Version

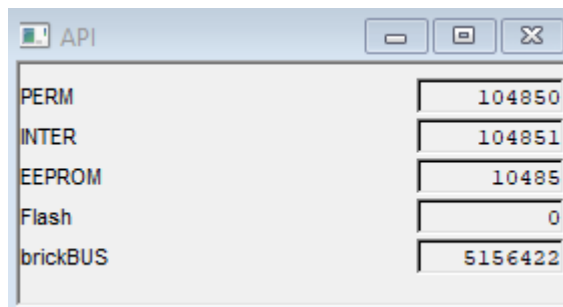
#### 6.2.5.4.3 ModBUS



In the ModBUS settings you can change the ModBUS communication type between RTU and TCP/IP. You can also set the baudrate when you use RTU over RS485, switch the communication on or off, set the ModBUS address and a timeout.

#### 6.2.5.4.4 API

The last point is the "API" in the mainwindow.



This point serves only for internal debugging purposes of IMACS manufacturing.

### 6.2.6 Plugin "LAN

See below LAN/RSxxx Communications for details of the protocol used.

### 6.2.7 Plugin "CAN"

See below CAN based remote Communications for details of the protocol used.

### 6.2.8 Plugin "Modbus Large Block"

See below ModBUS Large Block for details of the protocol used.

### 6.2.9 Plugin "Modbus Nativ" (planned)

See below ModBUS Nativ for details of the protocol used.

### 6.2.10 Plugin "Local Application" (on demand)

This plugin allows Programmers to code their own local application running e.g. in case the remote communication is off.

The remote communication plugins have a time counter which indicates the time elapsed since the last update from remote. This timer can be used to detect a break in the communication. In the default implementation, for pure bridge applications all brick outputs are set to 0 in case the communication is off for a certain time (settable as "Remote Timeout" between 0.01 and 9.99 seconds).

Care should be taken when both the remote communication and the local application manipulate the data for the brickBUS simultaneously. In this case it is not predictable which data is transferred via the brickBUS.LAN/RSxxx Communications.

### 6.2.11 Software History and internal repository

Software Version	Changes
0.35	internal, not released
0.36	internal, not released
0.37	released: problem with IP settings via DIP-switches fixed
0.03	First firmware with integrated ModBUS interface (communication changeable between ModBUS TCP or RTU)
0.50	First firmware including communication over LAN and ModBUS LB
0.51	Fixed crc calculation in ModBUS LB
0.52	Fixed connection problems with ModBUS TCP
0.53	Fixed problem with failing connections between pc-target programs a remote master
0.54	Fixed bug with checksum errors by ModBUS RTU
0.55	Special version without Ethernet interface possible

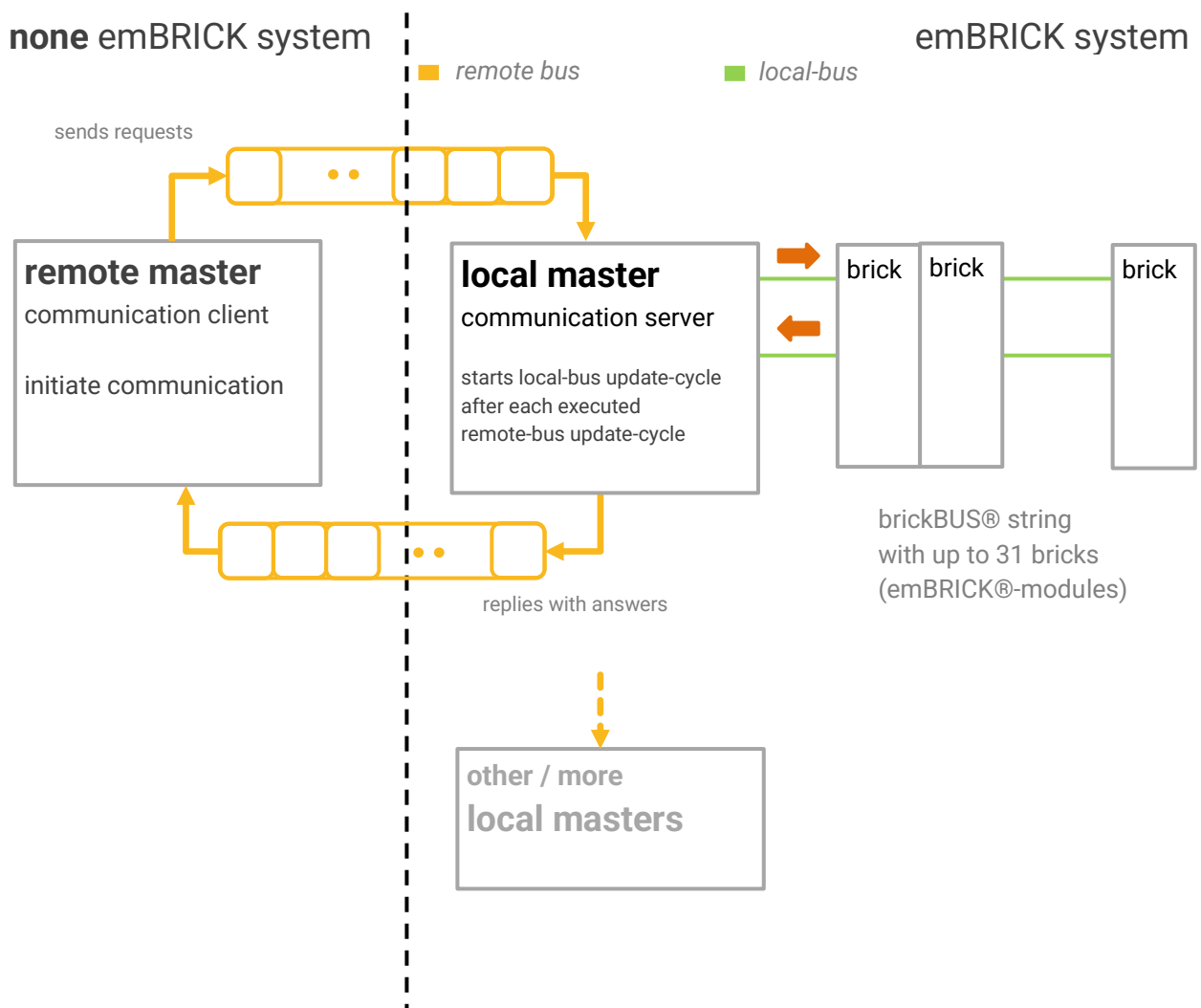


## 6.3 LAN/RSxxx Communications

### 6.3.1 Basics

This communication is used for different physical transportation layers that supports larger data blocks like Serial (RSxxx) and TCP/IP (LAN, WLAN). The common part of all transportation layers are the structure of the application data.

The remote-bus allows a connection between a *Remote-Master* (e.g. an ES, EPC, PC, PLC or other device) and a *Local-Master*. The *local-master* (= coupling master) distributes the incoming data to up to 32 *emBRICK*®-modules and get the state from the modules to send it (on request) to the remote master.



## 6.3.2 Remote Bus Protocol Definition V4

### 6.3.2.1 Communication block overview:

The protocol uses the same header/data structure throughout all communications:

header - 6 bytes	data - n bytes ( $n \leq \text{tcpip socket buffer size} - 6$ )
------------------	--

### 6.3.2.2 Header overview

LSB - length of message	MSB - length of message	command	reserved	reserved	reserved
----------------------------	----------------------------	---------	----------	----------	----------

The Header consists of the Command Byte (see below for possible Command Bytes), and the size of the message, including the Header. Thus, the length of a message is always equal to or greater than 6. Example: 6 Bytes Header + 200 Bytes Data → length of message = 206.

### 6.3.2.3 Available commands

Command	Effect
1	alive request: Local-Master answers „Hello World“, coded in ASCII, back to the client
2	configuration request: Local-Master returns the Configuration-Data of all connected bricks (modules)
3	Reset local bus
16	data update request with ... a) ... sending the Out-Image to the Local-Master. b) whereas the Local-Master answers by sending the last received In-image of the brickBUS to the Remote-Master. Then a new brickBUS update cycle is started (by writing the received data from the Remote-Master onto the brickBUS and buffer the received brickBUS data)
254	close request: The Local-Master closes the communication (e.g. a TCP/IP socket) that was opened by the Remote-Master. This command is in TCP/IP communication normally not required, since simply closing the communication from the master has exactly the same effect.

### 6.3.2.4 Command “1” – alive request

#### 6.3.2.4.1 Request

Send the following message (no data needed):

6	0	1	0	0	0
---	---	---	---	---	---

#### 6.3.2.4.2 Response

The Localmaster should respond with:

Header [6 Bytes] 18, 0 1, 0, 0, 0	Data [12 bytes] “Hello World”
--------------------------------------	----------------------------------

### 6.3.2.5 Command “2” – Configuration request

#### 6.3.2.5.1 Request

To request the configuration of the connected brickBUS-string (information about the modules found and initialized), the following message has to be sent to the *Local-Master* (no data needed):

6	0	2	0	0	0	0
---	---	---	---	---	---	---

#### 6.3.2.5.2 Response

Header [6 Bytes] (6+9+n*11)%256, (6+9+n*11)/256, 2, 0, 0, 0	Data [9+n*11 Bytes] Output Data, see below
--	---

Where n is the number of emBricks.

Most figures in the response are only usable if the status of the local master is “1”, i.e. if the Master is operating. Therefore, the response should be discarded if status of local master is “0”.

1<sup>st</sup> local-master data

Purpose	Byte no.	Example / Note
number of connected <i>bricks</i>	0 [1... 32]	01h (1 module found)
status of the <i>local-master</i>	1 [0...1]	01h (operating) / 00h (not operating)
component ID of the <i>local-master</i> (here 1-603 = 0643h)	2 (MSB)	06h
	3 (LSB)	43h
remote bus protocol version	4	04h (see below “Version History” for details)
software version of <i>local-master</i>	5 [1...255]	14h
Local Master manufacturer ID	6 [1...255]	01h (1=IMACS)
Reserved	7 (MSB)	00h (currently fix)
Reserved	8 (MSB)	00h (currently fix)

2<sup>nd</sup> followed by n blocks of the configuration of all connected (and initialized) bricks (I/O-modules). n is equal to the *number of connected bricks* received in the 1<sup>st</sup> part

Here: b = is the number of the corresponding brick [0...30]  
the example data are of the module *P-2Rel4Di2Ai-01*

Purpose	Byte no.	Example / Note
status of slave-module	9+(b*11) + 0	01h
Data Length MOSI (local Master -> Slave) [bytes]	9+(b*11) + 1	01h
Data length MISO (Slave -> Local Master) [bytes]	9+(b*11) + 2	06h (here: 1 + 2 + 2 + 1)
Slave local bus protocol version	9+(b*11) + 3	0Bh
Slave hardware revision	9+(b*11) + 4	12h
Slave device ID (here P-2Rel4Di2Ai-01: 5-131 = 140Bh)	9+(b*11) + 5 (MSB)	14h
	9+(b*11) + 6 (LSB)	0Bh
Slave manufacturer ID	9+(b*11) + 7 (MSB)	01h (1=IMACS)
	9+(b*11) + 8 (LSB)	
Data offset MOSI (local Master ->	9+(b*11) + 9	00h rising according used modules

Slave)		
Data offset MISO (Slave -> Local Master)	$9+(b*11) + 10$	00h rising according used modules

### 6.3.2.6 Command “3” – reset local bus

This command triggers a new initialisation sequence of the local bus

#### 6.3.2.6.1 Request

Send the following message (no data needed):

6	0	3	0	0	0	0
---	---	---	---	---	---	---

#### 6.3.2.6.2 Response

The Localmaster should respond with an echo of the transmitted request (no data needed)

6	0	3	0	0	0	0
---	---	---	---	---	---	---

### 6.3.2.7 Command “16” – update

#### 6.3.2.7.1 Request

To update the data (write, read) of the *Local-Master* and the *Slaves*, the following message has to be sent to the *Local-Master*.

Header [6 Bytes] (n+6)%256, (n+6)/256, 16, 0, 0, 0	Data [n Bytes] Output Data, see below
---	--

The output data consists of 3 parts:

Master Data Offset 2 Bytes (LSB MSB)	Slave Output Data i Bytes	Master Data Area 62 Bytes
---	------------------------------	------------------------------

i→dependant on the type and number of modules

See Product Documentation for a complete list of individual output data for all em-BRICK®-modules.

The master data offset is calculated from the start of the data area (master data offset = 2 + i). If the Slave output data area is 200 bytes wide, the master data offset would be 2 + 200 = 202.

#### Slave output data

All output bytes are transferred as raw data containing the values of the different outputs (DO, AO). The data is sorted by the sequence of the bricks. The data of each brick is sorted as defined for the brick (see product documentation).

Therefore, the slave output data contains

Brick 0 data bytes 0...n1 (see brick docu for number of bytes)

Brick 1 data bytes 0...n2 (see brick docu for number of bytes)

etc. until the number of bricks connected to the bus is reached.

### Master data area

All output bytes are transferred as raw data containing the values of the different outputs (DO, AO) of the *Local-Master*. The data of the local master is sorted as defined for the local master (see product documentation).

#### 6.3.2.7.2 Response

The response works similar to the request.

Header [6 Bytes] (n+6)%256, (n+6)/256, 16, 0, 0, 0	Data [n Bytes] Input Data, see below
---	---

The input data consists of 3 parts

Master Data Offset 2 Bytes (LSB MSB)	Slave Input Data i Bytes	Master Data Area 62 Bytes
---	-----------------------------	------------------------------

i→dependant on the type and number of modules

See Product Documentation for a complete list of individual output data for all em-BRICK®-modules.

The master data offset is calculated from the start of the data area (master data offset = 2 + i). If the Slave input data area is 200 bytes wide, the master data offset would be 2 + 200 = 202.

### Slave input data

For each Slave the first byte transferred is the status of this Slave. The status is followed by the input bytes as raw data containing the values of the different inputs (DI, AI, CNT). The data is sorted by the data of bricks. The data of each brick is sorted as defined for the brick (see product documentation).

If no valid input data is available, e.g. during brickBUS init sequence, the Slave input data field is empty and the Master data offset is 2.

### Master data area

All input bytes are transferred as raw data containing the values of the different inputs (DI, AI, CNT) of the *Local-Master*. The data of the local master is sorted as defined for the local master (see product documentation).

### 6.3.2.8 Command “254” – close request

This command is used to order the CouplingBrick to properly close the connection.

This command is in TCP/IP communication normally not required, since simply closing the communication from the master has exactly the same effect.

#### 6.3.2.8.1 Request

6	0	254	0	0	0
---	---	-----	---	---	---

#### 6.3.2.8.2 Response

The CouplingBrick will not respond like the previous commands. Instead it will respond with a FIN-message.

## 6.3.3 Data Transportation

### 6.3.3.1 TCP/IP via LAN/WLAN

By using TCP/IP, the communication blocks described above are transferred as described without additional information. The connection, routing, error detection/correction will be done automatically by the TCP/IP socket.

Each TCP/IP-message transfers one communication block.

Recommended communication process:

- Connect to the *local-master*. Request Configuration Data from *local-master*.
- Exchange Data with the *local-master* (every 5ms ... 50ms).
- Close the connection.

Typically, the Local-Master waits on Port 7086 for incoming connections.

For more information about connection, addressing and setup refer to the available hardware components (i.e. CAE\_Z-LWCS-M32-xx).

### 6.3.3.2 Serial via RS232/RS485 (planned)

#### 6.3.3.2.1 Basics

When using a RSxxx, the communication blocks of **Fehler! Verweisquelle konnte nicht gefunden werden.** will be used. Therefore, a RSxxx connection has no native routing or error detection/correction an additional protocol will be used as a transportation frame as described below.

Beside the data content, the physical parameters are:

- 9600/38400/115000 Baud, no parity, 8 data bits, 1 stop bit, half-duplex
- The gap between two bytes of one message have to be < 100 x Bit-Time. Otherwise a message will be dropped and the receiver expects a new message. This behaviour can also be used to "reset" the communication protocol machine after a communication error.

#### 6.3.3.2.2 Framing of Message to Local Master

Remote-Master (client) **sends to** Local Master(s) (server):

Purpose	Byte No.	Example / Note
source ID (actual not sup.)	0	actual don't care
drain ID (actual not sup.)	1	actual don't care
<data > (according <b>Fehler! Verweisquelle konnte nicht gefunden werden.</b> )	2 ... s	254
checksum	s+1 (LSB)	CS L (see below)
	s+2 (HSB)	CS H (see below)

If the data were received correctly by the local-master (CRC is ok), the local-master returns the checksum (2 bytes) to the remote-master. If the Remote-Master do not receive this checksum ter it is an indication for an error and the remote-master can react (i.e. wait the gap and send the



message again). If a reply from the local-master is necessary, its data are follow after the checksum by a reply message (see below).

### 6.3.3.2.3 Framing of Reply from Local Master

Local Master (server)      **answers to**      Remote Master (client):

Purpose	Byte No.	Example
<data > (according <b>Fehler! Verweis- quelle konnte nicht gefunden wer- den.</b> )	1 ... s	254
checksum	s+1 (LSB)	CS L (see below)
	s+2 (HSB)	CS H (see below)

Also, the reply message has to be checked with the checksum by the reomote-master about the corectness. If the remote-master detects a error, he i.e. drop the message and repeat the communication cycle. Calculation of the Checksum

### 6.3.3.3 Checksum

The caclutatioft he checksum corresponds with the standard of Modbus.

```

unsigned char *pMessage = ??????; // initialize pointing to message
short lenMessage= ???;           // initialize with len of message

unsigned short CRC = 0xFFFF;
short i,k;

for (i=0; i < lenMessage; i++, pMessage++)
{
    unsigned short x;
    unsigned short x2;
    x = (unsigned short)(*pMessage);
    CRC ^= x ;
    for (k=0; k < 8; k++)
    {
        x2 = CRC & 0x0001;
        CRC >>= 1;
        if (x2)
            CRC ^= 0xA001;
    }
}
// now "CRC" is the result

```

### 6.3.3.4 Serial via I2C

t.b.d.

## 6.3.4 Error detection and handling

### 6.3.4.1 Local Master

A local master (e.g. CAE\_Z-PatBridge#, CAE\_Z-UniBridge#,) has a "Remote Timeout" parameter. If the local master does not receive a data message wihtin that time from the remote master, it

sets the all output data of all connected IO-bricks to 0. This value value should be set to a value according your safety criterias (e.g. to 1 second).

### 6.3.4.2 Remote Master (General)

The description herein refers to the IMACS implementation of a remote master stack. It is recommended to follow this strategy if implementing your own remote master.

The remote master runs in its own thread using a state machine which is called periodically. It is recommended to call this state machine every 10 ms, a range of 2..50 ms should not be exceeded. Values mentioned hereafter must probably be modified if another rate then 10ms is choosen. In the following, every call of the state machine is referenced as "cycle".

The remote mater stack can service more than one local master (referenced as "node"). Every node is handelt separately, i.e. the behavior described in this chapter references always one node if not stated otherwise.

The remote master logs all messages in a logfile "brickbus.log". This logfile exists once for all nodes.

The remote master stack has an *error counter*, which is set to a predefined value e.g. when a request from the remote master to the local master is started and counted down every cycle.

### 6.3.4.3 Proceedings in remote master

First, the stack generates a welcome message indicating several version numbers.

Then the file eB\_LAN.cnf is parsed. Every encountered error generates a message in the logfile.

Then the LAN connection is established. The error counter is set to 500 cycles. If it reaches 0, a message „Connection to ... timed out.“ Is generated and a reconnect is started

Then the Init data is sent / requested. The error counter is set to 10 cycles. If it reaches 0, a message „Receiving init data from ... timed out.“ Is generated and a reconnect is started

Then the data is sent and requested. The error counter is set to 3 cycles on every transmission of a data send/request from the remote master If it reaches 0 before a response from the local master is received, a message „Receiving data from ... timed out." " is generated and the next transmission of data sent/request is started. Also after receiving a response (either correct or erroneous) the data is sent and requested as described above

All numbers are default values. They can be overwritten by the application program.

### 6.3.4.4 Warning data not received in time

On receiving data, the remote master stack expects the response to arrive within one cycle. Every cycle, data is not received a message "Warning on try " << numTry << ": data for node " << node.name << " not received in time." Is generated. The remote master stack continues to listen for incoming messages in the next cycle(s).

## 6.3.5 Protocol History

Versions of Remote-Bus-Protocal LAN/RSxxx

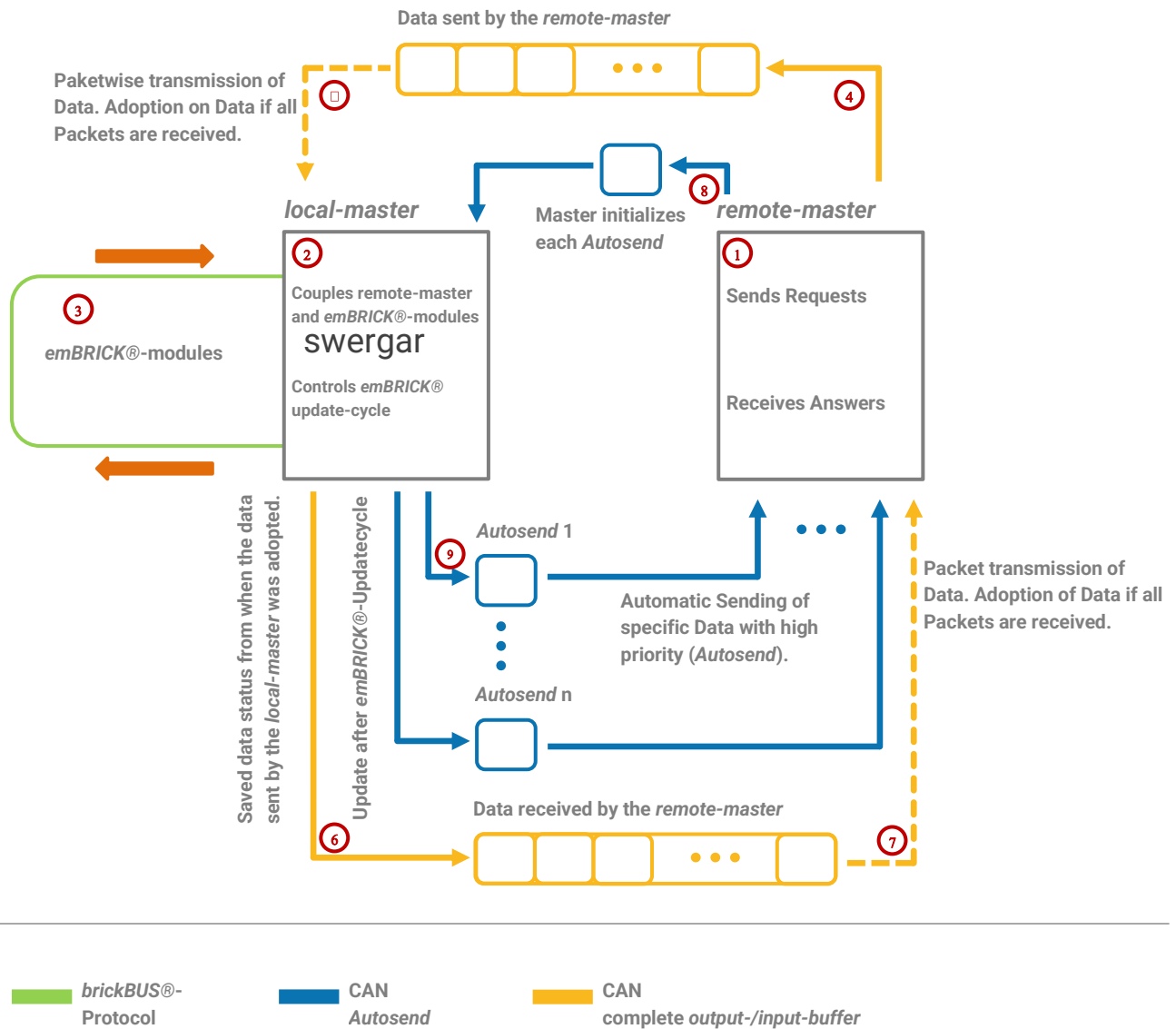
Version	availbale since	Modifications / Enhancements
2	12/2013	First released implementation

3	02/2014	Added 2 bytes to the header for standard data exchange.
4	08/2017	Further defintions in protocol

## 6.4 CAN based remote communication (planned)

### 6.4.1 Basics

The CAN-Connection is used to establish a communication from the *remote-master* (1) to the *local-master* (2). The *local-master* will act as a coupler, distributing the data onto the *emBRICK®*-modules (3).



## 6.4.2 Mode of Operation

Two methods of communication between the *local-master* and the *remote-master* are implemented, the *standard communication* and the *Autosend*.

The *standard communication* is used to update all *emBRICK®*-modules. To execute a standard communication, the *remote-master* has to send a buffer containing all data needed to completely describe the output status of all *emBRICK®*-modules (4). The buffer will most likely not fit into the 8 byte data area of a CAN-message. Thus it has to be split up into multiple packets (5). The first packet gets the *packet-index* 0. The *packet-index* will increase consecutively with each packet sent. The last packet gets the *packet-index* 255. Once the *local-master* receives a packet with the *packet-index* 255, it will distribute the received data to the *emBRICK®* -modules with the next update-cycle and send the last known input status of all *emBRICK®* -modules back to the *remote-master* (6) in the same way (7).

Sending and receiving the complete buffer is often unnecessary. Also, the *emBRICK®* -modules will get updated faster than the CAN-Communication is able to exchange the data with the *remote-master*, effectively limiting the whole system. To read some inputs more often than others, a second method of communication has been implemented:

The *Autosend* has to be set up once (9) by the *remote-master* and runs independent on the *local-master* (10). It will periodically send 6 bytes or 4 words of the *input-buffer*. The offsets of these bytes or words can be defined in the setup. To setup an *Autosend*, an *autosend-initialization* packet has to be sent. *Autosends* will get priority over the *standard-communication*.

## 6.4.3 Description of Operation

The Buffer for the *standard communication* will be split into multiple CAN-Messages if necessary. The *packet-index* for the first message is ,0'. Each following packet has its Packet-Index increased by 1. The last Packet has the *packet-index* '255'.

After each *brickBUS®*-Cycle the CAN-Slave checks if a message has been completely received. In this case, the message will be written to the *brickBUS®-output-buffer*. Simultaneously, the *brickBUS®-input-buffer* will be copied into a CAN-Buffer. This Buffer is then sent back to the *remote-master*. This ensures integrity of the Data while it is transmitted via CAN.

The *local-master* can send specific areas of the *brickBUS®-input-buffer* automatically. This is called *Autosend*. The *local-master* supports up to 32 different *Autosends*. Each of those *Autosends* can either transmit 6 Bytes or 4 Words. Every *Autosend* has to be set up with its Repetition Rate, *Autosend-ID* and Byte-/Word-Offsets in the *brickBUS®-input-buffer*. The fastest repetition time is 1ms. However, it is not recommended to set the *Autosend* Repetition Time to be lower than the *brickBUS®* Cycle Time. The recommended minimum *brickBUS®* Cycle Time is depending on the number and type of connected Slave-Modules. The standard setting for the *brickBUS®* Cycle Time is 10ms.

*Autosends* will get priority before the Standard Communication.

When multiple *Autosends* are scheduled for the same timeframe, the *Autosend* with the earliest scheduled transmission date will be sent first. When multiple *Autosends* share the same scheduled transmission date, the *Autosend* that has been set up first will be sent first.

Standard-Bitrate is 500 kBit/s.

### 6.4.3.1 Composition of the 29 Bit Arbitration Field (CAN2.0B)

The Arbitration Field is used to identify different Packets. The Arbitration Field is structured as follows:

28 - 21	20	19 - 16	15 - 8	7 - 0
AAAA AAAA	M	CCCC	PPPP PPPP	BBBB BBBB

B → fix 8-Bit Base-ID, can be changed by the customer (Standard = 1)

P → Package-Index 0...255 (consecutive number for linear data exchange)

C → Command-ID 0...15

M → 1 = Master sends, 0 = Slave sends

A → Node-Address 0...255 (in case of multiple CAN-Slaves)

### 6.4.3.2 Commands

Command	Function
<b>0x2</b> (must consist of 2 Packages)	Exchange the complete <i>emBRICK®</i> -output/input-buffer; Packet-Index 0xFF signals end of transmission
<b>0x3</b>	Configures how fast the <i>emBRICK®</i> -Master sends out CAN-Messages [0...0xFFFF]; '0' = as fast as possible, other Values = Time in ms between CAN-messages
<b>0x4</b>	Set up <i>brickBUS®</i> Cycle Time [0...0xFFFF] in ms; Standard = 10. Value ,0' stops the BUS.
<b>0x7</b> (LEN=0)	Clear all Autosends
<b>0x8</b> (LEN=6)	Setup a Word Autosend (1x) Byte 0 = Autosend-Frame ID 0...31 Byte 1 = Autosend-Rep-Rate (1...255 ms) Byte 2 = offset of Autosend-Word 1 (of input-Buffer) Byte 3 = offset of Autosend-Word 2 Byte 4 = offset of Autosend-Word 3 Byte 5 = offset of Autosend-Word 4
<b>0x9</b> (LEN=8)	Setup a Byte Autosend (1x) Byte 0 = Autosend-Frame ID 0...31 Byte 1 = Autosend-Rep-Rate (1...255ms) Byte 2 = offset of Autosend-Byte 1 (of input-Buffer) Byte 3 = offset of Autosend-Byte 2 Byte 4 = offset of Autosend-Byte 3 Byte 5 = offset of Autosend-Byte 4 Byte 6 = offset of Autosend-Byte 5 Byte 7 = offset of Autosend-Byte 6
<b>0xA</b> (LEN=0)	Initialize <i>emBRICK®</i> -String and receive configuration Data
<b>0xB</b> (LEN=0)	Send and receive command Data
<b>0xF</b> (LEN=1)	Set base-ID Byte 0 = fix 8-Bit base-ID

### 6.4.3.3 Command Data Field

The Command Data Field is a 8 Byte Data Field that contains all available commands that can be sent to the Master. Upon receiving the Command Data Field, the *local master* will send back the Status Data field which contains various Data about the current State of the brickBUS and itself.

#### 6.4.3.3.1 Description of Command Data Field (remote master → local master)

Byte	Function
0	<i>Bit 0:</i> clear EEPROM Error <i>Bit 1:</i> restart brickBUS <i>Bit 2:</i> stop brickBUS
1	-
2	-
3	Set expected number of <i>emBRICK®</i> -modules in the <i>string</i> ; The <i>local master</i> will continue to initialize the brickBUS until the expected number of <i>emBRICK®</i> -modules are found; deactivate this feature by sending '0' on this Byte. (0...31)
4	-
5	-
6	-
7	-

#### 6.4.3.3.2 Description of Status Data Field (local-master → remote master)

Byte	Function
0	<i>Bit 0:</i> atleast one checksum-error occurred in the last message <i>Bit 1:</i> EEPROM could not be read; if set, this must be cleared before sending Data
1	Number of <i>emBRICK®</i> modules found in the <i>string</i> . (0...31)
2	-
3	-
4	-
5	-
6	-
7	-

### 6.4.3.4 Configuration data *local-master* → *remote-master*

1x *emBRICK®*-Master

Offset	Purpose
0	size
1	status
2	master_id – high Byte
3	master_id – low Byte
4	local-/remote-master protocol version
5	local-master software version
6	manufacturer ID
7	peripheral_id high byte
8	peripheral_id low Byte

Slaves (n = Number of Slave)

Offset	Purpose
(9*(n+1))+0	status



$(9*(n+1))+1$	data length output data
$(9*(n+1))+2$	data length input data
$(9*(n+1))+3$	protocol version
$(9*(n+1))+4$	hardware version
$(9*(n+1))+5$	deviceID – high byte
$(9*(n+1))+6$	deviceID – low byte
$(9*(n+1))+7$	manufacturer ID – high byte
$(9*(n+1))+8$	manufacturer ID – low byte
$(9*(n+1))+9$	offset of output-buffer
$(9*(n+1))+10$	offset of input-buffer

Use the offset of the *input/output-buffer* to extract/insert IO Data for each Slave out of the received/transmitted data buffer.

## 6.4.4 Example of a Configuration Message

### 6.4.4.1 Data received by Master

Time	Chn ID	Name	Dir	DLC Data
45.701062	1	1A0001x	Tx	0
45.701914	1	A0001x	Rx	8 0B 00 06 42 01 12 01 00
45.702442	1	A0101x	Rx	8 00 00 01 01 0B 01 08 FE
45.702974	1	A0201x	Rx	8 00 01 00 00 01 01 01 0B
45.703505	1	A0301x	Rx	8 01 08 FE 00 01 01 01 01
45.704041	1	A0401x	Rx	8 01 01 0B 0F 08 FD 00 01
45.704567	1	A0501x	Rx	8 02 02 00 01 01 0B 0F 08
45.705107	1	A0601x	Rx	8 FD 00 01 03 03 00 01 01
45.705558	1	A0701x	Rx	8 0B 0F 08 FD 00 01 04 04
45.706018	1	A0801x	Rx	8 00 01 01 0B 0F 08 FD 00
45.706600	1	A0901x	Rx	8 01 05 05 00 11 12 0B 03
45.706890	1	A0A01x	Rx	8 09 9D 00 01 06 06 00 11
45.707175	1	A0B01x	Rx	8 12 0B 03 09 9D 00 01 17
45.707459	1	A0C01x	Rx	8 18 00 01 01 0B 0F 08 FD
45.707747	1	A0D01x	Rx	8 00 01 28 2A 00 01 01 0B
45.708031	1	A0E01x	Rx	8 0F 08 FD 00 01 29 2B 01
45.708319	1	A0F01x	Rx	8 01 01 0B 0F 08 FD 00 01
45.708498	1	AFF01x	Rx	2 2A 2C

### 6.4.4.2 Data formatted into Blocks for Master and Slaves:

Configuration Data of the emBRICK®-Master	0B 00 06 42 01 12 01 00 00
Configuration Data of emBRICK®-Module 1	00 01 01 0B 01 08 FE 00 01 00 00
Configuration Data of emBRICK®-Module 2	01 01 01 0B 01 08 FE 00 01 01 01
Configuration Data of emBRICK®-Module 3	01 01 01 0B 0F 08 FD 00 01 02 02
Configuration Data of emBRICK®-Module 4	00 01 01 0B 0F 08 FD 00 01 03 03
Configuration Data of emBRICK®-Module 5	00 01 01 0B 0F 08 FD 00 01 04 04
Configuration Data of emBRICK®-Module 6	00 01 01 0B 0F 08 FD 00 01 05 05
Configuration Data of emBRICK®-Module 7	00 11 12 0B 03 09 9D 00 01 06 06
Configuration Data of emBRICK®-Module 8	00 11 12 0B 03 09 9D 00 01 17 18
Configuration Data of emBRICK®-Module 9	00 01 01 0B 0F 08 FD 00 01 28 2A
Configuration Data of emBRICK®-Module 10	00 01 01 0B 0F 08 FD 00 01 29 2B
Configuration Data of emBRICK®-Module 11	01 01 01 0B 0F 08 FD 00 01 2A 2C

## 6.4.5 Examples of a Data Message

### 6.4.5.1 Data received by Master

Time	Chn ID	Name	Dir	DLC Data
5.688078	1	20001x	Rx	8 01 01 01 01 01 01 00
5.687784	1	20101x	Rx	8 0D 00 0D 00 0E 00 0E 00
5.687548	1	20201x	Rx	8 0D 00 00 00 00 00 00 07
5.687298	1	20301x	Rx	8 01 00 0D 00 0D 00 0E 00
5.687050	1	20401x	Rx	8 0D 00 0D 00 00 00 00 00
5.686808	1	2FF01x	Rx	5 00 C7 01 01 01

#### 6.4.5.1.1 Data formatted for each Slave:

Slave-Module	offset of <i>input-buffer</i>	data length <i>input-data</i>	Data
1	0x0	0x01	0x01
2	0x1	0x01	0x01
3	0x2	0x01	0x01
4	0x3	0x01	0x01
5	0x4	0x01	0x01
6	0x5	0x01	0x01
7	0x6	0x12	0x01 0x00 0x0D 0x00 0x0D 0x00 0x0E 0x00 0x0E 0x00 0x0D 0x00 0x00 0x00 0x00 0x00 0x00 0x07
8	0x18	0x012	0x01 0x00 0x0D 0x00 0x0D 0x00 0x0E 0x00 0x0D 0x00 0x0D 0x00 0x00 0x00 0x00 0x00 0x00 0xC7
9	0x2A	0x01	0x01
10	0x2B	0x01	0x01
11	0x2C	0x01	0x01

Offset of *input-buffer* and data length input data are being taken from the Configuration message.

#### 6.4.5.1.2 Structure of Slave Data received by Master

On the first byte of every data field, a Status-Byte will be transmitted. The Status-Byte can be used to identify problems with the BUS-communication. The Status-Byte is followed by the data area of the Slave as described in the Product Overview. It is advised to display the Status-Byte somewhere in the application. If the Status-Byte has the value '1' everything is as expected. If the status-byte frequently switches to other values, check your environment for interference fields.

Byte 0	Byte 1...n (if available)
Status-Byte	Data

### 6.4.5.2 Data sent to Slave

Sending Data to the *local-master* (writing to the emBRICK-*output-buffer*) works similarly with the difference in being  $M = 1$ , use of offset of *output-buffer* instead of offset of *input-buffer* and use of data length output data instead of data length input data.

Time	Chn ID	Name	Dir	DLC Data
5.688078	1	120001x	Rx	8 03 00 07 D0 0B 0F 0F 11
5.687784	1	120101x	Rx	8 02 01 06 A8 0A BC 0E 10
5.687548	1	120201x	Rx	8 21 33 44 45 66 34 88 69
5.687298	1	120301x	Rx	8 3A 3B CC 37 02 01 00 01
5.687050	1	120401x	Rx	8 03 02 0F 05 06 D0 80 09
5.686808	1	12FF01x	Rx	5 0A 02 0C

#### 6.4.5.2.1 Data formatted for each Slave

Slave-Module	offset of <i>output-buffer</i>	data length out-put data	Data
1	0x0	0x01	0x03
2	0x1	0x01	0x00
3	0x2	0x01	0x07
4	0x3	0x01	0xD0
5	0x4	0x01	0x0B
6	0x5	0x01	0x0F
7	0x6	0x11	0x0F 0x11 0x02 0x01 0x06 0xA8 0x0A 0xBC 0x0E 0x10 0x21 0x33 0x44 0x45 0x66 0x34 0x88
8	0x17	0x011	0x69 0x3A 0x3B 0xCC 0x37 0x02 0x01 0x00 0x01 0x03 0x02 0x0F 0x05 0x06 0xD0 0x80 0x09
9	0x28	0x01	0x0A
10	0x29	0x01	0x02
11	0x2A	0x01	0x0C

Offset of *output-buffer* and data length output data are being taken from the Configuration message.

#### 6.4.5.2.2 Structure of Slave Data sent by Master

The data-field of each Module only consists of the actual Data as described in the Product Documentation.

## 6.4.6 Examples of a Command/Status Message

### 6.4.6.1 Data sent by remote master

This example will reset the brickBUS

ID	Length	Data [hex]
0x001B0001	8	02 00 00 00 00 00 00 00

Data received by remote master

ID	Length	Data [hex]
0x000B0001	8	00 04 00 00 00 00 00 00

See 6.4.3.3.1 and 6.4.3.3.2.

## 6.4.7 Error detection and handling

t.b.d.

## 6.4.8 Protocol History

CAN Protocol Version	Modifications
1	First released implementation

## 6.5 ModBUS Large Block

In this case, the coupling bridge (local master) acts as ModBUS slave while the remote master acts as ModBUS master.

### 6.5.1 Data representation

Only the data types “Input Registers” (for input data and init data) and “Holding Registers” (for output data) are used.

CAUTION: The Data field must not exceed 250 bytes (due to ModBUS restrictions).

The byte stream of the data is mapped to the 16bit registers without reformatting to network byte order.

#### 6.5.1.1 Data Mapping Input Registers (Slave input Data)

The Slave input data is in the address range 0000h ... 007Ch. It consecutively lists all slave input data. The number of bytes of the slave input data is depending on the type and number of modules, see Product Documentation for a complete list of individual input data for all emBRICK®-modules.

The byte stream of the input data is mapped to the 16bit input registers without reformatting to network byte order.

For each Slave the first byte transferred is the status of this Slave. The status is followed by the input bytes as raw data containing the values of the different inputs (DI, AI, CNT). The data is sorted by the data of bricks. The data of each brick is sorted as defined for the brick (see product documentation).

If no valid input data is available, e.g. during brickBUS init sequence, all values are set to 0.

#### 6.5.1.2 Data Mapping Input Registers (Master input Data)

The Master input data is in the address range 0100h .. 011Eh. It consecutively lists 62 bytes master input data. All input bytes are transferred as raw data containing the values of the different inputs (DI, AI, CNT) of the *Local-Master*. The data of the local master is sorted as defined for the local master (see product documentation).

#### 6.5.1.3 Data Mapping Input Registers (Init Data)

The init data is in the address range 1000h ... 107Ch. It contains init data of the local master followed by n blocks of init data of all connected (and initialized) bricks (I/O-modules).

Most figures in the response are only usable if the status of the local master is “1”, i.e. if the Master is operating. Therefore, the response should be discarded if status of local master is “0”.

1<sup>st</sup> local-master data

Purpose	Byte no.	Example / Note
number of connected <i>bricks</i>	0 [1... 32]	01h (1 module found)
status of the <i>local-master</i>	1 [0...1]	01h (operating) / 00h (not operating)
component ID of the <i>local-master</i> (here 1-603 = 0643h)	2 (MSB)	06h
	3 (LSB)	43h
remote bus protocol version	4	01h (see below “Version History” for details)
software version of <i>local-master</i>	5 [1...255]	14h

Local Master manufacturer ID	6 [1...255]	01h (1=IMACS)
Reserved	7 (MSB)	00h (currently fix)
Reserved	8 (MSB)	00h (currently fix)

2<sup>nd</sup> followed by n blocks of the configuration of all connected (and initialized) bricks (I/O-modules). n is equal to the *number of connected bricks* received in the 1<sup>st</sup> part

Here:        b = is the number of the corresponding brick[0..30]  
               the example data are of the module *P-2Rel4Di2Ai-01*

Purpose	Byte no.	Example / Note
status of slave-module	$9+(b*11) + 0$	01h
Data Length MOSI (local Master -> Slave) [bytes]	$9+(b*11) + 1$	01h
Data length MISO (Slave -> Local Master) [bytes]	$9+(b*11) + 2$	06h (here: 1 + 2 + 2 + 1)
Slave local bus protocol version	$9+(b*11) + 3$	0Bh
Slave hardware revision	$9+(b*11) + 4$	12h
Slave device ID (here P-2Rel4Di2Ai-01: 5-131 = 140Bh)	$9+(b*11) + 5$ (MSB)	14h
	$9+(b*11) + 6$ (LSB)	0Bh
Slave manufacturer ID	$9+(b*11) + 7$ (MSB)	01h (1=IMACS)
	$9+(b*11) + 8$ (LSB)	
Data offset MOSI (local Master -> Slave)	$9+(b*11) + 9$	00h rising according used modules
Data offset MISO (Slave -> Local Master)	$9+(b*11) + 10$	00h rising according used modules

#### 6.5.1.4 Data Mapping Holding Registers (Slave output Data)

The slave output data is in the holding registers address range 0000h .. 007Ch. It consecutively lists all slave output data. The number of bytes of the slave output data is depending on the type and number of modules, see Product Documentation for a complete list of individual output data for all emBRICK®-modules.

All output bytes are transferred as raw data containing the values of the different outputs (DO, AO). The data is sorted by the sequence of the bricks. The data of each brick is sorted as defined for the brick (see product documentation).

Therefore, the slave output data contains

Brick 0 data bytes 0...n1 (see brick docu for number of bytes), followed by  
 Brick 1 data bytes 0...n2 (see brick docu for number of bytes) followed by  
 Brick 2 data etc. until the number of bricks connected to the bus is reached.

#### 6.5.1.5 Data Mapping Holding Registers (Master output Data)

The output data is in the holding registers address range 0100h ... 011Eh. It lists 62 bytes master output data.

### 6.5.2 Supported commands

Only the commands "Read Input Registers" (0x04), "Write multiple Registers" (0x10) and "Read/Write multiple Registers" (0x17) are supported.

### 6.5.2.1 Read Input Registers (0x04)

The number of bytes returned is always the double of the number requested in the bytes 3 and 4 of the command ("Quantity"). If this number is higher than the length of the actual input (MISO) data of the bricks string, the remaining bytes are padded with zeroes. If it is shorter, the bytes exceeding the requested size of the request are skipped.

### 6.5.2.2 Write multiple Registers (0x10)

If the number requested in the bytes 3 and 4 of the command ("Quantity") is less than half the number of bytes in the actual output (MOSI), only the first bytes in the MOSI buffer are updated. If the number exceeds the actual output length, remaining bytes are ignored. The command returns always the number in bytes 3 and 4 in the response bytes 3 and 4.

### 6.5.2.3 Read/Write multiple Registers (0x17)

If the number requested in the bytes 7 and 8 of the command ("Quantity to Write") is less than half the number of bytes in the actual output (MOSI), only the first bytes in the MOSI buffer are updated. If the number exceeds the actual output length, remaining bytes are ignored.

The number of bytes returned is always the double of the number requested in the bytes 3 and 4 of the command ("Quantity to Read"). If this number is higher than the length of the actual input (MISO) data of the bricks string, the remaining bytes are padded with zeroes. If it is shorter, the bytes exceeding the requested size of the request are skipped.

## 6.5.3 Restrictions

The following restrictions apply to the different header elements of a command:

Byte	Meaning	Restrictions
1	Function Code	Only 0x04, 0x10 allowed
2,3	Starting address	Only 0x0000, 0x0100, 0x1000 allowed, see above
4,5	Quantity	Must not exceed the size of the fields as stated above

## 6.5.4 Data Transportation

### 6.5.4.1 LAN (ModBUS TCP)

The Ethernet address is to be set separately, see "Switching IP Address..." above.

As standard, Port 501 is used.

The Transportation Layer specifications of Modbus\_Application\_Protocol\_V1\_1b are met.

### 6.5.4.2 RS485 (ModBUS RTU)

The RS485 serial port is operating at 1Mbaud (other baudrates are available upon request), 8bit data width, no parity, 1 stop bit.

The Transportation Layer specifications of Modbus\_Application\_Protocol\_V1\_1b are met.

## 6.5.5 Error detection and handling

### 6.5.5.1 ModBUS TCP

The ModBUS Timeout is the time within a message must be completed. Typically, this is approx. 0.01 .. 0.05 seconds for ModBUS TCP.

The remote master should have a timeout for receiving a response. In case the response is not received within that time, the message should be repeated. After several tries, it is recommended to re-establish the connection. This typical timeout should typically be approx. 0.1 seconds. Please note that the Ethernet line automatically repeats a message if the message was not transmitted correctly. The timeout should be set large enough to allow this Ethernet-implied repeats to take place.

Connection timeout beim Aufbau der Socket Verbindung

### 6.5.5.2 ModBUS RTU

If the ModBUS message is not completed within the time set as "ModBUS Timeout", the message is discarded and the receiver waits for the beginning of the next message. A reasonable value for the ModBUS Timeout is approx. 0.01 .. 0.05 seconds for high baudrates (>100 000 baud) and approx. 0.15 seconds for 38400 baud.

Verhalten Koppel-Master (als ModBUS Slave):

Timeout**Ms**Rec typ. 500 Bytelängen (z.B. 5ms bei 1MBaud) wenn der Empfang begonnen hat, und seit dem letzten Byte für dieses Dauer keine vollkorrekte Nachricht erkannt wurde, wird der Modbus-Empfangs und Sendebuffer gelöscht

Timeout**B**Shutdown typ. 100ms ... 1s wenn für diese Dauer keine neuen gültigen Nachricht vom Master kommen, werden die BrickBUS-Out-Daten alle auf 0 (oder einen geteachten Notzustand) gestellt

Verhalten des externen Modbus-Masters:

Timeout**Mb**MAnswer typ. 10ms + 500 Bytelängen (z.B. bei 1MBaud somit ca. 15ms) wenn nach dem Versenden einer Message vom Master an den Slave die Antwort vom Slave innerhalb dieser Zeit nicht vollständig eingeht, werden alle seriellen Buffer gelöscht und der Master geht in den Zustand "Waiting after Timeout"

Verhalten des Steuerungssystems (Remote-Master), das den ModBUS Master verwendet:

Timeout**RM**Answer typ. 100 ms wenn in dieser Zeit vom Kommunikationskanal (emBRICK-remoteBus via LAN, Modbus-TCP, Modbus-RTU, CAN) auf einen Buszyklus keine korrekte Antwort vom Remote-Slave bzw. Modbus-Master vorliegt wird ein RMCycleAnswerError und ein RMRepearCounter hochgezählt. Beide Zähler werden auch bei anderen Empfangsfehler hochgezählt. Nach einer korrekten Kommunikation wird der RMRepearCounter gelöscht.

MaxRep**RM**ReInittyp. 5 wenn der RMRepearCounter diesen Wert überschreitet erfolgt auf dem Remote-Master ein Shutdown (d.h. die der Applikation vom



Brickbus-EA weitergeleiteten Daten werden auf 0 gesetzt) und es wird versucht einen neue Kommunikation aufzubauen

#### 6.5.5.2.1 Störungs-Szenarien zum Testen

(bei 1MBaud und 80ms Modbus-Zykluszeit)

alle 2s für 10ms      => nur Fehler erkennen, kein Shut-Down

alle 250ms für 1ms    => nur Fehler erkennen, kein Shut-Down

alle 10ms für 100µs   => Shut-Down permanent solange Störung anliegt

aus-/einstecken      => Shutdown nach 1s ausgesteckt

#### 6.5.5.2.2 Infos über den radCASE Modbus-RTU Master (selbst)

siehe: ..\rc\_lib\LIB\ModBUS.rad: ModulDef.MODUL:MModbusMaster <<

#### 6.5.5.2.3 ModbusMasterState:

- Init,
- Waiting for Request
- Busy
- Waiting (vorher Receive Timeout    Param: Modbus.Timeout)
- Waiting (vorher Error)
- Waiting (vorher flasche Daten)]
- 

Parameter:

Timeout einer Answer,

Derzeitiger Zeiten/Zyklus bei 1MBaud

1ms	OutDaten an Slave
3ms	Anzwortsverz. von Slave
<1ms	Empfangsbestätigung von Slave
27ms	Pause seitens Master
<1ms	In-Daten von Slave anfordern
3ms	Anzwortsverz. von Slave
<1ms	Indaten werden von Slave an Master gesendet
50ms	Pause seitens Master
	weiter mit oben

#### 6.5.5.2.4 Infos über RemoteMaster (der den ModBUS-RTU Master verwendet)

beim Init wird 1000ms gewartet bei 38400 danach kommt sofort ein Reconnect  
Zähler für Retry und Zähler für Reconnect

RemoteMaster-State: [TryInit, Connected] bei 38400 / 1MBaud  
falls (1000/ xxxxx ms) keine/falsche Antwort => TryInit  
die erste korrekten Nutzdaten wurden empfangen => Connected

Sowohl der ModBUS als auch der Master laufen in einer PERM-Schleife mit einer festen Wiederholzeit.

Beim LWCS betraegt die Wiederholzeit 2 ms (fest).

Beim Remote Master sind die Wiederholzeiten – getrennt fuer ModBUS und Remote Master – als Parameter einstellbar (nur mit neuestem API.WinPC und API.BBB oder dem von H. Leitner gebauten Spezial-API des DIA5500-II). Diese Werte stehen momentan noch auf 10ms, hier sind sicherlich 1..2 ms sinnvoll – wenn der Rest-Applikation dann noch genuegend Rechenzeit bleibt.

The remote master should have a timeout for receiving a response. In case the response is not completely received within that time, the message should be repeated. After several tries, it is recommended to re-init the connection, i.e. to repeat the request for the Init data. The typical timeout to receive a response depends on the transmission speed, it can be roughly calculated with 2 times the transmission time (see above) + approx. 0.03 seconds for the internal calculation of the local master.

#### 6.5.5.3 ModBUS (all versions)

Typically, the following errors may happen:

- Erroneous request to or response from local master
- Exception message
- Internal errors

The local master discards erroneous messages and waits for the next incoming request. If applicable, it responds with a exception message as defined by the ModBUS standard.

Exception messages from the master are simply ignored but reported on the visu.

The local master (e.g. Patbridge) has a "Remote Timeout". If the local master does not receive a data message within that time, it sets all connected Outputs to 0. Typically, its value should be approx. 2 seconds.

The remote master shall repeat the request in case it receives an erroneous response (i.e. wrong checksum). Incoming exception messages should be notified to the user.

### 6.5.6 Protocol History

Version	Modifications
1	First released implementation

## 6.6 Modbus Nativ (further feature)

In this case, the coupling bridge (local master) acts as ModBUS slave while the remote master acts as ModBUS master.

## 7. Local Bus Access Programming

### 7.1 Using C/C++ via Raspberry Pi (Raspian, Geany)

refer to starterkit manual

### 7.2 Using C/C++ via BeagleBoneBlack (Angstrom)

coming soon

## 7.3 Using Model-based/C/C++ via radCASE

Platforms: ..... Embedded native, Embedded OS, Linux, Windows  
 Language Platforms: ..... Embedded native, Embedded OS, Linux, Windows  
 OE: ..... Embedded native, Embedded OS, Linux, Windows

### 7.3.1 radCASE-Assign-Strings

#### 7.3.1.1 Basic Structure

Corresponding to the *Reference\_Manual*, the following assign-strings exist:

CTR <Port>, ..., <HardwareID> (for DI, DO)  
 CTR <Index>, ..., <HardwareID> (for AO, CNT)  
 CTR <Index>, ..., <Mode>, ..., <HardwareID> (for AI)

#### 7.3.1.2 Port / Index

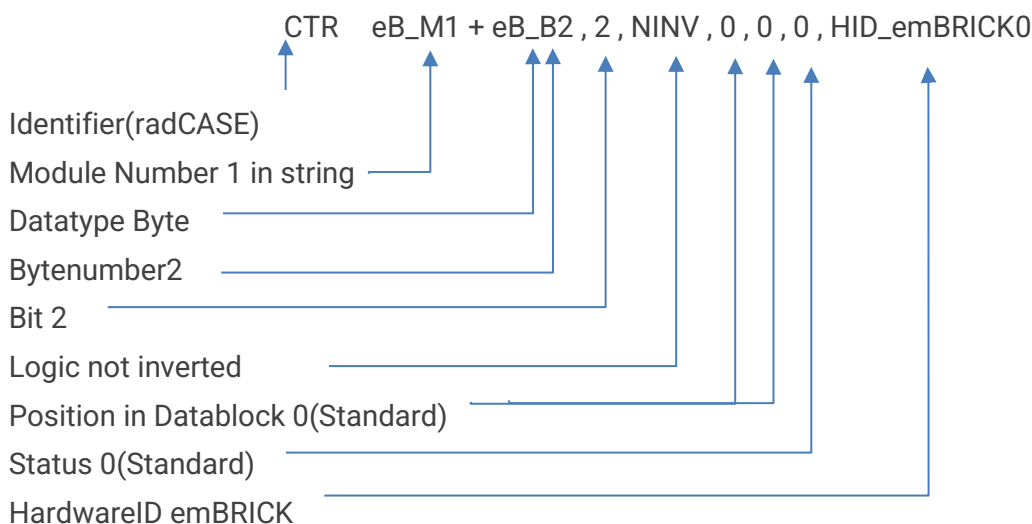
In the case of *emBRICK®*, the Port/Index is composed as follows: **eB\_M<x> + eB\_<T><y>**  
 where

x = module number in the string[1..32] (255 für Dummy)  
 T = coverage of the accessed data (B=byte, W=word, L=long)  
 y = consecutive number of the data in the module (s. *Product\_Catalogue*) [0...15]

Examples:

CTR **eB\_M1 + eB\_B2**, ..., HID\_emBRICK0  
 → Access to Node 0, module 1, byte 2  
 CTR **eB\_M2 + eB\_W0**, ..., HID\_emBRICK0  
 → Access to Node 0, module 2, word 0  
 CTR **eB\_M1 + eB\_B8**, ..., HID\_emBRICK1  
 → Access to Node 1, module 1, byte 8

Example of an assign-string to control a digital input(I2):



#### 7.3.1.3 Application for non *emBRICK®* I/Os

In systems with I/Os not realized via *emBRICK®* but also with radCASE, port and index can be freely chosen. It is recommended to number ports according to the port-number on the hardware, according to the Index as defined in the hardware.

For example:

P1 (on uP) => Port 1

P5 (on uP) => Port 5

Analog Input 1 => Index 1

PWM3 => Index 3

## 7.3.2 Setting Outputs

Since the dynamic configuration of the *brickBUS®* takes some time, the application developer needs to ensure that no outputs are set as long as the *brickBUS®* is not completely initialized.

## 7.3.3 Flowmeter

For the assign string of the flow meter on the *brickBUS®* is defined, that both E\_AI as well as E\_CNT have to get the sum value of the address from the structure. The sum value is the start address of the two input words.

It is necessary to enter both assign strings as the program in the *rc\_lib* is designed in a way that, if applicable, modules with only E\_CNT or only E\_AI can be defined.

## 7.4 Using IEC61131 via CODESYS

for Local-Bus-Access: under development

## 7.5 Using Middldeware based via Gamma

under development



## 8. Remote Bus Access Programming

### 8.1 Using native C/C++ Programming (i.e. via MSVC)

The most easy and recommended way is to use the coupling master with a C/C++ program (i.e. via PC/IPC and Windows, using MSVC).

A sample project is available under

[http://embrick.de/downloads/remotemaster/windows/Z-CouplingBrick\\_MSVC\\_V0.04.zip](http://embrick.de/downloads/remotemaster/windows/Z-CouplingBrick_MSVC_V0.04.zip)

[http://embrick.de/downloads/remotemaster/\\_PD\\_CouplingBrick%20Starterkit.pdf](http://embrick.de/downloads/remotemaster/_PD_CouplingBrick%20Starterkit.pdf)

### 8.2 Model-based/C/C++ with radCASE (IMACS GmbH)

The model-based developing suite **radCASE** offers an enhanced UML-Modeling framework and efficient code-generator for all embedded C/C++ platforms.

In this case radCASE generates an *emBRICK RemoteMaster* running as an application on a Windows- or Linux PC as a target (like a Soft-PLC).

For more information about radCASE see [www.radcase.de](http://www.radcase.de).

#### 8.2.1 Basics

The application will be generated as an autarkic Windows application radCASE-Application.exe

#### 8.2.2 Handling

##### 8.2.2.1 Configuration-File (eB\_LAN.cfg)

This file describes the *local-masters* and their connected *emBRICK@*-modules. A *remote-master* with radCASE can connect to several *local-masters*. Each *local-master* has to be assigned a name and his IP-adress. Additionally, *emBRICK@*-modules that are expected to be connected to the specific *local-master* can be described in the same line. Differences in connected *emBRICK@*-modules to expected *emBRICK@*-modules will throw an error in the log-file but otherwise have no effect in the current implementation.

The file has to be put into ...\\OSDL\\eB\_LAN.cnf.

Comments can be made by writing a '\*' in the very first column of a line.

Example of a Configuration-file with 2 strings (*local-masters*) :

```
* Node, IP-Adr,      ID-Coupler,  ID-Modules
1: HW_1, 192.168.3.10, 1601,      2181, 2181, 2181, 2301, 4401
2: HW_2, 127.168.3.11, 1601,      7100, 7100, 7100, 7010, 7001
```

For using the remote master from radCASE the HAL has to support it (refer to Integration manual). Also, the feature has to be activated using the Define RD\_EB\_REMOTEHOST.

To use the IOs of a local master (node) the Assign string has to be set accordingly. The node is selected by providing a hardwareID of HID\_EBR\_NODE1 – HID\_EBR\_NODE32, where the number

must match the ID in the config file (refer to Configuration-File). The index has to match the index of the IO on the local master.

Additionally to the access of the IOs there is an additional virtual module available for each node identified by the according hardwareID, granting additional information.

The following counters (CNT) are available:

eB_MNodeControl + eB_NC_FIRSTWARNING	
eB_MNodeControl + eB_NC_FURTHERWARNING	
eB_MNodeControl + eB_NC_RECONNECTS	
eB_MNodeControl + eB_NC_RM_VERSION	
eB_MNodeControl + eB_NC_NC_VERSION	
eB_MNodeControl + eB_NC_BUSSTATUS	
eB_MNodeControl + eB_NC_ERRORMODULE	

### 8.2.2.2 Log-File

The log-file logs all detected *local-masters* and their connected *emBRICK®*-modules. It also compares the connected *emBRICK®*-modules with the expected ones, as described in *eB\_LAN.cnf*. It also logs permanently the connection establishments and breakdowns or communication errors and retries with their timestamps.

This log-file can be found under ...\\CTR\\API.WinPC\\binary\\Release\\brickBUS.log.

Example of a log-file with one *local-master*.

```
2017-12-25 21:50:46.311 -> Remote-Master, Version 29
on PC Target
Revisionnumbers:
API_SVN: 3799
radCASE_SVN: 10002
2017-12-25 21:50:46.319 -> Found Master HW_1 with IP-Address 192.168.3.10
2017-12-25 21:50:46.331 -> Starting connection to emBRICK node HW_1 on IP-Address 192.168.3.10
2017-12-25 21:50:46.341 -> Established connection to HW_1 on IP-Address 192.168.3.10
2017-12-25 21:50:46.341 -> Requesting initialization data from HW_1

2017-12-25 21:50:46.851 -> Connection closed for HW_1

2017-12-25 21:50:51.849 -> Starting connection to emBRICK node HW_1 on IP-Address 192.168.3.10
2017-12-25 21:50:51.857 -> Established connection to HW_1 on IP-Address 192.168.3.10
2017-12-25 21:50:51.858 -> Requesting initialization data from HW_1
2017-12-25 21:50:51.868 -> #Received unexpected number of modules in node HW_1: Expected: 5 Received: 2
2017-12-25 21:50:51.869 -> #Received unexpected component ID in node HW_1: Expected: 1601 Received: 1602
2017-12-25 21:50:51.872 -> #Received unexpected SlaveID for module number 1 in node HW_1: Expected: 2181 Received: 5131
2017-12-25 21:50:51.875 -> Recognized emBRICK node:
```

---

Node number	Node name	Component ID (BusProtocolVersion, SoftwareVersion, Manufacturer ID)	IP address	Number of Slaves
1	HW_1	1602 ( 4, 20, 1)	192.168.3.10	2

Slave Device ID [OffsetOut, OffsetIn, HW-Revision, ProtocolVersion, Manufacturer ID]  
 {5131[0, 0, 18, 11, 1]}  
 {2181[1, 6, 13, 11, 1]}

---

```
2017-12-25 21:50:51.879 -> Warning on try 1: data for node HW_1 not received in time.

2017-12-25 21:51:22.580 -> Warning on try 1: data for node HW_1 not received in time.
2017-12-25 21:51:22.589 -> Warning on try 2: data for node HW_1 not received in time.
2017-12-25 21:51:22.599 -> Warning on try 3: data for node HW_1 not received in time.
2017-12-25 21:51:22.600 -> Receiving data from HW_1 timed out.
2017-12-25 21:51:22.609 -> Starting connection to emBRICK node HW_1 on IP-Address 192.168.3.10

2017-12-25 21:51:27.058 -> Established connection to HW_1 on IP-Address 192.168.3.10
2017-12-25 21:51:40.442 -> Application terminated
```

Note:

This log is also online available on the PC-Target as a separate message window area when enabling the option "View" -> "Error console".

### 8.3 Middleware Gamma (RST GmbH)

details t.b.d

### 8.4 Modelbased Programing with eTrice (PROTOS GmbH)

details t.b.d

### 8.5 IEC61131 Soft-PLC with logi.CAD3 (logi.cals)

details t.b.d

### 8.6 IEC61499 Soft PLC with 4diac (fortiss GmbH)

details t.b.d

### 8.7 IEC61131Soft-PLC with CODESYS (3S GmbH)

#### 8.7.1 Create your own brick description

The devdesc.xml files are used to describe the Brick itself and its I/Os.

To create your own description, follow these steps.

- Open the [Product Catalogue](#).
- Find in the catalogue your brick description. As an example, we use the P\_2Rel4Di2Ai-01 Brick, that is included in our Starterkit.

6.5 CAE P Bricks (Process Control) .....	178
6.5.1 P-112Rel#-01 .....	179
6.5.2 P-2Rel6Di-01 .....	183
6.5.3 P-2Rel4Di2Ai-0# .....	187
6.5.4 P-6Rel5DiPow-01 .....	191
6.5.5 P-5Rel3DiPoPow-0# .....	197
6.5.6 P-LfTmpAoDAioImp-0# .....	203

- Scroll down to the “Process Data Image” section of your brick.

### 6.5.3.7 Process Data Image

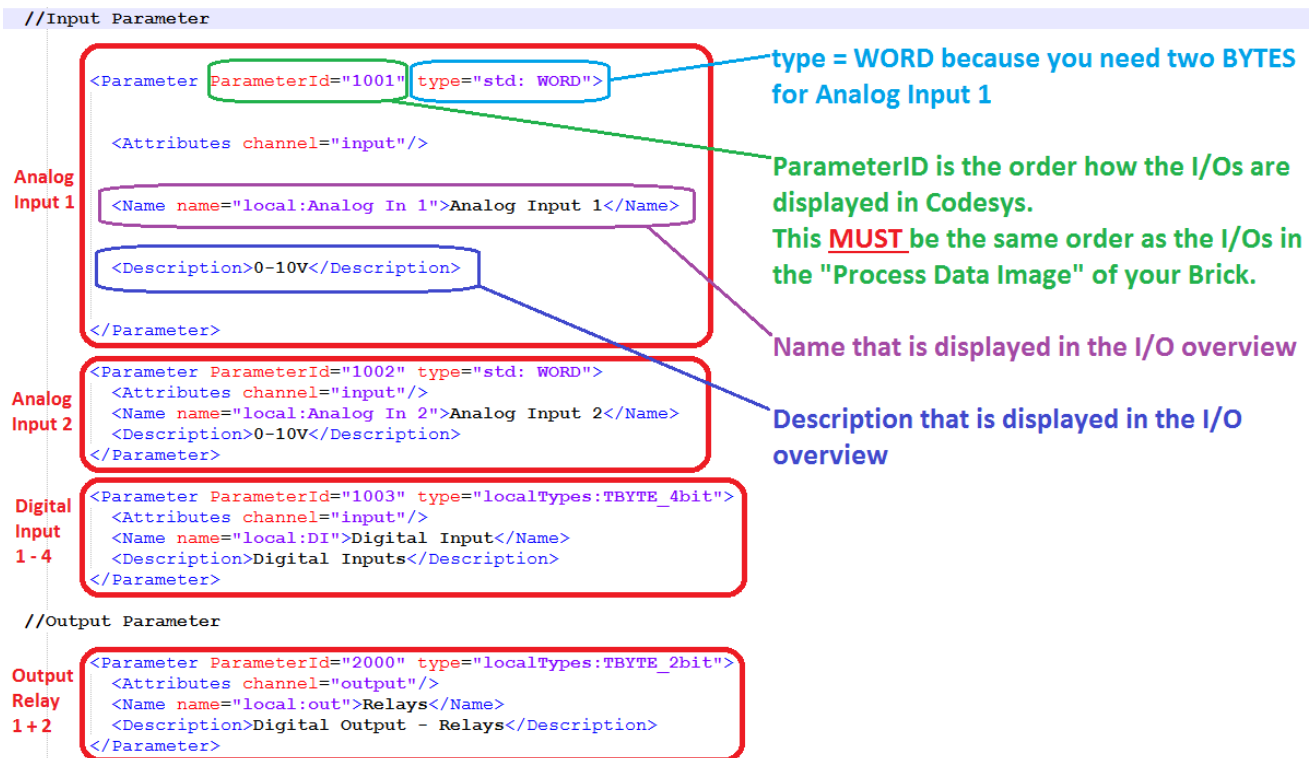
Out-Userdata from Master to Slave total: 1 Bytes			In-Userdata to Master from Slave total: 5 Bytes		
Byt	Function	rC Assign...	Byte	Function	rC Assign...
0	Output setting Bit 0 = Relay 1 Bit 1 = Relay 2	... + eB_B0, 0, ... ... ... + eB_B0, 1, ...	0, 1	Analog-Input 1 0..783 = 0 .. 10V 0..898 = 0 .. 20mA	... + eB_W0, 0, ...
			2, 3	Analog- Input 2 0..783 = 0 .. 10V 0..898 = 0 .. 20mA	... + eB_W1, 0, ...
			4	Digital Input Bit 0 = Digital-Input 1 ... Bit 3 = Digital-Input 4	... + eB_B4, 0, ...

- Make a copy of an existing devdesc.xml file.
- Open the copy and search for the line that says “//Hardware Description”.  
Change as you please.

```
//Hardware Description
<Strings namespace="local">
  <Language lang="en">
    <String identifier="ModelName">CAE_P-2Rel4Di2Ai-01</String>
    <String identifier="DeviceDescription">emBRICK-Module with 2 Relais, 4 Digital Inputs and 2 Analog Inputs.</String>
    <String identifier="VendorName">emBRICK GmbH</String>
    <String identifier="typename">MyDevice_With_Lib</String>
    <String identifier="typedescription">
      A device that includes libs and automatically generates some FB instances from this libs.
    </String>
  </Language>
</Strings>
<Device hideInCatalogue="false">
  <DeviceIdentification>
    <Type>5131</Type>
    <Id>0001 5-131</Id>
    <Version>1.0</Version>
  </DeviceIdentification>
  <DeviceInfo>
    <Name name="local:ModelName">CAE_P-2Rel4Di2Ai-01</Name>
    <Description name="local:DeviceDescription">emBRICK-Module with 2 Relais, 4 Digital Inputs and 2 Analog Inputs.</Description>
    <Vendor name="local:VendorName">emBRICK GmbH</Vendor>
    <OrderNumber>VIM0-0150A00</OrderNumber>
    <Image name="localFiles:DeviceBmp" />
    <Icon name="localFiles:DeviceIcon" />
  </DeviceInfo>
</Device>
</Device>
```

Name of the Brick  
 Manufacturer name of the Brick  
 Description of the Brick  
 ID of the Brick  
 Manufacturer ID followed by the Brick ID  
 Version of the descdev.xml file  
 Name of the Brick  
 Manufacturer name of the Brick  
 Order number of the Brick

Now search for the line “//Input Parameter”.



- Now you must look at the table from the “Process Data Image” of your Brick. The left table lists the Outputs and the right table lists the Inputs. As you see in our example the Output section is only two bits. Relay 1 and 2. The Input section is two BYTES (one WORD = two BYTES) for the Analog Input 1. Also two BYTES (e.g. one WORD) for the Analog Input 2. Followed by four BITS for Digital Input 1, 2, 3 and 4.

- With that Information we can now change the “//Input Parameter” section.

First comes the Analog Input 1. Because it is two BYTES long our type becomes “std: WORD”.

The ParameterID for Inputs starts with “1000”. For Outputs with “2000”.

And because that’s the first entry in the Input section of the “Process Data Image” the ParameterID becomes “1001”.

It does not become “1000” because that is always reserved for our status byte of every Brick.

The analog Input 2 is also two BYTES. Our type is also “std: WORD” and ParameterID is “1002” because it is the second entry.

Next are the four Digital Inputs. They only need four BITS. So our type becomes “localTypes:TBYTE\_4bit” and the ParameterID is “1003”

The “localTypes” is needed because this type is declared inside our “devdesc.xml”. When you are looking at the upper part of the “devdesc.xml” you can see the various declared types.

“localTypes:TBYTE\_2bit”

“localTypes:TBYTE\_4bit”

“localTypes:TBYTE\_6bit”

“localTypes:TBYTE” (8 individual BITS)

- There are also the global types. "std:BYTE" (one BYTE no individual BITS) often used for the Status Byte of every Brick.  
And theres the "std:WORD" that equals two BYTES and is often used for Analog Inputs.
- Now to the "///Output Parameter" section.  
We only have two Relays. They need two BITS. So our type becomes "localTypes:TBYTE\_2bit" and the ParameterID is "2000" because it's the first entry in the Output section of the "Process Data Image" and the "2000" is not reserved.
- If you followed the correct order from the "Process Data Image" Table your "devdesc.xml" should now work. Save the copy under a new name and install the new device in CodeSys.

## Phyton

details t.b.d

## 8.9 Node-RED

details t.b.d

## 8.10 Labview

details t.b.d

## 9. Troubleshooting

### 9.1 Slavemodul state LED

Each Brick has its own state LED. It is placed in the upper area and marked with state LED.

This LED can flash different codes as listed here.

Please make sure that all Bricks are flashing the **running** code.

### Bus State and Status-LED Blink Codes

A *brickBUS*® member (slave module) can have the following bus-states and signalize this with a corresponding LED-flash code:

<b>uninitialized</b> .....	Status-LED: continuous fast flashing approx. 5 Hz = the slave module has not been initialized since last power on.
<b>no_com</b> .....	Status-LED: Morse code • – • = no running communication / communication breaks down. For >2s no messages received.
<b>under_com</b> .....	Status-LED: Morse code • • • = the slave is initialized and communication is running but not stable/to less messages (< 20 messages/s received)
<b>disturbed_com</b> .....	Status-LED: Morse code • • – = the slave is initialized and communication is running but to many error/wrong messages are received (> 1 com-error/s detected)
<b>running</b> .....	Status-LED: continuous flashing approx. 0.8 Hz = correct operation with sufficient correct messages
<b>defect</b> .....	Status-LED: <b>no flashing</b> (continuous on or off lightning) = the module is defect. Operation of the BUS is no longer safe, please replace.

This info is also noted in the System Manual.



## 9.2 Local Mode Operation

### 9.2.1 Check-List

Status - LED? exact in State "running"

Init-Counter?

Number of found bricks

IDs of found bricks

### 9.2.2 radCASE Project

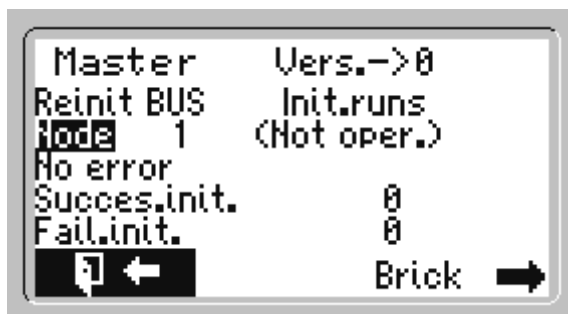
Projects on uniMIND, uniBRAIN, uniCORE or a custom localmaster hardware

#### 9.2.2.1 emBRICK Diagnosis Menü

In the menu „Service “->„Brick Overview “all information is available. This is the bricks firmware version, protokoll version and ID. This layout is different for different display sizes.

Here is a example with the 128x64dot display:

Master



Slave



(all is zero because the pictures are taken in the simulation)

Please check this:

- ➔ can you find all the bricks you have connected with their appropriate ID?  
(the brick ID is on the label on the module)
- ➔ how many success.init. are shown? Normally this has to show a 1. Is this upwards counting every second a bus error is present.
- ➔ How many Fail.init. are shown. This number should be 0, if it is counting up, at least one of the bricks is defective.

### 9.2.3 BeagleboneBrick/RaspberryBrick Projects

In the BoardSupportPackage is an example. At the start all modules found are listed.

For details take a look in the starterkit manual. A similar way should be provided by the programmer.

## 9.3 Remote Mode Operation

This test is only available for systems with the original emBRICK Remote Master Stack code in connection with a CouplingBrick.

### 9.3.1 Log file

#### 8.1.1.1.1 Log-File

This log-file logs all detected *local-masters* and their connected *emBRICK@*-modules. It also compares the connected *emBRICK@*-modules with the expected ones, as described in *eB\_LAN.cnf*. Connection Establishments and breakdowns get logged with their timestamps. This log-file can be found under ...\\CTR\\API.WinPC\\binary\\Release\\brickBUS.log.

#### Example of a log-file with one *local-master*.

```
Start: 2015-12-09 15:51:17
PC-Target, Version 13
Revisionsnummern:
API SVN: 1902
radCASE SVN: 7278
radLib SVN: 672
List of all recognized emBRICK nodes:
```

Node number	Node name	Node ID (EthProtocolVersion, SoftwareVersion, Manufacturer ID, Periphery)	IP address	Number of Slaves
2	KK_EG	0 ( 0, 0, 0, none)	192.168.3.11	0

Node number	Node name	Node ID (EthProtocolVersion, SoftwareVersion, Manufacturer ID, Periphery)	IP address	Number of Slaves
1	HW_1	1602 ( 4, 21, 1, none)	192.168.100.156	1

```
SlaveIDs [OffsetOut, OffsetIn, SW-HW-Version, ProtocolVersion, Manufacturer ID]
{4411[0, 0, 15, 11, 1]}
```

```
Nodes expected: 2
Nodes found: 1

#Received unexpected number of modules in node KK_EG: Expected: 1 Received: 0
#Received unexpected MasterID in node KK_EG: Expected: 1601 Received: 0
#Received unexpected number of modules in node HW_1: Expected: 12 Received: 1
#Received unexpected MasterID in node HW_1: Expected: 1601 Received: 1602
#Received unexpected SlaveID for module number 1 in node HW_1: Expected: 2181 Received: 4411
```

This info is in the Programmers Manual noted.