Manual for the BSB-LPB-LAN Adapter and the BSB-LAN Software - Printer Friendly Version of the Site https://1coderookie.github.io/BSB-LPB-LAN_EN

Table of contents

Back to Introduction

Table of Contents

Users of the outdated setup adapter v2 + Mega: please note appendix D!

Back to Introduction

Introduction

This manual was written to make the start and the usage of the BSB-LPB-LAN adapter (schematic layout v4, Arduino Due and ESP32 version) and the BSB-LAN software easier.

It is suggested to read the manual completely before starting the installation and usage of the adapter and the software.

Jump straight to the TOC

Download the PDF version of this manual

WATCH OUT:
There is NO WARRANTY or GUARANTEE of any kind that this adapter will NOT damage your heating system!
Any implementation of the steps described here, any replica of the adapter and any use of the hardware and software described is at your own risk!
None of the contributors or authors can be held liable for any damages of any kind!

BSB-LPB-LAN - A Short Introduction

"BSB-LPB-LAN" is a community based hardware and software project, which originally had the goal, to access the controllers of different heat generators (oil and gas heating, heat pumps, solar thermal etc.) of certain manufacturers (initially mainly Brötje and Elco) via PC / laptop / tablet / smartphone.
Later on it should be possible to read out data, process it further (eg log and graphically) or even influence the control system and integrate the system into existing SmartHome systems.

All this has now been implemented: With the help of an inbuilt adapter, an Arduino Due and a LAN shield or an ESP32, a suitable heat generator can now be inexpensively integrated into the domestic network. The controller of the heat generator must be equipped with a "Boiler System Bus" (BSB), a "Local Process Bus" (LPB) or a "Point-to-Point Interface" (PPS). These are systems in which a SIEMENS controller is used (or, depending on the heater manufacturer, usually a branded OEM version).

With the usage of the BSB-LPB-LAN adapter and the BSB-LAN software, various functions, values and parameters can now be easily monitored, logged and (if wanted) web-based controlled and changed. An optional integration into existing SmartHome systems such as FHEM, openHAB, HomeMatic, IoBroker, Loxone, IP-Symcon, EDOMI or Home Assistant can be done via HTTPMOD, JSON or MQTT. In addition, the use of the adapter as a standalone logger without LAN or Internet connection when using a microSD card is also possible. Furthermore, optional temperature and humidity sensors can be connected and their data also logged and evaluated. By using an Arduino and the ability to integrate your own code into the BSB-LAN software, there is also a wide range of expansion options.

As a first rough orientation, whether your own heating system is compatible or not, you can search for a connection option for optional room units in the operating instructions of the heater. If room units of the QAA55 / QAA75 type are listed as compatible (Broetje also refers to these as "RGB Basic" and "RGT B Top"), then the adapter can be connected via BSB and the full functionality of BSB-LAN is given. This is the case with most oil, gas and heat pump systems of the last years.
If other room units are listed, see the chapter Room Units.
However, accurate information if the adapter could be connected only provides the actual controller name and the manual of the controller (search for "BSB" and "room unit").

The following overview shows the most common used controllers of the different heating systems which will work with BSB-LAN. As a basic rule we can say, that the controller types of the last years which are named with an **S** at the end (RV**S** and LM**S**) are compatible with BSB-LAN and offer (mostly) the full range of funtionality. For further and more detailed informations about the different controllers and the connection see the corresponding chapters.

Gas-fired heating systems controllers:

- LMU74/LMU75 and LMS14/LMS15 (latest models), connection via BSB, complete functionality

- LMU54/LMU64, connection via PPS, limited functionality

Oil-fired heating systems controllers / solarthermic controllers / zone controllers:

- RVS43/RVS63/RVS46, connection via BSB, full functionality
- RVA/RVP, connection via PPS (modelspecific sometimes LPB), limited functionality

Heat pump controllers:

- RVS21/RVS61, connection via BSB, full functionality

Weishaupt (model WTU):

- RVS23, connection via LPB, (nearly) full functionality

To see a more detailed listing of the reported systems which are sucessfully used with BSB-LAN please follow the corresponding link:

- **Broetje**
- **Elco**
- **Other Manufacturers (e.g. Fujitsu, Atlantic, Weishaupt)**

The software is available here.

Authors:

- Software, schematics v1, first documentation EN, support up to v0.16:
  Gero Schumacher (gero.schumacher [ät] gmail.com)
- Software, PCB schematics v1 & v2, first documentation EN, support since v0.17:
  Frederik Holst (bsb [ät] code-it.de)
- Debugging, manuals, translations, support since v0.17:
  Ulf Dieckmann (adapter [ät] quantentunnel.de)

Based upon the code and the work of many other users and developers! Thanks!

Further to the TOC

Introduction

Introduction

This manual was written to make the start and the usage of the BSB-LPB-LAN adapter (schematic layout v4, Arduino Due and ESP32 version) and the BSB-LAN software easier.

It is suggested to read the manual completely before starting the installation and usage of the adapter and the software.

Jump straight to the TOC

Download the PDF version of this manual

WATCH OUT:
There is NO WARRANTY or GUARANTEE of any kind that this adapter will NOT damage your heating system!
Any implementation of the steps described here, any replica of the adapter and any use of the hardware and software described is at your own risk!
None of the contributors or authors can be held liable for any damages of any kind!

BSB-LPB-LAN - A Short Introduction

"BSB-LPB-LAN" is a community based hardware and software project, which originally had the goal, to access the controllers of different heat generators (oil and gas heating, heat pumps, solar thermal etc.) of certain manufacturers (initially mainly Brötje and Elco) via PC / laptop / tablet / smartphone.
Later on it should be possible to read out data, process it further (eg log and graphically) or even influence the control system and integrate the system into existing SmartHome systems.

All this has now been implemented: With the help of an inbuilt adapter, an Arduino Due and a LAN shield or an ESP32, a suitable heat generator can now be inexpensively integrated into the domestic network. The controller of the heat generator must be equipped with a "Boiler System Bus" (BSB), a "Local Process Bus" (LPB) or a "Point-to-Point Interface" (PPS). These are systems in which a SIEMENS controller is used (or, depending on the heater manufacturer, usually a branded OEM version).

With the usage of the BSB-LPB-LAN adapter and the BSB-LAN software, various functions, values and parameters can now be easily monitored, logged and (if wanted) web-based controlled and changed. An optional integration into existing SmartHome systems such as FHEM, openHAB, HomeMatic, IoBroker, Loxone, IP-Symcon, EDOMI or Home Assistant can be done via HTTPMOD, JSON or MQTT. In addition, the use of the adapter as a standalone logger without LAN or Internet connection when using a microSD card is also possible. Furthermore, optional temperature and humidity sensors can be connected and their data also logged and evaluated. By using an Arduino and the ability to integrate your own code into the BSB-LAN software, there is also a wide range of expansion options.

As a first rough orientation, whether your own heating system is compatible or not, you can search for a connection option for optional room units in the operating instructions of the heater. If room units of the QAA55 / QAA75 type are listed as compatible (Broetje also refers to these as "RGB Basic" and "RGT B Top"), then the adapter can be connected via BSB and the full functionality of BSB-LAN is given. This is the case with most oil, gas and heat pump systems of the last years.
If other room units are listed, see the chapter Room Units.
However, accurate information if the adapter could be connected only provides the actual controller name and the manual of the controller (search for "BSB" and "room unit").

The following overview shows the most common used controllers of the different heating systems which will work with BSB-LAN. As a basic rule we can say, that the controller types of the last years which are named with an **S** at the end (RV**S** and LM**S**) are compatible with BSB-LAN and offer (mostly) the full range of funtionality. For further and more detailed informations about the different controllers and the connection see the corresponding chapters.

Gas-fired heating systems controllers:

- LMU74/LMU75 and LMS14/LMS15 (latest models), connection via BSB, complete functionality

- LMU54/LMU64, connection via PPS, limited functionality

Oil-fired heating systems controllers / solarthermic controllers / zone controllers:

- RVS43/RVS63/RVS46, connection via BSB, full functionality
- RVA/RVP, connection via PPS (modelspecific sometimes LPB), limited functionality

Heat pump controllers:

- RVS21/RVS61, connection via BSB, full functionality

Weishaupt (model WTU):

- RVS23, connection via LPB, (nearly) full functionality

To see a more detailed listing of the reported systems which are sucessfully used with BSB-LAN please follow the corresponding link:

- **Broetje**
- **Elco**
- **Other Manufacturers (e.g. Fujitsu, Atlantic, Weishaupt)**

The software is available here.

Authors:

- Software, schematics v1, first documentation EN, support up to v0.16:
  Gero Schumacher (gero.schumacher [ät] gmail.com)
- Software, PCB schematics v1 & v2, first documentation EN, support since v0.17:
  Frederik Holst (bsb [ät] code-it.de)
- Debugging, manuals, translations, support since v0.17:
  Ulf Dieckmann (adapter [ät] quantentunnel.de)

Based upon the code and the work of many other users and developers! Thanks!

Further to the TOC

1. The BSB-LPB-LAN Adapter and the BSB-LAN Software

Back to TOC
Back to Introduction

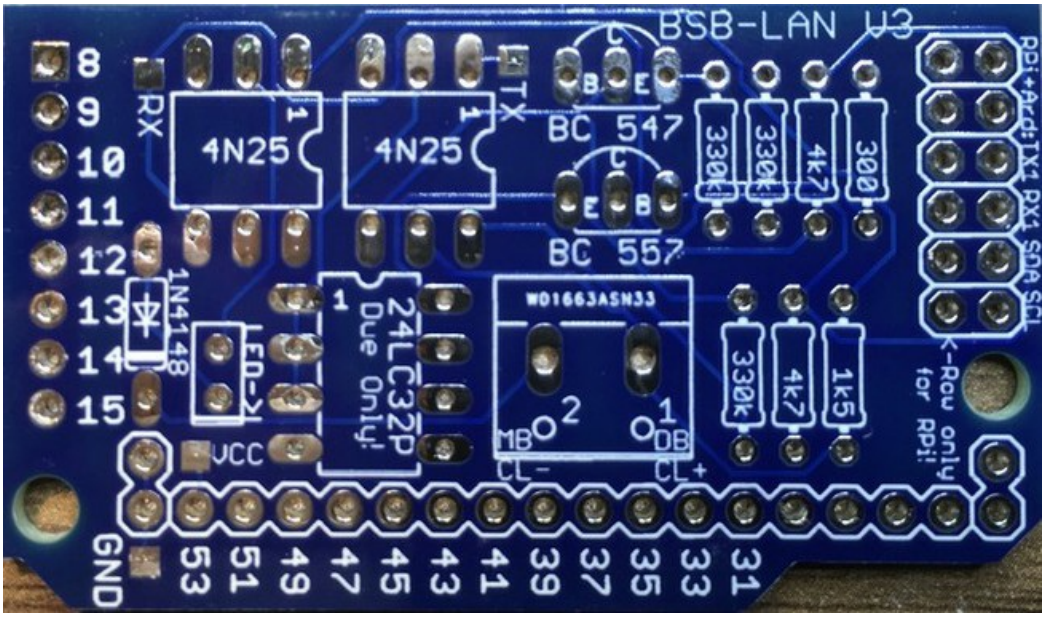
1. The BSB-LPB-LAN Adapter and the BSB-LAN Software

The BSB-LPB-LAN adapter and the BSB-LAN software were developed to realize remote access to the controllers of heating systems.
Besides that it's possible to e.g. add additional temperature sensors (DHT22, DS18B20, BME280) or relais boards or log parameters to an optional microSD card.
You can also use additional individual code by using the file `BSB_LAN_custom.h`.
Of course BSB-LAN can be integrated in existing home automation solutions like FHEM, openHAB, nodeRed, ioBroker and so on by using the supported solutions MQTT, JSON and HTTPMOD.

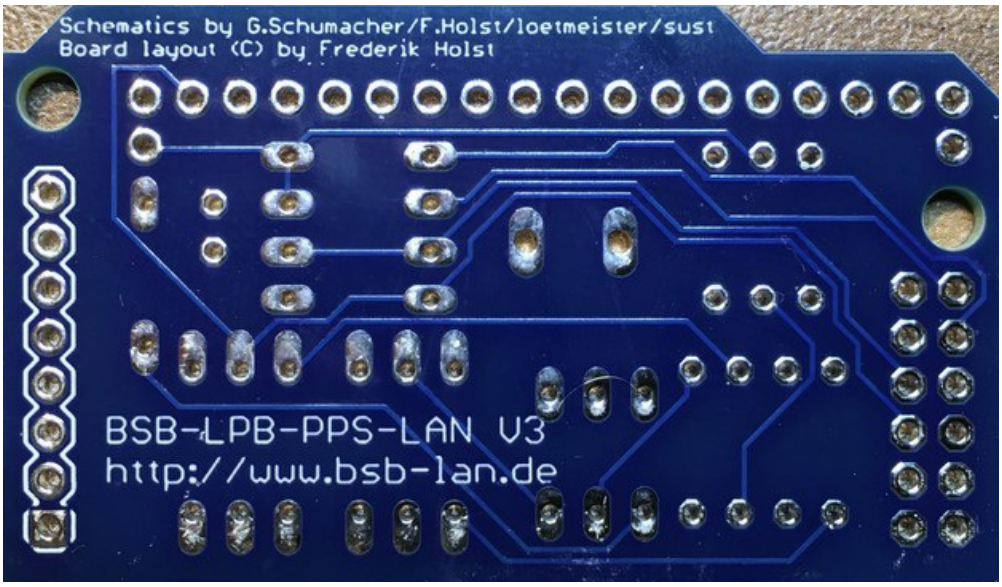
The software runs on an Arduino Due plus a LAN shield and also on an ESP32 - that's (of course besides the adapter itself) already everything you need!
Due to the limited space of flash memory you can't use boards like Arduino Uno or Nano or so.

For using the BSB-LAN system, the controller of your heating system has to be provided with a BSB (Boiler System Bus) or a LPB (Local Process Bus).
Most of the up to date controllers produced by SIEMENS which are used by manufacturers like Broetje or Elco offer at least one of these bus types.
Older controllers which offer only a PPS (point to point connection) may work but mostly with (very) limited functionality.
You can see an overview of the reported heating systems which are successfully used in combination with BSB-LAN here.
Please read the manual of your heating system to check if the controller offers this kind of connector(s).

You can find the schematic for the adapter in the appendix A1.
If you don't want to build it by your own, you can contact Frederik Holst (bsb [at] code-it.de) and ask if a PCB is available. Please mention if you are looking for the ESP32 or the Arduino Due version of the adapter!

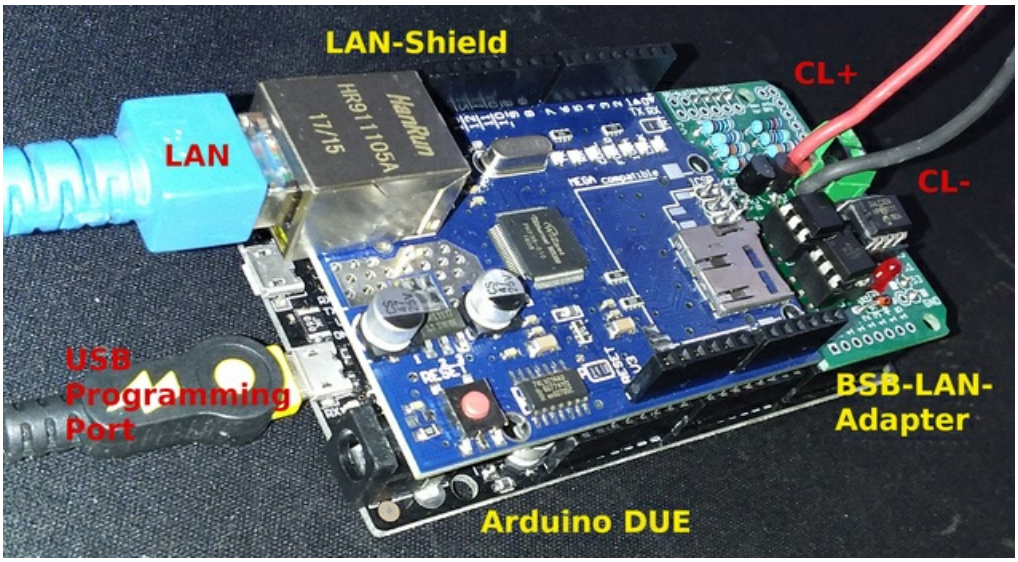


The PCB of the BSB-LPB-LAN adapter v3, top view, not assembled.



The PCB of the BSB-LPB-LAN adapter v3, bottom view, not assembled.

The BSB-LPB-LAN adapter v4, fully assembled, mounted on an Arduino Due (Clone) plus LAN shield.



The complete setup (Arduino Due + LAN shield + BSB-LPB-LAN adapter v3), belonging cables included.

Further on to chapter 2
Back to TOC

2. General Informations about BSB, LPB and PPS

Back to TOC
Back to chapter 1

2. General Informations about BSB, LPB and PPS

BSB (Boiler System Bus), LPB (Local Process Bus) and PPS (point to point connection) are different types of bus systems (well, PPS isn't really a bus though). They aren't compatible between each other, so e.g. you can't connect a BSB unit to a LPB.
Every of the controllers mentioned in this manual which are versions of RVS and LMS (and LMU7x) have at least one BSB port to offer. LPB isn't available at each type of these controllers, but for the usage of BSB-LAN it's not necessary - just use the BSB.
PPS isn't used anymore at younger controllers, mostly old ones like RVA, RVP or LMU5x/6x are based on this type of connection system.

In the following subchapters I'll give a short overview of the main aspects and differences of these bus/connection systems.

2.1 BSB and LPB

BSB (Boiler System Bus) and LPB (Local Process Bus) are two different bus types, which can be divided into two different usage purposes:

1. The BSB is a 'local' bus, where e.g. parts like the operating unit or a room unit are connected to the controller of the heating system. It offers 'local' access to the controller.

2. The LPB is a bus, which offers access across connected controllers (if the installation was set up right!). Using the LPB you could e.g. connect two or more heating units to realize a burner cascade or to connect the controller of you heating system with the controller of your solarthermic system.
   If you have an existing installtion like that you could connect the BSB-LPB-LAN adapter to one of the mentioned controllers and would have access to certain parameters of both controllers. in that case you would have to pay attention to use the correct bus address of the units to make sure you reach the desired controller.

Even though it's possible to use one adapter in an existing LPB structure with different controllers and query each controller by its own address, it's advisable to use one adapter-setup (Arduino + LAN shield + adapter) for each controller if they also offer a BSB port. It's just more comfortable because you wouldn't have to change the destination address every time you want to query another controller.

2.1.1 Addressing within the BSB

Because of the bus structure, each participant gets a specific address. The following addresses are already defined:

bus address	device address	device (name in the serial monitor)
0x00	0	controller itself („HEIZ“)
0x03	3	expansion module 1 („EM1“) / mixer-ClipIn AGU
0x04	4	expansion module 2 („EM2“) / mixer-ClipIn AGU
0x06	6	room unit 1 („RGT1: QAA55, QAA75, IDA“)
0x07	7	room unit 2 („RGT2: QAA55, QAA75“)
0x08	8	room unit 3/P and/or OCI700 servicetool („CNTR“)
0x0A	10	operating unit (with display) („DISP“)
0x0B	11	service unit (QAA75 defined as service unit) („SRVC“)
0x31	49	OZW672 webserver
0x32	50	(presumably) wireless receiver („FE“)
0x36	54	Remocon Net B („REMO")
0x42	**66**	**BSB-LPB-LAN adapter („LAN“)**

bus address	device address	device (name in the serial monitor)
0x7F	127	broadcast message („INF“)

Note:
The preset bus address `0x42` *of the BSB-LPB-LAN adapter is the BSB device address 66. This address is set in the file* `BSB_lan_config.h`.

2.1.2 Addressing within the LPB

The addressing within the LPB is different than the one within the BSB. Basically there are two 'addresses': an address of a segment and an address of a unit. Both have different meanings. Because the topic LPB is pretty complex, please search for further informations by yourself. Especially the documents about the LPB of "Siemens Building Technologies - Landis & Staefa Division" should be regarded as they are the main sources for these informations.

Note:
The preset bus address `0x42` *of the BSB-LPB-LAN adapter is the LPB segment address 4 with device address 3. This address is set in the file* `BSB_lan_config.h`.

2.2 PPS

Right now, the PPS will just be mentioned really short here, because it's only available at *old* controllers and therefore not relevant for most of the users. As already said, PPS is not a real bus. It's more a point-to-point communication protocol for the usage of connecting a room unit to a controller for example. So if you have an old heating system like a Broetje WGB 2N.x and you have (or can connect) a room unit like a QAA50 or QAA70, then you are using PPS.
The adapter has to be connected the same way the room unit would have to be. Please read the manual of your heating system to find out about that. In most cases though the two pins of the connectors at the controller are labeled as "A6" and "MD" (or just "M"). In that case, you have to connect "A6" to "CL+" and "MD"/"M" to "CL-" of the adapter.

Connectors "A6" and "MD" at a Siemens RVA53 controller.

The functionality of this 'bus' is very limited, so you probably only have a dozen of parameters available. In the webinterface of BSB-LAN you only have access to the category "PPS-Bus".

Note:
If there's already a room unit like QAA70 connected to the controller, BSB-LAN only can read values. If you want BSB-LAN to be able to set certain values, you would have to disconnect the room unit for the time you want to have the BSB-LAN adapter connected!
Please take notice of the comments at the specific PPS definements in the file `BSB_lan_config.h` when using PPS!

Important note for users of the old (retired) setup adapter v2 + Arduino Mega 2560:
Because of the time-critical communication of the PPS, it is recommended to adjust the setup for using the hardware serial. Therefore the following adaptions have to be done:

- The adapter has to be *fully* assembled.
- Only the solder jumer SJ1 has to be set.
- The adapter has to be plugged in one pinrow towards the center of the the arduino.

- The configuration has to be changed: set the pins within the variable "BSBbus" to 19 (RX) and 18 (TX) (instead of "68,69").

2.3 Connecting the Adapter to the Controller

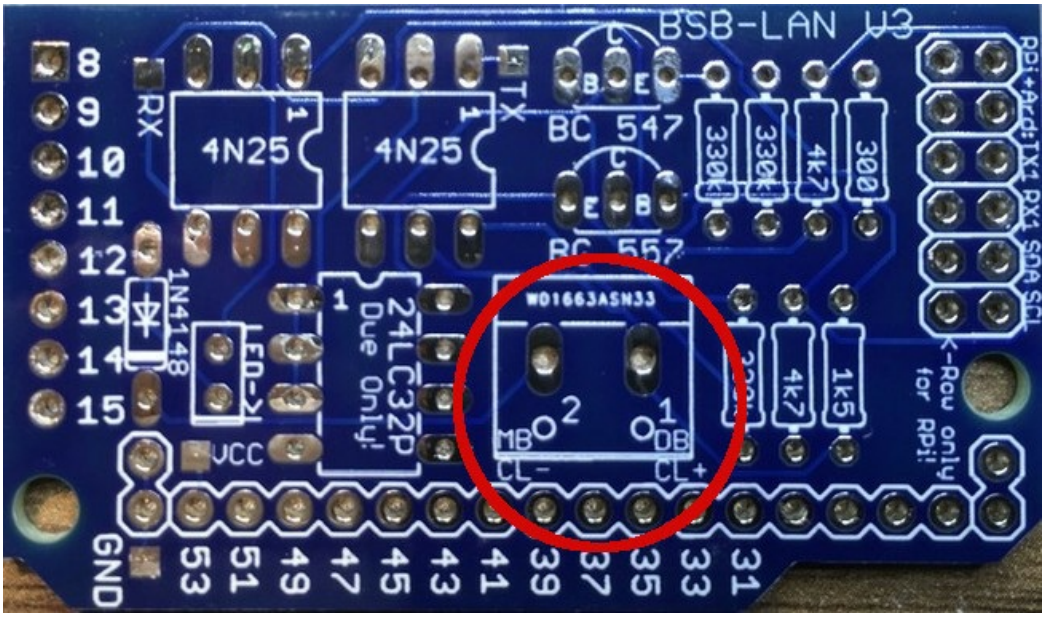
Basically the connection of the BSB-LPB-LAN adapter to the controller is made in the same way and at the same port where a room unit will be connected. To localize the specific port at your controller, please read the manual of your heating system.

In cases where only one BSB port is available at the controller (e.g. RVS21 controller within heat pumps) you can connect the adapter parallel to an already installed room unit.

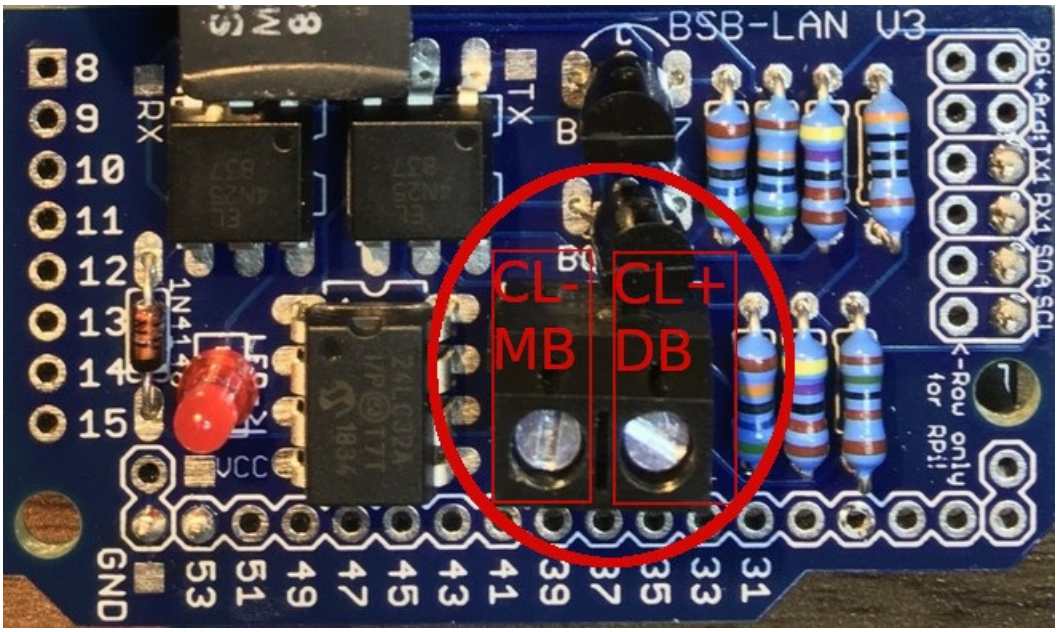
Note:
Because BSB is a real bus, you can also connect the adapter in your living area if there's already a wired room unit installed.
If you don't already have a wired room unit, you can still think about if it's maybe easier to put a long thin bus cable to the heater than a LAN cable.
So it's not necessary at all to connect the adapter exactly at the place where the heater is located.

When connecting or disconnecting the adapter, please make sure that you switched off both units before (Arduino and controller of your heating system)!
Please make sure you are using the right pins and regard the polarity!

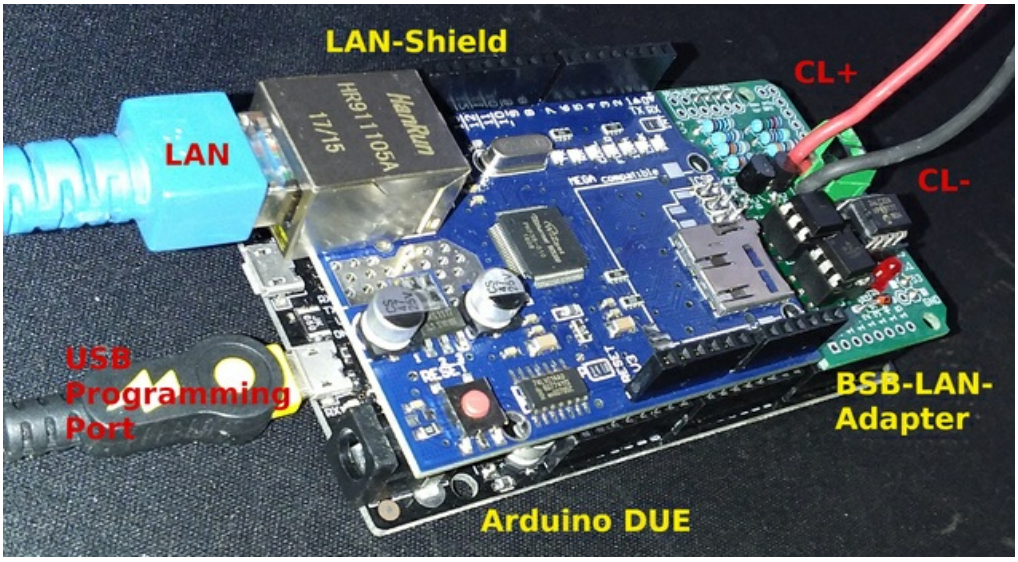
Adapter:
The PCB of the adapter is already labeled with "CL+ / DB" and "CL- / MB".
If you are building an adapter completely by your own, please look at the schematics.



The plain PCB.



Fully assembled PCB.



The complete setup (Arduino Due, LAN shield, BSB-LAN adapter), belonging cables included.

BSB:
The connection of the adapter takes places at the already described ports and pins.
Please connect "CL+" (adapter) to "CL+" (controller) and "CL-" (adapter) to "CL-" (controller).

An additional pin "G+" which could be found sometimes at the controller is only for the backlight of a QAA75 room unit (because it offers 12V) - please make sure that you DON'T use that pin by accident!

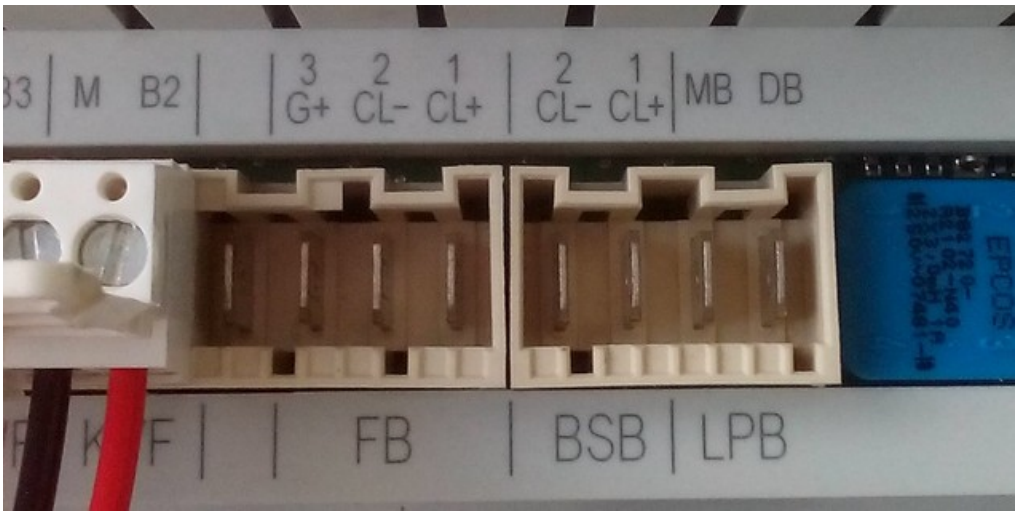
LPB:
The connection of the adapter takes places at the already described ports and pins.
Please connect
"DB (adapter)" to "DB (controller)" and
"MB (adapter)" to "MB (controller)".

PPS:
The connection of the adapter takes places at the already described ports and pins.
In most of the cases it's "A6" and "M", therefore please connect
"CL+" (adapter) to "A6" (controller) and
"CL-" (adapter) to "M" (controller).

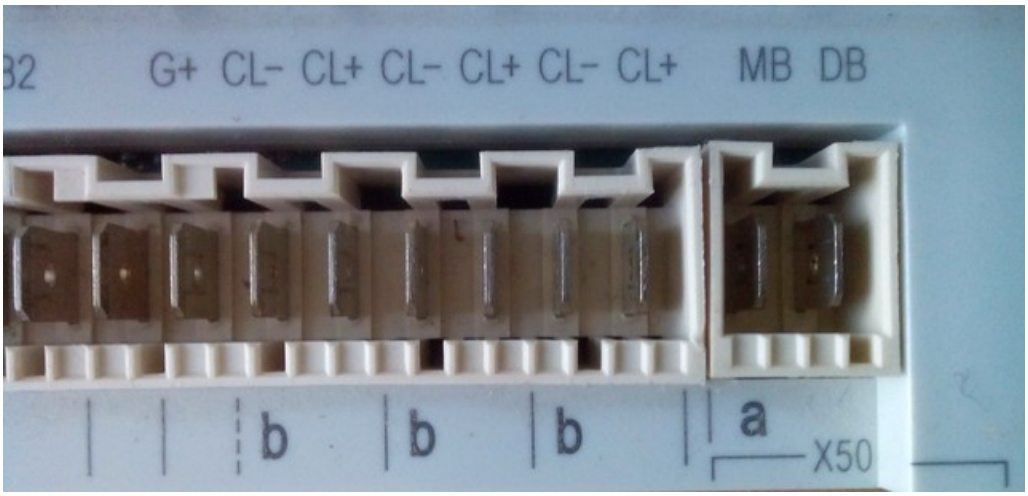
Connectors:
Both the BSB and LPB ports are double-pole and are labeled different sometimes by certain manufacturers. The most common names are:

- BSB port: BSB, FB, BSB & M, CL+ & CL-
- LPB port: LPB, DB & MB

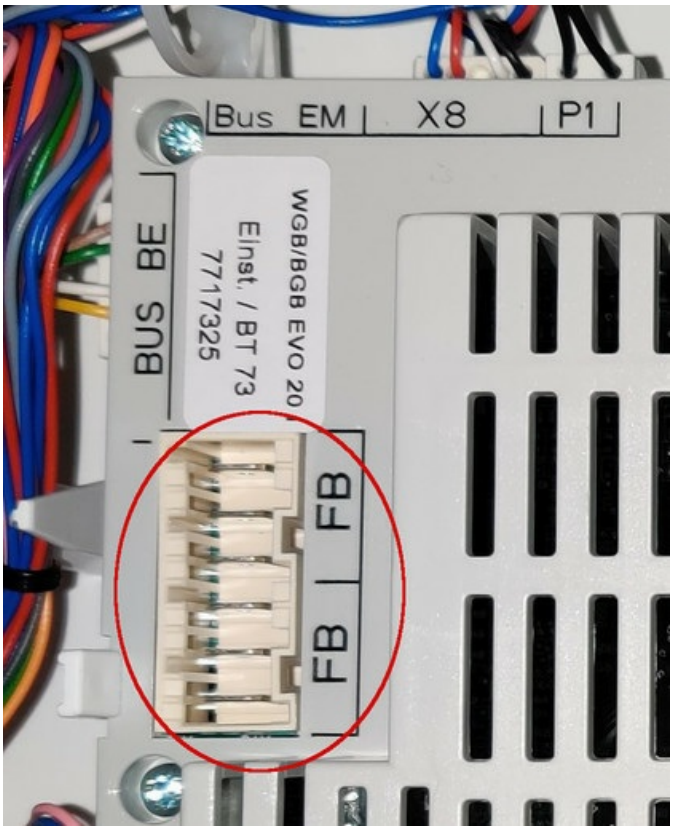
The following pictures show some examples of these connectors at different controllers:



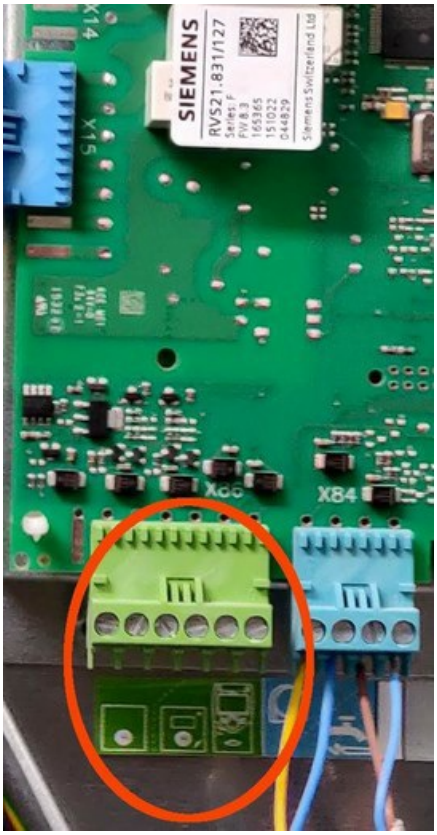
BSB (FB with CL+ & CL-) and LPB (DB & MB) at a Broetje ISR-RVS43.222 controller.



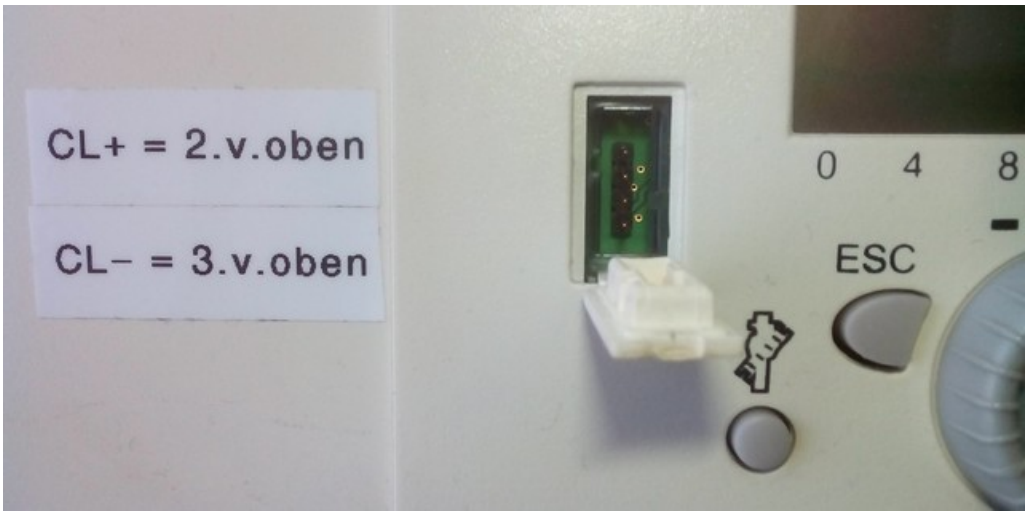
Connectors b = BSB (CL+ & CL-) and a = LPB (DB & MB) at a Siemens RVS63.283 controller.



BSB at connector "FB" at a LMS1x controller.



BSB at connector "X86" at a RVS21 controller (Note: only certain pins!).



BSB (CL+ & CL-) at the four pin service plug at the front of the operating unit ISR Plus. The (permament) usage of this connector isn't advisable though.

Notes on connectors:

The connection of the cables to the respective contacts should always be done with the specific connectors if available. A general list of the respective connectors can't be named here though, because some controllers need special connectors.
For the most common three poled "FB" port (connector for the room unit) which is available at most of the controllers, this connector seem to fit though: Broetje Connector Room Unit ISR, Rast 5- 3pol. = 627528.

BSB / LPB / PPS: If the original connectors are not available, (insulated) 6,3mm cable lugs can be used instead.

Four pin service plug: For the (temporary) connection at the four pin service plug at the front of the operating unit, 2,54mm DuPont connectors (female) can be used. You can find them (e.g.) at the typical breadboard connection cables or at many cables used within the internal parts of desktop computer hardware (e.g. internal speaker, fan).

Notes on cables:

LPB: In order to be as protected as possible from interference, the connection cables for the *LPB* connection should have a cross-section of 1.5mm² in accordance with LPB design principles, twisted two-core and shielded (cable length 250m max per bus node, max total length 1000m).

BSB: For the *BSB* connection, Cu cables with a minimum cross-sectional area of 0.8mm² (up to 20m) should be selected, eg LIYY or LiYCY 2 x 0.8. For cable lengths up to 80m 1mm² should be selected, up to 120m 1,5mm² cross section2. In general, a parallel installation with mains cables should be avoided (interference signals); shielded cables should always be preferred to unshielded cables.

Even though these are the official notes, users reported success with cables like phone installation cables, 0.5-0.75mm speaker cables and so on. Before you have to buy something new, you probably can just give it a try and see if you have some cables already at home which will do the job.

Further on to chapter 3
Back to TOC

3. Supported Heating Systems and Controllers

Back to TOC
Back to chapter 2

3. Supported Heating Systems and Controllers

In general BSB-LAN works with controllers built by SIEMENS which are supported with a BSB and/or a LPB. These controllers are branded and used by different manufacturers of heating systems (e.g. Broetje, Elco). Please read the manual of your heating system to find out if the controller offers a BSB and/or LPB.

Clearification:
Whenever I'm talking about the "controller", I mean the so called "BMU" (boiler management unit). That's the device with all the electronics inside, which controls the whole function of the heating system and which is located inside the housing of the heating system. At this device the sensors, pumps and the operating and room units are connected to.
The 'operating unit' and the optional room units are the devices located outside at the housing of the heating system, the ones with a display and some buttons to interact with the BMU/controller.

Note:
Some recent models of Broetje don't have a BSB and are NOT compatible with BSB-LAN. Please see chapter 3.3 for further informations.

3.1 Successfully Tested Heating Systems

The following chapters are giving an overview of heating systems which have been successfully tested with BSB-LAN and reported by the users. Because not every user reports his heating system, it can be assumed that in practise even more systems work successfully with the BSB-LPB-LAN adapter and the BSB-LAN software.

3.1.1 Broetje

- Broetje BBK 22E [LMS14] (gas fired) {BSB}
- Broetje BBK 22F [LMS14] (gas fired) {BSB}
- Broetje BBK EVO 22I [LMS15] (gas fired) {BSB}
- Broetje BBS EVO 20H [LMS15] (gas fired) {BSB}
- Broetje BBS Pro Evo 15C [LMU74] (gas fired) {BSB}
- Broetje BGB EVO 20H [LMS15] (gas fired) {BSB}
- Broetje BGB EVO 20I [LMS15] (gas fired) {BSB}
- Broetje BMR 20/24 [LMS14] (gas fired) {BSB}
- Broetje BSK 20 [LMS14] (gas fired) {BSB}
- Broetje EcoCondens BBS 15E [LMS14] (gas fired) {BSB}
- Broetje EcoCondens BBS 20E [LMS14] (gas fired) {BSB}
- Broetje EcoCondens BBS 28C [LMU7] (gas fired) {BSB}
- Broetje EcoCondens BBS EVO 20G [LMS15] (gas fired) {BSB}
- Broetje EcoCondens BBS EVO 20H [LMS15] (gas fired) {BSB}
- Broetje EcoCondens Kompakt BBK 22D [LMU7] (gas fired) {BSB}
- Broetje EcoCondens Kompakt BMK 20/24 RSP 160 [LMS15] (gas fired) {BSB}

- Broetje EcoSolar Kompakt BMR 20/24 [LMS15] (gas fired + solar) {BSB}
- Broetje EcoTherm Kompakt WMS 12 [LMS 15] (gas fired) {BSB}
- Broetje EcoTherm Kompakt WMS 24 [LMS 15] (gas fired) {BSB}
- Broetje EcoTherm Plus BBS2N.28 [LMU 64] (gas fired) {+ OCI420 via LPB}
- Broetje EcoTherm Plus WGB2N.20 [LMU 64] (gas fired) {+ OCI420 via LPB}
- Broetje EcoTherm Plus WGB 15-38H [LMS14] (gas fired) {BSB}
- Broetje EcoTherm Plus WGB-M EVO 20H [LMS15] (gas fired) {BSB}
- Broetje EuroCondens BBS EVO 15H [LMS15] (gas fired) {BSB}
- Broetje ISR-SSR [RVS63.283] (solar system controller) {BSB}
- Broetje ISR-ZR1, ZR2 [RVS46.530] (zone controller) {BSB}
- Broetje LogoBloc Unit L-UB 17C [RVS43.122] (Ölbrenner) {BSB}
- Broetje LogoBloc Unit L-UB 25C [RVS43.122] (oil fired) {BSB}
- Broetje NovoCondens BOB 20 [RVS43.325] (oil fired) {BSB}
- Broetje NovoCondens BOB 20B [RVS43] (oil fired) {BSB}
- Broetje NovoCondens BOB 25 [RVS43] (oil fired) {BSB}
- Broetje NovoCondens BOB 25B [RVS43] (oil fired) {BSB}
- Broetje NovoCondens SOB 22 [RVA63.242] (oil fired) {PPS}
- Broetje NovoCondens SOB 26 [RVA63.242] (oil fired) {PPS}
- Broetje NovoCondens SOB 22C [RVS43.222] (oil fired) {BSB}
- Broetje NovoCondens SOB 26C [RVS43.222] (oil fired) + EWM [RVS75.390] {BSB}
- Broetje NovoCondens WOB 20D [RVS43.325] (oil fired) {BSB}
- Broetje SensoTherm BLW Split B [RVS21] (heat pump) {BSB}
- Broetje SensoTherm BLW 12B [RVS21.825] (heat pump) {BSB}
- Broetje SensoTherm BLW 15B [RVS21.825] (heat pump) {BSB}
- Broetje SensoTherm BSW-K [RVS61.843] (heat pump) {BSB}
- Broetje SensoTherm BSW-6A [RVS51] (heat pump) {BSB}
- Broetje SensoTherm BSW-8K [RVS61] (heat pump) {BSB}
- Broetje SGB 260H [LMS14] (Gasbrenner) {BSB}
- Broetje TrioCondens BGB 20E [LMS14] (gas fired) {BSB}
- Broetje WBC 22/24 [RVS43.345] (gas fired) {BSB}
- Broetje WBS 14D [LMU74] (gas fired) {BSB}
- Broetje WBS 14F [LMS14] (gas fired) {BSB}
- Broetje WBS 22 [LMS14] (gas fired) {BSB}
- Broetje WBS 22D [LMU74] (gas fired) {BSB}
- Broetje WBS 22E [LMS14] (gas fired) {BSB}

- Broetje WGB 15E [LMS14] (gas fired) {BSB}
- Broetje WGB 20C [LMU74] (gas fired) {BSB}
- Broetje WGB 20E [LMS14] (gas fired) {BSB}
- Broetje WGB 28E [LMS14] (gas fired) {BSB}
- Broetje WGB-C 20/24H [LMS14] (gas fired) {BSB}
- Broetje WGB 20 Eco [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 15H [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 15I [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 20 [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 20H [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 20I [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 28H [LMS15] (gas fired) {BSB}
- Broetje WGB EVO 38I [LMS15] (gas fired) {BSB}
- Broetje WGB-M EVO 20H [LMS15] (gas fired) {BSB}
- Broetje WGB-M EVO 20I [LMS15] (gas fired) {BSB}
- Broetje WGB Pro EVO 20C [LMU75] (gas fired) {BSB}
- Broetje WGB S 17/20E EcoTherm Plus [LMS14] (gas fired) {BSB}
- Broetje WGB-U 15H [LMS14] (gas fired) {BSB}
- Broetje WGB-U 15I [LMS14] (gas fired) {BSB}
- Broetje WGB-U 20I [LMS14] (gas fired) {BSB}
- Broetje WMC [LMS15] (gas fired) {BSB}
- Broetje WMS 12B [LMS15] (Gasbrenner) {BSB}

The new models WLS/WLC and BOK by Broetje are NOT compatible with BSB-LAN!

3.1.2 Elco

- Elco Aerotop G07-14 [RVS61.843] (heat pump) {BSB}
- Elco Aerotop S [RVS61] (heat pump) {BSB}
- Elco Aerotop T07-16 [RVS61.843] (heat pump) {BSB}
- Elco Aerotop T10C [RVS61.843] (heat pump) {BSB}
- Elco Aquatop 8es [RVS51.843] (heat pump) {BSB}
- Elco Aquatop S08 [RVS61.843] (heat pump) {BSB}
- Elco Aquatop T10C [RVS61.843] (heat pump) {BSB}
- Elco Straton 17 [RVS63] (oil fired) {BSB}
- Elco Straton 21 [RVS63.283] (oil fired) {BSB}
- Elco Straton S [RVS63] (oil fired) {BSB}
- Elco Straton S 21.2 [RVS43.345] (oil fired) {BSB}

- Elco Thision 25S [RVS63]] (gas fired) {BSB}
- Elco Thision 9 [LMU7] (gas fired) {BSB}
- Elco Thision S Plus 13 [LMS14] (gas fired) {BSB}
- Elco Thision S Plus 19 [LMS14] (gas fired) {BSB}
- Elco Thision S Plus 24 [LMS14] (gas fired) {BSB}
- Elco Thision S Plus 24 Compact [?] (gas fired) {BSB}
- Elco Thision S9.1 [LMU7] (gas fired) {BSB}
- Elco Thision S13.1 E [LMU7x] (gas fired) {BSB}
- Elco Thision S17.1 [LMU74] (gas fired) {BSB}
- Elco Thision S17.1 [RVS63.283] (gas fired) {BSB}
- Elco Thision S25.1 [RSV63.283] (gas fired) + MM [AVS75.390] {BSB}

3.1.3 Other Manufacturers

- ATAG Q38CR [LMS14] (gas fired) {BSB}
- Atlantic Alféa Evolution 2 [RVS21] (heat pump) {BSB}
- Atlantic Alféa Excellia A.I.TRI 16 [RVS21] (heat pump) {BSB}
- Atlantic Alféa Excellia Duo [RVS21] (heat pump) {BSB}
- Atlantic Alféa Extensa + [RVS21.831] (heat pump) {BSB}
- Atlantic Alféa Extensa 6+ [RVS21.831] (heat pump) {BSB}
- Atlantic Alféa Extensa AOYA 18 LALL / AOYA 30 LBTL [RVS21] (heat pump) {BSB}
- Atlantic Alféa Extensa Duo [RVS21] (heat pump) {BSB}
- Atlantic Alféa Extensa Duo + [RVS21.831] (heatp pump) {BSB}
- Atlantic Perfinox condens Duo 5024 [LMS14] (gas heater) {BSB}
- Austria Email LWPK 8 [RVS21.831] (heat pump) {BSB}
- Baxi Luna Platinum 1.18 [LMS15] (gas fired) {BSB}
- Baxi Luna Platinum+ [LMS15] (gas fired) {BSB}
- Baxi Luna Platinum+ 1.126A [LMS15] (gas fired) {BSB}
- Baxi Luna Platinum+ 24 [LMS15] (gas fired) {BSB}
- Boesch (unknown model) [RVS63] (heat pump] {BSB}
- CTA Optiheat 1-5es [RVS61] (heat pump) {BSB}
- CTA Optiheat 1-18es [RVS61] (heat pump) {BSB}
- CTC 380 IC [RVS43.143] (oil fired) {BSB}
- District Heating [RVD230] {LPB}
- Deville 9942 [RVA53] (?) (PPS)
- Deville 9981 [RVA53.140] (oil fired) {PPS}
- EVI HEAT Combi-7 [RVA43] (heat pump) {LPB}

- Froeling Rendagas Plus [RVA63.244] (gas fired) {LPB}
- Fujitsu Waterstage Comfort 10 [RVS21.827] (heat pump) {BSB}
- Fujitsu Waterstage WSHA 050 DA [RVS41.813] (heat pump) {BSB}
- Fujitsu Waterstage WSYA 100 DG 6 [RVS21.831] (heat pump) {BSB}
- Fujitsu Waterstage WSYK 160 DC 9 [RVS21.827] (heat pump) {BSB}
- Fujitsu Waterstage WSYP 100 DG 6 [RVS21.831] (heat pump) {BSB}
- Geminox Thrs 19 [LMS14] (gas fired) {BSB}
- Gruenenwald GREENHEAT GH10 ZP 41 E [RVA63] (heat pump) [+ RVA46] {PPS/LPB}
- HANSA SND 30TK [RVS13] (oil fired) {BSB}
- Interdomo Domostar GBK 25 H/SH [LMS15] (gas fired) {BSB}
- MHG Ecostar 200 [RVS53] (oil fired) {BSB}
- MHG Procon E25 [LMS14] (gas fired) {BSB}
- MHG Procon E 25 HS [LMS14] (gas fired) {BSB}
- Olymp SHS 730 [RVS63] (oil fired) {BSB}
- Olymp WHS 500 [RVS61] (heat pump) {BSB}
- Sieger TG11 [RVP54.100] (oil fired) {PPS}
- Šildymo Technologijų Centras ŠTC STC9 [RVS51] (heat pump) {BSB}
- Sixmadun TG11 BE [RVA63] (?) {PPS/LPB}
- Termomax Termo ÖV Color 35/E [RVA63.2424] (oil fired) {LPB}
- Thermics Energie 9PWW [RVS61] (heat pump) {BSB}
- Thermital TBox Clima TOP [RVS63] (gas fired + solar + pellet stove) {BSB/LPB}
- Viessmann Vitotwin 300-W {BSB}
- Weishaupt WTU 15 S [WRS-CPU-B1 = RVS23] (oil fired) {LPB}
- Weishaupt WTU 25 G [WRS-CPU B2/E = RVS23] (oil fired) {LPB}
- Weishaupt WTU 25 G [WRS-CPU-B3 = RVS23] (oil fired) {LPB}
- Weishaupt WTU 25 S [WRS-CPU-B3 = RVS23] (oil fired) {LPB}
- Weishaupt WTU 30 S [WRS-CPU-B1 = RVS23] (oil fired) {LPB}

3.2 Detailed Listing and Description of the Supported Controllers

The following list of controllers and descriptions should give a short overview of a selection of devices already supported by BSB-LAN and their rudimentary differences. The different controller-specific availability of special parameters will not further received. It should be noted, however, that via BSB-LAN several parameters are available which aren't available by the regular operating unit of the heating system itself.

3.2.1 LMx Controllers

The following subchapters are about the LMU and LMS controller types. These seem to be used within gas fired heating systems.

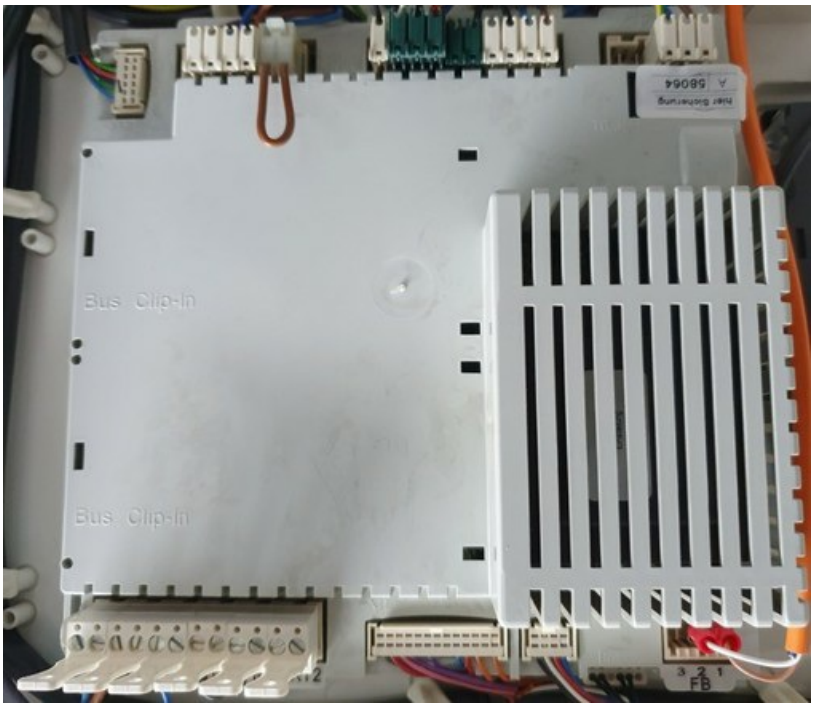
3.2.1.1 LMU Controllers

Controllers of the series **LMU54/LMU64** are installed in older systems, they are out of date. According to experience, these controllers have neither a BSB nor a LPB, only a PPS interface is available here. Sometimes LPB can be retrofitted by means of a ClipIn module (OCI420).

A LMU64 controller with an installed OCI420 ClipIn module.

Using BSB-LAN with these controller models is, according to experience, only possible to a limited extent. More detailed information can be found in chapter 3.4.

Controllers of the series **LMU74/LMU75** appear to be the successors of the LMU54/LMU64 controller series and are also no longer installed.



A LMU7x controller.

The LMU7x controller type usually just offers BSB connection. If needed, LPB needs to be retrofitted using a ClipIn module (OCI420) (this is not necessary for using BSB-LAN though!).

The control unit usually is a variant of the Siemens AVS37.294 (so called "ISR Plus" whithin Broetje).

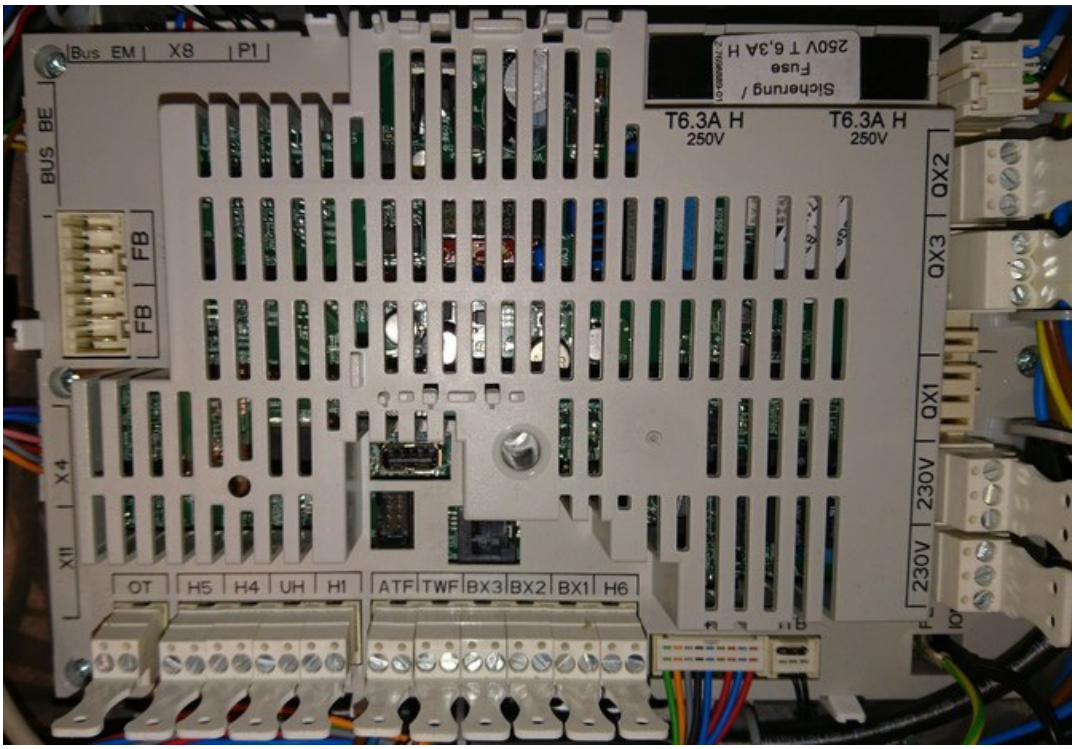
Usually NTC10k (QAD36, QAZ36) and NTC1k (QAC34 = outdoor temperature sensor) are used as sensors.

3.2.1.2 LMS Controllers

Controllers of the series **LMS** seem to be the successors of the LMU series and thus the current controller generation.

The (functional) difference between the LMS14 and the LMS15 seems to be the "Sitherm Pro" application to optimize the overall combustion process, which apparently only the LMS15 controller seems to offer.

The LMS controller type usually just offers a BSB connection. If needed, LPB can be retrofitted using a ClipIn module (OCI345) (this is not necessary for using BSB-LAN though!).



A LMS15 controller.

The operating unit usually is a variant of the Siemens AVS37.294 (so called "ISR Plus" whithin Broetje).

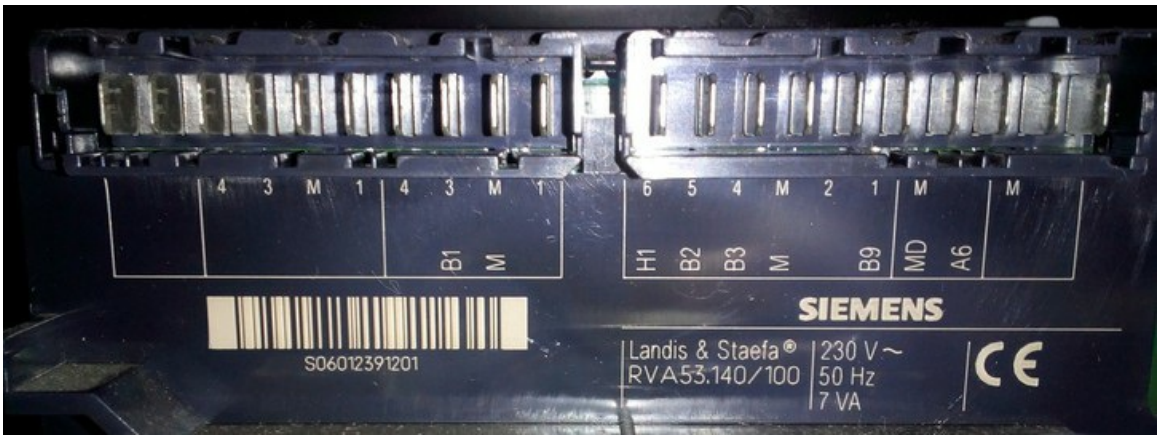
Usually NTC10k (QAD36, QAZ36) and NTC1k (QAC34 = outdoor temperature sensor) are used as sensors.

3.2.2 RVx Controllers

The following subchapters are about the RVA, RVP and RVS (current one) controller types. These seem to be used within oil fired heating systems, heat pumps and different 'standalone' systems (like solar or zone controllers).

3.2.2.1 RVA and RVP Controllers

Controllers of the type **RVA** seem to belong to the previous controller generation and, depending on the model, only offer a PPS (RVA53) or a PPS and a LPB connection (RVA63) but no BSB.
As an (included) operating unit usually a variant of the so called "Eurocontrol" (Broetje) is installed.



A RVA53 controller.

Frontside view: Operating unit of a RVA53 controller.

Controllers of the type *RVP* seem to be even older than RVA controllers and only offer a PPS interface.

3.2.2.2 RVS Controllers

Controllers of the type **RVS** seem to be the current controller generation.
They usually offer both a LPB and several BSB connections.
Exceptions seem to be the controllers of the series RVS21, RVS41, RVS51, RVS61 and RVS23:

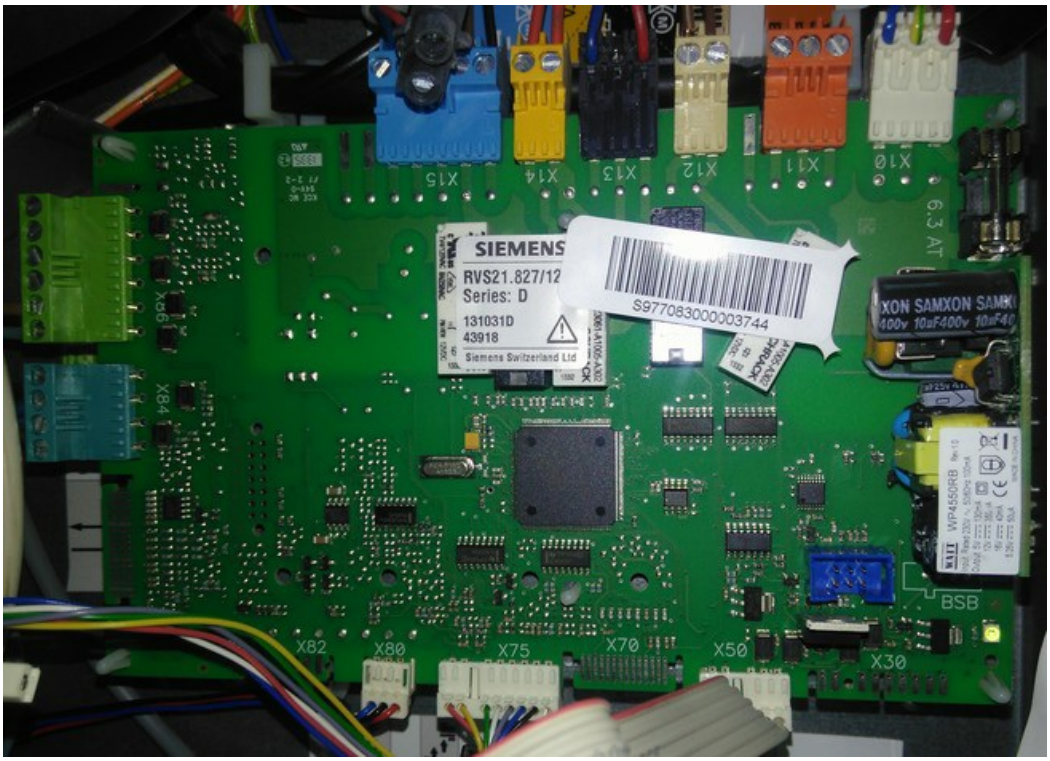
- RVSx1 controllers are used within heat pumps, the RVS21 seems to offer only a BSB connector.
- RVS23 controllers are used on a particular Weishaupt model (WTU) and seem to only offer a LPB. These controllers seem to be labeled by Weishaupt as "WRS-CPU Bx". Further information on this controller model can be found in [chapter 3.5] (chap03.md#35-special-case-weishaupt-heating-systems).

The operating unit usually is a variant of the Siemens AVS37.294 (so called "ISR Plus" whithin Broetje).

Usually NTC10k (QAD36, QAZ36) and NTC1k (QAC34 = outdoor temperature sensor) are used as sensors.

The following gives a short overview of the main RVS controller types.

RVS21.xxx
The RVS21 is the type of controller which is used in heatpumps. It offers BSB and a pair of connectors for an optional room unit.



A RVS21 controller.

If needed, LPB can be retrofitted using a ClipIn module (OCI345) (this is not necessary for using BSB-LAN though!).

RVS41.xxx
The RVS41 is another type of controller which is used within heat pumps. it offers BSB and LPB and seems to be pretty identical to the RVS43 (at least judging by the look of it).

RVS43.xxx
The RVS43 is the type that usually is built in oil fired burner systems. The number of connectors and functions could be expanded with an AVS75 expansion module.

A RVS43 controller.

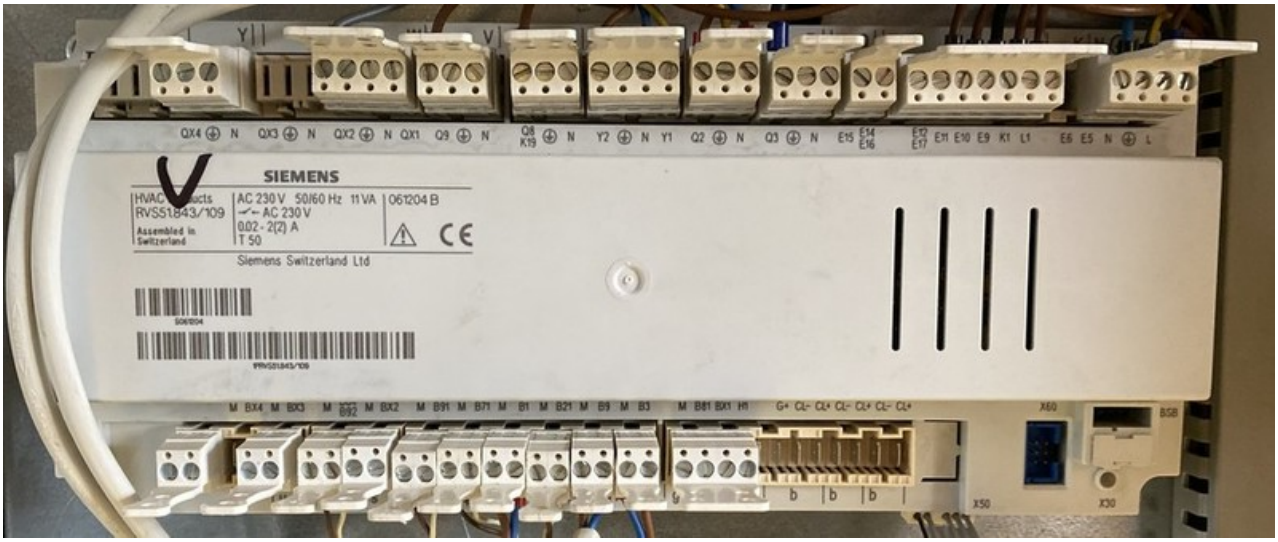
RVS46.xxx

The RVS46 is a small zone controller, which offers one (ZR1) or two (ZR2) connections for a pump/heating circuit. The RVS46 can control zones/circuits by it's own, or integrated in the system via LPB connection to a main controller. It offers BSB and LPB.

The 'small' zone controller ZR1.

The ZR1/2 is not designed for controlling the whole functionality of e.g. a complete oil fired burner.

RVS51.xxx
The RVS51 is the 'bigger' type of controller which is used in heatpumps. It offers BSB and LPB and seems to be pretty identical to the RVS63 (at least judging by the look of it).



A RVS51.843 controller.

RVS61.xxx
The RVS61 is the 'bigger' type of controller which is used in heatpumps. It offers BSB and LPB and seems to be pretty identical to the RVS63 (at least judging by the look of it).

RVS63.xxx
The RVS63 seems to be the 'biggest' controller with the most connectors and functions. Basically he is designed to control systems which are more complex, e.g. additionally solar thermic systems or an integrated oven. Therefore it is named "Solar System Controller" within Broetje. The RVS63 can already be built in within complex heating systems or it could optionally added. In that case it comes with an external housing and must be connected via LPB to the already existing controller. In that case, all the sensors, pumps etc. of the main system have to be

connected to the RVS63, because it becomes the 'main' controller for the whole system.

A RVS63 controller.

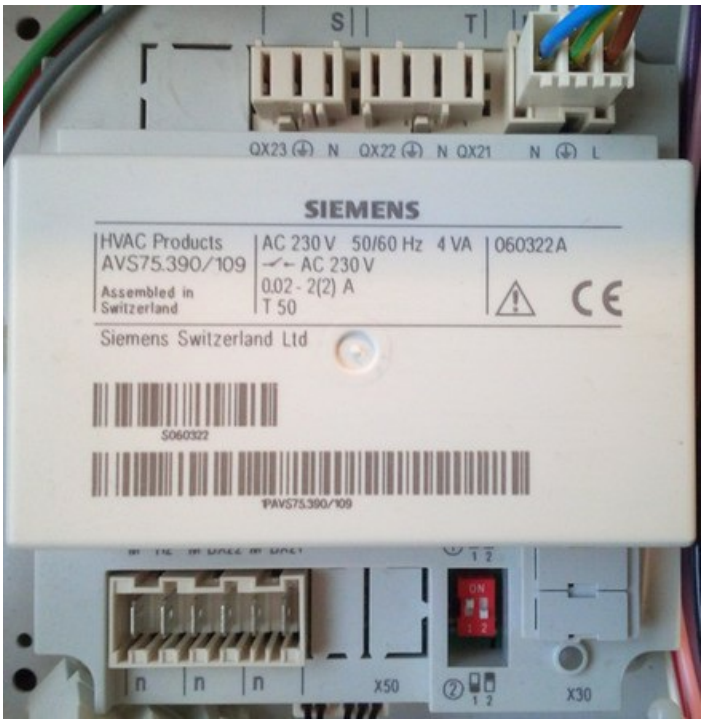
The **RVS65.xxx** seems to be pretty identical to the RVS63 and -until now- was reported only once by a user as being a wall-mounted "Solar System Controller" from Broetje.

3.2.3 Expansion- and ClipIn-Modules

If the available connectors and the range of function of the specific controller aren't enough (e.g. retrofitting of a solarthermic system), one can expand the system by using an expansion- or ClipIn-module. An expansion module offers connectors for (e.g.) a pump circuit and the belonging sensors.
These modules are being connected at the main controller by using a special bus cable and the dedicated connector. Internally they are communicating with the controller via BSB (an exception seems to be the used controller type within the named Weishaupt heating systems). The parameterization takes places via the operating unit of the connected controller.
Therefore, access to an extension module is only possible via the specific parameters within the main controller. Because the expansion modules are listed within a query of `ip/Q`, I'll present the two main types really short in the follwing.

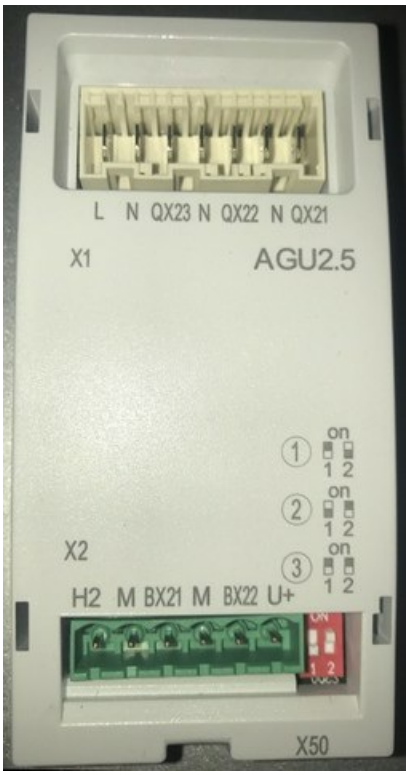
Note:
If you want to retrofit an expansion module, of course see the specific manual of your heating system for further informations and call a heating engineer for the installation.

Expansion modules of the type **AVS75.xxx** are used within the RVS and LMS controller types. The bus connection usually takes places via the connector "Bus-EM".



Expansion module AVS75.390.

Expansion modules for LMU controller types are named "ClipIn-module". There seem to be different types for the specific needs (e.g. relay module, solar module). In general, the main appelation seems to be **AGU2.5x** (where the "x" seems to label the respective version), the bus connection usually takes place via the connector "X50".



ClipIn-module AGU2.55.

3.2.4 Operating Units

The operating unit (located at the heating system itself) within the systems of the recent years (with controller types LMU7x, LMS1x, RVS) usually are types of the **AVS37.xxx**. They look pretty much the same within the different manufacturers, within specific systems (e.g. heat pumps) certain buttons or functions can differ though.
If you compare the look of the AVS37 operating unit and the QAA75.61x room unit, you can see that they actually also look pretty identical and the usage of both devices is also almost the same. In most cases the heater sided operating unit constantly shows the temperature of the heating

device and the room unit shows the room temperature. Both units spread these values regularly (approx. every 10 seconds) over the BSB as a broadcast (INF-message).

A typical operating unit AVS37.

Recently some manufacturers are using a new type of operating unit though, it's called **QAA75.91x**. It seems to be possible to detach these units from the heater itself and -by using an optional connection setup- to install them in your living area. In that case they are still working as the main operating unit for the controller, but with the additional benefits of a room unit.

A QAA75.91x operating unit.

3.3 New Type: NOT Supported Controller from Broetje

It should be noted that the heating manufacturers introduced new device models to the market. According to current knowledge this type of controller is NOT compatible with BSB-LAN.

Within Broetje these seem to be the heating system series

- WLS / WLC (gas fired),
- BOK (oil fired),
- BLW Split-P, BLW Split C and BLW Split-K C (heat pump).

These systems seem to have 'IWR CAN'-based controllers built in (at the heater unit "IWR Alpha", room unit "IWR IDA"), which have neither a

LPB nor a BSB.

The following image of a WLC24 board shows the existing connections.

Connectors of the new controller model at a Broetje WLC24 - this controller is incompatible with BSB-LAN!

In addition to a service socket (probably IWR CAN) there are not further documented 'L-Bus' and 'R-Bus'.
At the 'R-bus' (room unit bus) either a room thermostat (on / off) or the new 'smart' room unit "Broetje IDA" can be connected.

WATCH OUT:
At none of these connectors the BSB-LPB-LAN adapter can be connected!

3.4 Special Case: LMU54/LMU64 Controller

LMU54 / LMU64 controllers are based on OpenTherm, which has different bus specifications and also a different communication protocol. Therefore, OpenTherm is not compatible with BSB-LAN.
However, often there is a possibility to connect this controller type anyway: as with the BSB controllers LMU7x and LMS1x, it is possible to retrofit a LPB by means of a so-called ClipIn module (OCI420). At this turn, the adapter can be connected.

A LMU64 controller with an installed OCI420 ClipIn module.

However, the functionality of this type of controller (even when using BSB-LAN) is relatively limited and also dependent to a certain extent on the software version of the controller (tested with LMU64, SW v2.08 vs. SW v3.0): controllers with SW from v3.0 seem to offer more functions

(controllable via BSB-LAN) than controllers with SW <v3.0. In particular, the two setpoint temperature parameters 709 and 711 should be mentioned here. On their basis the burner behavior could be determined to a certain extent - these can only be used or changed with SW from v3.0. (Note: There is still an attempt if the burner behavior can be satisfactorily influenced by relays on another contact, but up to now we didn't find a solution for that.)

However, according to current knowledge, parameters such as outside temperature, boiler temperature, DHW temperature, flow temperature, etc. can be accessed within both software versions mentioned.

To be fair, it must be said here that the additional financial expense for purchasing an OCI420 LPB-ClipIn module may not be 'worthwhile'. However, this depends on the pursued goal. If you only want to log temperatures to get a rough overview of the actual state of the heating system, a more reasonable solution with a corresponding DS18B20 temperature sensor installation would be sufficient.

Hints for connection and configuration of an OCI420-ClipIn are given in chapter 3.8.

3.5 Special Case: Weishaupt Heating Systems

Some Weishaupt devices (see list of successfully tested devices: Weishaupt WTU with WRS-CPU control unit) have RVS23 controllers installed. This controller type has a LPB on which the existing installation of Weishaupt systems is already based: room units, operating units and extension modules are already connected to each other via LPB. The adapter can also be connected to this LPB, but it must be correctly integrated into the existing LPB installation. In general, this isn't a problem with the default LPB address of the adapter (segment 4, device address 3), but it should be checked again if there are any communication problems.

The Weishaupt devices also seem to have a service socket in addition to the regular operating unit, with two of the four pins provided and led out. According to the statement of a Weishaupt user (*Thanks to BSB-LAN user Philippe!*), the upper one of the two pins seems to be MB and the lower one seems to be DB.

3.6 Conventional Room Units for the Listed Controllers

The following briefly describes the different room units. These are also manufactured by SIEMENS and branded by the different heating manufacturers. Thus, they can be used across manufacturers, e.g. a corresponding QAA room unit of Elco can be used on a Broetje heater (of course, always provided that it is the right type of room unit). It's not yet known if there are certain restrictions in individual cases.

As optional 'local' accessory parts of the heating system, they are connected to the BSB. That's why the connector for room units is what you are looking for, when you want to connect the adapter. So if you connect a room unit and adjusted the settings of the specific parameters (e.g. usage and influence of the room unit and room temperature), you can directly access the measured room temp. If you don't have an external room unit, but you can or want to measure your room temperature(s) in a different way, then you can imitate a room unit by transmitting these measured temperatures via BSB-LAN to the controller and influence the behaviour. For that, look up the function itself and the description of the URL command `/Ixxx=yyy`.

The following description starts with the room units for the current heating system controllers (RVS and LMS), which are also fully supported by BSB-LAN (so called "Broetje ISR").

Note: It seems as if the product portfolio has been supplemented with new room units and other accessories. On occasion, I'll add relevant products here.

3.6.1 QAA55 / QAA58

The QAA55 is the 'smallest' and most affordable ISR room unit model. At Broetje it is called "RGB B", sometimes it is also called "Basic Room Unit", "ISR RGB" or similar. It is quite limited in functionality and is basically 'just' a room temperature sensor with a few additional operating options.

The QAA 55 room unit.

In addition to the optional measurement of the room temperature, it offers a presence button and the options for switching the operating mode and for changing the room set temperature. It only has a small LCD display that shows the current room temperature. It is connected via a two-pole cable to the BSB.

The QAA58 is the wireless version of the QAA55. It is battery operated, the AVS71.390 radio frequency receiver (868 MHz frequency) must in turn be connected to the X60 connection of the boiler controller via cable.

3.6.2 QAA75 / QAA78

The QAA75.61x is the 'big' ISR room unit. In addition to the integrated temperature sensor, it has the full functionality of the boiler-side control unit. In addition, there is a presence button and a manual DHW push can be triggered by pressing the DHW mode button for a longer time.

The QAA75.61x room unit.

At Broetje the QAA75.61x is called "room unit RGT", sometimes it is also called "room unit RGT B Top", "ISR RGT" or similar. It is also connected by cable to the BSB, with a third connection for the optional backlight available (terminal "G +" on the controller).

The QAA78.61x is the wireless version of the QAA75.61x. It is battery operated, the AVS71.390 radio frequency receiver (868 MHz frequency) must in turn be connected to the X60 connection of the boiler controller via cable. The above named "RGT" is extended by an "F" at Broetje, so it's "RGTF".

Note:
At this point it has to be mentioned, that obviously two different versions of the QAA75 are available: the already mentioned room unit QAA75.61x and the different looking QAA75.91x.
Whenever I'm referring to the "QAA75" in this manual, I mean the above described model QAA75.61x.

The QAA75.91x seems to offer the same functionality like the QAA75.61x, but it seems to be used only with some types of heating systems by certain manufacturers (e.g. Broetje WMS/WMC C, BMK B, BMR B and Baxi Luna Platinum+). At these types of heating systems, it seems to be used as the operating unit which is located at the housing of the heating system itself, but (in conjunction with an optional adapter, e.g. Broetje "ISR RGA") could also be used as a room unit. In that case it seems to be still used as the operating unit, just with the additional benefit of the functions of a room unit.

A QAA75.91x operating unit, with optional equipment useable as a room unit.

3.6.3 QAA74

The QAA74 is a pretty new type of room unit at the market, which should/will replace the QAA75 in long term. At Broetje it's called "ISR RGP" (room unit premium), at Siemens "UI400". It is equipped with a 3,8" LCD display and a turn/push button for control purposes. Within some specific types of heating system, it's also used as the main operating unit, named AVS74.

3.6.4 Broetje IDA

The "Broetje IDA" is a room unit which, in addition to an integrated temperature sensor and some functions, also offers a certain range of functions for controlling the heating system via app with a smartphone. A presence button is not available.

IDA is integrated into the domestic WLAN and requires Internet access, if you want to control the unit via app. In the case of purely local use of the room unit (without remote access via the app), no WLAN access is required. Incidentally, the WLAN access also updates the IDA firmware. An interesting analysis of the traffic was made here by FHEM forum member "freetz".

For connection to the BSB of the boiler controller, a BSB interface (GTW17) must be connected. Interested user must look for "ISR IDA" in this case, so that the GTW17 is included in the package.
For controllers with the communication protocol OpenTherm (e.g. the older controller generation Broetje LMU6x), the OT interface (GTW16) must be used.
IWR-CAN-based controllers (see chapter 3.3) can directly be connected to the service dongle GW05 (WLAN gateway).

The exact functionality and installation steps of IDA are to be taken from the corresponding instructions of the manufacturer.

The parallel use of IDA and BSB-LAN is possible in principle, however, due to the report of a user (*Thanks to FHEM-Foums member "mifh"!*) a few restrictions regarding the functional scope of BSB-LAN are known:
If IDA is connected to the BSB, then it is the master for the settings or values of

- time and date,
- heating or switching programs and the
- room temperature.
  If these settings / values are changed via BSB-LAN, they will be overwritten with the settings / values from IDA after a short time.
  It is thus no longer possible, for example, to detect the room temperatures from different rooms and to transmit them to the controller via BSB-LAN, since IDA overwrites this.

The function of the presence button via BSB-LAN should still be available.

3.6.5 QAA53 / QAA73

The room units QAA 53 and QAA 73 also differ in their functional scope. They are used in the OpenTherm-based LMU6x controllers.
Further information on these room units can be found in the corresponding instructions.

3.6.6 QAA50 / QAA70

In principle, the QAA50 and QAA70 also have the same difference in functionality. These room units are used in the old controller generations, which offers only one PPS connector. When using the adapter parallel to an already existing room unit it's only possible to read values via BSB-LAN. In that case no values and settings of the heating controller can be changed via BSB-LAN.

A QAA70 room unit.

Further information on these room units can be found in the corresponding instructions.

3.7 Special Accessories: Webserver OZW672 and Servicetool OCI700

For the sake of completeness there should two commercial solutions be mentioned, which offer access to the controller of the heating system. Thats the webserver OZW672 and the servicetool OCI700.

The webserver OZW672 (Broetje: "ISR OZW") is connected via bus cable to the controller and with a LAN cable to the network (and, if desired, also to the internet). If desired, one could connect with the fee-based dataportal of Broetje and offers remote access (via PC, tablet or smartphone+app) to the controller.

The OCI700 is the servicetool used by the installation engineer. It is connected to a local computer running a special software and offers an overview of the settings of the controller.

3.8 Retrofitting an LPB by Using an OCI420 ClipIn

If an OCI420 should be connected and used with a LMx controller, the installation and the connection must be made in accordance with the respective operating instructions.

There are, however, a few key points that usually can't be found in the operating instructions although they are necessary for a successful operation. This mainly concerns the settings that have to be made for the LPB power supply. Furthermore, the LPB device address 1 with segment address 0 must be set and the setting as the time master has to be made.
As always, the following information comes without any guarantee!

If you follow the instructions on the OCI420, you will most likely encounter error 81, which means "short circuit in the LPB bus or missing power supply". If the OCI420 has been connected correctly, the LPB bus power must be activated in this case. The parameter is "LPBKonfig0".

The following settings are described for controllers of type LMU64. Except for the parameter numbers, the settings of the bits are identical for other LMx controllers.
For the LMU64, the relevant parameter has the number 604 (for LMU74: parameter number 6006). Here are eight bits (604.0 to 604.7) available to be set as follows (where "0" = OFF and "1" = ON):

604.0 = 0 → time master
604.1 = 1 → time master
604.2 = 1 → distributed bus supply AUTOMATIC
604.3 = 1 → status LPB bus supply: 1 = active

604.4 = 1 → event behavior allowed
604.5 = 0 → domestic hot water allocation
604.6 = 0 → domestic hot water allocation
604.7 = 0 → no priority of LMU request before external power specification

If you call up the 'overview' of the LPBKonfig0 settings, however, the bit order is displayed from back to front (from bit 7 to bit 0!) and should be as follows after the successful setting: 00011110.
Furthermore, the following settings have to be made:

605 LPB device address = 1
606 LPB segment address = 0

After successful setting, no error code should occur and the green LED on the OCI420 should flash at regular intervals.

Further on to chapter 4
Back to TOC

4. Installation of the Arduino IDE and Configuration of the Adapter

Back to TOC
Back to chapter 3

4. Installation of the Arduino IDE and Configuration of the Adapter

Note: The following description is for the *Arduino Due!* If you want to install BSB-LAN on an *ESP32* , please see the chapter 12.2!

- Download and install the latest version of the Arduino IDE from https://www.arduino.cc/en/Main/Software for your OS.
- Connect the Arduino Due (plus installed LAN shield and BSB-LPB-LAN adapter!) via USB to your computer. Make sure that you are using the "Programming Port" of the Due!
- Download the latest version of BSB-LAN and extract the downloaded file *BSB-LAN-master.zip*.
- Enter the folder "BSB-LAN-master"/"BSB_LAN" and rename the file *BSB_LAN_config.h.default* to ***BSB_LAN_config.h*** !
- If you want to implement your own individual code, rename the file *BSB_LAN_custom.h.default* to ***BSB_LAN_custom.h*** !
- Open the BSB_LAN sketch by double clicking the file *BSB_LAN.ino*. The necessary files like *BSB_LAN_config.h* and *BSB_LAN_defs.h* will automatically loaded within.
- Switch to the tab "BSB_LAN_config.h" and configure the necessary parameters like IP address etc. corresponding to your network (if you don't want to use DHCP which is activated by default). Check if the IP you are typing in isn't already used by your router. You can also use DHCP though. Adjust the further settings of BSB-LAN in this file to your needs, e.g. logging, optional installed temperature sensors and so on. See chap. 5.2 for further informations.
- Make sure, that you are using the current Ethernet Library (min. v2). Therefore, open „Sketch“ → „Include Library“ → „Manage Libraries“ and check if an update or a newer version of the „Ethernet Library“ is available. If so, update to that version or install the newer one.
- Now select "Arduino Due (Programming Port)" in "Tools/Board" in the main menu of the Arduino IDE.
  If the board doesn't appear in the list, you have to add the Atmel SAM Core to it. Simply choose Tools/Board/Boards Manager, search for 'Arduino SAM Boards' where the Due is included, click on it and then hit the 'Install' button. After doing that you will find the Arduino Due in Tools/Board.
- Select the correct serial port in "Tools/Serial Port".
- If you are using Windows, you probably have to install further drivers. Please see https://www.arduino.cc/en/Guide/ArduinoDue for further informations.
- Upload/flash the sketch to your Arduino by selecting "Sketch/Upload".
- Connect the Arduino with a LAN cable with your router/switch. Make sure that a you have a working power supply attached or that the Arduino is powered by USB (use the "Programming Port").
- Open the page `http://<chosen-ip-address>/` (or `http://<chosen-ip-address>/<passkey>/` if you are using the optional passkey feature).
  Now the landing page of the BSB-LAN webinterface should appear. If not, reboot the Arduino by pressing the reset button on it and try again after a while.
  You can check your configuration of BSB-LAN by querying the page `http://<chosen-ip-address>/C`.

After you configured BSB-LAN by adjusting the file BSB_LAN_config.h to your needs and access to the webinterface of BSB-LAN was successful, you can now continue with checking the function of the adapter.

Note: Once the adapter is connected to the bus of the controller of the heating system, you can let it be connected if you want to flash the Due again. There's no need to disconnect it if you want to update BSB-LAN.

Further on to chapter 5
Back to TOC

5. Configuration of the BSB-LAN Software v2.x

Back to TOC
Back to chapter 4

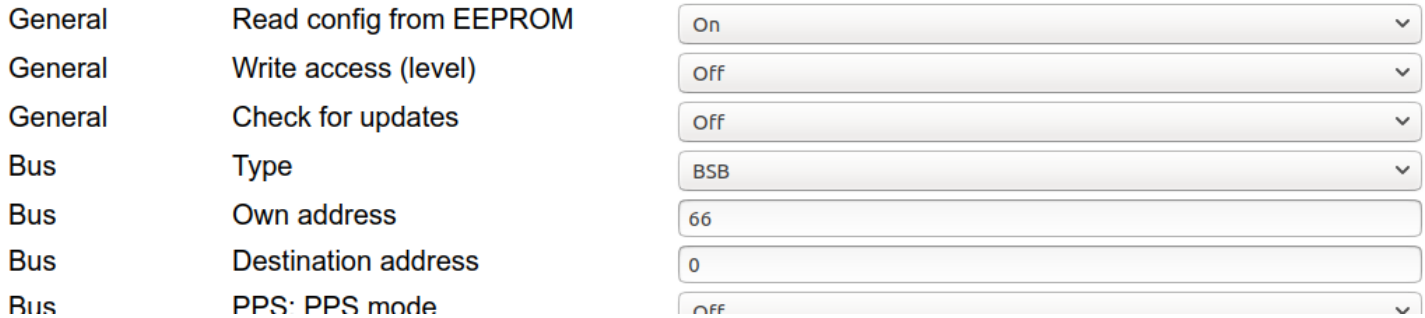
5. Configuration of the BSB-LAN Software v2.x

Note: Since v2.x it is now also possible to make almost the complete configuration of the BSB-LAN software via the web interface (menu "Settings").

5.1 Configuration via Webinterface

The settings overview or the web configuration interface is in principle self-explanatory, nevertheless the individual points are listed here again with a short explanation.
For a more detailed explanation of the individual functions, please refer to chap. 5.2.

The overview of the web configuration is divided into three columns:



- For the sake of clarity, a rough category is displayed in the left column (e.g. "General", "Bus", etc.), so that the assignment of the respective entry is already apparent at first glance.
- In the middle column the function is named.
- In the right column is the corresponding field, which shows the current entry or setting. The entries from the file *BSB_lan_config.h* are taken over, that means that also with deactivated functions the default settings are visible, so that it becomes clear, how (e.g.) parameters should be entered. Depending on the type of setting either a pull down menu with the available settings or only a field is displayed.

Important:
To apply changed settings, you must finally click on the button "Save parameter" at the bottom!

In the following, the tabular overview of the functions with the (default) settings and the corresponding explanations (unfortunately, the naming of the left column "Category" must be omitted here for reasons of space and presentation):

Function	(default) Setting	Explanation
Read config from EEPROM	On	Reads the stored configuration from the EEPROM when starting the Due (Off/On). These settings can deviate from the default settings, which were made in the file *BSB_lan_config.h*. *If the settings stored in the EEPROM should be overwritten, e.g. during an update, set to "Off" and save the setting before flashing!* If the setting is "Off", changes will only remain active until the Due is restarted.
Write access (level)	Off	Write access of the adapter to the heating controller (Off/Standard/Complete). **If write access to the heating controller should be granted, it is recommended to select the 'Standard' setting, in which case almost all available parameters can be written.** In contrast to 'Complete', however, some function-critical parameters cannot be changed, as they are protected again inside the controller. *The setting 'Complete' should therefore only be selected in exceptional cases and with caution and a very good knowledge of the controller functionality!*

Function	(default) Setting	Explanation
Check for updates	Off	Automatically check for updates of BSB-LAN (Off/On)
Type	BSB	Used bustype (BSB/LPB/PPS)
Own address	66	Own address of the adapter
Destination address	0	Destination address for queries
PPS: PPS mode	passive	PPS only: Users who use the adapter on the PPS interface must make two settings: First, the mode in which the bus is to be accessed (passive/as room unit) must be selected. When using a QAA room device, "passive" must be selected here. Then only the values that go via the bus are displayed in the web interface, writing of values is then not possible. If "as room unit" is selected here, values can also be sent to the heating system via the web interface. The type of the room device to be emulated must then still be selected (see below). *There should then be no other room device on the bus, otherwise both transmitters send their own values to the heater, so that no consistent operation is possible.*
PPS: QAA model	QAA70	PPS only: Type of the room unit that should be imitated (QAA50/QAA70).
URL Passkey	-no default setting-	Optional security function: "URL Passkey"
HTTP authentification	-no default setting-	Optional security function: "User-Pass" (Basic HTTP Auth). Syntax: Username:Password
DHCP usage	On	DHCP usage (= automatic allocation of the IP address by the router) (Off/On)
IP address (fixed)	192.168.178.88	Manual network configuration: fixed IP address
Subnet	255.255.255.0	Manual network configuration: Subnet
Gateway	192.168.178.1	Manual network configuration: IP address of the gateway
DNS Server	192.168.178.1	Manual network configuration: IP address of the DNS server
TCP Port	80	TCP port of the setup
MAC address	00:80:41:19:69:90	(Preset) MAC address of the LAN shield or MAC address of the ESP
Trusted IP address	0.0.0.0	Optional security function: "Trusted IP", access is only possible from this IP
Trusted IP address	0.0.0.0	Optional security function: "Trusted IP", access is only possible from this IP
WLAN SSID	-no default setting-	SSID of the WLAN when using the WiFi-ESP-solution
WLAN password	-no default setting-	Password of the WLAN when using the WiFi-ESP-solution
Usage	Off	Use MQTT function (Off/On)
IP address broker	192.168.178.20	IP-Adresse des MQTT-Brokers
Username	User	MQTT: Username when using username/password
Password	Pass	MQTT: Passwort when using username/password
Device ID	MyHeater	Device name (header in JSON payload)
Topic prefix	BSB-LAN	Topic prefix of the MQTT messages

Function	(default) Setting	Explanation
Calculation	Off	Calculation of 24h average values of selected parameters (Off/On)
Parameter	8700,8326	Parameters for the 24h average calsulation
Bus telegrams	Off	Logging of bus telegrams activated (Off/-various options-), the desired setting is to be made according to the respective option description.
To SD card	Off	Save values to be logged on the microSD card (Off/On)
Interval (seconds)	3600	Loginterval in seconds
Parameters	8700,8743,8314	Parameters to be logged
Pins	7	Used pin(s) for OneWire sensors (DS18B20)
Pins	2,3	Used pin(s) for DHT22 sensors
DHW push button: pin	0	Room unit emulation: used pin for the DHW push
RU1 temperature sensor parameter	-no default setting-	Room unit 1 emulation: enter the specific parameter number(s) for the optional room temperature sensor(s) here. Up to five sensors are possible, parameter numbers must be separated only with a comma. If more than one sensor is used, an automatic average will be calculated.
RU1 presence button: pin	0	Room unit 1 emulation: used pin for the presence button for HC1
RU2 temperature sensor parameter	-no default setting-	Room unit 2 emulation: enter the specific parameter number(s) for the optional room temperature sensor(s) here. Up to five sensors are possible, parameter numbers must be separated only with a comma. If more than one sensor is used, an automatic average will be calculated.
RU2 presence button: pin	0	Room unit 2 emulation: used pin for the presence button for HC2
RU3 temperature sensor parameter	-no default setting-	Room unit 3 emulation: enter the specific parameter number(s) for the optional room temperature sensor(s) here. Up to five sensors are possible, parameter numbers must be separated only with a comma. If more than one sensor is used, an automatic average will be calculated.
RU3 presence button: pin	0	Room unit 3 emulation: used pin for the presence button for HC3
Usage	Off	Use MAX! devices (Off/On)
IP address cube	192.168.178.5	IP address of the CUNO/CUNX/modified MAX!Cube
Devices	KEQ0502326,KEQ0505080	Serial numbers of the MAX! devices to be used
Usage	Off	Use IPWE extension (URL/ipwe.cgi) (Off/On)
Parameters	8700,8743,8314	Parameters that should be displayed within the IPWE extension
Usage	Serial	Use debug function (Off/Serial/Telnet)
Verbosity mode	On	Verbosity mode activated (Off/On)
Monitor mode	Off	Monitor mode activated (Off/On)

5.2 Configuration by Adjusting the Settings Within *BSB_LAN_config.h*

The BSB-LAN software can be configured by adjusting the settings in the file *BSB_LAN_config.h*. All settings are listed below in the same way as

they are listed and preset in the file. It is therefore advisable to work through the settings point by point with this manual at hand.

Note:
To 'activate' or a definement you have to delete the two slashes in front of the hashtag, to 'deactivate' a definement you have to add two slashes in front of the hashtag. E.g.:
A deactivated definement: `//#define XYZ`
An activated definement: `#define XYZ`

- The **language of the user interface** of the web interface of the adapter as well as the category and parameter designations must be selected or defined. For "English" the following definition must be selected: `#define LANG EN`
  Starting with BSB-LAN v.042 it is possible to use BSB-LAN in other languages, too, whereby in principle any language can be supported (only' the corresponding translations have to be created).
  Currently available are: Czech (CZ), German (DE), Danish (DK), English (EN), Spanish (ES), Finnish (FI), French (FR), Greek (GR), Hungarian (HU), Italian (IT), Dutch (NL), Polish (PL), Russian (RU), Swedish (SE), Slovenian (SI) and Turkish (TR). If certain expressions are not available in the specific language, the English expression is automatically displayed. If this is also not available, the German expression is finally displayed.

- **Load configuration settings from EEPROM or from the file *BSB_LAN_config.h*:**
  `byte UseEEPROM = 1;`
  According to the default setting, the configuration settings are read from the EEPROM when BSB-LAN is started. As a fallback the variable can be set to '0', then the settings are read from the file *BSB_LAN_config.h*.

Network settings:
Note: By default, the usage of DHCP is activated, so you don't have to change any network settings. If you want to use a fixed IP though, deactivate DHCP and set the IP and the addresses of the Gateway and the Subnet accordingly.

- **MAC address of the ethernet shield:**
  `byte mac[] = { 0x00, 0x80, 0x41, 0x19, 0x69, 0x90 };`
  The default MAC address can be kept. A change is usually only necessary if more than one adapter is used (in any case, you should make sure that each MAC address only occurs *once* in the network!). In this case, changes should only be made to the last byte (e.g. 0x91, if a second adapter is used).
  Important note:
  The MAC address assigned here also influences the host name (or is a part of it), which is assigned by the router when using DHCP (see below): The host name consists of the identifier "WIZnet" and the last three bytes of the MAC address.
  The MAC address which can be set here doesn't apply to the WiFi-ESP-solution! There the MAC address can't be set!

  For the default MAC address mentioned above, the host name is thus "WIZnet196990". This host name is usually also displayed as such in the router. In this case the web interface of BSB-LAN can be reached in the browser under `http://wiznet196990`.
  If a second adapter is used and the MAC address will be changed to
  `byte mac[] = { 0x00, 0x80, 0x41, 0x19, 0x69, 0x91 };`
  the host name is "WIZnet196991" or `http://wiznet196991`.

- **Ethernet port:**
  `uint16_t HTTPPort = 80;`
  Port 80 for HTTP is preset.

- **DHCP:**
  `bool useDHCP = true;`
  By default DHCP is used. If this is not desired and you want to assign a fixed IP address by yourself, set the variable to *false*.

  Important note:
  Please see the notes above regarding the hostname based on the MAC address. The IP given by the router will also appear within the start process of the Arduino Due within the serial monitor of the Arduino IDE.

- **IP address:**
  `byte ip_addr[4] = {192,168,178,88};`
  Fixed IP address of the adapter, if DHCP is not used - *please note the commas instead of dots!*
  Note: If you want to give the adapter a fixed IP, please make sure that it occurs only once in your network!

- **Gateway address:**
  `byte gateway_addr[4] = {192,168,178,1};`
  IP address of the gateway (usually the one of the router itself) - *please note the commas instead of dots!*

- **Subnet:**
  `byte subnet_addr[4] = {255,255,255,0};`
  Address of the subnet - *please note the commas instead of dots!*

- **WiFi by additional ESP8266:**
  `//#define WIFI`
  This definement has to be activated if the WiFi function of the ESP8266-WiFi-solution or the ESP32 should be used.

  `char wifi_ssid[32] = "YourWiFiNetwork";` For the usage of WiFi, *YourWiFiNetwork* has to be replaced by the SSID of the WiFi network.

  `char wifi_pass[64] = "YourWiFiPassword";`
  For the usage of WiFi, *YourWiFiPassword* has to be replaced by the password of the WiFi network.

  `#define WIFI_SPI_SS_PIN 12`
  The SS pin to be used at the DUE when using the ESP8266-WiFi-solution is defined here. It is advisable to leave the default setting. If, however, another pin should be used, it is essential to ensure that the desired pin is neither used elsewhere nor is included in the list of protected pins.

  Note: The MAC address can't be set within the WiFi-ESP-solution!

- **Using Multicast DNS:**
  `#define MDNS_HOSTNAME "BSB-LAN"`
  By default the usage of Multicast DNS with the hostname "BSB-LAN" is activated, so that you can find the adaptersetup under this name within your network.
  Please note: mDNS is only available when using LAN, it is not available if you are using the WiFi solution using an ESP8266!

- **Debugging and related settings:**
  - `#define DEBUG` → the debug module will be compiled (activated by default)
  - `byte debug_mode = 1;` → The following debug options are available:
    0 - debugging deactivated
    1 - send debug messages to the serial interface (e.g. for using the aerial monitor of the Arduino IDE); default setting
    2 - send debug messages to a TelNet client instead of the serial interface
  - `byte verbose = 1;` → By default the verbose mode is activated (= 1), so that (besides the raw data) the respective plaintext (if available) of parameters and values is displayed. It is advisable to leave this setting as it facilitates possible trouble shooting. Furthermore, this setting is necessary if telegrams and command IDs of new parameters should be decoded.
  - `byte monitor = 0;` → Bus monitor mode, deactivated by default; set to '1' to activate
  - `bool show_unknown = true;` → All parameters including the *unknown parameters* (error message "error 7 (parameter not supported)") are displayed when querying via web interface (e.g. when querying a complete category); default setting.
    If you want to hide the 'unknown' parameters that are not supported by the controller of your heating system (e.g. when querying a complete category), you have to set the variable to 'false' (`bool show_unknown = false;`). *The parameters are still queried in such a query (e.g. for a complete category) though.*

Security functions:
There are several options to control and protect access to your heating system. However, keep in mind, that even activating all three options are no guarantee that a versatile intruder with access to your (W)LAN won't be able to gain access. In any case, no encryption of data streams is provided by the Arduino itself. Use VPN or a SSL proxy if that is a must for you and connect the Arduino wired to the VPN server or SSL proxy. The following three security options are available within BSB-LAN:

- **Passkey:**
  To protect the system from unwanted access from outside, the **function of the security key (PASSKEY)** can be used (very easy and not

really secure!): `char PASSKEY[64] = "";`

To use this function, add a certain sequence of alphanumerical characters as a simple security function, e.g. `char PASSKEY[64] = "1234";` → in this example the passkey is '1234'. If no alphanumerical sequence is set (default), the passkey function remains deactivated.

Note:
If PASSKEY is defined, the URL has to contain the defined passkey as first element, e.g.: `URL/1234/` to view the main website (don't forget the trailing slash!). Only within the URL of the optional IPWE extension the passkey has NOT to be added!

- **Trusted IP:**
  `byte trusted_ip_addr[4] = {0,0,0,0};`
  `byte trusted_ip_addr2[4] = {0,0,0,0};`

  Within these variables you can define up to two IP addresses from which the access to BSB-LAN will then be possible (e.g. sever of your home automation system).
  If the default setting will not be changed or if the first number is a '0', this function is deactivated (default setting).

- **User-Pass:**
  `char USER_PASS[64] = "";`
  Provides a (base64-coded) username:password based access (default setting: deactivated). No encryption! Syntax is Username:Password as shown in the deactivated example:
  `//char USER_PASS[64] = "User:Password";`

Settings for optional sensors:

- **OneWire temperature sensors (DS18B20):**
  `#define ONE_WIRE_BUS`
  `bool enableOneWireBus = false;`
  `byte One_Wire_Pin = 7;`

  If you want to use OneWire temperature sensors (DS18B20), the definition must be activated, the variable must be set to *true* and the corresponding pin (DATA connection of the sensor on the adapter board / Arduino Due) must be defined. *Note: Make sure that you don't use any of the protected pins which are listed further down below!*
  By default, the module is activated, the variable is set to *false* (= no usage) and pin 7 for DATA is set.

- **DHT22 sensors:**
  `#define DHT_BUS`
  `byte DHT_Pins[10] = {5};`

  If you want to use DHT22 sensors (temperature & humidity), the definement must be activated and the corresponding pin(s) must be be defined. *Note: Make sure that you don't use any of the protected pins which are listed further down below!*
  By default, the module is activated and the pin 5 is set for the DATA of one sensor (note: each sensor has to be connected to a different pin).

- **BME280 sensors:**
  `//#define BME280 1`

  If you want to use BME280 sensors (temperature, humidity & barometric pressure), the definement must be activated and the corresponding amount of sensors (default 1, maximum 2!) must be be defined. The sensors have to be connected to the I2C bus. The address of the first sensor mus be 0x76, the one of the second sensor 0x77.

- **24h averages:**
  `#define AVERAGES`
  If you want to create 24h averages from certain parameters, the definement must be activated (default setting).

  `bool logAverageValues = false;` If you want the averages to be logged to a microSD card within the file *averages.txt*, you need to change the default setting and change the variable to `true`. If you don't want to have these values logged, the variable must be set to `false` as per default.

  Further more you have to list the specific numbers of the parameters you want to be calculated. E.g.:

```
int avg_parameters[40] = {
8700,    // outside temperature
8830    // DHW (warm water) temperature
};
```

- **Logging (also to microSD card) and/or usage of MQTT:**
  `#define LOGGER` → The logging module will be compiled.
  Note: This is a requirement for logging to a microSD card as well as for using MQTT!

  In the following, various settings can/should be made:

  - If you are using a microSD card adapter on an ESP32-based board and want to log data to the card instead of the SPIFFs flash storage, activate the following definement:
    `//#define ESP32_USE_SD`
  - If 'raw' *bus telegrams* should be logged, the selection can be specified. The telegrams are stored within the file *journal.txt* on the microSD card. By default the logging of these bus messages is deactivated: `int logTelegram = LOGTELEGRAM_OFF;`

  The following options are available:
  `LOGTELEGRAM_OFF` → no logging of bus telegrams (default setting)
  `LOGTELEGRAM_ON` → all bus telegrams are logged
  `LOGTELEGRAM_ON + LOGTELEGRAM_UNKNOWN_ONLY` → only unknown bus telegrams are logged
  `LOGTELEGRAM_ON + LOGTELEGRAM_BROADCAST_ONLY` → only broadcast telegrams are logged
  `LOGTELEGRAM_ON + LOGTELEGRAM_UNKNOWNBROADCAST_ONLY` → only unknown broadcast telegrams are logged

  - `bool logCurrentValues = false;`
    The data of the parameters to be logged are stored in the file 'datalog.txt' on the microSD card (deactivated by default). For activating this function the variable must be set to 'true'.
  - `unsigned long log_interval = 3600;`
    The desired logging interval in seconds.
    Note: This interval must also be set for using MQTT, even though if no data should be logged!

  The parameters that should be logged must be listed:

```
int log_parameters[40] = {
8700,    // outside temperature
8830    // DHW (warm water) temperature
};
```

- **MQTT:**
  If you want to use MQTT the belonging varaibles and settings have to be adjusted:

  - `define MQTT` → The MQTT module will be compiled (default setting).
  - `byte mqtt_mode = 0;` → MQTT is deactivated (default setting); the following options are available:
    1 = send messages in plain text format
    2 = send messages in JSON format. Use this if you want a json package of your logging information printed to the mqtt topic
    Structure of the JSON payload: {"MQTTDeviceID": {"status":{"log_param1":"value1","log_param2":"value2"}, ...}}
    3 = send messages in rich JSON format. Use this if you want a json package of your logging information printed to the mqtt topic
    Structure of the rich JSON payload: {"MQTTDeviceID": {"id": one_of_logvalues, "name": "program_name_from_logvalues", "value": "query_result", "desc": "enum value description", "unit": "unit of measurement", "error", error_code}}
  - `byte mqtt_broker_ip_addr[4] = {192,168,1,20};` → IP of the MQTT broker (standard port 1883). *Please note the commas insted of dots!*
  - `char MQTTUsername[32] = "User";` → Set username for MQTT broker here or set zero-length string if no username/password is used.
  - `char MQTTPassword[32] = "Pass";` → Set password for MQTT broker here or set zero-length string if no password is used.
  - `char MQTTTopicPrefix[32] = "BSB-LAN";` → Optional: Choose the "topic" for MQTT messages here. If zero-length string here, default topic name used.

- `char MQTTDeviceID[32] = "MyHeater";` → Optional: Define a device name to use as header in json payload. If zero-length string here, "BSB-LAN" will be used.

  Important Note:
  The parameters that should be queried and the interval for sending the values must be defined within the ***logger definement*** *as mentioned above.*

- **IPWE:**
  `#define IPWE` → The ipwe module will be compiled.
  `bool enable_ipwe = false;`
  By default, the usage of the ipwe extension (URL/ipwe.cgi) is deactivated. If you want to use it, set the variable to 'true'.
  Define the parameters that should be displayed (max 40):

```
int ipwe_parameters[40] = {
8700,    // outside temperature
8830    // DHW (warm water) temperature
};
```

- **MAX! (CUNO/CUNX/modified MAX!Cube):**
  If you want to use MAX! thermostats, adjust the following settings:
  - `//#define MAX_CUL` → activate the definement (deactivated by default)
  - `bool enable_max_cul = false;` → set the variable to 'true' (default value: 'false')
  - `byte max_cul_ip_addr[4] = {192,168,178,5};` → Set the IP address of the CUNO/CUNX/modified MAX!Cube - *please note the commas instead of dots!*
    - Define the MAX! thermostats that should be queried (max 20) by entering the 10 digit serial number / MAX! ID:

```
char max_device_list[20][11] = {
"KEQ0502326",
"KEQ0505080"
};
```

  See chapter 12.5 for further informations about MAX! components.

- Define the number of retries for the query command (default value is 3, doesn't need to be changed usually):
  #define QUERY_RETRIES 3

Settings of the bus pins and bus type:

- **RX/TX pinconfiguration:**
  `byte bus_pins[2] = {0,0};` → automatic detection and selection of the used pins (RX,TX); possible options:
  - Hardware serial (since adapter v3 & Arduino Due): RX = 19, TX = 18 (`{19,18}`)
  - Software serial (up to adapter v2 & Arduino Mega 2560): RX = 68, TX = 69 (`{68,69}`)
- **Bus type / protocol:**
  `uint8_t bus_type = 0;` → Depending on the connection of the adapter to the controller of your heating system (BSB/LPB/PPS), the corresponding bus type must be set (default value is 0 = BSB). Possible options:
  0 = BSB
  1 = LPB
  2 = PPS
- **Bus settings:**
  Depending on the bus type, you can/must adjust certain settings:
  - **BSB:**
    `byte own_address = 0x42;` → sets own address of the BSB-LAN adapter; default setting is '0x42' = 66, which is 'LAN' in serial monitor

`byte dest_address = 0x00;` → destination address of the heating system; preset: 0 See chap. 2.1.1 for further informations.

- **LPB:**
  `byte own_address = 0x42;` → own address of the BSB-LAN adapter; preset: segment 4, device 3
  `byte dest_address = 0x00;` → destination address of the heating system; preset: segment 0, device 1
  See chap. 2.1.2 for further informations.

- **PPS:**
  `bool pps_write = 0;` → Readonly access (default setting); if you want to enable writing to the controller of the heating system, set the variable to '1'. *Note: Only enable writing if there is no other 'real' room unit such as QAA50/QAA70!*
  `byte QAA_TYPE = 0x53;` → type of the room unit which should be imitated; 0x53 = QAA70 (default setting), 0x52 = QAA50

- **Protected GPIO pins:**
  Usually there is no need to change these settings if the standard configuration of the BSB-LAN ahrdware is used. However, if you can adjust these settings though, please refer to the listing within the file *BSB_lan_config.h*.

- **Detection or fixed setting of the controller type of the heating system:**
  `static const int fixed_device_family = 0;`
  `static const int fixed_device_variant = 0;`
  By default, the automatic detection of the controller type is active. Usually there is no need to change this setting. However, you can set the type manually though, but you should *only* change this if you *really* know what you are doing! In that case set the variables of `fixed_device_family` and `fixed_device_variant` to your device family and variant (parameters 6225 and 6226).

- **Read/write access to the controller:**
  `#define DEFAULT_FLAG FL_SW_CTL_RONLY`
  By default, only read-access to the controller of the heating system is granted for the BSB-LAN adapter. If you want to make all parameters writeable / settable, then you can adjust this setting within the webinterface of BSB-LAN (menu "settings").
  Note for Mega-user:
  The possibility to configure BSB-LAN via the webinterface doesn't exist within the usage of the Mega 2560, because the module WEBCONFIG can't be compiled and used due to the limited memory of the Mega. In this case you still have to grant write access by setting the flag '0': `#define DEFAULT_FLAG 0`

- **Include own code:**
  `//#define CUSTOM_COMMANDS`
  This includes commands from the file *BSB_lan_custom.h* to be executed at the end of each main loop (deactivated by default).

- **Check for Updates of BSB-LAN:**
  `#define VERSION_CHECK`
  `bool enable_version_check = false;`
  Check for new versions when accessing BSB-LAN's main page (internet access needed). Doing so will poll the most recent version number from the BSB-LAN server. This function is deactivated by default; to activate this function, set the variable to 'true'.

  Note: In this process, it is unavoidable that your IP address will be transferred to the server, obviously. We nevertheless mention this here because this constitutes as 'personal data' and this feature is therefore disabled by default. Activating this feature means you are consenting to transmitting your IP address to the BSB-LAN server where it will be stored for up to two weeks in the server's log files to allow for technical as well as abuse analaysis. No other data (such as anything related to your heating system) is transmitted in this process, as you can see in the source code.

- **"External" webserver:**
  `//#define WEBSERVER`
  Usage of the "external" web server if definement is active. Please see chapter 8.2.10 for further informations.

- **Store configuration in EEPROM:**

`#define CONFIG_IN_EEPROM`
Stores the configuration in the EEPROM. If you don't want to use this function, deactivate the definement.

- **Compile web-based configuration and EEPROM config store module extension:**
  `#define WEBCONFIG`
  Activates the configuration via webinterface.

- **Compile JSON-based configuration and EEPROM config store module extension.**
  `#define JSONCONFIG`

- **Variables for future use, no function yet (November 2020):**

  `#define ROOM_UNIT` → compile room unit replacement extension
  `byte UdpIP[4] = {0,0,0,0};` → destination IP address for sending UDP packets to
  `uint16_t UdpDelay = 15;` → interval in seconds to send UDP packets

  `#define OFF_SITE_LOGGER` → compile off-site logger extension
  `byte destinationServer[128] = "";` → URL string to periodically send values to an off-site logger
  `uint16_t destinationPort = 80;` → port number for abovementioned server
  `uint32_t destinationDelay = 84600;` → interval in seconds to send values

For users of the outdated setup based on an Arduino Mega 2560:
Due to the lack of memory of the Mega 2560 it is neccessary to deactivate certain modules (or functions). Please also see *appendix d* *for further informations.*

- **Disabling functions:**

  If you use CONFIG_IN_EEPROM and WEBCONFIG modules then you can enable I_DO_NOT_WANT_URL_CONFIG for saving flash memory (~1.2Kb). This will disable configuration through URL commands (/A, /L, /P).
  `#define I_DO_NOT_WANT_URL_CONFIG`

  Enable I_WILL_USE_EXTERNAL_INTERFACE for saving flash memory (~6,8Kb). /DG command will be disabled.
  `#define I_WILL_USE_EXTERNAL_INTERFACE`

  Enabling I_DO_NOT_NEED_NATIVE_WEB_INTERFACE will eliminate native web interface and save up to 13 Kb of flash memory. /N[E] and /Q command still work. You can use this if you are using third-party software for BSB-LAN management. Do not forget to enable other required modules (JSONCONFIG, MQTT, WEBSERVER).
  `#define I_DO_NOT_NEED_NATIVE_WEB_INTERFACE`

- **Disabling modules:**

  If you want to try this version of BSB-LAN to run on an Arduino Mega 2560, you can change the modules which should be compiled so that they fit your needs and the Mega's memory constraints.
  Note: This overwrites any definements above.

```
#if defined(__AVR__)
//#undef CONFIG_IN_EEPROM
//#undef WEBCONFIG
#undef JSONCONFIG
//#undef WEBSERVER
#undef AVERAGES
#undef DEBUG
#undef IPWE
#undef MQTT
#undef MDNS_HOSTNAME
#undef OFF_SITE_LOGGER
#undef ROOM_UNIT
#undef VERSION_CHECK
#undef MAX_CUL
#undef BME280
#endif
```

Further on to chapter 6
Back to TOC

6. Examining the Correct Functionality and First Usage of the Adapter

Back to TOC
Back to chapter 5

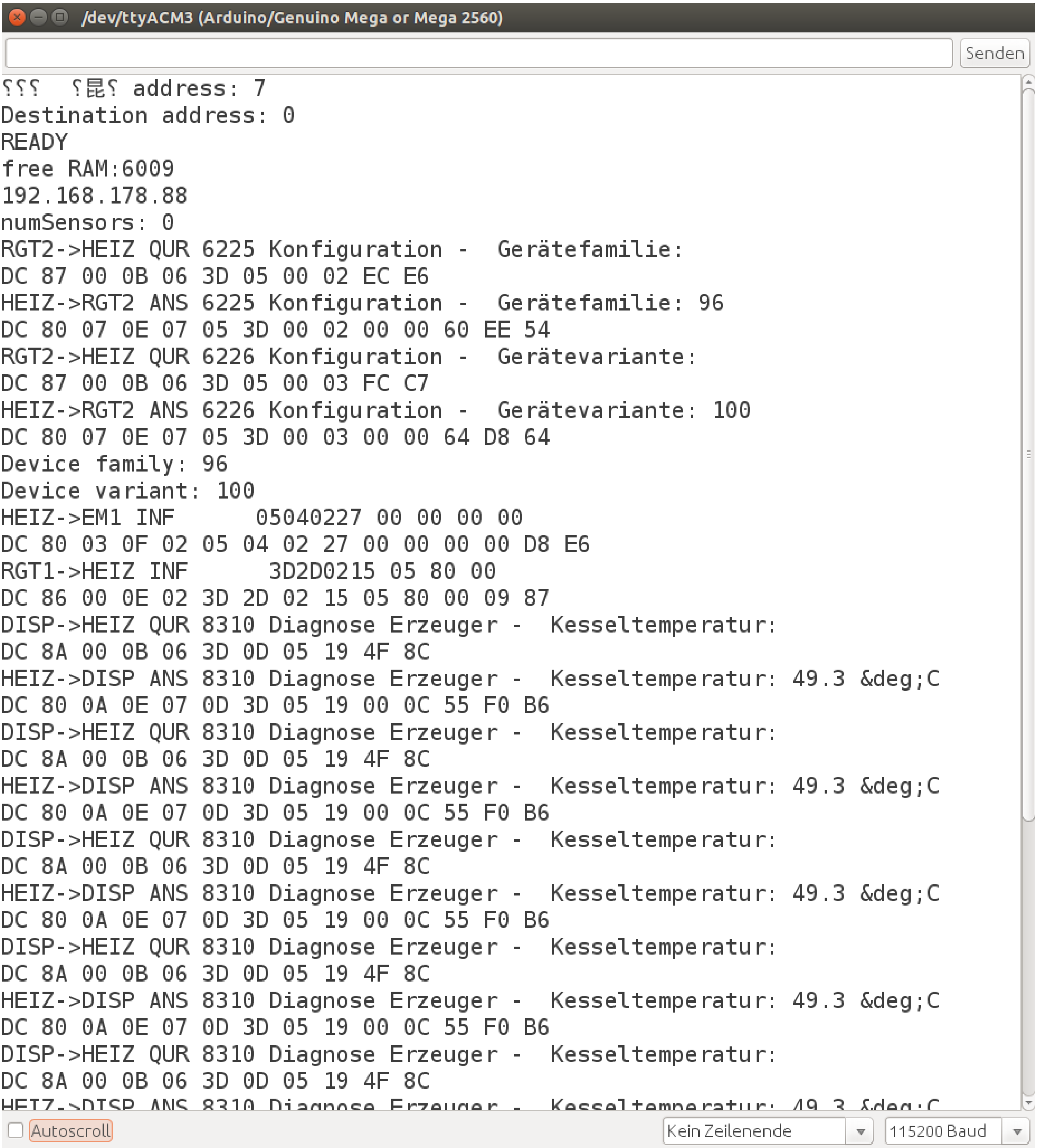
6. Examining the Correct Functionality and First Usage of the Adapter

To check if the adapter works correctly and recognizes your controller automatically, it's adviseable to follow these steps:

1. Switch off the controller of the heater and connect the adapter at the right pins to the BSB (or LPB / PPS). Watch the polarity!
2. Switch the controller back on and check if the red LED at the adapter is lit. If you see the LED flackering a little bit from time to time then that's no malfunction - it schows activity on the bus.
3. Connect the Arduino Due (of course fully assembled with the lan shield and the adapter) via USB (use the "Programming Port" in the center) with your computer and via LAN with your network.
4. Now start the Arduino IDE, choose the right COM port where the Arduino is connected to and start the serial monitor (menu "Tools" or the little magnifying glass symbol at the top right corner).
5. If the connected controller has successfully been detected automatically by BSB-LAN it should appear an output in the serial monitor where the value/number behind "Device family" and "Device variant" is NOT 0.
   A correct output looks like that (with different numbers due to a different controller type):

```
[...]
Device family: 96
Device variant: 100
[...]
```

The following screenshot shows an output of the serial monitor right after the start (and a little runtime). The adapter is configured as room unit 2 (RGT2) and queries the parameters 6225 and 6226 initially for autodetection of the controller. The following lines already are telegrams. The display of the operating unit of the controller shows the temperature of the boiler unit (here: "Kesseltemperatur") which comes in periodically as a so called broadcast message (BC).



```
/dev/ttyACM3 (Arduino/Genuino Mega or Mega 2560)
Senden
⸮⸮⸮  ⸮昆⸮ address: 7
Destination address: 0
READY
free RAM:6009
192.168.178.88
numSensors: 0
RGT2->HEIZ QUR 6225 Konfiguration -  Gerätefamilie:
DC 87 00 0B 06 3D 05 00 02 EC E6
HEIZ->RGT2 ANS 6225 Konfiguration -  Gerätefamilie: 96
DC 80 07 0E 07 05 3D 00 02 00 00 60 EE 54
RGT2->HEIZ QUR 6226 Konfiguration -  Gerätevariante:
DC 87 00 0B 06 3D 05 00 03 FC C7
HEIZ->RGT2 ANS 6226 Konfiguration -  Gerätevariante: 100
DC 80 07 0E 07 05 3D 00 03 00 00 64 D8 64
Device family: 96
Device variant: 100
HEIZ->EM1 INF      05040227 00 00 00 00
DC 80 03 0F 02 05 04 02 27 00 00 00 00 D8 E6
RGT1->HEIZ INF      3D2D0215 05 80 00
DC 86 00 0E 02 3D 2D 02 15 05 80 00 09 87
DISP->HEIZ QUR 8310 Diagnose Erzeuger -  Kesseltemperatur:
DC 8A 00 0B 06 3D 0D 05 19 4F 8C
HEIZ->DISP ANS 8310 Diagnose Erzeuger -  Kesseltemperatur: 49.3 &deg;C
DC 80 0A 0E 07 0D 3D 05 19 00 0C 55 F0 B6
DISP->HEIZ QUR 8310 Diagnose Erzeuger -  Kesseltemperatur:
DC 8A 00 0B 06 3D 0D 05 19 4F 8C
HEIZ->DISP ANS 8310 Diagnose Erzeuger -  Kesseltemperatur: 49.3 &deg;C
DC 80 0A 0E 07 0D 3D 05 19 00 0C 55 F0 B6
DISP->HEIZ QUR 8310 Diagnose Erzeuger -  Kesseltemperatur:
DC 8A 00 0B 06 3D 0D 05 19 4F 8C
HEIZ->DISP ANS 8310 Diagnose Erzeuger -  Kesseltemperatur: 49.3 &deg;C
DC 80 0A 0E 07 0D 3D 05 19 00 0C 55 F0 B6
DISP->HEIZ QUR 8310 Diagnose Erzeuger -  Kesseltemperatur:
DC 8A 00 0B 06 3D 0D 05 19 4F 8C
HEIZ->DISP ANS 8310 Diagnose Erzeuger -  Kesseltemperatur: 49.3 &deg;C
DC 80 0A 0E 07 0D 3D 05 19 00 0C 55 F0 B6
DISP->HEIZ QUR 8310 Diagnose Erzeuger -  Kesseltemperatur:
DC 8A 00 0B 06 3D 0D 05 19 4F 8C
HEIZ->DISP ANS 8310 Diagnose Erzeuger -  Kesseltemperatur: 49.3 &deg;C
Autoscroll    Kein Zeilenende    115200 Baud
```

Note:
If only weird character strings appear in the serial monitor, check the baud rate at the lower right corner of the serial monitor window. It should be set to 115200 baud.

If the connected controller hasn't been detected correct, the number behind "Device family" and "Device variant" will be a "0". Additionally to that six lines of "query failed" appear before the line "Device family".
This is how it would look like:

```
[...]
query failed
query failed
query failed
query failed
query failed
query failed
Device family: 0
Device variant: 0
[...]
```

In most cases there is a problem in the wiring or with certain components of the used harware or the adapter itself.

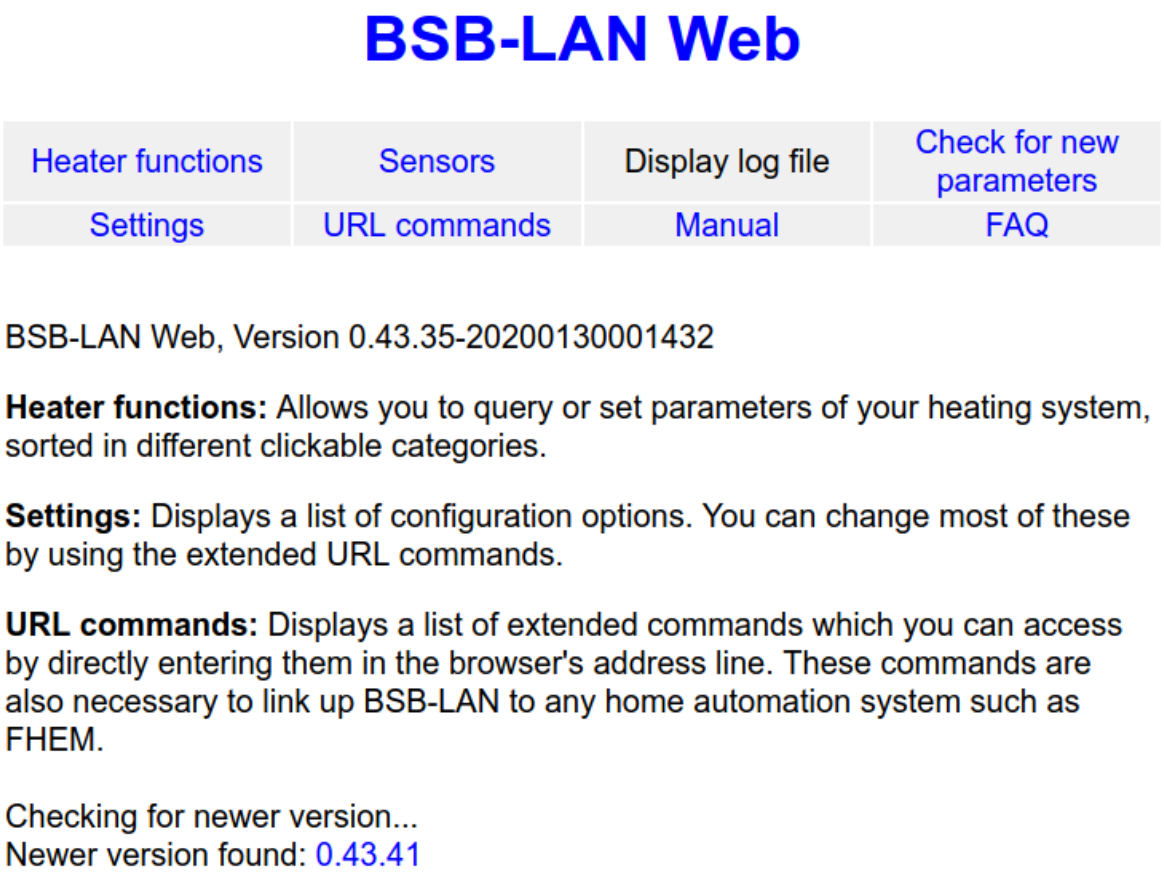
Further on to chapter 7
Back to TOC

7. BSB-LAN Web - the Webinterface of the Adapter

Back to TOC
Back to chapter 6

7. BSB-LAN Web - the Webinterface of the Adapter

By accessing the adapters IP (`http://<IP-address>`), the starting page of the webinterface "BSB-LAN Web" is displayed.
If you're using the passkey function (`http://<IP-address>/<passkey>/`) or additional security options, of course the URL has to be specifically expanded.



Within the webinterface there are some buttons at the top for an easy and direct access to certain functions:

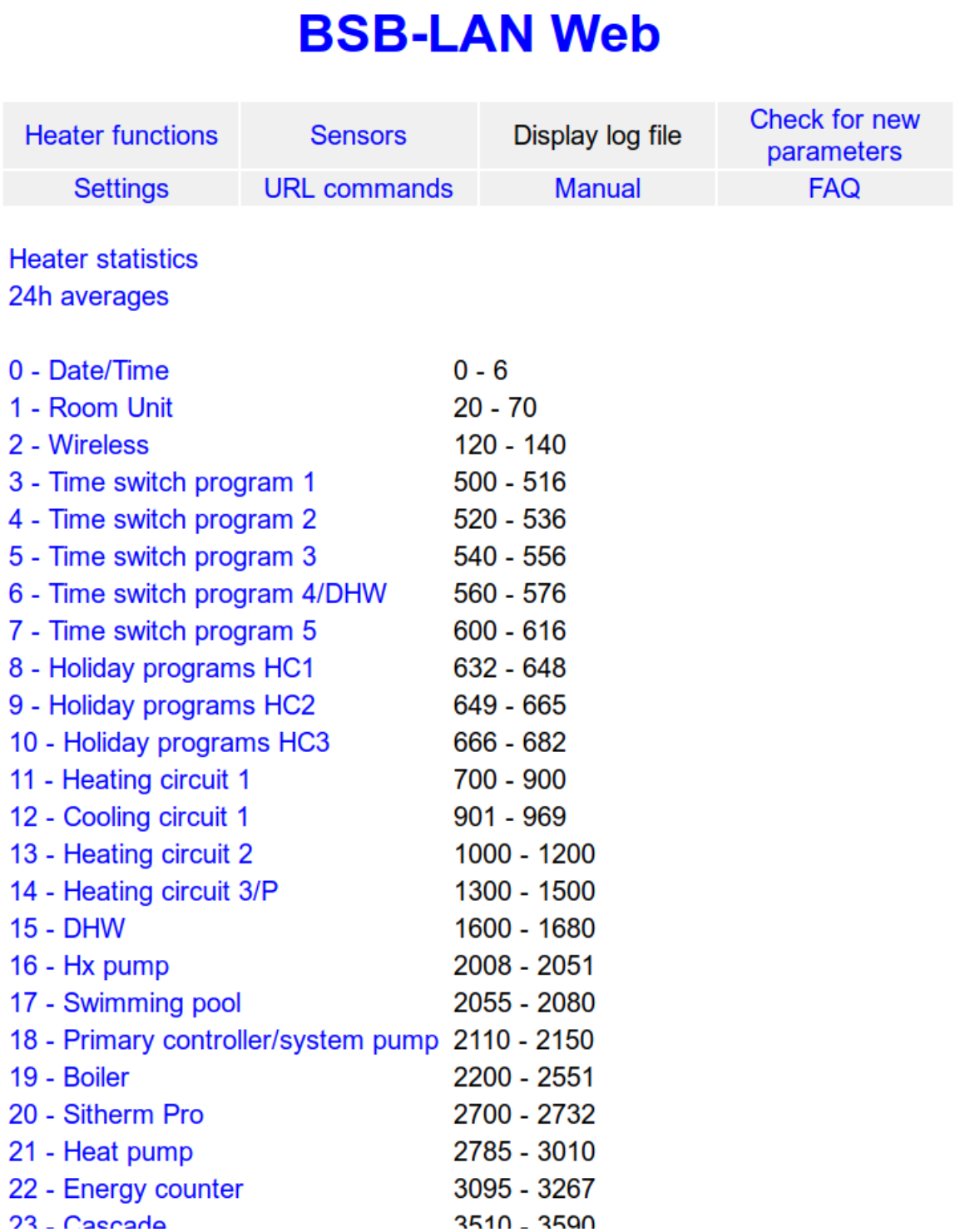
- Heater functions
- Sensors
- Display log file
- Check for new parameters
- Settings
- URL commands
- Manual
- FAQ

The button "Display log file" will be displayed in black letters, if the logging function isn't active (like shown in the screenshot above). If logging ist activated, the button is named "Plot log file".

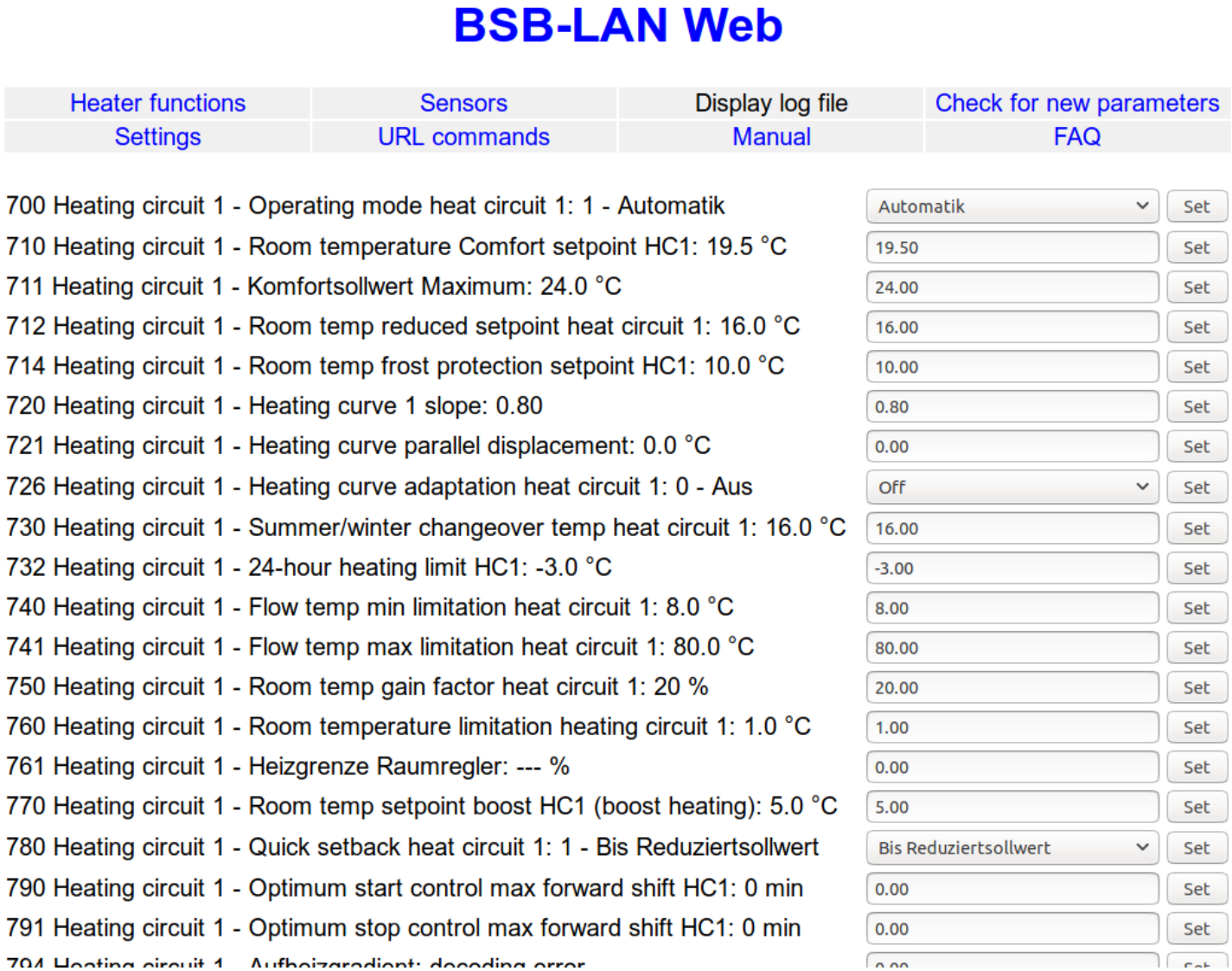
Underneath the header area the installed version of BSB-LAN is shown.
BSB-LAN checks by default if a newer version is available. If there is a newer version, the link leads to the ZIP file of the repo, so that you can save it directly from within the webinterface.

Note: If you don't want this function to be active because BSB-LAN connects automatically to the internet, you can deactivate it by uncommenting the belonging definement (`//#define VERSION_CHECK 1`) in the file BSB_lan_config.h.

Heater functions (URL command: /K):
The button "heater functions" displays a list of all categories within the supported controllers (therefore also categories which aren't supported by certain controller types).

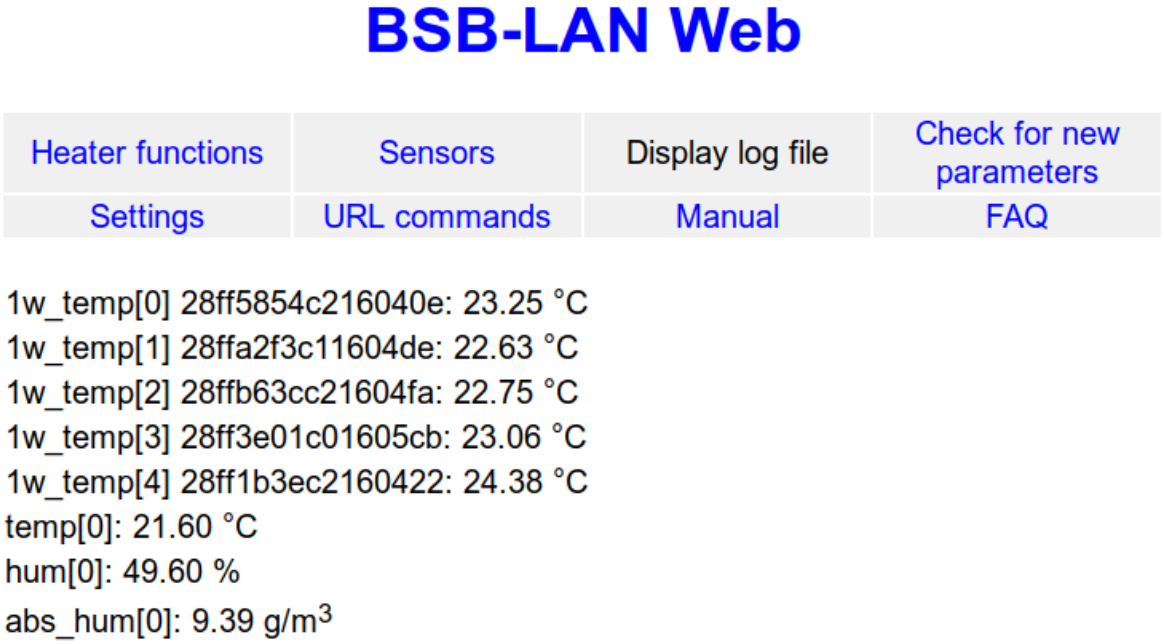


A click on the category name queries all supported parameters and displays them in the webinterface.



Sensors (URL command: /K49):

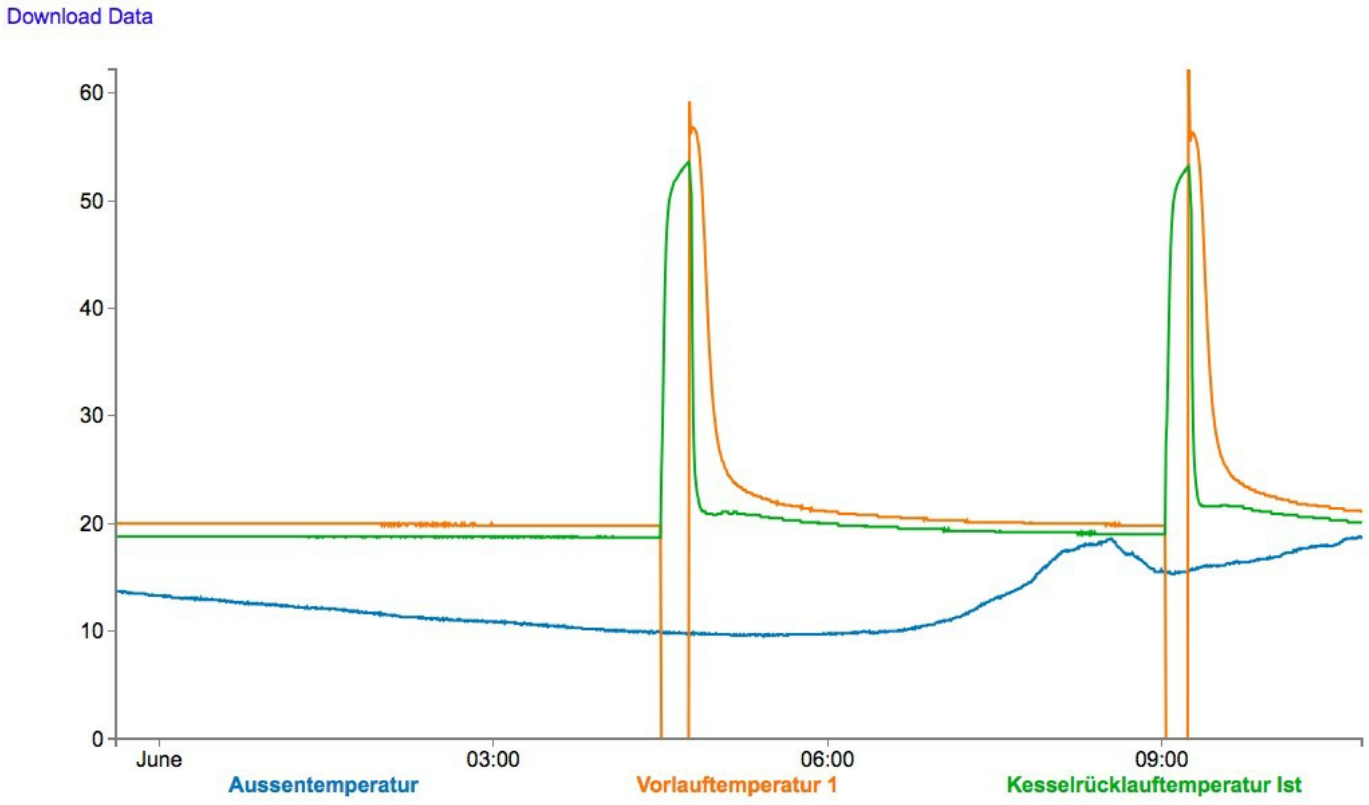
If optional sensors (DS18B20 / DHT22) are connected and configured in *BSB_lan_config.h*, the sensors will be listed after clicking this button.



DS18B20 sensors are named "1w_temp[x]" and are listed with their individual sensor ID.

DHT22 sensors show the temperature, humidity and absolute humidity.

Display/Plot log file (URL command: /D and /DG):
If the logging function to the microSD card is set and active, the belonging button is named "Plot log file". Once you click on it, the logfile (file *datalog.txt*) will be graphically displayed. If the logging function is deaktivated, the button is named "Display log file" and is shown in black letters. To display the logfile graphically it's neccessary to allow the JavaScriptFramework from d3js.org to work, so please don't use adblockers on that, if you want to use this function.



Check for new parameters (URL command: /Q):
This function queries all known parameters and checks, if any parameter would be supported by that special controller which isn't released yet.

BSB-LAN Web

Heater functions	Sensors	Display log file	Check for new parameters
Settings	URL commands	Manual	FAQ

Scanning for devices.....
Device address found: 0

Teste Geräteadresse 0:
Device family: 90
Device variant: 109
Geräte-Identifikation: RVS63.283/109
Device SW version: 3.4
Entwicklungs-Index:
Device OV version: 101.4
Bootloader-Version:
EEPROM-Version: 50.0
Konfiguration - Info 2 OEM:
Zugangscode Inbetriebnahme?:
Zugangscode Fachmannebene ?:
Zugangscode OEM?:
Zugangscode OEM2?:
Bisher unbekannte Geräteabfrage: 20
Hersteller-ID (letzten vier Bytes): 17680
Bisher unbekannte Geräteabfrage: 00010001F4 - unknown type
Outside temp sensor local (10003): 25.4 °C
Outside temp sensor local (10004): 25.4 °C

6225;6226;6224;6220;6221;6227;6229;6231;6232;6233;6234;6235;6223;6236;6237;
90 ;109 ;RVS63.283/109;3.4 ;;101.4 ;50.0 ;;;;;;20 ;17680;00010001F4 - unknown
type;

Start test...
Test finished.

Settings (URL command: /C):

It shows an overview of certain functions that have been set.

You get a quick overview of (e.g.) the used version of BSB-LAN, the uptime, the used bus type, the address, the readonly or read/write state of the adapter, about parameters that are set to log, protected GPIO pins and so on.

BSB-LAN Web

Heater functions	Sensors	Plot log file	Check for new parameters
Settings	URL commands	Manual	FAQ

Settings

Version: 0.43.35-20200130001432
Free memory: 3592 Bytes
Uptime: 10929
Bus system: BSB (66, 0) Read/Write
Monitor mode: 0
Verbosity level: 1
1-Wire bus pins: 3
DHT22 bus pins: 2
Protected GPIO pins: 0 1 4 10 11 12 13 18 19 20 21 22 23 50 51 52 53 62 63 64 65 66 67 68 69
MAC-Address: 00:80:41:19:69:90
Calculating 24h averages for the following parameters:
8700 - Outside temp sensor local

Logging the following parameters every 3600 seconds:
8700 - Outside temp sensor local
700 - Operating mode

URL commands:
The button leads to the chapter "Cheatsheet URL Commands" of this manual, where the URL commands are listed in a short overview. Internetaccess needed.

Manual:
The button leads to the table of content of this manual. Internetaccess needed.

FAQ:
The button leads to the chapter "FAQ" of this manual. Internetaccess needed.

Further on to chapter 8
Back to TOC

8. URL Commands and Special Functions

Back to TOC
Back to chapter 7

8. URL Commands and Special Functions

Because the webinterface basically is just set 'on top' to achieve access without further programs like FHEM or openHAB, it's possible to access the functions and parameters with external programs.

8.1 Listing and Description of the URL Commands

Note:
The values and parameters in the following list of the URL commands must be written without the brackets. E.g.: URL command `/<x>` for the simple query of parameter 8700 = `/8700`.

URL Command	Effect
`/<x>`	`Query value/setting of parameter <x>`
`/<x>!<addr>`	`Query value/setting of parameter <x> for destination address <addr>`
`/<x>/<y>/<z>`	`Query values/settings of parameters <x>, <y> and <z>`
`/<x>-<y>`	`Query values/settings of parameters <x> to <y>`
`/<x>!<addr>-<y>`	`Query values/settings of parameters <x> to <y> for destination address <addr>`
`/A=0`	`Disable 24h average calculation temporarily` `Disables the 24h average calculation temporarily (until the next reboot of the Arduino). For a complete deactivation, uncomment all parameters for that function in the file BSB_LAN_config.h.`
`/A=<x>,<y>,<z>`	`Change 24h average value calculation of parameters <x>, <y>, <z>` `During runtime up to 20 new parameters can be defined for the 24h average calculation. These parameters are kept until the next reboot of the Arduino.`
`/B0`	`Reset counter of accumulated burner-runtime and -cycles`
`/C`	`Display configuration of BSB-LAN`
`/D or /DD`	`Display logfile from the microSD-card` `Displays the logfile datalog.txt which contains the values of the logged parameters defined in the file BSB_LAN_config.h.`

`/DG`	`Graphical display of the logfile from microSD-card`
	`Shows graphical output (graphs) of the logged values.`
	`Note: If you use Javascript blockers, make sure you allow access to d3js.org, because the Arduino just loads the csv-file into the browser and the D3-framework converts the data.`

`/DJ`	`Display logfile journal.txt from the microSD-card` `Displays the logfile journal.txt which shows the content of received and transmitted telegrams. This log is useful for debugging and the search for unknown parameters. To use this function, you must enable the LOGGER module in the file BSB_LAN_config.h and set the first element of the log_parameters array to 30000.`
`/D0`	`Reset both logfiles & create new header` `This command deletes the content of the files datalog.txt and journal.txt and creates a new csv-header for datalog.txt. This command should be executed before first logging.`
`/DD0`	`Remove logfile datalog.txt only`
`/DJ0`	`Remove logfile journal.txt only`

URL Command	Effect
/E<x>	Display ENUM-values of parameter <x> At this command the adapter doesn't communicate with the controller, it's a software sided internal function of BSB-LAN. This command is only available for parameters of the type VT_ENUM, VT_CUSTOM_ENUM, VT_BITS and VT_CUSTOM_BITS.
/G<x>	GPIO: Query pin <x> Displays the actual state of GPIO pin <x>, where <y>=0 is LOW and <y>=1 is HIGH.
/G<x>=<y>	GPIO: Set pin <x> to HIGH (<y> = 1) or LOW (<y> = 0) Sets GPIO pin <x> to LOW (<y>=0) or HIGH (<y>=1). Reserved pins which shouldn't be allowed to be set can be defined previously at GPIO_exclude in the file BSB_lan_config.h.
/G<x>,I	GPIO: Query pin <x> while setting to INPUT If e.g. a coupling relay is connected to a GPIO pin and the state should just be queried, this command should be used. It sets the GPIO pin to input (default they are set to output) and keeps this as long until it's changed by using G<x>=<y>. After that, it's set to output again until the next "I" sets it to input again.
/I<x>=<y>	Send INF-message to parameter <x> with value <y> Some values can't be set directly, the controller gets these values by a TYPE_INF-message. As an example, the room temperature of 19.5°C should be transmitted: http://<ip-address>/I10000=19.5.
/JC=<x>,<y>,<z>	JSON: Query possible values for parameters <x>, <y> and <z> for ENUM type parameters The format of the returned data is the same as the command /JK=<x>. Unlike the /JQ command, it does not return the current parameter values.
/JI	JSON: Display configuration of BSB-LAN
/JK=<x>	JSON: Query all parameters of category <x>
/JK=ALL	JSON: List all categories with corresponding parameter numbers
/JL	JSON: Creates a list of the configuration in JSON format
/JQ=<x>,<y>,<z>	JSON: Query parameters <x>, <y> and <z>
/JQ	→ with JSON-structure (see chap. 8.2.4) via HTTP-POST request: Query parameters
/JR<x>	JSON: Query reset-value of parameter <x> Within the integrated operational unit of the heating system there are reset options available for some parameters. A reset is done by asking the system for the reset value and setting it afterwards.
/JS	→ with JSON-structure (see chap. 8.2.4) via HTTP-POST request: Set parameters
/JV	Queries the version of the JSON-API. Payload: {"api_version": "major.minor"}
/JW	JSON: Reads the list of configuration created with /JL and adjusts the settings.
/K	List all categories At this command the adapter doesn't communicate with the controller, it's a software sided internal function of BSB-LAN.
/K<x>	Query all parameters and values of category <x> At this command the adapter doesn't communicate with the controller, it's a software sided internal function of BSB-LAN.
/L=0,0	Deactivate logging to microSD-card temporarily In general, the activation/deactivation of the logging function should be done in the file BSB_lan_config.h before flashing the Arduino. During runtime the logging can be temporarily deactivated by using L=0,0 though, but it only has an effect until the next reboot of the Arduino.

URL Command	Effect
/L=<x>,<y1>,<y2>,<y3>	Set logging interval to <x> seconds with (optional) logging parameter <y1>,<y2>,<y3> This command can be used to change the logging interval and the parameters that should be logged during runtime. All parameters that should be logged have to be set. After a reboot of the Arduino, again only the parameters are logged which has been defined in BSB_lan_config.h initially.
/LB=<x>	Configure logging of bus-telegrams: only broadcasts (<x>=1) or all (<x>=0) When logging bus telegrams (log parameter 30000 as the only parameter), only the broadcast messages (<x>=1) or all telegrams (<x>=0) are logged.
/LD	Disable logging of telegrams to journal.txt
/LE	Enable logging of telegrams to journal.txt
/LU=<x>	Configure logging of bus-telegrams: only unknown (<x>=1) or all (<x>=0) When logging bus telegrams (log parameter 30000 as the only parameter), only unknown command ids (<x>=1) or all telegrams (<x>=0) are logged.
/M<x>	Activate (<x> = 1) or deactivate (<x> = 0) bus monitor mode By default bus monitor mode is deactivated (<x>=0). When setting <x> to 1, all bytes on the bus are monitored. Each telegram is displayed in hex format with a timestamp in miliseconds at the serial monitor. The html output isn't affected though. To deactivate the monitor mode, set <x> back to 0: /M0.
/N	Reset & reboot arduino (takes approx. 15 seconds) Reset and reboot of the Arduino. Note: Function must be activated in BSB_lan_config.h by #define RESET
/NE	Reset & reboot arduino (takes approx. 15 seconds) and erase EEPROM Reset and reboot of the Arduino with additional erasing of the EEPROM. Note: Function must be activated in BSB_lan_config.h by #define RESET
/P<x>	Set bus type/protocol (temporarily): <x> = 0 → BSB / 1 → LPB / 2 → PPS Changes between BSB (<x>=0), LPB (<x>=1) and PPS (<x>=2). After a reboot of the Arduino, the initially defined bus type will be used again. To change the bus type permanently, adjust the setting setBusType config in BSB_lan_config.h accordingly.
/P<x>,<y>,<z>	Set bus type/protocol <x>, own address <y>, target address <z> (temporarily) Temporarily change of the set bus type and addresses: <x> = bus type (0 = BSB, 1 = LPB, 2 = PPS), <y> = own address and <z> = destination address Empty values leave the address as it is already set.
/Q	Check for unreleased controller-specific parameters
/R<x>	Query reset-value of parameter <x> Within the integrated operational unit of the heating system there are reset options available for some parameters. A reset is done by asking the system for the reset value and setting it afterwards.
/S<x>=<y!z>	Set value <y> for parameter <x> with optional destination address <z> Command for setting values (therefore, write-access must be defined previously in BSB_lan_config.h!). Additionally a destination address can be set by using <z>. If <!z> isn't used, the standard destination address will be used. To set a parameter to 'off/deactivated', just use an empty value: http://<ip-address>/S<x>=
/U	Displays the user-defined variables if used in BSB_lan_custom.h For the creation of one's own subroutines in BSB_lan_config.h two arrays of 20 bytes size, custom_floats[] und custom_longs[], are available. If used, these can be displayed via URL command /U and can be useful to query own sensors in BSB_lan_custom.h and display the results on the web-interface via /U.

URL Command	Effect

`/V<x>`	`Activate (<x> = 1) or deactivate (<x> = 0) verbose output mode`
	`The preset verbosity level is 1, so the bus is 'observed' and all data are displayed in raw hex format additionally.`
	`If the mode should be deactivated, <x> has to be set to 0: /V0`
	`Verbosity mode affects the output of the serial monitor as well as the (optional) logging of bus data to microSD card. Therefore the card could run out of space quickly, so it's advisable to deactivate the verbosity mode already in the BSB_lan_config.h: byte verbose = 0`
	`The html output isn't affected by /V1.`

`/W`	`With a preceding /W the URL commands C, S and Q return data without HTML header and footer (e.g.: /WC or /WS<x>=<y!z>); module WEBSERVER has to be compiled!`

`/X`	`Query optional MAX!-thermostats`
	`Queries and displays the temperatures of optional MAX!-thermostats.`
	`Note: MAX!-components have to be defined in BSB_lan_config.h before!`

8.2 Special Functions

8.2.1 Transmitting a Room Temperature

By using an INF-message, a room temperature can be transmitted to the controller. Therefore you have to activate the function 'room influence' (e.g. parameter 750 for circuit 1, parameter 1050 for circuit 2) before.
Write-access has to be granted for BSB-LAN.
The room temperatures have to be sent regularly in a 'short' interval, like every one or two minutes.

The following parameters have to be used:

- 10000 = heat circuit 1
- 10001 = heat circuit 2
- 10002 = heat circuit 3/P

Example:
This command transmits a room temperature of 19.5°C to the circuit 1: `http://<ip-address/I10000=19.5`

Note: Room Influence Regarding the Room Temperature
FHEM forum user "freetz" has decoded the model behind the "room influence" (parameter 750), so that the effects on the flow temperature became more clear. Thanks a lot for this!
His article as well as an Excel spreadsheet can be found here.

8.2.2 Simulating the Presence Function

The function of the presence button is implemented with the special parameters

- 701 = heat circuit 1,
- 1001 = heat circuit 2 and
- 1301 = heat circuit 3/P
  and has to be executed as a SET-command.

Therefore BSB-LAN needs write-access.
These special parameters (701, 1001, 1301) can not be queried!

With an active *automatic* heating mode one has to use
`http://<ip-address>/S<parameter>=1` to change to the mode 'reduced' and
`http://<ip-address>/S<parameter>=2` to the change to the mode 'comfort'.

The setting is active until the next changement of the heating mode occurs, which is triggered by the time schedule.

Example: The command `<URL>/S701=2` switches HC1 to the comfort mode within the automatic mode.

Note: The function of the presence button is only available when the heater is in automatic mode!

8.2.3 Triggering a Manual DHW-Push

Within many controllers there is a (nearly) undocumented function available: a manual DHW push. To initiate a manual DHW push, one has to press and hold the DHW-mode-button at the operational unit. After approx. three seconds a message appears at the display and the heating process starts.

With some controllers this function can also be used with BSB-LAN using a SET-command: `http://<ip-address>/S1603=1` - of course the parameter 1603 must be made settable before (see chap. 05).

8.2.4 Retrieving and Controlling via JSON

This function is still 'under development', so changes can occur!

It's also possible to use JSON to query or set parameters.

- **Query possible values for parameters:** `http://<ip-address>/JC=<x>,<y>,<z>`
  Query possible values for parameters `<x>,<y>,<z>`. The format of the returned data is the same as the command `/JK=<x>`.
- **Query the configuration of BSB-LAN:** `http://<ip-address>/JI`
  Query configuration of BSB-LAN. Configuration will be reported in a JSON friendly structure.
- **Query of categories:**

  `http://<ip-address>/JK=<xx>`
  Query of a specific category (`<xx>` = number of category)

  `http://<ip-address>/JK=ALL`
  Query of all categories (including min. and max.)

- **Query and set parameters via HTTP POST:**

  For this the following URL commands have to be used:
  `http://<ip-address>/JQ` to query parameters
  `http://<ip-address>/JS` to set parameters

  The following parameters are usable within these URL commands:

```
http://<ip-address>/JQ
Send: "Parameter"
Receive: "Parameter", "Value", "Unit", "DataType" (0 = plain value (number), 1 = ENUM (value (8/16 Bit) followed by space follo
wed by text), 2 = bit value (bit value (decimal) followed by bitmask followed by text/chosen option), 3 = weekday, 4 = hour:min
ute, 5 = date and time, 6 = day and month, 7 = string, 8 = PPS time (day of week, hour:minute)), "readonly" (0 = read/write, 1
= read only parameter), "error" (0 - ok, 7 - parameter not supported, 1-255 - LPB/BSB bus errors, 256 - decoding error, 257 - un
known command, 258 - not found, 259 - no enum str, 260 - unknown type, 261 - query failed), "isswitch" (1 = it VT_ONOFF or VT_Y
ESNO data type (subtype of ENUM), 0 = all other cases)

http://<ip-address>/JS
Send: "Parameter", "Value", "Type" (0 = INF, 1 = SET)
Receive: "Parameter", "Status" (0 = error, 1 = OK, 2 = parameter read-only)
```

  The query of multiple parameters within one command is also possible:
  The command `http://<ip-address>/JQ=<x>,<y>,<z>` queries the parameters `<x>, <y>, <z>`.

- **Set parameters via Linux command line or „Curl for Windows“**
  Exemplary for parameter 700 (operating mode heating circuit 1) → set to 1 (= automatic mode):

  Linux command line:

```
curl -v -H "Content-Type: application/json" -X POST -d '{"Parameter":"700", "Value":"1", "Type":"1"}' http://<ip-address>/JS
```

Curl for Windows:

```
curl -v -H "Content-Type: application/json" -X POST -d "{\"Parameter\":\"700\", \"Value\":\"1\", \"Type\":\"1\"}" http://<ip-add
ress>/JS
```

- **Query the reset value of a parameter:**
  `http://<IP-Adresse>/JR<x>` → Queries the reset-value of parameter . Within the integrated operational unit of the heating system there are reset options available for some parameters. A reset is done by asking the system for the reset value and setting it afterwards (JSON: via /JS).

- **Backup and restore the config settings of BSB-LAN:**

  `http://<IP-Adresse>/JL` → Creates a list of the configuration in JSON format.

  `http://<IP-Adresse>/JW` → Reads the list of configuration created with /JL and adjusts the settings.

  Note: For the usage of this function the module "JSONCONFIG" (see file *BSB_lan_config.h*) has to be compiled!

8.2.5 Checking for Non-Released Controller Specific Command IDs

Note: It is adviseable to execute this one time query at the initial usage of BSB-LAN.

`http://<ip-address>/Q`

This function queries all command ids from the file *BSB_lan_defs.h* and sends those ones which aren't marked for the own type of controller as a query to the controller (type QUR, 0x06).

This already happens regularly within parameters for which only one command id is known and creates the already known "error 7" messages. As soon as more than one command id is known for a specific parameter, the first known command id still stays at "DEV_ALL" and is still the standard command id for all controllers. The new command id is then only approved for the new type of controller where that command id comes from. But: It's possible that this new command id also works with other controllers or that it's the 'regular' id. So the URL command /Q now checks all command ids which aren't approved for the own type of controller. By this it's often possible that 'new' parameters becoming available for the own controller.

Note:
Within this command, only queries occur - so no changes of any settings within the controller will be changed!

If all command ids are already known and approved for the own type of controller, no "error 7" messages occur within the output of the /Q command.
As an example, this is how the output of the webinterface looks in this case:

```
Gerätefamilie: 92
Gerätevariante: 100
Geräte-Identifikation: AVS37.294/100
Software-Version: 2.0
Entwicklungs-Index:
Objektverzeichnis-Version: 1.3
Bootloader-Version:
EEPROM-Version:
Konfiguration - Info 2 OEM:
Zugangscode Inbetriebnahme?:
Zugangscode Fachmannebene ?:
Zugangscode OEM?:
Zugangscode OEM2?:
Bisher unbekannte Geräteabfrage: 20
Hersteller-ID (letzten vier Bytes): 58469
Bisher unbekannte Geräteabfrage:
Außentemperatur (10003):
Außentemperatur (10004):

6225;6226;6224;6220;6221;6227;6229;6231;6232;6233;6234;6235;6223;6236;6237;
92;100;AVS37.294/100;2.0;;1.3;;;;;;;20;58469;;

Starte Test...

Test beendet.

Fertig.
```

If some 'new' parameters have been identified by the function of /Q, the output of the webinterface looks like this (e.g.):

```
Gerätefamilie: 92
Gerätevariante: 100
Geräte-Identifikation: AVS37.294/100
Software-Version: 2.0
Entwicklungs-Index:
Objektverzeichnis-Version: 1.3
Bootloader-Version:
EEPROM-Version:
Konfiguration - Info 2 OEM:
Zugangscode Inbetriebnahme?:
Zugangscode Fachmannebene ?:
Zugangscode OEM?:
Zugangscode OEM2?:
Bisher unbekannte Geräteabfrage: 20
Hersteller-ID (letzten vier Bytes): 58469
Bisher unbekannte Geräteabfrage:
Außentemperatur (10003):
Außentemperatur (10004):

6225;6226;6224;6220;6221;6227;6229;6231;6232;6233;6234;6235;6223;6236;6237;
92;100;AVS37.294/100;2.0;;1.3;;;;;;;20;58469;;


Starte Test...

5
5 Uhrzeit und Datum - Sommerzeitbeginn Tag/Monat: error 7 (parameter not supported)
DC C2 0A 0B 06 3D 05 04 B3 DA F8
DC 8A 42 14 07 05 3D 04 B3 00 FF 03 19 FF FF FF FF 16 C4 C8
6
6 Uhrzeit und Datum - Sommerzeitende Tag/Monat: error 7 (parameter not supported)
DC C2 0A 0B 06 3D 05 04 B2 CA D9
DC 8A 42 14 07 05 3D 04 B2 00 FF 0A 19 FF FF FF FF 16 80 41

Test beendet.

Fertig.
```

In general, the output of /Q (together with the brand and the specific name of that type of heating system) should be reported in any case, so that we can add that system to the list of reported systems which work with BSB-LAN.
But: Especially if any error7-messages occur this should be done, so that the reported error7-parameters can be approved for that specific type of controller and can be made available.
For reporting the system, please copy and paste the output of the webinterface (like the examples above) and post it either in the german FHEM-Forum or send it via email to Frederik or me (Ulf). Please don't forget to add the name of the brand and the specific type of your heating system! Thanks.

8.2.6 IPWE Extension

The IPWE extension offers the presentation of various previously defined parameters by the usage of just one short URL. To access this table overview, the following URL has to be used:
`<ip-address>/ipwe.cgi`.
Note: If the optional security function of the passkey is used, the passkey has NOT to be added within the URL in this case!

Sensortyp	Adresse	Beschreibung	Temperatur	Luftfeuchtigkeit	Windgeschwindigkeit	Regenmenge
T	1	Außentemperatur	6.00	0	0	0
T	2	Trinkwassertemperatur 1	59.40	0	0	0
T	3	Kesseltemperatur	52.30	0	0	0
T	4	Status Heizkreis 1	114.00	0	0	0
T	5	1. Brennerstufe T2	255.00	0	0	0
T	6	2. Brennerstufe T8	0.00	0	0	0
T	7	Historie 1 Datum/Zeit	1.01	0	0	0
T	8	Historie 1 Fehlercode	98.00	0	0	0
T	9	284c453d07000082	21.25	0	0	0

Example of an IPWE output.

To use the function of the IPWE extension, one has to make two settings in the file `BSB_LAN_config.h` before flashing the arduino:

- The definement `#define IPWE` has to be active.

- The desired parameters which should be displayed have to be listed.

Additionally to the set parameters, the values of optional connected sensors (DHT22 / DS18B20) will be listed automatically.
If DS18B20 sensors are connected, the specific sensor id of each sensor will also be displayed. You can see this in the last row of the IPWE example above, the specific sensor id of the one and only connected DS18B20 sensor is "284c453d07000082".

Notes:

- If there are -by accident- parameters defined which aren't supported by the specific heating system, the non-existent values will be displayed as "0.00". So don't get that wrong - it doesn't mean, that that specific value is "0.00"! Before you define the parameters, it's best to check before and make sure, that the heating system really offers these parameters.
- Because the IPWE extension was originally designed to implement values of a specific wireless weather station, there will be some columns which doesn't seem to make sense, like "Windgeschwindigkeit" (wind speed) or "Regenmenge" (amount of rain water) - you can just ignore that. Basically, there are just two columns which are relevant for the normal parameters: "Beschreibung" and "Temperatur", because here you can find the name of the parameter and the belonging value.
- The states of certain parameters aren't shown in clear text, only in numerical values. In the above example you can see this e.g. in the row with the parameter "1. Brennerstufe T1": There doesn't appear "Ein" (on), it only shows the value "255" - which means "Ein" (on).

8.2.7 Changing the Date, Time and Time Programs

Changing the date, time and time programs is only possible by using a special URL command, it is not possible via webinterface.
To use this feature, BSB-LAN needs write access (see chap. 5).

Changing the date and time
The following URL command sets the date to the fourth of january 2019 and the time to 08:15pm:
`/S0=04.01.2019_20:15:00`
Using this function it is possible to sychronize the time with (e.g.) a NTP time server.

Changing time programs
The following URL command sets the time program for wednesday at heating circuit 1 (parameter 502) to 05:00am-10:00pm:
`/S502=05:00-22:00_xx:xx-xx:xx_xx:xx-xx:xx`

8.2.8 Transmitting an Alternative Outside Temperature

Certain specific controller types allow the usage of different wireless components. Amongst other things there is also a wireless outside temperature sensor available. Within these compatible controllers it is possible to use BSB-LAN to transmit an alternative outside temperature.

Until now it seems that only controllers of the types LMS and RVS are compatible. Older types of controllers (e.g. LMU and RVA) don't seem to be compatible.

For using this function the wired temperature sensor has to be deconnected from the controller. The transmission of the alternative outside temperature has to be done regularly within a time windows of (approx.) max. 10 minutes, but it is adviseable to transmit the value every 60 to 120 seconds.

To use this function, BSB-LAN needs write access (see chap. 5). The outside temperature has to be transmitted as an INF message (with the virtual parameter 10003) using the URL command
`<ip-address>/I10003=xx`
where xx is the outside temperature in °C (degrees celcius); fractional values are possible.

Example:
With `<ip-address>/I10003=16.4` the outside temperature of 16.4°C is transmitted; `<ip-address>/I10003=9` transmits 9°C.

8.2.9 Integrating Own Code in BSB-LAN

BSB-LAN offers the possibility to integrate your own code. For this purpose, the corresponding definition in the file `BSB_lan_config.h` must be activated and the code must be added in the files `BSB_lan_custom.h.default`, `BSB_lan_custom_global.h` and `BSB_lan_custom_setup.h`. The file `BSB_lan_custom.h.default` must be renamed to `BSB_lan_custom.h` for use. An example and corresponding notes can be found in the respective files.

FHEM-Forumuser "Scherheinz" has provided another example (see forum post).
Many thanks for this!

In the following the above mentioned example:
Description:
"Every 20 seconds the battery voltage is read in via a voltage divider. Then a moving average is calculated from the last 10 values and forwarded to FHEM via MQTT" (quote from the above linked post).

Integration:
The following code must be inserted into the file `BSB_lan_custom_global.h`:

```
const int akkuPin = A0;
int akkuWert = 0;
float akkuSpg = 12.00;
char tempBuffer[100];
int j;

void Filtern(float &FiltVal, int NewVal, int FF){ //gleitender Mittelwert bilden aus den 10 letzten Werten
  FiltVal= ((FiltVal * FF) + NewVal) / (FF +1);
}
```

The following code must be inserted into the file `BSB_lan_custom.h`:

```
if (custom_timer > custom_timer_compare + 20000) {    // alle 20 Sekunden
  custom_timer_compare = millis();

akkuWert = analogRead(akkuPin); // Spannung messen

akkuWert = map(akkuWert, 500, 1023, 0, 150); // umwandeln auf 0 - 15V
akkuWert = akkuWert / 10.00;

Filtern(akkuSpg, akkuWert, 9); //gleitender Mittelwert bilden aus den 10 letzten Werten
if (j++ > 10) akkuWert=1;  // nach 10 Werten Sprung auf 1

if (!MQTTClient.connected()) {
      MQTTClient.setServer(MQTTBroker, 1883);
      int retries = 0;
      while (!MQTTClient.connected() && retries < 3) {
        MQTTClient.connect("BSB-LAN", MQTTUser, MQTTPass);
        retries++;
        if (!MQTTClient.connected()) {
          delay(1000);
          DebugOutput.println(F("Failed to connect to MQTT broker, retrying..."));
        }
        MQTTClient.publish("AkkuSpannung",dtostrf(akkuSpg, 6, 1, tempBuffer));
        MQTTClient.disconnect();
      }
    }
}
```

8.2.10 Using the Webserver Function

The webserver function has been developed and added by the user "dukess", who also gave the following informations about the usage.
Thanks a lot!

By activating the belonging definement '#define webserver' within *BSB_LAN_config.h*, BSB-LAN can act as a webserver which even supports static compression. For using that function, a few points must be noticed:

- All files are / must be stored on the microSD card, but files can be placed in different directories and subdirectories though. E.g.: `http://<bsb-lan-ip-address>/foo/bar.html` gets the file `bar.html` from the directory `foo` of the microSD card.
- Only static content is supported.
- Supported file types are: html, htm, css, js, xml, txt, jpg, gif, svg, png, ico, gz.
- The web server supports the following headers: ETag, Last-Modified, Content-Length, Cache-Control.
- As already mentioned, the web server supports static compression. If possible (if the client's browser supports gzip), it's always trying to deliver gzipped content (e.g. /d3d.js.gz for the URL /d3d.js).

The following examples show the usage:

- If there's no file named `index.html` located in the root directory of the microSD card, the regular web interface of BSB-LAN will be displayed by the query of the URL `http://<ip-address>/`.
- If there's a file named `index.html` located in the root directory of the microSD card, that file will be displayed by the query of the URL `http://<ip-address>/` instead of the regular webinterface of BSB-LAN.
- If the file `index.html` is located in a subdirectory of the microSD card, that file will only be displayed when the complete URL will be queried: `http://<ip-address>/foo/bar/index.html`. If (in this case) only `http://<ip-address>/foo/bar/` would be queried, the regular webinterface of BSB-LAN would still be displayed, because directory listing or URL rewriting isn't implemented in the special webserver function.

Note: As always, if you are using the PASSKEY function, you have to add the passkey to the URL.

8.2.11 Using the Alternative AJAX Web Interface

The AJAX webserver function has also been developed and added by the user "dukess".
Please see the informations in his AJAX repo for usage.
Thanks a lot!

8.2.12 MQTT

BSB-LAN supports the MQTT protocol, so the values and settings of the heating controller can be retrieved via MQTT.
To use MQTT with BSB-LAN, it is mandatory that the definition "#define LOGGER" in the file *BSB_LAN_config.h* is activated. This is already the case in the default setting.

The parameters to be sent (queried by BSB-LAN, the transmission interval (only one interval possible for all parameters!) and the other MQTT-specific settings (broker, topic, etc.) are to be set either via web configuration or directly in the file *BSB_LAN_config.h*. Please refer to the explanations in the corresponding subchapters of chap. 5.

Examples for an integration of BSB-LAN can be found in the corresponding subchapters of chap. 11.

BSB-LAN uses the subtopic "status" below the defined "MQTTTopicPrefix" to publish its online state. Based on the default setting this would be "BSB-LAN/status". This allows you to track whether BSB-LAN is actually publishing current readings and able to receive commands.
If BSB-LAN is available, the topic contains the value "online", otherwise you'll see "offline". The message is made persistant via the retain-flag, thus, the subscriber does not have to have the topic subscribed during BSB-LAN startup.
Any restart initiated by the firmware (e.g. the URL-command /N) will immediately set the topic to "offline". Any uncontrolled shutdown (e.g. a power outage or some firmware flashing) will cause the broker to transmit the offline-message after a (broker specific) timeout.

In addition to (broker-side) pure receiving, it is also possible to query and/or send control commands (URL commands /S and /I) to BSB-LAN from the broker via MQTT. Of course, BSB-LAN must be granted write access to the controller if one wants to change settings.

The command syntax is:
`set <MQTT server> publish <topic> <command>`

- `<MQTT server>` = The name of the MQTT server.
- `<topic>` = Default setting is "BSB-LAN", otherwise the defined "MQTTTopicPrefix" in the file *BSB_LAN_config.h* accordingly. If no topic is defined (not advisable), "FromBroker" must be taken as topic.
- `<command>` = The query of the specific parameter or the corresponding parameter-specific URL command /S or /I.
  Note: Only one query/command is possible at a time, so no parameter ranges can be queried!

Subsequently BSB-LAN sends back an acknowledgement of receipt ("ACK_\<command>").

Example:
The command `set mqtt2Server publish BSB-LAN /S700=1` sends from the MQTT broker named "mqtt2Server" the command "/S700=1" with the topic "BSB-LAN" and causes a mode switch to automatic mode.

The command `set mqtt2Server publish BSB-LAN /700` sends from the MQTT broker named "mqtt2Server" the command "/700" with the topic "BSB-LAN" and causes a query of parameter 700.

8.2.13 Room Unit Emulation

With the setup of the BSB-LAN adapter a room unit can be emulated, therefore additional hardware is needed.

The following functions are implemented in the code:

- Integration fo connected sensors for measuring and transmitting the room temperature(s) to the desired heating circuit(s),
- triggering a DHW push by using a pushbutton and
- using the presence function for the heating circuits 1-3 by using a pushbutton (automatic detection of the present state with the corresponding change between comfort and reduced mode in the automatic mode).

To use the functions, the corresponding entries must be made in the configuration. This can be done either by changes in the file *BSB_LAN_config.h* or via the web interface (menu item "Settings").

In the following some notes about each function.

Room temperature

- Up to five connected sensors can be specified for the room temperature measurements.
- If more than one sensor is used, an average value is automatically calculated and transmitted to the heating controller.
- To assign the respective sensors to the desired heating circuits, the specific parameter numbers of the respective sensors must be entered. An overview of the connected sensors together with the associated parameter number can be found in the category "One Wire, DHT & MAX! Sensors" (menu item "Heating functions" or by clicking on the menu item "Sensors").
- When entering several sensors for one HC, the parameter numbers are only to be separated from each other by a comma, no space may be used after the comma.

Pushbutton for TWW push and presence button function

- The GPIO pins used for connecting the pushbuttons (one pin per pushbutton) must be set in the configuration.
- DIGITAL pins must be used!
- Please make sure that you do not use any other pins (e.g. those of connected sensors)! For Due-users: explicitly *don't* use the pins 12, 16-21, 31, 33, 53!
- The pushbuttons are to be connected arduino-typically for HIGH, that means you must connect a pull-down resistor (approx. 100kOhm) additionally to the respective pin.
- You can find a pinout diagram of the Due in appendix B.
- If you are not sure how to connect a pushbutton to an Arduino for HIGH, please have a look at the internet, where you can find countless examples.
  Nevertheless, it should be briefly mentioned here how to proceed:
  - The push button with the two connectors A and B has to be connected at one connector (A) to the desired GPIO digital pin of the Due.
  - Additionaly, to the same terminal of the pushbutton (A) the pull-down resistor has to be connected, which in turn has to be connected to GND of the Due.
    This is important, the use of the resistor must not be omitted! Due to the pull-down resistor a defined potential is applied to the GPIO when the button is not operated and the so-called 'floating' of the input is prevented. If the pull-down is not used and the input would 'float', unwanted level changes could occur at the pin, which in turn would result in the respective function (DHW push or heating mode switchover) being triggered unintentionally.
  - The other pin of the button (B) is connected to a **3.3V** pin of the Due.
    Caution: The inputs of the Due are only 3.3V tolerant, so *don't ever* connect the pushbutton to a 5V pin of the Due!
    If the button is pressed now, the circuit is closed - the signal is recognized as HIGH and the respective command (TWW push/presence button) is triggered.
- Additional note: If you disconnect the pushbutton(s) (e.g. because you don't want to use them anymore) make sure that you set the belonging pin to "0" again and save the changed configuration, so that no floating could occur at that previously used pin!

8.2.14 Erasing EEPROM Using Pincontacts

In principle, the EEPROM can be erased via the web interface with the command /NE. However, in certain situations (e.g. if no access to the web interface is possible) it may be necessary to delete the EEPROM without using the URL command.
For this, the following pins must be connected to each other when starting or rebooting the device:

- Due: pins 31 & 33
- ESP32: pins 18 and GND
  After successful erase, the LED of the Arduino/ESP32 flashes for four seconds.
  At restart the (pre-)settings from the file *BSB_LAN_config.h* are taken over, an adjustment can be done afterwards as usual via (e.g.) the web interface.

Further on to chapter 9
Back to TOC

9. Logging Data

Back to TOC
Back to chapter 8

9. Logging Data

9.1 Usage of the Adapter as a Standalone Logger with BSB-LAN

Insert a FAT32-formatted microSD card into the memory card slot of the ethernet shield before powering up the Arduino.

Before flashing, activate the definement `#define LOGGER` in the file *BSB_lan_config.h*, add the parameters to be logged to the variable `log_parameters` and determine the log interval with the variable `log_interval`. Please also note the corresponding points in chapter 8.1.
Later, during the runtime, both the interval and the logging parameters can be changed by using the command `"/L=[Interval],[Parameter1],...,[Parameter20]"`.

All data is stored within the file *datalog.txt* on the card in csv format. Thus the data can easily be imported in Excel or OpenOffice Calc.

The file contents can be viewed with the URL command `/D`, a graphical representation of the log files is done by `/DG`.

To delete and rebuild the file *datalog.txt*, use the URL command `/D0`.

The URL command `/D0` should also be executed on first use! This will initiate the file with the appropriate CSV header.

Notes:

Occasionally it may happen that certain microSD cards are not easily recognized by the LAN shield. Should this Problem occur, the usage of cards with memory sizes from 1GB, 2GB to max. 4GB is recommended. Should these cards also be problematic, try formatting it as FAT16.

Please note that the Arduino is not an exact clock. Even if the interval has been set up to e.g. 60 seconds, the time displayed in the file (which is received by the heating control) possibly will differ - this can take up to a second per minute.
If an exact log time is absolutely necessary, you can measure the average time difference between the Arduino time and the real time and adjust the log interval accordingly (e.g. set 59 seconds instead of 60 seconds).

9.2 Usage of the Adapter as a Remote Logger

In addition to the use of complex systems such as FHEM and the specific logging solutions, you can e.g. execute the following command periodically (for example via cron job):

```
DATE=`date +%Y%m%d%H%M%S`; wget -qO- http://192.168.178.88/8310/720/710 | egrep "(8310|720|710)" | sed "s/^/$DATE /" >> log.txt
```

The log file *log.txt* resulting from this example contains the recorded values of parameters 8310, 720 and 710.
Later you can sort the log file based on the parameter numbers, use the command \'sort\' for this:

```
sort -k2 log.txt
```

Further on to chapter 10
Back to TOC

10. Reading Out New Parameter Telegrams

Back to TOC
Back to chapter 9

10. Reading Out New Parameter Telegrams

If your heating system has parameters which aren't implemented in BSB-LAN yet, you can help us to make BSB-LAN even better. For that you have to use the serial monitor of the ArduinoIDE and read out the specific telegram / command ID of each parameter including the state or value which is present at the time you are getting that command ID and the different options (if available). Here are some instructions and explanations how to do it, please read it completely before you start.

To now read out the telegram / command ID of a new parameter, all you need is to connect your Arduino to a Laptop/PC via USB while it is connected to your heating system and follow these steps (BSB only, LPB is similar, but telegram structure is a bit different):

- Start the Arduino IDE and turn on the serial monitor.
- Enable logging to the serial console and turn on verbose output with URL-Parameter /V1 (if you deactivated the verbose mode in the file BSB_lan_config.h before) on the Arduino, e.g. http://192.168.178.88/V1. Alternatively, you can log bus telegrams to SD card by using (only) logging parameter 30000 (see logging section above) and set variable `log_unknown_only` to 1 (URL command /LU=1) and follow logging entries with URL command /D.
- On the heating system, switch to the parameter you want to analyze (using the command wheel, arrows or whatever input mode your heating system has).
- Wait for "silence" on the bus and then switch forward one parameter and then back again to the parameter you want. You should now have something like this on the log:

```
DISP->HEIZ QUR      113D305F
DC 8A 00 0B 06 3D 11 30 5F AB EC
HEIZ->DISP ANS      113D305F 00 00
DC 80 0A 0D 07 11 3D 30 5F 00 00 03 A1
DISP->HEIZ QUR      113D3063
DC 8A 00 0B 06 3D 11 30 63 5C 33
HEIZ->DISP ANS      113D3063 00 00 16
DC 80 0A 0E 07 11 3D 30 63 00 00 16 AD 0B
```

  The first four lines are from the parameter you switched forward to, the last four lines are from the parameter you want to analyze (doing the switching back and forth only makes sure that the last message on the bus is really the parameter you are looking for). Instead of DISP you might see RGT1, depending on what device you are using to select the parameter.
  Note:
  If the parameter you want to read out has different optional setting between which you can choose, you should try to read out the command ID for each option. For that you (usually) have to change the optional setting and confirm the change by pressing the OK-button. **But:** Please only do this if you are sure that these settings are not critical for the function of your heating system or your installation! If you change the settings, then you also have to write down the specific name for the chosen option where the command ID belongs to. At the end you should go back to the preset setting. If the parameter you are decoding has a unit like degrees, volt or so, please also write that one down.

Important:
Write down the command ID with the specific name and unit (if available) of the new function. If you also read out the different optional settings for a new parameter, don't forget to also write it down. In the end you should have a list which exactly shows the specific command ID together with the name of the parameter, the state or shown value at the time you read it out and (if available) the unit (like degrees or so). The same goes for the optional settings. Your work will be useless if e.g. you just write down the command ID togehter with the name of the new parameter - because then we still don't know what the specific command ID means (in terms like on/off etc.).

Further on to chapter 11
Back to TOC

11. Usage of External Programs

Back to TOC
Back to chapter 10

11. Usage of External Programs

Note:
The code examples and/or modules presented in the following subchapters may not yet have been adapted to the changes that accompanied the changeover of the BSB-LAN version to the v2.x. It may therefore be that some examples can no longer be adopted 1:1, but must be adapted (e.g.: the URL command /T isn't available anymore in BSB-LAN v2.x but can still be found in some examples listed below).
If you should discover a code example or a module, which is not executable in the represented form with the current BSB-LAN version v2.x, then please inform the author of the example/module and (if possible) send me an adapted version of that example, so that I can place the corrected version here.
Thanks.

Because the adapter is just an interface which makes it possible to gain access to the controller of the heating system, it is of course possible to use external programs in addition to BSB-LAN. By this you can integrate your heating system in complex home automation systems and e.g. create comprehensive logfiles and realize their graphical output.

Due to the many different software solutions out there, neither a comprehensive presentation, nor a comprehensive description about the integration in specific programs can be given here. Also the whole topic about heating systems in general and how to optimize their functionality can not be treated here.
However, the following subchapters supported by kind users offer some example code and scripts for some common software solutions to show you how the integration of BSB-LAN works. Hopefully this will be helpful for your first steps.

If any question about these example scripts arises, please try to find the answer by yourself (e.g. in a specific forum). Maybe the author of the specific example may also be willing to help you, so you can try to contact him, but please don't send me (Ulf) any questions about these topics.

If you are a programmer or a user of a system which isn't already mentioned here and you want to contribute an example script to make the first steps easier for other users, of course you can contact me by email!

At this point, there's just one advise I'd like to give you: always make sure, that the whole process of heat generation and the whole functionality of your heating system will still work without any problems in the case that your home automation system may be faulty or without functionality at all! Every intervention you may want to trigger with this system should not have any impact on the basic functionality of your heating system. Keep in mind that it's (in most of the cases) still a certain kind of controlled explosion and a burning process which generates the heat!

Notes:
Of course the following examples always have to be adjusted to your individual needs and settings. Especially the correct IP, probably activated security options and the setting that gives BSB-LAN writeable access should be mentioned here (see chapter 5 for the specific options).

11.1 FHEM

11.1.1 Integration via BSB-LAN Module

FHEM forum member „justme1968" is working on a module for an easy integration in FHEM:
https://forum.fhem.de/index.php/topic,84381.0.html
Vielen Dank!

UPDATE:
A first version is already available, but it's limited in functionality and probably not yet fully stable. If you are using FHEM, it's maybe better to use an implemention method like HTTPMOD until the module is completely done.

11.1.2 Integration via HTTPMOD Module

The example scripts for the FHEM integration were contributed by FHEM forum member „freetz".
Thanks a lot!

To gain access to the adapter via FHEM, you can use the module HTTPMOD.

Example script for the query of parameters and the transmission of the room temperature:

The example script queries the parameters 8700, 8743 and 8314 every 300 seconds. It assigns them to the device \"THISION\" (name of the heating system) and the readings \"Aussentemperatur\", \"Vorlauftemperatur\" und \"Ruecklauftemperatur\".

Furthermore it provides a reading \"Istwert\" which can be set via FHEM to transmit the current room temperature to the heating system (parameter 10000).

Last but not least it calculates the difference between \"Vorlauftemperatur\" and \"Rücklauftemperatur\" and assigns this difference to the reading \"Spreizung\".

Please note:
The RegEx-conditions must match from the beginning of the string (therefore the number of the parameter, e.g. 8700).

There seems to be a problem within the HTTPMOD-module and readings which start with "0" (`reading0Name` *and* `reading0Regex`*), so please start the readings counting from "1".*

```
define THISION HTTPMOD http://192.168.178.88/8700/8743/8314 300
attr THISION userattr reading1Name reading1Regex reading2Name reading2Regex reading3Name reading3Regex readingOExpr set1Name set1URL
attr THISION event-on-change-reading .*
attr THISION reading1Name Aussentemperatur
attr THISION reading1Regex 8700 .*:[ \t]+([-]?[\d\.]+)
attr THISION reading2Name Vorlauftemperatur
attr THISION reading2Regex 8743 .*:[ \t]+([-]?[\d\.]+)
attr THISION reading3Name Ruecklauftemperatur
attr THISION reading3Regex 8314 .*:[ \t]+([-]?[\d\.]+)
attr THISION readingOExpr $val=~s/[\r\n]//g;;$val
attr THISION set1Name Istwert
attr THISION set1URL http://192.168.178.88/I10000=$val
attr THISION timeout 5
attr THISION userReadings Spreizung { sprintf("%.1f",ReadingsVal("THISION","Vorlauftemperatur",0)-ReadingsVal("THISION","Ruecklaufte
mperatur",0));; }
```

With that example the readings are displayed as numerical values. If you want them to be displayed in plain text like it's done within the webinterface of BSB-LAN, you have to adjust the regular expressions according to that. The following example shows for the parameters '700 - Betriebsart' and '8000 - Staus Heizkreis 1'.

```
attr THISION reading4Name Betriebsart
attr THISION reading4Regex 700 .*-[ \t]+(.*)
attr THISION reading5Name Status Heizkreis 1
attr THISION reading5Regex 8000 .*-[ \t]+(.*)
```

The numbering of the previously listed readings are continued with this, the readings should be added to the line 'attr THISION userattr'. Furthermore the URL has to be expanded with the parameters 700 and 8000. At the end it looks like this:

```
define THISION HTTPMOD http://192.168.178.88/8700/8743/8314/700/8000 300
attr THISION userattr reading1Name reading1Regex reading2Name reading2Regex reading3Name reading3Regex reading4Name reading4Regex re
ading5Name reading5Regex readingOExpr set1Name set1URL
attr THISION event-on-change-reading .*
attr THISION reading1Name Aussentemperatur
attr THISION reading1Regex 8700 .*:[ \t]+([-]?[\d\.]+)
attr THISION reading2Name Vorlauftemperatur
attr THISION reading2Regex 8743 .*:[ \t]+([-]?[\d\.]+)
attr THISION reading3Name Ruecklauftemperatur
attr THISION reading3Regex 8314 .*:[ \t]+([-]?[\d\.]+)
attr THISION reading4Name Betriebsart
attr THISION reading4Regex 700 .*-[ \t]+(.*)
attr THISION reading5Name Status Heizkreis 1
attr THISION reading5Regex 8000 .*-[ \t]+(.*)
attr THISION readingOExpr $val=~s/[\r\n]//g;;$val
attr THISION set1Name Istwert
attr THISION set1URL http://192.168.178.88/I10000=$val
attr THISION timeout 5
attr THISION userReadings Spreizung { sprintf("%.1f",ReadingsVal("THISION","Vorlauftemperatur",0)-ReadingsVal("THISION","Ruecklaufte
mperatur",0));; }
```

Example script for the query and control of a connected relay board:

The following script is an example for a FHEM configuration, where the three relay ports named \"Heater\", \"Fan\" and \"Bell\" are queried and controlled, with the relays connected to the GPIO pins 7, 6 and 5.

```
define EthRelais HTTPMOD http://192.168.178.88/G05/G06/G07 30
attr EthRelais userattr reading1Name reading1Regex reading2Name reading2Regex reading3Name reading3Regex readingOExpr readingOMap se
t1Name set1URL set2Name set2URL set3Name set3URL setIMap setParseResponse:0,1 setRegex
attr EthRelais event-on-change-reading .*
attr EthRelais reading1Name Heater
attr EthRelais reading1Regex GPIO7:[ \t](\d)
attr EthRelais reading2Name Fan
attr EthRelais reading2Regex GPIO6:[ \t](\d)
attr EthRelais reading3Name Bell
attr EthRelais reading3Regex GPIO5:[ \t](\d)
attr EthRelais room Heizung
attr EthRelais set1Name Heater
attr EthRelais set1URL http://192.168.178.88/G07=$val
attr EthRelais set2Name Fan
attr EthRelais set2URL http://192.168.178.88/G06=$val
attr EthRelais set3Name Bell
attr EthRelais set3URL http://192.168.178.88/G05=$val
attr EthRelais setParseResponse 1
attr EthRelais setRegex GPIO[0-9]+:[ \t](\d)
attr EthRelais timeout 5
```

11.2 OpenHAB

Right now there is no complete binding for BSB-LAN (but there is a binding for openHAB2 though!). But using the bindings for HTTPMOD and Javascript Transformation it's possible to read and set parameters.

Logging can be realized by (e.g.) InfluxDB and visualisation by (e.g.) Grafana.

11.2.1 OpenHAB with Javascript Transformation

The example scripts for the openHAB integration were contributed by FHEM forum member „acfischer42". Based on that, user "sihui" contributed two corrections/suggestions for a change and the script to display the values in a sitemap.
Thanks a lot!

NOTE:
The neccessary addons like the the Javascript Transformations have to be installed previously!

Example of an item configuration:

```
Number hz_aussentemp "Aussentemperatur [%.1f °C]" <temperature> (Heizunglog) { http="<[http://192.168.178.88/8700:60000:JS(bsbinput.
js)]" }
String hz_700 "Heizkreis 1 Betriebsart [%s]" <temperature> (Heizunglog){ http="<[http://192.168.178.88/700:1000:JS(bsbinput_string.j
s)]" }
```

The following Javascript has to be put as *bsbinput.js* in the folder *transform*.

Example script for the query of parameters which report a value (bsbinput.js):

```
(function(i) {
    var outputres;
    var results = [];
    value1 = i;
    // define regex to search for the line in the http response
    var regEx = 'input type=text id=\'value[0-9]+\' VALUE=\'[-]*[0-9]+\.[0-9]+';
    var re = new RegExp(regEx, 'gim');

 do {
    match = re.exec(value1);
    if (match){
        results.push(match[0]);
    }
} while (match);

    outputres = results[0]
    //extract actual value from the output
    var output=outputres.substr(outputres.indexOf("VALUE='")+7,outputres.length);
    return output;
})(input)
```

Example script for a direct query of enum-values (bsbinput_string.js):

```
(function(i) {
        var outputres;
    var results = [];
    value1 = i;
    // define regex to search for the line in the http response
    var regEx = '<option value=\'[0-9]+\' SELECTED>.*</option>';
    var re = new RegExp(regEx, 'gim');

 do {
    match = re.exec(value1);
    if (match){
        results.push(match[0]);
    }
} while (match);

    outputres = results[0]
    //extract actual value from the output
    var l=outputres.indexOf("</o")-outputres.indexOf(">")-1
    var output=outputres.substr(outputres.indexOf(">")+1,l);
        return output;
})(input)
```

Writing data is done via rules:

```
rule "RoomTemp"

when
    Item iSet_temp changed
then
    sendHttpGetRequest("http://192.168.178.88/I10000="+iSet_temp.state.toString)
end
```

Display of values in a sitemap (BasicUI, ClassicUI, iOS und Android App):

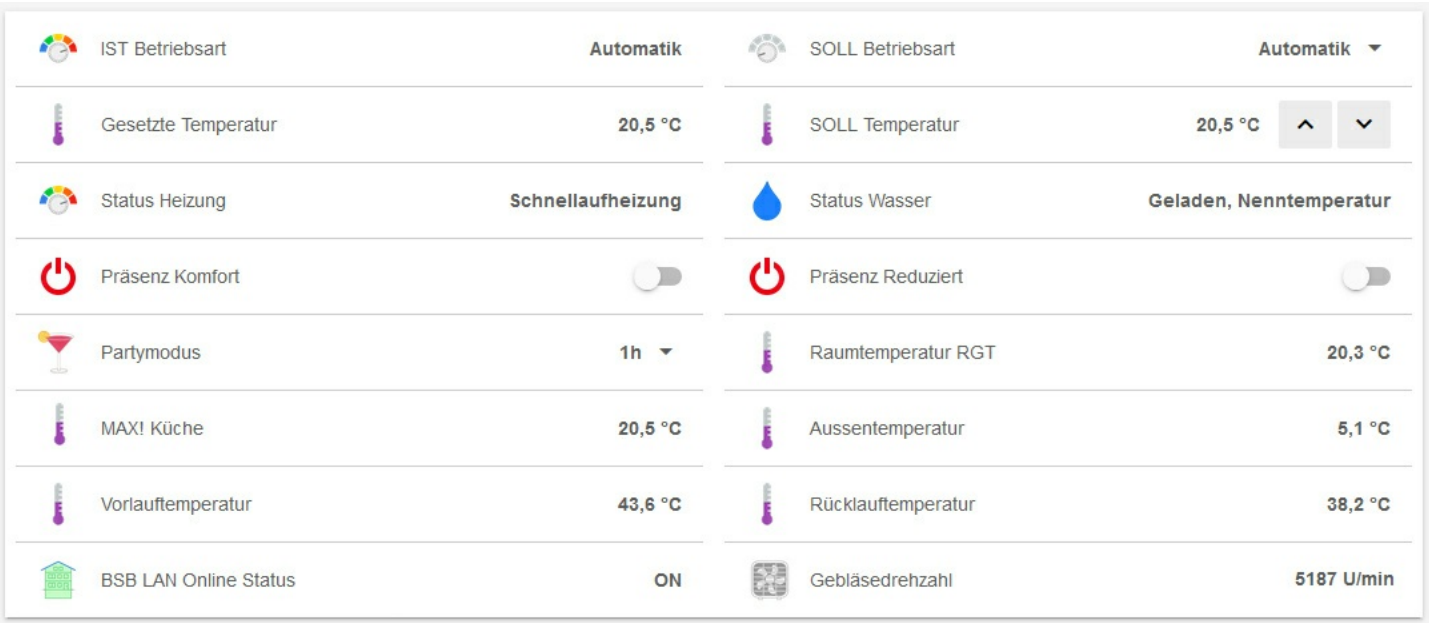
```
sitemap demo label="Mein BSB LAN" {
    Frame label="Heizung" {
        Text item=hz_aussentemp
        Text item=hz_700
    }
}
```

11.2.2 OpenHAB with Javascript Transformation, MQTT, Network and Expire

Based on the previous example, FHEM forum member „sihui" (GitHub: sihui62) wrote an expanded example.
Thanks a lot!

NOTE:
The neccessary addons like the the Javascript Transformation, MQTT, Network and Expire have to be installed previously!

The following example is shown as a sitemap in BasicUI like in the following screenshot:



Example of an item configuration (/items/bsblan.items):

```
Number hz_mode_cmd <heating> //change heating mode
String hz_mode_state <heating> { http="<[http://192.168.178.88/700:10000:JS(bsbinput_string.js)]" } //read heating mode from BSB LAN
 Adapter
Number hz_temperature_cmd <temperature> //change target temperature
Number hz_temperature_state <temperature> { http="<[http://192.168.178.88/8741:15000:JS(bsbinput.js)]" } //read current target tempe
rature from BSB LAN Adapter
String hz_status <heating> { http="<[http://192.168.178.88/8000:20000:JS(bsbinput_string.js)]" } //read current heating status from
BSB LAN Adapter
String hz_status_water <water> { http="<[http://192.168.178.88/8003:25000:JS(bsbinput_string.js)]" } //read current hot water status
 from BSB LAN Adapter
Switch hz_mode_komfort <switch> { expire="1s,command=OFF" } //ONLY if Parameter 48 is available on your controller: set temporary Ko
mfort state during Automatik mode, switch item to OFF after one second (momentary switch)
Switch hz_mode_reduziert <switch> { expire="1s,command=OFF" } //ONLY if Parameter 48 is available on your controller: set temporary
Reduziert state during Automatik mode, switch item to OFF after one second (momentary switch)
Number hz_temperature_rgt <temperature> { http="<[http://192.168.178.88/8740:25000:JS(bsbinput.js)]" } //read current room temperatu
re for remote RGT from BSB LAN Adapter
Number hz_fan_speed <fan> { http="<[http://192.168.178.88/8323:30000:JS(bsbinput.js)]" } //read current fan speed from BSB LAN Adapt
er
Number hz_aussentemp <temperature> { http="<[http://192.168.178.88/8700:20000:JS(bsbinput.js)]" } //read current outside temperature
 from BSB LAN Adapter via Javascript Transformation (not used here)
Number hz_kitchen_maxActual "MAX! Küche [%.1f °C]" {channel="max:thermostat:KEQ0565026:KEQ0648949:actual_temp"} //read temperature f
rom MAX!
Number BSBLAN_Aussentemp <temperature> { channel="mqtt:topic:bsblan:aussentemp" } //read current outside temperature from BSB LAN Ad
apter via MQTT2
Number BSBLAN_Vorlauftemp <temperature> { channel="mqtt:topic:bsblan:vorlauftemp" } //read current flow temperature from BSB LAN Ada
pter via MQTT2
Number BSBLAN_Ruecklauftemp <temperature> { channel="mqtt:topic:bsblan:ruecklauftemp" } //read current return temperature from BSB L
AN Adapter via MQTT2
Switch bsb_lan_presence <presence> { channel="network:pingdevice:192_168_178_88:online" } //check online status of BSB LAN through N
etwork binding
Number hz_mode_party <party> //enable or disable Party mode for 1-5 hours
```

The following Javascript has to be put as *bsbinput.js* in the folder *transform*.

Example script for the query of parameters which report a value (/transform/bsbinput.js):

```
(function(i) {
    var outputres;
    var results = [];
    value1 = i;
    // define regex to search for the line in the http response
    var regEx = 'input type=text id=\'value[0-9]+\' VALUE=\'[-]*[0-9]+\.[0-9]+';
    var re = new RegExp(regEx, 'gim');

 do {
    match = re.exec(value1);
    if (match){
        results.push(match[0]);
    }
} while (match);

    outputres = results[0]
    //extract actual value from the output
    var output=outputres.substr(outputres.indexOf("VALUE='")+7,outputres.length);
    return output;
})(input)
```

Example script for a direct query of enum-values (/transform/bsbinput_string.js):

```
(function(i) {
        var outputres;
    var results = [];
    value1 = i;
    // define regex to search for the line in the http response
    var regEx = '<option value=\'[0-9]+\' SELECTED>.*</option>';
    var re = new RegExp(regEx, 'gim');

 do {
    match = re.exec(value1);
    if (match){
        results.push(match[0]);
    }
} while (match);

    outputres = results[0]
    //extract actual value from the output
    var l=outputres.indexOf("</o")-outputres.indexOf(">")-1
    var output=outputres.substr(outputres.indexOf(">")+1,l);
        return output;
})(input)
```

Writing and reading data is done via rules (/rules/bsblan.rules):

```
var Timer PartyModeTimer = null //initialize a timer for party mode

rule "HeatingTempTarget" //change target temperature
when
    Item hz_temperature_cmd changed
then
    sendHttpGetRequest("http://192.168.178.88/S710="+hz_temperature_cmd.state.toString)
end

rule "HeatingMode" //change heating mode
when
    Item hz_mode_cmd changed
then
    sendHttpGetRequest("http://192.168.178.88/S700="+hz_mode_cmd.state.toString)
end

rule "UpdateHeatingMode" //reflect manual RGT remote changes on UI
when
    Item hz_mode_state changed
then
    hz_mode_cmd.postUpdate(transform("MAP","heatingmode.map",hz_mode_state.state.toString))
end

rule "SetModeKomfort" //set mode temporary to Komfort during Automatik mode
when
    Item hz_mode_komfort changed to ON
then
    sendHttpGetRequest("http://192.168.178.88/S701=0")
end

rule "SetModeReduziert" //set mode temporary to Reduziert during Automatik mode
when
    Item hz_mode_reduziert changed to ON
then
    sendHttpGetRequest("http://192.168.178.88/S701=1")
end

rule "SetPartyMode" //extends heating Komfort time for 1-5 hours
when
    Item hz_status changed
then
    // to do: read shutdown times for Absenkung Reduziert dynamically from BSB LAN Adapter
    if (hz_status.state.toString=="Absenkung Reduziert" && (now.getHourOfDay()>=22 && (now.getHourOfDay()<=23))) { //only trigger ru
le content during normal Reduziert shutdown times
        switch (hz_mode_party.state) {
                case 1: {
                if(PartyModeTimer!==null) {
                   PartyModeTimer.cancel
                   PartyModeTimer = null
                }
                    PartyModeTimer = createTimer(now.plusHours(1)) [ |
                    hz_mode_cmd.sendCommand(1)
                    logInfo("BSBLAN","Party Mode disabled")
                    ]
                hz_mode_cmd.sendCommand(3)
                hz_mode_party.postUpdate(0)
                logInfo("BSBLAN","Party Mode 1h")
                }
                case 2: {
                if(PartyModeTimer!==null) {
                   PartyModeTimer.cancel
```

```
                    PartyModeTimer = null
                }
                    PartyModeTimer = createTimer(now.plusHours(2)) [ |
                    hz_mode_cmd.sendCommand(1)
                    logInfo("BSBLAN","Party Mode disabled")
                    ]
                hz_mode_cmd.sendCommand(3)
                hz_mode_party.postUpdate(0)
                logInfo("BSBLAN","Party Mode 2h")
                }
                case 3: {
                if(PartyModeTimer!==null) {
                    PartyModeTimer.cancel
                    PartyModeTimer = null
                }
                    PartyModeTimer = createTimer(now.plusHours(3)) [ |
                    hz_mode_cmd.sendCommand(1)
                    logInfo("BSBLAN","Party Mode disabled")
                    ]
                hz_mode_cmd.sendCommand(3)
                hz_mode_party.postUpdate(0)
                logInfo("BSBLAN","Party Mode 3h")
                }
                case 4: {
                if(PartyModeTimer!==null) {
                    PartyModeTimer.cancel
                    PartyModeTimer = null
                }
                    PartyModeTimer = createTimer(now.plusHours(4)) [ |
                    hz_mode_cmd.sendCommand(1)
                    logInfo("BSBLAN","Party Mode disabled")
                    ]
                hz_mode_cmd.sendCommand(3)
                hz_mode_party.postUpdate(0)
                logInfo("BSBLAN","Party Mode 4h")
                }
                case 5: {
                if(PartyModeTimer!==null) {
                    PartyModeTimer.cancel
                    PartyModeTimer = null
                }
                    PartyModeTimer = createTimer(now.plusHours(5)) [ |
                    hz_mode_cmd.sendCommand(1)
                    logInfo("BSBLAN","Party Mode disabled")
                    ]
                hz_mode_cmd.sendCommand(3)
                hz_mode_party.postUpdate(0)
                logInfo("BSBLAN","Party Mode 5h")
                }
            }
    }
end

rule "ConsiderRoomTempFromKitchen" //feed external temperatures to controller, for example MAX!
when
    Item hz_kitchen_maxActual changed
then
    sendHttpGetRequest("http://192.168.178.88/I10000="+hz_kitchen_maxActual.state.toString)
end
```

Transformation of number values to readable text (/transform/heatingmode.map):

```
Automatik=1
Reduziert=2
Komfort=3
Schutzbetrieb=0
```

Display of the values in a sitemap (/sitemaps/bsblan.sitemap, e.g. for BasicUI, ClassicUI, iOS and Android App):

```
sitemap bsblan label="Mein BSB LAN"
{
Frame    {
            Text label="Heizung" icon="heating"
                {
                Text item=hz_mode_state label="IST Betriebsart [%s]"
                Selection item=hz_mode_cmd label="SOLL Betriebsart [%s]" mappings=[1="Automatik",3="Komfort",2="Reduziert"]
                Text item=hz_temperature_state label="Gesetzte Temperatur [%.1f °C]"
                Setpoint item=hz_temperature_cmd label="SOLL Temperatur [%.1f °C]" minValue=16 maxValue=24 step=0.5
                Text item=hz_status label="Status Heizung [%s]"
                Text item=hz_status_water label="Status Wasser [%s]"
                Switch item=hz_mode_komfort label="Präsenz Komfort"
                Switch item=hz_mode_reduziert label="Präsenz Reduziert"
                Selection item=hz_mode_party label="Partymodus [%s]" mappings=[0="Aus",1="1h",2="2h",3="3h",4="4h",5="5h"]
                Text item=hz_temperature_rgt label="Raumtemperatur RGT [%.1f °C]"
                Text item=hz_kitchen_maxActual label="MAX! Küche [%.1f °C]"
                Text item=BSBLAN_Aussentemp label="Aussentemperatur [%.1f °C]"
                Text item=BSBLAN_Vorlauftemp label="Vorlauftemperatur [%.1f °C]"
                Text item=BSBLAN_Ruecklauftemp label="Rücklauftemperatur [%.1f °C]"
                Text item=bsb_lan_presence label="BSB LAN Online Status [%s]"
                Text item=hz_fan_speed label="Gebläsedrehzahl [%d U/min]"
                }
        }
}
```

11.2.3 OpenHAB2 Binding

BSB-LAN user „hypetsch" developed a binding for openHAB2, which is officially part of openHAB2 since v2.5.4!
Thanks a lot!

11.3 HomeMatic (EQ3)

The following example scripts are written by FHEM forum member „Bratmaxe".
After that, the example scripts of "PaulM" are listed.
Thanks a lot!

Example script for a query:

There are only 6 parameters need to be set:
CuxGeraetAbfrage = Device address of the CuxD execute device, which does the queries
CuxGeraetLogging = Device address of the CuxD execute device, which does the logging (emptyr==deactivated)
IPAdresseBSB = IP-address of the BSB-LPB-LAN-adapter
Wort = Number of the desired parameter: e.g. outside temperature = 8700
Variablename = Name of the system variable in the CCU
Durchschnitt24h = true == Durchschnittswert 24h holen, false == aktuellen Wert holen

It's not necessary to create a variable before, that's done by the script.
The type of the variable (number, bool, value list) will be adjusted automatically to the queried parameter.

```
! BSB-Adapter Wert abfragen by Bratmaxe
! 29.10.2018 - V0.1 - Erste Version

string CuxGeraetAbfrage = "CUX2801001:1"; ! GeräteAdresse des CuxD Execute Gerätes, welches die Abfragen ausführt
string CuxGeraetLogging = "CUX2801001:1"; ! GeräteAdresse des CuxD Execute Gerätes, welches das Logging ausführt, Leer ("") lassen,
wenn kein Cuxd-Highcharts Logging gewünscht
string IPAdresseBSB = "192.168.178.100"; !IP_Adresse des BSB-Adapters
string Wort = "8700"; !Parameternummer: Beispiel Außentemperatur = 8700, Betriebsmodus = 700
string Variablename = "Wetter_Temperatur_Heizung"; ! Name der Systemvariable
boolean Durchschnitt24h = false; ! true = Durchschnittswert holen, false = aktuellen Wert holen - diese muss vorher in der BSB_lan_c
onfig.h konfiguriert wurden sein!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!Ab hier keine Anpassungen mehr notwendig!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! URL Zusammenführen
string url="";
if (Durchschnitt24h) { url="http://" # IPAdresseBSB # "/A" # Wort; }
else { url="http://" # IPAdresseBSB # "/" # Wort; }
! Variable anlegen, wenn nicht vorhanden:
object svObject = dom.GetObject(Variablename);
object svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
if (!svObject)
```

```
{
    svObject = dom.CreateObject(OT_VARDP);
    svObjectlist.Add(svObject.ID());
    svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
    svObject.Name(Variablename);
    svObject.Internal(false);
    svObject.Visible(true);
}

! Werte holen
dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #"'")
;
dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();

! Prüfe, ob eine Ausgabe vorhanden ist, sonst z.B. IP-Adresse falsch, oder Netzwerkfehler
if (stdout != null && stdout != "")
{
    ! Ausgabe filtern
    integer pos = (stdout.Find(Wort# " "));
    if (pos == -1)
    {
        WriteLine("Position vom Wort '" # Wort # "' konnte nicht ermittelt werden");
    }

    stdout = stdout.Substr(pos, stdout.Length());
    pos = stdout.Find("/td");
    stdout = stdout.Substr(0, pos);

    ! Sonderzeichen ersetzen
    if (stdout.Contains("&deg;")){ stdout = stdout.Replace("&deg;","°"); }
    if (stdout.Contains("&#037;")){ stdout = stdout.Replace("&#037;","%"); }
    stdout = stdout.ToLatin();
    !WriteLine("Nach Sonderzeichenumwandlung: " # stdout); !Debug: Welchen Wert hat stdout aktuell

    ! Systemvariabel Info ermitteln
    string Info = stdout.Substr(0,stdout.Find(":"));
    !Info = Info.Substr(Wort.Length(), stdout.Length()); !Parameterzahl vor der Info entfernen
    !WriteLine("DPInfo = " # Info); !Debug: Welcher DPInfo-Wert wurde gefunden

    ! Systemvariabel Wert ermitteln
    string Wert = stdout.Substr(stdout.Find(": ") + 2,stdout.Length());
    Wert = Wert.Substr(0,Wert.Find(" "));
    !WriteLine("Wert = " # Wert); !Debug: Welcher Wert wurde gefunden

    ! Systemvariabel Einheit ermitteln
    string Einheit = stdout.Substr(stdout.Find(Info) + Info.Length() + 1, stdout.Length());
    Einheit = Einheit.Substr(Einheit.Find(Wert) + Wert.Length() + 1,Einheit.Length());
    Einheit = Einheit.RTrim();
    if (Einheit.Contains("- "))    { Einheit = ""; }
    !WriteLine("Einheit = " # Einheit); !Debug: Welche Einheit wurde gefunden

    ! Systemvariable Typ und Werte setzen
    svObject.DPInfo(Info);
    svObject.ValueUnit(Einheit);

    ! Enums des Parameters ermitteln, wenn vorhanden
    url="http://" # IPAdresseBSB # "/E" # Wort;

    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #
"'");
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
    stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();
    ! Prüfe, ob es sich um einen Parameter mit Enum-Werten handelt.
    if (!stdout.Contains("FEHLER: Falscher Typ!"))
    {
        ! Setzen des Systemvariabel Wertetyp und Ermitteln der Enum-Werte des Parameters
        stdout = (stdout.Substr(stdout.Find("0 - "), stdout.Length())).ToLatin();
        string value = "";
        string newvalues = "";
        integer inewvalues=0;
        foreach (value, stdout.Split("\r"))
        {
            if (value.Contains(" - "))
            {
                if (newvalues == "") { newvalues = newvalues # value.Substr(value.Find(" - ") + 3,value.Length()); }
                else { newvalues = newvalues # ";" # value.Substr(value.Find(" - ") + 3,value.Length()); }
                inewvalues = inewvalues + 1;
            }
        }

        svObject.ValueType(ivtInteger);
        svObject.ValueSubType(istEnum);
        svObject.ValueList(newvalues);
        !prüft, ob der ermittelte Wert innerhalbe der möglichen Werte liegt
        if (Wert < inewvalues) { if (Wert != svObject.Value()) { svObject.State(Wert); } }
        else { WriteLine("Der ermittelte Wert entspricht keinem gültigen Enum-Wert. Bitte Ausgabe prüfen!") }
    }
```

```
    }
    elseif (Einheit.Contains("- Aus") || Einheit.Contains("- Ein"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtBinary);
        svObject.ValueSubType(istBool);
        svObject.ValueName0("Aus");
        svObject.ValueName1("Ein");
        if (Wert != svObject.Value()) {    svObject.State(Wert); }
    }
    elseif (Einheit.Contains("°"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        svObject.ValueMin(-50);
        svObject.ValueMax(100);
        if (Wert != svObject.Value()) {    svObject.State(Wert); }
    }
    elseif (Einheit.Contains("%"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        svObject.ValueMin(0);
        svObject.ValueMax(100);
        if (Wert != svObject.Value()) {    svObject.State(Wert); }
    }
    else
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        if (Wert != svObject.Value()) {    svObject.State(Wert); }
    }
    dom.RTUpdate(0); ! Interne Aktualisierung der Systemvariabelen

    ! Logging
    if (CuxGeraetLogging != "")    { dom.GetObject("CUxD."#CuxGeraetLogging#".LOGIT").State(dom.GetObject(ID_SYSTEM_VARIABLES).Get(V
ariablename).Name()#";"#dom.GetObject(ID_SYSTEM_VARIABLES).Get(Variablename).Value());    }
}
```

Script for setting parameters:

Create a program where all system variables that should be observed are associated with ODER and bigger than or equal with 0 and "bei Aktualisierung auslösen" (triggered by actualisation).
Example:

```
WENN Variablename größer oder gleich 0 "bei Aktualisierung auslösen"
DANN Dieses SKRIPT sofort ausführen
```

This variable need to include the value of the parameter in the info first (will be named by the read-out-script from above automatically). Example: 700 heating circuit 1 - operating mode
The number of the parameter will be detected automatically from the system variable 'Info'.
If the variable will be changed, the changed value will be send and actualized to the adapter automatically!

```
! BSB-Adapter Wert setzen by Bratmaxe
! 29.10.2018 - V0.1 - Erste Version

! Funktionsbeschreibung:
! Ein Programm, wo alle Systemvariabeln die Überwacht werden sollen mit ODER Verknüpft und größer oder gleich 0 und  "bei Aktualisie
rung auslösen" , anlegen.
! Beispiel:
! WENN Variablename größer oder gleich 0 "bei Aktualisierung auslösen"
! DANN Dieses SKRIPT sofort ausführen
! die Variable muss in der Info zuerst den Parameter-Wert enthalten (wird von meinem Auslese Skript automatisch so benannt. Beispiel
: 700 Heizkreis 1 - Betriebsart
! Die Parameternummer wird dann automatisch aus der Systemvariable Info ermittelt.
! Wird die Variable geändert, so wird der geänderte Wert automatisch an den BSB-Adapter übermittelt und aktualisiert!

string CuxGeraetSetzen = "CUX2801001:12"; ! GeräteAdresse des CuxD Execute Gerätes
string IPAdresseBSB = "192.168.2.200"; !IP_Adresse des BSB-Adapters

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!Ab hier keine Anpassungen mehr notwendig!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Hole Auslösende Variabel
var source = dom.GetObject("$src$"); !Funktioniert nur beim automatischen Aufruf
! Zum manuellen Aufruf/testen nächste Zeile einkommentieren
!source = dom.GetObject(ID_SYSTEM_VARIABLES).Get("VARIABLENAMEN");
```

```
if (source)
{
    ! Wort ermitteln
    string Wort = source.DPInfo().ToString().Substr(0,source.DPInfo().Find(" "));
    !WriteLine("Wort: "#Wort);
    if (Wort != null && Wort != "")
    {
        string Wert = source.Value().ToString();
        !WriteLine("Wert: "#Wert);
        if (Wert != null && Wert != "")
        {
            ! Anweisung senden
            string urlset="http://" # IPAdresseBSB # "/S" # Wort # "=" # Wert;
            dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_SETS").State("wget -t 5 -T 20 -q -O - '"# urlset #"'");
            dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_QUERY_RET").State(1);
            var stdout = dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_RETS").State();
            if (stdout != null && stdout != "")
            {
                if (stdout.Contains("FEHLER: "))
                {
                    stdout = stdout.Substr(stdout.Find("FEHLER: "), stdout.Length());
                    stdout = stdout.Substr(0, stdout.Find("/td"));
                    WriteLine("Fehlermeldung: "# stdout);
                    WriteLine("Wurde der BSB-Adapter zum Schreiben berechtigt? Handbuch Seite 26 beachten...");
                }
                else
                {
                    ! Kontrollabfrage
                    string url="http://" # IPAdresseBSB # "/" # Wort;
                    dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_SETS").State("wget -t 5 -T 20 -q -O - '"# url #"'");
                    dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_QUERY_RET").State(1);
                    stdout = dom.GetObject("CUxD." # CuxGeraetSetzen # ".CMD_RETS").State();

                    ! Ausgabe filtern
                    integer pos = (stdout.Find("tr  td \r\n" # Wort # " ") + 9);
                    stdout = stdout.Substr(pos, stdout.Length());
                    pos = stdout.Find("/td");
                    stdout = stdout.Substr(0, pos);

                    ! Sonderzeichen ersetzen
                    if (stdout.Contains("&deg;")){ stdout = stdout.Replace("&deg;","°"); }
                    if (stdout.Contains("&#037;")){ stdout = stdout.Replace("&#037;","%"); }
                    !WriteLine("Nach Sonderzeichenumwandlung: " # stdout); !Debug: Welchen Wert hat stdout aktuell

                    ! Systemvariabel oldWert ermitteln
                    string oldWert = stdout.Substr(stdout.Find(": ") + 2,stdout.Length());
                    oldWert = oldWert.Substr(0,oldWert.Find(" "));
                    !WriteLine("oldWert = " # oldWert.ToFloat()); !Debug: Welcher oldWert wurde gefunden
                    !WriteLine("newWert = " # Wert.ToFloat()); !Debug: Welcher oldWert wurde gefunden

                    if (Wert.ToFloat() != oldWert.ToFloat()) { WriteLine("Fehler: Werte stimmen nach setzen nicht überein!"); }
                    else { WriteLine("Wert wurde erfolgreich gesetzt");    }
                }
            }
            else { WriteLine("Keine Ausgabe gefunden. IP-Adresse und Verkabelung prüfen.");    }
        }
        else { WriteLine("Der neue Wert konnte nicht ermittelt werden.");    }
    }
    else { WriteLine("Wort konnte nicht ermittelt werden, Steht der Wert in der SystemvariableInfo am Anfang gefolgt von einem Leerz
eichen?");    }
}
else { WriteLine("Auslösende Variable nicht erkannt! - Skript wird nicht ausgeführt.");    }
```

Query of the errors of the controller to realize a notification if an error occurs:

```
! BSB-Adapter Wert abfragen Fehlercodes by Bratmaxe
! 05.11.2018 - V0.1 - Erste Version

string CuxGeraetAbfrage = "CUX2801001:1"; ! GeräteAdresse des CuxD Execute Gerätes, welches die Abfragen ausführt
string IPAdresseBSB = "192.168.178.100"; !IP_Adresse des BSB-Adapters
string Variablename = "Heizung_Fehlercodes"; ! Name der Systemvariable
integer AnzahlFehler = 10;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!Ab hier keine Anpassungen mehr notwendig!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Parameter Zusammenbauen
integer i =0;
string Woerter ="";
while (i < AnzahlFehler)
{
    if (Woerter != "")
    {
        Woerter = Woerter + " " + ((6801) + (10 * i)).ToString();
```

```
    }
    else { Woerter = Woerter + ((6801) + (10 * i)).ToString(); }
    i = i + 1;
}

! URL Zusammenführen
string Ergebnis = "";
string Wort = "";

foreach(Wort, Woerter.Split(","))
{
    string url="http://" # IPAdresseBSB # "/" # ((Wort.ToInteger() - 1).ToString());

    ! Werte holen
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #
"'");
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
    var stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();

    ! Prüfe, ob eine Ausgabe vorhanden ist, sonst z.B. IP-Adresse falsch, oder Netzwerkfehler
    if (stdout != null && stdout != "")
    {
        ! Ausgabe filtern
        integer pos = (stdout.Find((Wort.ToInteger() - 1).ToString() #  " "));
        stdout = stdout.Substr(pos, stdout.Length());
        pos = stdout.Find("/td");
        stdout = stdout.Substr(0, pos);

        ! Sonderzeichen ersetzen
        if (stdout.Contains("°")){ stdout = stdout.Replace("°","°"); }
        if (stdout.Contains("%")){ stdout = stdout.Replace("%","%"); }
        stdout = stdout.ToLatin();
        Ergebnis = Ergebnis # stdout.RTrim() # "\n\r";
    }

    url="http://" # IPAdresseBSB # "/" # Wort;
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #
"'");
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
    stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();

    ! Prüfe, ob eine Ausgabe vorhanden ist, sonst z.B. IP-Adresse falsch, oder Netzwerkfehler
    if (stdout != null && stdout != "")
    {
        ! Ausgabe filtern
        integer pos = (stdout.Find(Wort# " "));
        stdout = stdout.Substr(pos, stdout.Length());
        pos = stdout.Find("/td");
        stdout = stdout.Substr(0, pos);

        ! Sonderzeichen ersetzen
        if (stdout.Contains("°")){ stdout = stdout.Replace("°","°"); }
        if (stdout.Contains("%")){ stdout = stdout.Replace("%","%"); }
        stdout = stdout.ToLatin();
        Ergebnis = Ergebnis # stdout.RTrim() # "\n\r\n\r";
    }
}

!Wenn noch keine Systemvarible vorhanden, diese anlegen
object svObject = dom.GetObject(Variablename);
object svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
if (!svObject)
{
    svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
    svObject = dom.CreateObject(OT_VARDP);
    svObjectlist.Add(svObject.ID());
    svObject.Name(Variablename);
    svObject.ValueType(ivtString);
    svObject.ValueSubType(istChar8859);
    svObject.DPInfo("Die letzen 20 Fehlercodes der Heizung");
    svObject.Internal(false);
    svObject.Visible(true);
    dom.RTUpdate(0);
}

if (Ergebnis.ToLatin() != svObject.Value().ToLatin()) { svObject.State(Ergebnis); }
```

Query of parameters and additionally connected DS18B20 temperature sensors, using the specific sensor ids and the query of /T:

```
! BSB-Adapter Wert abfragen by Bratmaxe
! 29.10.2018 - V0.1 - Erste Version
! 11.11.2019 - V0.2 - Auslesen von Temperatursensoren hinzugefügt
! 15.11.2019 - V0.3 - Änderung der Ausleseart der Temperatursensoren mithilfe der ID

string CuxGeraetAbfrage = "CUX2801001:11"; ! GeräteAdresse des CuxD Execute Gerätes, welches die Abfragen ausführt
```

```
string CuxGeraetLogging = "CUX2801001:10"; ! GeräteAdresse des CuxD Execute Gerätes, welches das Logging ausführt, Leer ("") lassen,
 wenn kein Cuxd-Highcharts Logging gewünscht
string IPAdresseBSB = "192.168.178.88"; !IP_Adresse des BSB-Adapters
string Wort = "T"; !Parameternummer: Beispiel Außentemperatur = 8700, Betriebsmodus = 700, eigene Temperatursensoren = T
string TemperatursensorID = "28aa44085414010b"; !Wenn Wort = "T", dann hier die ID des auszulesenden Temperatursensors eingeben, wir
d sonst ignoriert!
string Variablename = "Wetter_Temperatur"; ! Name der Systemvariable
boolean Durchschnitt24h = false; ! true = Durchschnittswert holen, false = aktuellen Wert holen - diese muss vorher in der BSB_lan_c
onfig.h konfiguriert wurden sein!!! (Bei Wort = T wird dieser Parameter ignoriert)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!Ab hier keine Anpassungen mehr notwendig!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! URL Zusammenführen
string url="";
if (Durchschnitt24h && Wort != "T")
{
    url="http://" # IPAdresseBSB # "/A" # Wort;
}
else
{
    url="http://" # IPAdresseBSB # "/" # Wort;
}

! Variable anlegen, wenn nicht vorhanden:
object svObject = dom.GetObject(Variablename);
object svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
if (!svObject)
{
    svObject = dom.CreateObject(OT_VARDP);
    svObjectlist.Add(svObject.ID());
    svObjectlist = dom.GetObject(ID_SYSTEM_VARIABLES);
    svObject.Name(Variablename);
    svObject.Internal(false);
    svObject.Visible(true);
}

! Werte holen
dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #"'")
;
dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();

! Prüfe, ob eine Ausgabe vorhanden ist, sonst z.B. IP-Adresse falsch, oder Netzwerkfehler
if (stdout != null && stdout != "")
{
    integer pos = (stdout.Find(Wort# " "));

    ! Ausgabe filtern
    if (Wort == "T")
    {
        pos = (stdout.Find(TemperatursensorID # ": "));
    }

    if (pos == -1)
    {
        WriteLine("Position vom Wort '" # Wort # "' konnte nicht ermittelt werden");
    }

    stdout = stdout.Substr(pos, stdout.Length());
    pos = stdout.Find("/td");
    stdout = stdout.Substr(0, pos);

    ! Sonderzeichen ersetzen
    if (stdout.Contains("&deg;")){ stdout = stdout.Replace("&deg;","°"); } !& d e g ; ohne Leerzeichen
    if (stdout.Contains("%")){ stdout = stdout.Replace("%","%"); } !& # 0 3 7 ; ohne Leerzeichen
    !WriteLine("Nach Sonderzeichenumwandlung: " # stdout); !Debug: Welchen Wert hat stdout aktuell

    ! Systemvariabel Info ermitteln
    string Info = "";
    if (Wort == "T")
    {
        Info = "SensorID: " + TemperatursensorID;
    }
    else
    {
        Info = stdout.Substr(0,stdout.Find(":"));
    }
    !Info = Info.Substr(Wort.Length(), stdout.Length());
    !WriteLine("DPInfo = " # Info); !Debug: Welcher DPInfo-Wert wurde gefunden

    ! Systemvariabel Wert ermitteln
    string Wert = stdout.Substr(stdout.Find(": ") + 2,stdout.Length());
    Wert = Wert.Substr(0,Wert.Find(" "));
    !WriteLine("Wert = " # Wert); !Debug: Welcher Wert wurde gefunden

    ! Systemvariabel Einheit ermitteln
```

```
    ! Systemvariabel Einheit ermitteln
    string Einheit = stdout.Substr(stdout.Find(Info) + Info.Length() + 1, stdout.Length());
    Einheit = Einheit.Substr(Einheit.Find(Wert) + Wert.Length() + 1,Einheit.Length());
    Einheit = Einheit.RTrim();
    if (Einheit.Contains("- "))    { Einheit = ""; }
    !WriteLine("Einheit = " # Einheit); !Debug: Welche Einheit wurde gefunden

    ! Systemvariable Typ und Werte setzen
    svObject.DPInfo(Info);
    svObject.ValueUnit(Einheit);

    ! Enums des Parameters ermitteln, wenn vorhanden
    url="http://" # IPAdresseBSB # "/E" # Wort;

    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #
"'");
    dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_QUERY_RET").State(1);
    stdout = dom.GetObject("CUxD." # CuxGeraetAbfrage # ".CMD_RETS").State();
    ! Prüfe, ob es sich um einen Parameter mit Enum-Werten handelt.
    if (!stdout.Contains("FEHLER: Falscher Typ!"))
    {
        ! Setzen des Systemvariabel Wertetyp und Ermitteln der Enum-Werte des Parameters
        stdout = (stdout.Substr(stdout.Find("0 - "), stdout.Length())).ToLatin();
        string value = "";
        string newvalues = "";
        integer inewvalues=0;
        foreach (value, stdout.Split("\r"))
        {
            if (value.Contains(" - "))
            {
                if (newvalues == "") { newvalues = newvalues # value.Substr(value.Find(" - ") + 3,value.Length()); }
                else { newvalues = newvalues # ";" # value.Substr(value.Find(" - ") + 3,value.Length()); }
                inewvalues = inewvalues + 1;
            }
        }

        svObject.ValueType(ivtInteger);
        svObject.ValueSubType(istEnum);
        svObject.ValueList(newvalues);
        !prüft, ob der ermittelte Wert innerhalbe der möglichen Werte liegt
        if (Wert < inewvalues) { if (Wert != svObject.Value()) { if (Wert != "") { svObject.State(Wert); }} }
        else { WriteLine("Der ermittelte Wert entspricht keinem gültigen Enum-Wert. Bitte Ausgabe prüfen!") }
    }
    elseif (Einheit.Contains("- Aus") || Einheit.Contains("- Ein"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtBinary);
        svObject.ValueSubType(istBool);
        svObject.ValueName0("Aus");
        svObject.ValueName1("Ein");
        if (Wert != svObject.Value()) {    if (Wert != "") { svObject.State(Wert); } }
    }
    elseif (Einheit.Contains("°"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        svObject.ValueMin(-50);
        svObject.ValueMax(100);
        if (Wert != svObject.Value()) {    if (Wert != "") { svObject.State(Wert); } }
    }
    elseif (Einheit.Contains("%"))
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        svObject.ValueMin(0);
        svObject.ValueMax(100);
        if (Wert != svObject.Value()) {    if (Wert != "") { svObject.State(Wert); } }
    }
    else
    {
        ! Setzen des Systemvariabel Wertetyp
        svObject.ValueType(ivtFloat);
        svObject.ValueSubType(istGeneric);
        if (Wert != svObject.Value()) {    if (Wert != "") { svObject.State(Wert); } }
    }
    dom.RTUpdate(0); ! Interne Aktualisierung der Systemvariabelen

    ! Logging
    if (CuxGeraetLogging != "")    { dom.GetObject("CUxD."#CuxGeraetLogging#".LOGIT").State(dom.GetObject(ID_SYSTEM_VARIABLES).Get(V
ariablename).Name()#";"#dom.GetObject(ID_SYSTEM_VARIABLES).Get(Variablename).Value());    }
}
```

The following HomeMatic-scripts are developed by FHEM forum member „PaulM".
Thanks a lot!

For implementation in HomeMatic the usage of CuxD and wget is a good option.

Example for the query of the parameter '8326 burner modulation' by using CuxD:

```
! Skriptanfang:
! BSB-Abfrage
string url='http://192.168.178.88/8326'; !IP anpassen
! WriteLine("URL: " # url); !nur Kontrolle
! --timeout=seconds  ! während der Dauer der Abfrage werden von der CCU
! keine anderen Skripte ausgeführt !!!
! siehe CUxD-Handbuch 5.8.2 System.Exec
! dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State
! ("wget -t 5 -T 20 -q -O - '"# url #"'");  ! abgekürzte Wget Syntax
dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #"'");
! ausführliche Wget Syntax
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
! WriteLine("Antwort: " # stdout);
! Beim stdout den ueberfluessigen Vorspann entfernen
integer laenge = stdout.Length();
integer pos = stdout.Find("body");
! ab hier kommen auswertbare Informationen
pos = pos + 10;
stdout = stdout.Substr(pos, (laenge - pos));
! wenn Rückmeldung mit allen Daten angezeigt werden soll,
! Ausrufezeichen nächste Zeile entfernen
! WriteLine("Antwort ohne Vorspann: " # stdout);
string wort = "8326"; !Brennermodulation
integer laenge = wort.Length();
! WriteLine("laenge: " # laenge); zum Testen für andere Parameter
! für Skripttest Ausrufezeichen entfernen
integer pos = stdout.Find(wort);
! WriteLine("pos:" #pos);
pos = pos + 39; !bei anderen Parametern meist zwischen 25 und 50
string daten = stdout.Substr((pos + laenge +1), 100);
! WriteLine("daten :"#daten);
integer pos = daten.Find(wort);
daten = daten.Substr(0, (pos));
integer zahl = daten.ToFloat();
! keine Prüfung, da auch 0 sein kann
dom.GetObject("H_Brennermodulation").State(zahl);
WriteLine("H_Brennermodulation: "#zahl);
! nur für Chart CUxD
dom.GetObject("CUxD.CUX2801001:1.LOGIT").State
("H_Brennermodulation"#";"#zahl);
WriteLine("Hallo Welt!");
! Skriptende:
```

Example for setting the mode to comfort with ‚S700=3' using CuxD:

```
! Skriptanfang:
! Heizung KOMFORT (=AN - keine Nachtabsenkung)
! Anweisung senden
string urlset ='http://192.168.178.88/S700=3'; !IP anpassen
WriteLine("Befehl: " # urlset);
dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State
("wget -t 5 -T 20 -q -O - '"# urlset #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
! Kontrollabfrage
string url='http://192.168.178.88/700'; !IP anpassen
! WriteLine("URL: " # url);
dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State
("wget -t 5 -T 20 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
! WriteLine("Antwort: " # stdout);
! Beim stdout den ueberfluessigen Vorspann entfernen
integer laenge = stdout.Length();
integer pos = stdout.Find("body");
pos = pos + 10;
stdout = stdout.Substr(pos, (laenge - pos));
! wenn Rückmeldung mit allen Daten angezeigt werden soll,
! Ausrufezeichen nächste Zeile entfernen
! WriteLine("Antwort ohne Vorspann: " # stdout);
string wort = "700"; !Heizbetrieb
integer laenge = wort.Length();
! WriteLine("laenge: " # laenge);
integer pos = stdout.Find(wort);
! WriteLine("pos:" #pos);
pos = pos + 28;
! WriteLine("pos: " # pos);
string daten = stdout.Substr((pos + laenge +1), 100);
! WriteLine("daten :"#daten);
string substring = daten.Substr(0, 1);
integer zahl = substring.ToInteger();
WriteLine("aktueller Heizbetrieb (0 bis 3): " # zahl);
if (zahl == 0) {dom.GetObject('H_Heizbetrieb').State("Heizung AUS");}
if (zahl == 1) {dom.GetObject('H_Heizbetrieb').State("Heizung Automatik");}
if (zahl == 2) {dom.GetObject('H_Heizbetrieb').State("Heizung Nachtabsenkung");}
if (zahl == 3) {dom.GetObject('H_Heizbetrieb').State("Heizung Komfort");}
! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Heizbetrieb"#";"#zahl);
! alle Werte aktualisieren
var programObj = dom.GetObject("Heizungswerte abfragen");
! Programmnamen ggf. anpassen
programObj.ProgramExecute();
WriteLine("Hallo Welt!");
! Skriptende:
```

Query and save as a system variable:

Ferienbetrieb von-bis / Absenkniveau (632/633/648)

Heiz- und Warmwasserbetrieb (700/1600/8700/8326/8743/8314/8830)

Übertrag der von einem One-Wire Sensor gemessenen Raumtemperatur zum BSB (I10000)

Protokollieren (für Auswertungen mit CUxD Highcharts)

```
!Heizung Abfrage V17 mit CUxD 2018-02-17

!wget mit ausführl. Syntax 2018-02-17
!inkl. Daten Ferienbetrieb und -niveau 2018-01-31

!Prüfung of Rückgabewerte ungleich Null
!auch senden an CUxD-Geräte Tempsens_H_*

!H_Trinkwassertemperatur korrigiert +1.3  2017-08-11
!mit Übertrag der Raumtemperatur an die Heizung Parameter 8740 2017-05-13
!mit OneWire Sensoren 2017-04-26

!als Systemvariablen sind in Homematic angelegt:
!H_Ferien_Beginn Zeichenkette
!H_Ferien_Ende Zeichenkette
!H_Ferienniveau Zeichenkette

!H_Aussentemperatur    Zahl
!H_Brennermodulation    Zahl
!H_Vorlauftemperatur    Zahl
!H_Kesselruecklauftemperatur    Zahl
!H_Trinkwassertemperatur    Zahl
!H_Heizbetrieb     Zeichenkette
!H_Trinkwasserbetrieb    Zeichenkette

!632    Beginn Ferienbetrieb    TT.MM
!633    Ende Ferienbetrieb    TT.MM
```

```
!648  Frostschutz / Reduziert

string url='http://192.168.10.13/632/633/648/700/1600/8700/8326/8743/8314/8830/T';

!WriteLine("URL: " # url);

!--timeout=seconds  ! während der Dauer der Abfrage werden von der CCU keine anderen Skripte ausgeführt
! CUxD-Handbuch 5.8.2 System.Exec "Es ist zu beachten, dass die Verarbeitung des HM-Scripts erst fortgesetzt wird, nachdem das aufge
rufene Programm beendet wurde. Während dieser Zeit werden keine anderen HM-Scripts ausgeführt!"

!dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");  ! abgekürzte Wget Syntax
dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget --tries=5 --timeout=20 --quiet --output-document=- '"# url #"'");  ! ausführ
liche Wget Syntax
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
!WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    !WriteLine("Antwort ohne Vorspann: " # stdout);

string wort = "632"; !Ferien Heizkreis 1 Beginn TT.MM
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 48;
    !WriteLine("pos: " # pos);
    string daten = stdout.Substr((pos + laenge +1), 13);
    daten = daten.Substr(2,5);
    !WriteLine("daten :"#daten);
    dom.GetObject('H_Ferien_Beginn').State("Ferien ab: ???"); ! setzt vorherigen Wert zurück
    dom.GetObject('H_Ferien_Beginn').State("Ferien ab: "# daten);
    var temp = dom.GetObject('H_Ferien_Beginn').Value();
    WriteLine("Ferien Heizkreis 1 Beginn TT.MM: " # temp);

string wort = "633"; !Ferien Heizkreis 1 Ende TT.MM
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 48;
    !WriteLine("pos: " # pos);
    string daten = stdout.Substr((pos + laenge +1), 15);
    daten = daten.Substr(0,5);
    !WriteLine("daten :"#daten);
    dom.GetObject('H_Ferien_Ende').State("Ferien bis: ???"); ! setzt vorherigen Wert zurück
    dom.GetObject('H_Ferien_Ende').State("Ferien bis: "# daten);
    var temp = dom.GetObject('H_Ferien_Ende').Value();
    WriteLine("Ferien Heizkreis 1 Ende TT.MM: " # temp);

string wort = "648"; !Betriebsniveau Ferien
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 48;
    !WriteLine("pos: " # pos);
    string daten = stdout.Substr((pos + laenge +1), 15);
    daten = daten.Substr(0,12);
    !WriteLine("daten :"#daten);
    dom.GetObject('H_Ferienniveau').State("Ferienniveau: ???"); ! setzt vorherigen Wert zurück
    dom.GetObject('H_Ferienniveau').State("Ferienniveau: "# daten);
    var temp = dom.GetObject('H_Ferienniveau').Value();
    WriteLine("Betriebsniveau Ferien: " # temp);


string wort = "700"; !Heizbetrieb
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 28;
    !WriteLine("pos: " # pos);
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    string substring = daten.Substr(0, 1);
    integer zahl = substring.ToInteger();
    !WriteLine("zahl: " # zahl);
        if (zahl == 0) {dom.GetObject('H_Heizbetrieb').State("Heizung AUS");}
        if (zahl == 1) {dom.GetObject('H_Heizbetrieb').State("Heizung Automatik");}
        if (zahl == 2) {dom.GetObject('H_Heizbetrieb').State("Heizung Nachtabsenkung");}
        if (zahl == 3) {dom.GetObject('H_Heizbetrieb').State("Heizung Komfort");}
```

```
! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Heizbetrieb"#";"#zahl);


string wort = "1600"; !Trinkwasserbetrieb
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 35;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    string substring = daten.Substr(0, 1);
    integer zahl = substring.ToInteger();
    !WriteLine("zahl: " # zahl);
        if (zahl == 0) {dom.GetObject('H_Trinkwasserbetrieb').State("Warmwasserbetrieb - AUS");}
        if (zahl == 1) {dom.GetObject('H_Trinkwasserbetrieb').State("Warmwasserbetrieb - EIN");}

    ! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Trinkwasserbetrieb"#";"#zahl);


string wort = "8700"; !Aussentemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 41;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    !Korrektur des angezeigten Wertes auf das Niveau der übrige Thermometer
    if (zahl<>0)
        {
        zahl = zahl - 3.5;
        dom.GetObject("H_Aussentemperatur").State(zahl);
        WriteLine("H_Aussentemperatur: "#zahl);

        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Aussentemperatur"#";"#zahl);

        ! für CUxD Gerät Tempsens_H_Aussen
        dom.GetObject("CUxD.CUX9002012:1.SET_TEMPERATURE").State(zahl);
        }

string wort = "8326"; !Brennermodulation
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 39;;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    !keine Prüfung, da auch 0 sein kann
    dom.GetObject("H_Brennermodulation").State(zahl);
    WriteLine("H_Brennermodulation: "#zahl);

    ! nur für Chart CUxD
    dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Brennermodulation"#";"#zahl);

string wort = "8743"; !Vorlauftemperatur 1:
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 45;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    if (zahl<>0)
        {
        dom.GetObject("H_Vorlauftemperatur").State(zahl);
        WriteLine("H_Vorlauftemperatur: "#zahl);

        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Vorlauftemperatur"#";"#zahl);

        ! für CUxD Gerät Tempsens_H_KesselVor
        dom.GetObject("CUxD.CUX9002014:1.SET_TEMPERATURE").State(zahl);
        }
```

```
string wort = "8314"; !Kesselrücklauftemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 52;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    if (zahl<>0)
        {
        dom.GetObject("H_Kesselruecklauftemperatur").State(zahl);
        WriteLine("H_Kesselruecklauftemperatur: "#zahl);

        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Kesselruecklauftemperatur"#";"#zahl);

        ! für CUxD Gerät Tempsens_H_KesselRue
        dom.GetObject("CUxD.CUX9002013:1.SET_TEMPERATURE").State(zahl);
        }

string wort = "8830"; !Trinkwassertemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 48;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    !Korrektur relativ zur Kesselvorlauftemperatur
    if (zahl<>0)
        {
        zahl = zahl + 1.3;
        dom.GetObject("H_Trinkwassertemperatur").State(zahl);
        WriteLine("H_Trinkwassertemperatur "#zahl);

        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Trinkwassertemperatur"#";"#zahl);

        ! für CUxD Gerät Tempsens_H_Trinkw
        dom.GetObject("CUxD.CUX9002015:1.SET_TEMPERATURE").State(zahl);
        }

!Übertrag der Raumtemperatur Esszimmer  ...4d6 an die Heizung als Parameter 8740

    real temp = dom.GetObject("T_Innentemperatur_Esszimmer").Value();
    temp = temp.ToString();

    string urlset = 'http://192.168.10.13/I10000='# temp;
    !WriteLine("url " # urlset);

    dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# urlset #"'");
    dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
    var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
    !WriteLine("stdout: " # stdout);

    !Abfrage, ob Setzen des Wertes ok
    string url='http://192.168.10.13/8740';

    dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");
    dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
    var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
    !WriteLine("stdout: " # stdout);

    string wort = "8740"; !Raumtemperatur 1
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 42;
    string daten = stdout.Substr((pos + laenge +1), 5);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    !integer zahl = daten.ToFloat();
    WriteLine("an die Heizung übertragene Raumtemperatur (8740) "#daten #"°C");

WriteLine("Hallo Welt!");
```

Commands to change the mode of the heater:

Damit syntax-sichere Anweisungen von CCU an BSB gegeben werden können (wichtig z.B. auch wenn via VPN kein direkter Zugang zum BSB-Adapter möglich ist).

Heizung AUS (= Frostschutzbetrieb):

```
!Heizung AUS (=Frostschutzbetrieb)  2017-03-09

string urlset ='http://192.168.10.13/S700=0';
WriteLine("Befehl: " # urlset);

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# urlset #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();

string url='http://192.168.10.13/700';

WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen

!Werte aktualisieren
var programObj = dom.GetObject("Heizungswerte abfragen");
programObj.ProgramExecute();

    WriteLine("Antwort ohne Vorspann: " # stdout);
WriteLine("Hallo Welt!");
```

Heizung Automatik (= AN - mit Nachtabsenkung):

```
!Heizung Automatik (=AN - mit Nachtabsenkung)  2017-03-09

!http://192.168.10.13/Passwort/S700=0 (Schutzbetrieb)
!http://192.168.10.13/Passwort/S700=1 (Automatik)
!http://192.168.10.13/Passwort/S700=2 (Reduziert)
!http://192.168.10.13/Passwort/S700=3 (Komfort)

string urlset ='http://192.168.10.13/S700=1';
WriteLine("Befehl: " # urlset);

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# urlset #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();

string url='http://192.168.10.13/700';

WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    WriteLine("Antwort ohne Vorspann: " # stdout);

!Werte aktualisieren
var programObj = dom.GetObject("Heizungswerte abfragen");
programObj.ProgramExecute();

WriteLine("Hallo Welt!");
```

Heizung KOMFORT (= AN - keine Nachtabsenkung):

```
!Heizung KOMFORT (=AN - keine Nachtabsenkung)  2017-03-09

!http://192.168.10.13/Passwort/S700=0 (Schutzbetrieb)
!http://192.168.10.13/Passwort/S700=1 (Automatik)
!http://192.168.10.13/Passwort/S700=2 (Reduziert)
!http://192.168.10.13/Passwort/S700=3 (Komfort)

! Anweisung senden
string urlset ='http://192.168.10.13/S700=3';
WriteLine("Befehl: " # urlset);

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# urlset #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();

! Kontrollabfrage
string url='http://192.168.10.13/700';

WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    WriteLine("Antwort ohne Vorspann: " # stdout);

!alle Werte aktualisieren
var programObj = dom.GetObject("Heizungswerte abfragen");
programObj.ProgramExecute();

WriteLine("Hallo Welt!");
```

Heizung NACHTABSENKUNG (dauernd, d.h. auch tagsüber!):

```
!Heizung Nachtabsenkung (dauernd, d.h. auch tagsüber Nachtabsenkung)  2017-03-09

!http://192.168.10.13/Passwort/S700=0 (Schutzbetrieb)
!http://192.168.10.13/Passwort/S700=1 (Automatik)
!http://192.168.10.13/Passwort/S700=2 (Reduziert)
!http://192.168.10.13/Passwort/S700=3 (Komfort)


string urlset ='http://192.168.10.13/S700=2';
WriteLine("Befehl: " # urlset);

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# urlset #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();

string url='http://192.168.10.13/700';

WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    WriteLine("Antwort ohne Vorspann: " # stdout);

!Werte aktualisieren
var programObj = dom.GetObject("Heizungswerte abfragen");
programObj.ProgramExecute();

WriteLine("Hallo Welt!");
```

Query of the daily averages /A and saving it as a system variable:

```
!Heizung Abfrage Tagesdurchschnitte V5 2017-10-07

!Raumtemperatur Ist und Trinkwassertemperatur ergänzt
!Variablen reduziert

!als Systemvariablen sind in Homematic angelegt:
!H_Aussentemperatur_24h                Zahl    8700
!H_Brennermodulation_24h            Zahl    8326
!H_Vorlauftemperatur_24h            Zahl    8743
!H_Kesselruecklauftemperatur_24h    Zahl    8314

!H_Raumtemperatur_Ist_24h            Zahl    8740
!H_Raumtemperatur_Soll_24h  ???
!H_Trinkwassertemperatur_24h        Zahl    8830

string url='http://192.168.10.13/A';

WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 -qO- '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
!WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    WriteLine("Antwort ohne Vorspann: " # stdout);

!-----------------------------------------------------------------------------

string wort = "8700"; !Aussentemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 20;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    !Korrektur des angezeigten Wertes auf das Niveau der übrige Thermometer
    zahl = zahl - 3.5;
    dom.GetObject("H_Aussentemperatur_24h").State(zahl);
    WriteLine("H_Aussentemperatur_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Aussentemperatur_24h"#";"#zahl);

!-----------------------------------------------------------------------------

string wort = "8326"; !Brennermodulation
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 21;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    dom.GetObject("H_Brennermodulation_24h").State(zahl);
    WriteLine("H_Brennermodulation_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Brennermodulation_24h"#";"#zahl);

!-----------------------------------------------------------------------------

string wort = "8743"; !Vorlauftemperatur 1:
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 23;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
```

```
    integer pos = daten.Find(wort);
    !daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    dom.GetObject("H_Vorlauftemperatur_24h").State(zahl);
    WriteLine("H_Vorlauftemperatur_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Vorlauftemperatur_24h"#";"#zahl);

!-------------------------------------------------------------------------

string wort = "8314"; !Kesselrücklauftemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 33;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    dom.GetObject("H_Kesselruecklauftemperatur_24h").State(zahl);
    WriteLine("H_Kesselruecklauftemperatur_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Kesselruecklauftemperatur_24h"#";"#zahl);

!-------------------------------------------------------------------------

string wort = "8740"; !Raumtemperatur 1
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 21;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    dom.GetObject("H_Raumtemperatur_Ist_24h").State(zahl);
    WriteLine("H_Raumtemperatur_Ist_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Raumtemperatur_Ist_24h"#";"#zahl);

!-------------------------------------------------------------------------
string wort = "8830"; !Trinkwassertemperatur
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 28;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    dom.GetObject("H_Trinkwassertemperatur_24h").State(zahl);
    WriteLine("H_Trinkwassertemperatur_24h: "#zahl);

! nur für Chart CUxD
!dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("H_Trinkwassertemperatur_24h"#";"#zahl);


WriteLine("Hallo Welt!");
```

Query connected DS18B20 temperature sensors and save the output as a system variable:

```
!Temperatursensoren Abfrage CUxD 2017-10-13 (Auszug)


!als Systemvariablen sind in Homematic angelegt:
!28ff99e890160456 T_Aussentemperatur_Nord
!28ffff85901604d6 T_Innentemperatur_Esszimmer

string url='http://192.168.10.13/ipwe.cgi';

!WriteLine("URL: " # url);

!--timeout=seconds

dom.GetObject("CUxD.CUX2801001:1.CMD_SETS").State("wget -t 30 --timeout=20 -q -O - '"# url #"'");
dom.GetObject("CUxD.CUX2801001:1.CMD_QUERY_RET").State(1);
var stdout = dom.GetObject("CUxD.CUX2801001:1.CMD_RETS").State();
!WriteLine("Antwort: " # stdout);

!Beim stdout den ueberfluessigen Vorspann entfernen
    integer laenge = stdout.Length();
    integer pos = stdout.Find("body");
    pos = pos + 10;
    stdout = stdout.Substr(pos, (laenge - pos));
    !wenn Rückmeldung mit allen Daten angezeigt werden soll, Ausrufezeichen nächste Zeile entfernen
    !WriteLine("Antwort ohne Vorspann: " # stdout);

string wort = "28ff99e890160456"; !T_Aussentemperatur_Nord
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 11;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    if (zahl<>0)
        {
        WriteLine("T_Aussentemperatur_Nord gemessen: "#zahl);
        dom.GetObject("T_Aussentemperatur_Nord").State(zahl);
        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("T_Aussentemperatur_Nord"#";"#zahl);
        }

string wort = "28ffff85901604d6"; !T_Innentemperatur_Esszimmer
    integer laenge = wort.Length();
    !WriteLine("laenge: " # laenge);
    integer pos = stdout.Find(wort);
    !WriteLine("pos:" #pos);
    pos = pos + 11;
    string daten = stdout.Substr((pos + laenge +1), 100);
    !WriteLine("daten :"#daten);
    integer pos = daten.Find(wort);
    daten = daten.Substr(0, (pos));
    integer zahl = daten.ToFloat();
    if (zahl<>0)
        {
        dom.GetObject("T_Innentemperatur_Esszimmer").State(zahl);
        WriteLine("T_Innentemperatur_Esszimmer: "#zahl);
        ! nur für Chart CUxD
        dom.GetObject("CUxD.CUX2801001:1.LOGIT").State("T_Innentemperatur_Esszimmer"#";"#zahl);
        }

WriteLine("Hallo Welt!");
```

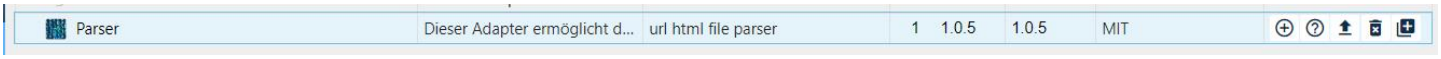
11.4 IoBroker

BSB-LAN-User „hacki11“ developed an adapter for ioBroker, which he kindly made available in his GitHub-Repo.
Thanks a lot!

Sorry, not fully translated yet.. :(
The following examples for ioBroker integration were written by FHEM forum member „Thomas_B".
Thanks a lot!

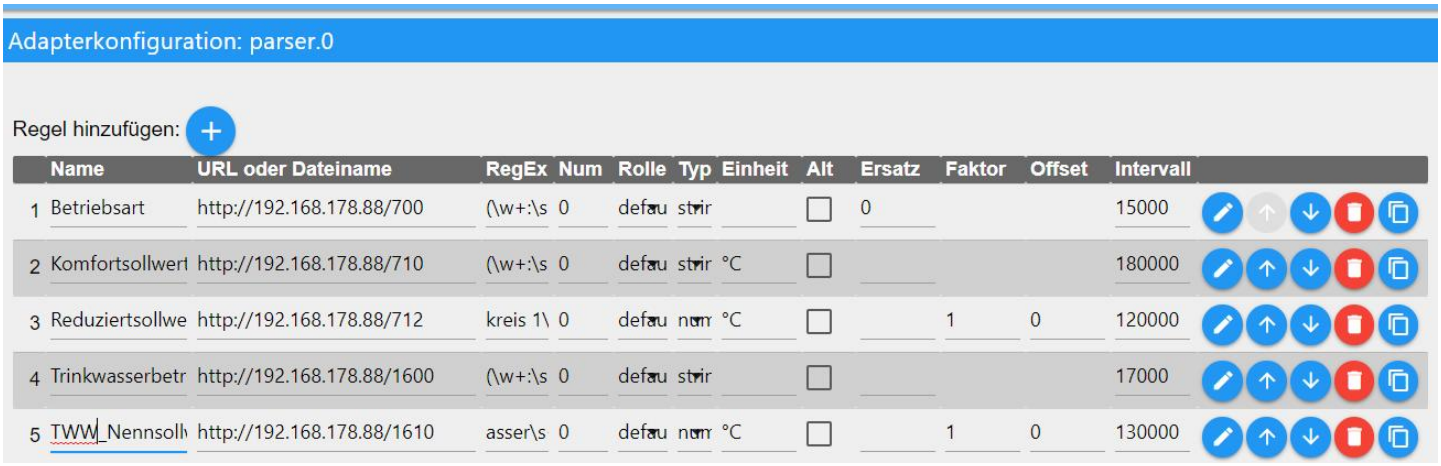
Query parameters/values and display them:

Unter ‚ioBroker Admin → Adapter' eine ‚Parser'-Instanz hinzufügen:



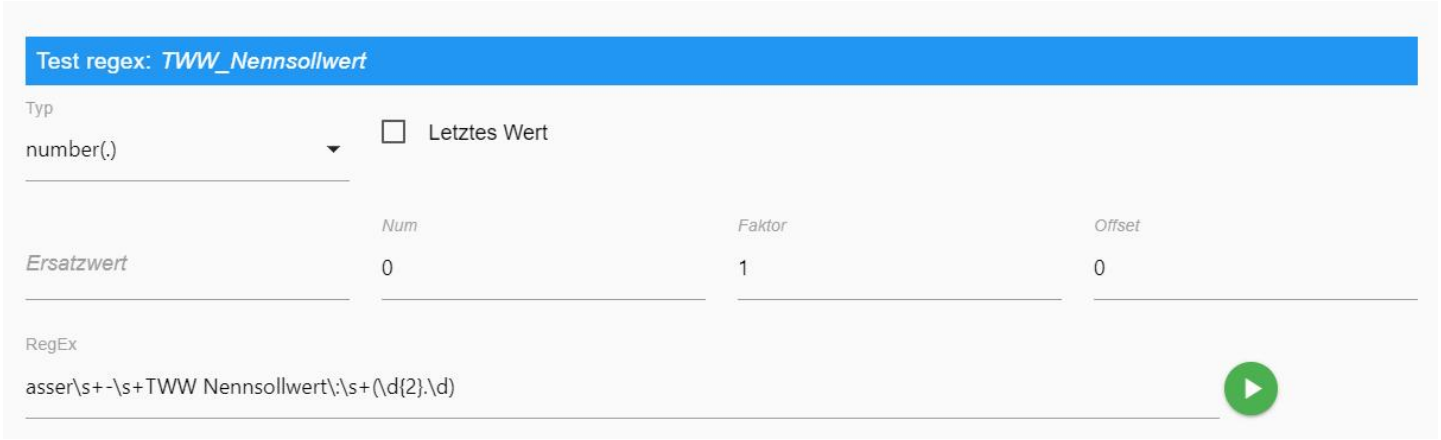
Danach unter ‚ioBroker Admin → Instanzen' die hinzugefügten Adapterinstanz ‚parser.0' zum Konfigurieren öffnen:

Dort auf das ‚+' klicken, danach unter ‚Namen' den Namen ‚TWW_Nennsollwert' vergeben. Unter ‚URL oder Dateiname' die IP des BSB-LAN-Adapters samt Parameternummer angeben. Anschließend auf das ‚Bearbeiten'-Icon klicken.



Es öffnet sich die Eingabemaske, in der unter ‚RegEx' folgende Zeichenfolge eingegeben werden muß:

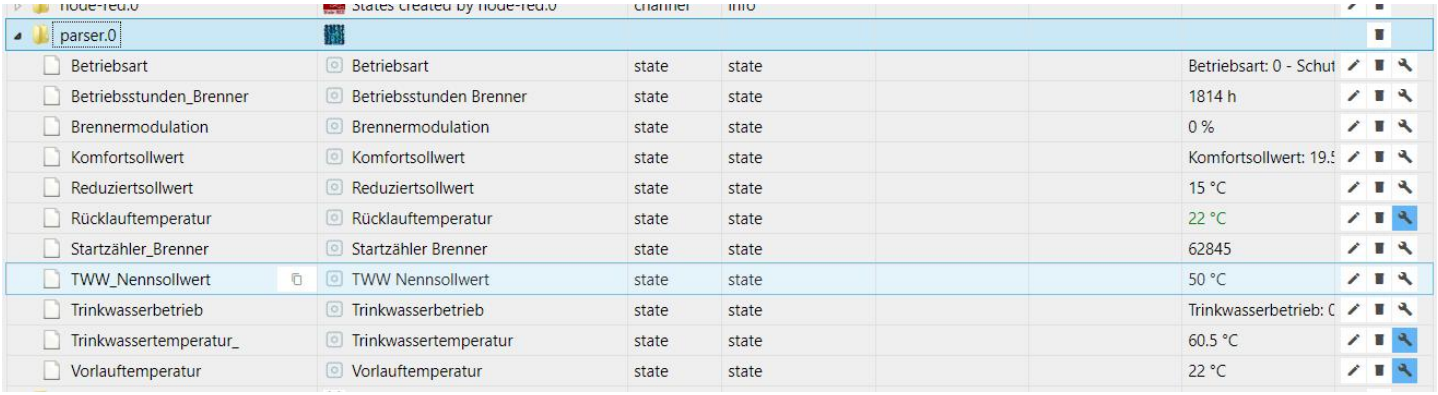
```
asser\s+-\s+TWW Nennsollwert\:\s+(\d{2}.\d)
```



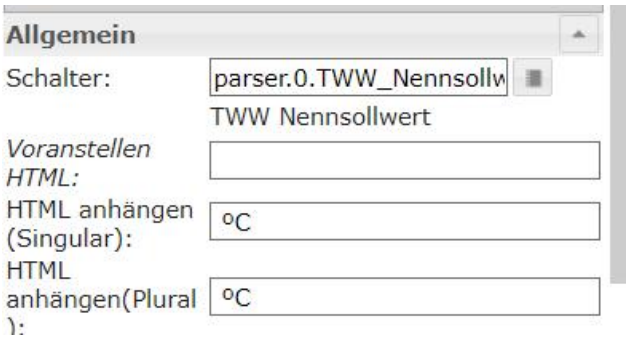
Danach kann die Eingabemaske mit ‚Speichern' geschlossen werden.

Das Abfrageintervall kann man nach Bedarf einstellen, danach auf ‚Speichern und Schließen' klicken. Damit ist die Adapterkonfiguration abgeschlossen.

Unter ‚ioBroker Admin → Objekte' findet sich nun der Ordner ‚parser.0' und die unter der Instanz ‚parser.0' angelegten Datennamen und deren Werte:



Die Werte können unter VIS mittels eines ‚Basic-Number'-Widgets mit folgenden Einstellungen angezeigt werden:

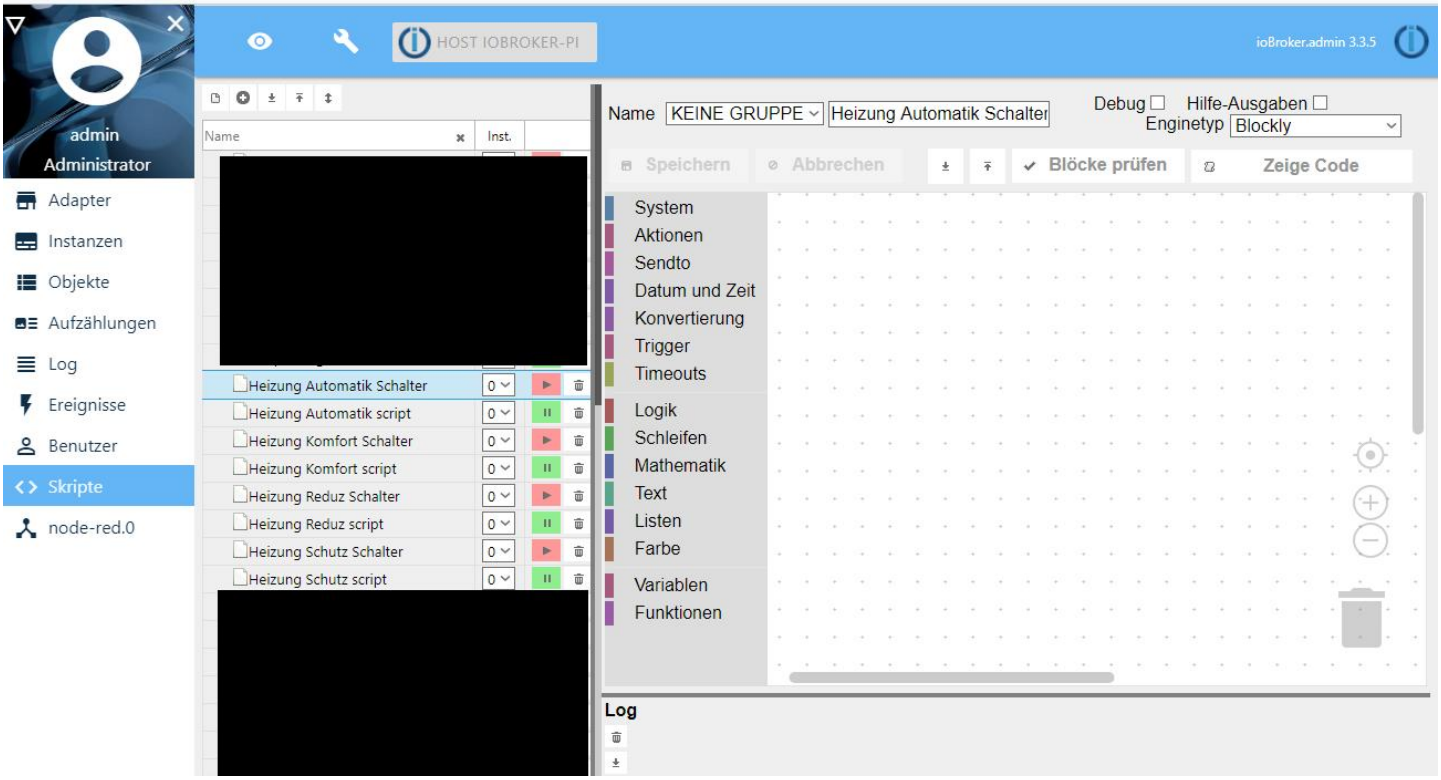


Widgetcode zum Importieren:

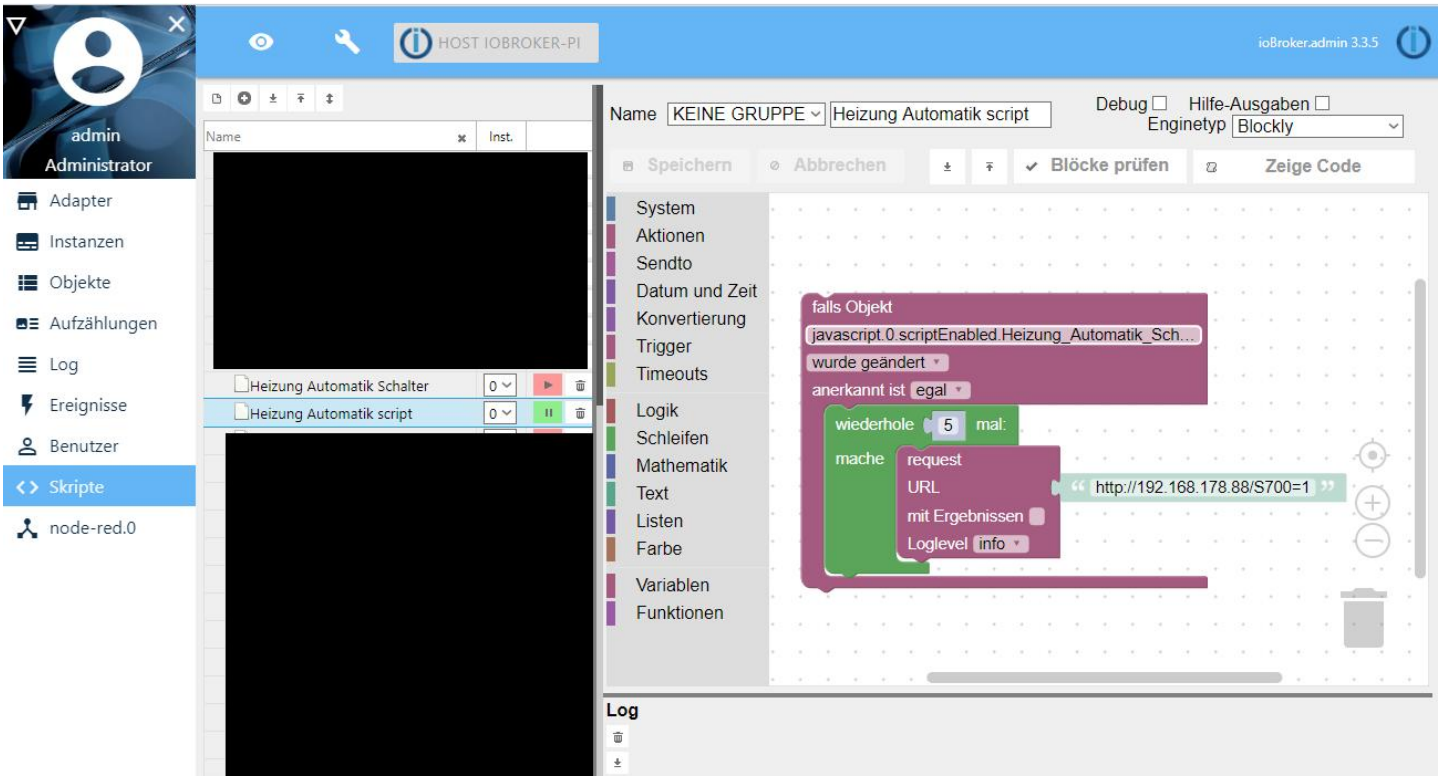
```
[{"tpl":"tplValueFloat","data":{"oid":"parser.0.TWW_Nennsollwert","visibility-cond":"==","visibility-val":1,"is_comma":"true","factor":"1","html_append_singular":" °C","html_append_plural":" °C","gestures-offsetX":0,"gestures-offsetY":0,"is_tdp":false,"signals-cond-0":"==","signals-val-0":true,"signals-icon-0":"/vis.0/bluefox_ehome/lowbattery.png","signals-icon-size-0":0,"signals-blink-0":false,"signals-horz-0":0,"signals-vert-0":0,"signals-hide-edit-0":false,"signals-cond-1":"==","signals-val-1":true,"signals-icon-1":"/vis.0/bluefox_ehome/lowbattery.png","signals-icon-size-1":0,"signals-blink-1":false,"signals-horz-1":0,"signals-vert-1":0,"signals-hide-edit-1":false,"signals-cond-2":"==","signals-val-2":true,"signals-icon-2":"/vis.0/bluefox_ehome/lowbattery.png","signals-icon-size-2":0,"signals-blink-2":false,"signals-horz-2":0,"signals-vert-2":0,"signals-hide-edit-2":false,"visibility-groups-action":"hide","lc-type":"last-change","lc-is-interval":true,"lc-is-moment":false,"lc-format":"","lc-position-vert":"top","lc-position-horz":"right","lc-offset-vert":0,"lc-offset-horz":0,"lc-font-size":"12px","lc-font-family":"","lc-font-style":"","lc-bkg-color":"","lc-color":"","lc-border-width":"0","lc-border-style":"","lc-border-color":"","lc-border-radius":10,"lc-zindex":0,"digits":"1"},"style":{"left":"398px","top":"428px","width":"59px","height":"18px","color":"white","text-align":"right","font-family":"Arial, Helvetica, sans-serif","font-style":"normal","font-variant":"normal","font-weight":"bold","font-size":"","z-index":"20"},"widgetSet":"basic"}]
```

Create a button switch:

Zunächst ein leeres Script ‚Heizung Automatik Schalter' anlegen:



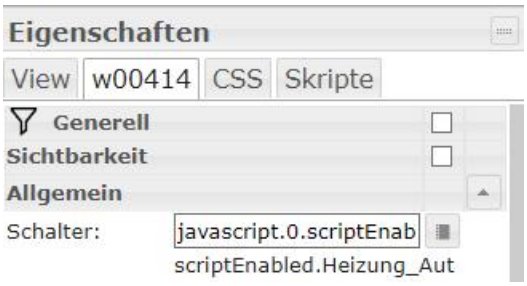
Dann ein Blocky-Script ‚Heizung Automatik script' mit folgendem Inhalt anlegen (der nachfolgende Code kann in Blocky importiert werden):




```
on({id: "javascript.0.scriptEnabled.Heizung_Automatik_Schalter", change: "ne"}, function (obj) {
  var value = obj.state.val;
  var oldValue = obj.oldState.val;
  for (var count = 0; count < 5; count++) {
    try {
      require("request")('http://<IP des BSB-LAN Adapters>/S700=1').on("error", function (e) {console.error(e);});
    } catch (e) { console.error(e); }
    console.log("request: " + 'http:// ://<IP des BSB-LAN Adapters>//S700=1');
  }
});
```

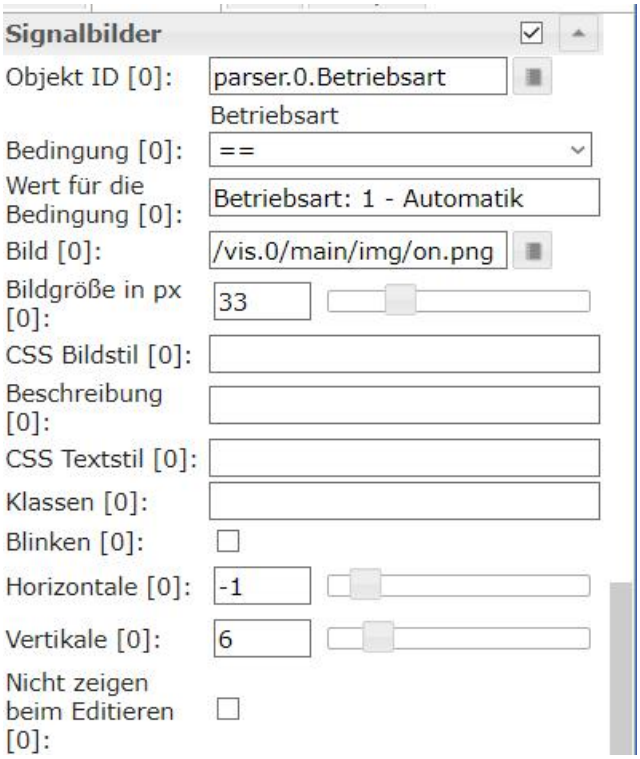
Dann ein ‚jqui -- Button State'-Widget in VIS anlegen:

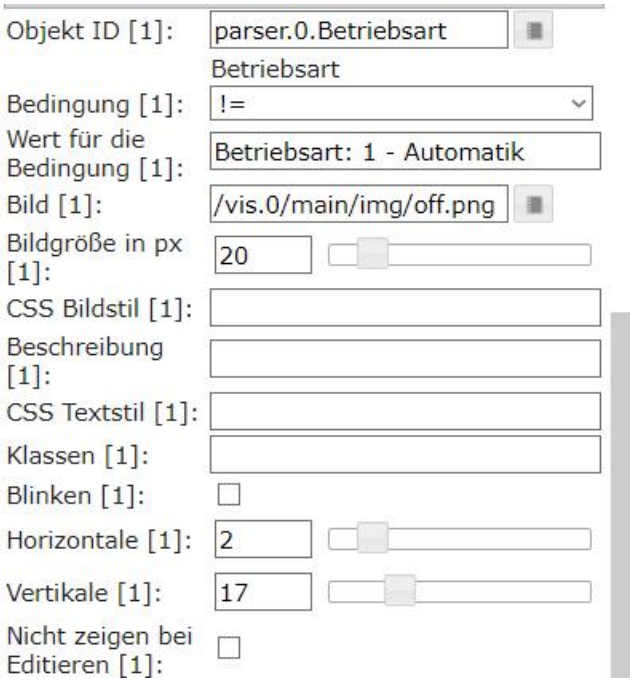
In den Eigenschaften unter ‚Schalter' das anfangs angelegte leere Script angeben:



Damit lässt sich die Betriebsart ‚Heizung Automatik' einschalten.

Damit der Schalterzustand durch oder entsprechend visualisiert wird, müssen noch folgende Signalbilder in dem Widget hinzugefügt werden, wobei die Bilder „on.png" und „off.png" in dem genannten Verzeichnispfad abgespeichert werden müssen.





Widgetcode zum Importieren:


```
[{"tpl":"tplJquiButtonState","data":{"oid":"javascript.0.scriptEnabled.Heizung_Automatik_Schalter(2)","g_fixed":false,"g_visibility":false,"g_css_font_text":true,"g_css_background":true,"g_css_shadow_padding":false,"g_css_border":false,"g_gestures":false,"g_signals":true,"g_last_change":false,"visibility-cond":"==","visibility-val":1,"visibility-groups-action":"hide","buttontext":"Heizung Automatik (AN)","signals-cond-0":"==","signals-val-0":"Betriebsart: 1 - Automatik","signals-icon-0":"/vis.0/main/img/on.png","signals-icon-size-0":"33","signals-blink-0":false,"signals-horz-0":"-1","signals-vert-0":"6","signals-hide-edit-0":false,"signals-cond-1":"!=","signals-val-1":"Betriebsart: 1 - Automatik","signals-icon-1":"/vis.0/main/img/off.png","signals-icon-size-1":"20","signals-blink-1":false,"signals-horz-1":"2","signals-vert-1":"17","signals-hide-edit-1":false,"signals-cond-2":"==","signals-val-2":true,"signals-icon-2":"/vis/signals/lowbattery.png","signals-icon-size-2":0,"signals-blink-2":false,"signals-horz-2":0,"signals-vert-2":0,"signals-hide-edit-2":false,"lc-type":"last-change","lc-is-interval":true,"lc-is-moment":false,"lc-format":"","lc-position-vert":"top","lc-position-horz":"right","lc-offset-vert":0,"lc-offset-horz":0,"lc-font-size":"12px","lc-font-family":"","lc-font-style":"","lc-bkg-color":"","lc-color":"","lc-border-width":"0","lc-border-style":"","lc-border-color":"","lc-border-radius":10,"lc-zindex":0,"value":"","signals-oid-1":"parser.0.Betriebsart","signals-oid-0":"parser.0.Betriebsart"},"style":{"left":"14px","top":"426px","width":"219px","height":"27px","z-index":"1","font-size":"x-small"},"widgetSet":"jqui"}]
```

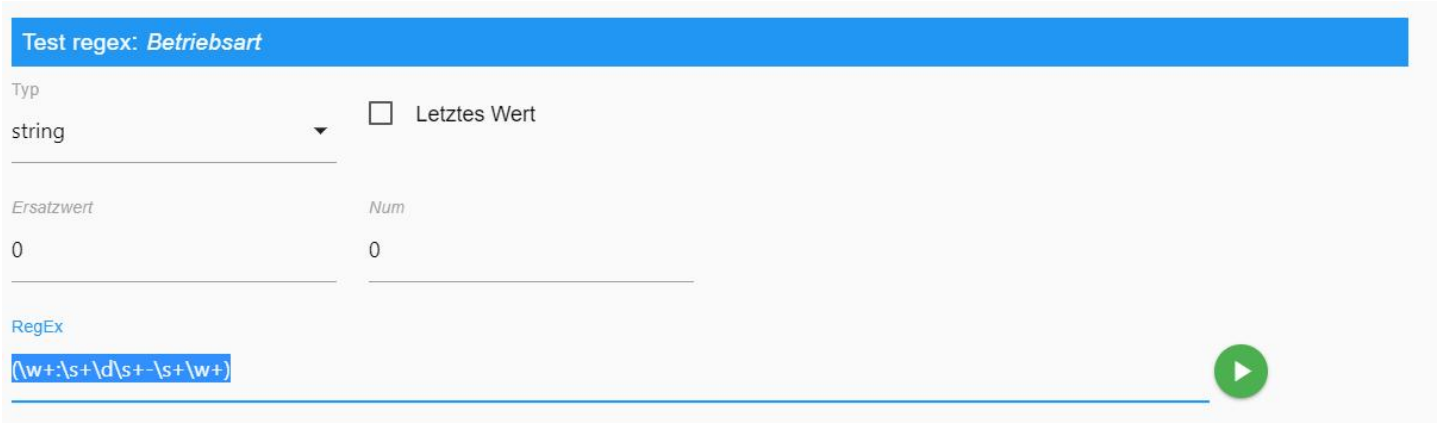

Die Einbindung der jeweiligen Werte bei ‚Objekt ID [0]' und ‚Objekt ID [1]' (‚parser.0.Betriebsart') wird nachfolgend erklärt.

Query the mode of the heater:

Bei der Adapterkonfiguration für ‚parser.0' eine Regel mit der Bezeichung ‚Betriebsart' erstellen, dann die IP (samt Parameternummer) des BSB-LAN-Adapters angeben und zum Bearbeiten öffnen.

Unter ‚RegEx' folgende Syntax angeben:

```
(\w+:\s+\d\s+-\s+\w+)
```



11.5 Loxone

Sorry, not yet translated.. :(
Die Loxone-Beispiele stammen vom FHEM-Forumsmitglied „Loxonaut".
Vielen Dank!

Abfrage von Parametern:
Die Abfrage von Parametern erfolgt in Loxone mittels virtuellen HTTP Eingängen und der JSON-Funktion von BSB-LAN (URL-Befehl `/JQ=`). Eine detailliertere Beschreibung der virtuellen HTTP Eingänge und der Befehle ist in den Dokumentationen zu Loxone und im Loxone-Wiki zu finden.

Achtung: Die Entwicklung der JSON-Funktion in BSB-LAN ist noch nicht endgültig abgeschlossen, es kann jederzeit zu Änderungen kommen. Die Konfiguration ist dann entsprechend anzupassen.

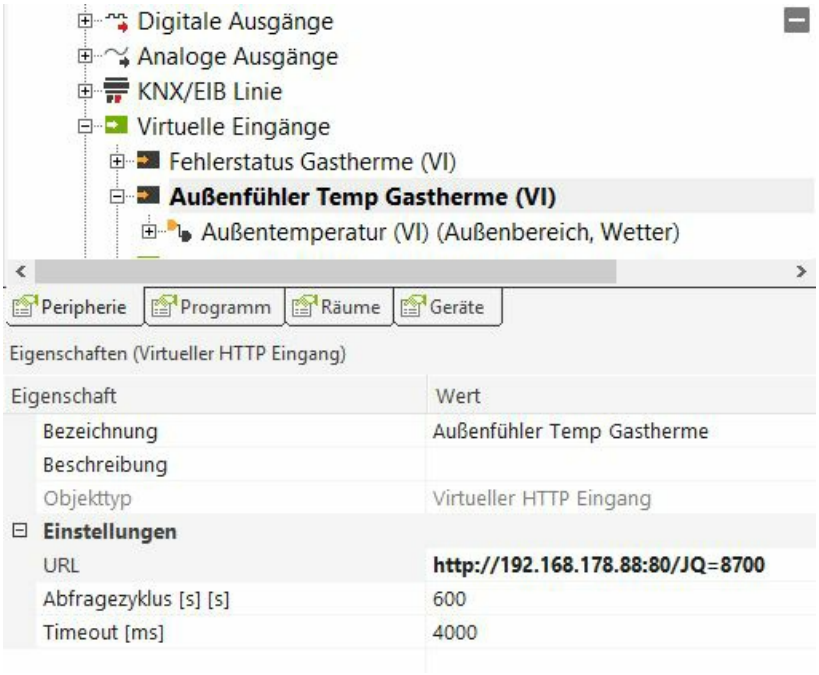
Das folgende Beispiel zeigt die Einrichtung anhand des Parameters "8700 Außentemperatur".

Zum Hinzufügen eines virtuellen HTTP Eingangs muss zunächst im Fenster "Peripherie" die Zeile "Virtuelle Eingänge" markiert werden. Nun klickt man auf die oben erschienene Schaltfläche "Virtueller HTTP Eingang":

Bei den Eigenschaften trägt man die Bezeichnung und die entsprechenden Werte ein (beim Abfragezyklus sollte ein entsprechend sinnvoller Wert gewählt werden), die URL des BSB-LAN-Adapters ist hierbei um den Befehl

`/JQ=8700`

für die Abfrage der Außentemperatur zu erweitern:



Anschließend fügt man dem vituellen HTTP Eingang noch einen virtuellen HTTP Eingang Befehl hinzu:

Hier definiert man, was aus dem JSON-Export ausgelesen werden soll. Der JSON-Export ist wie folgt aufgebaut:

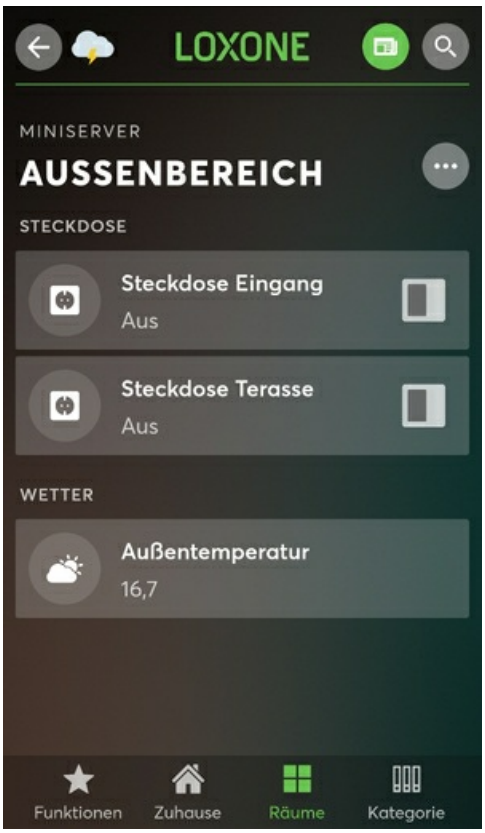
```
{
"8700": {
"name": "Aussentemperatur","value": "16.6",
"unit": "&deg;C",
"desc": "",
"dataType": 0
}
}
```

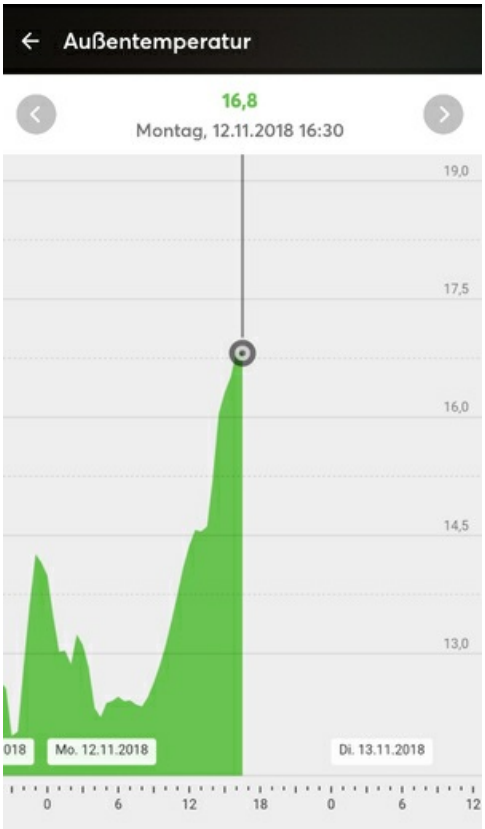
Mittels Loxone-Befehlserkennung

`value": "\v`

wird der Wert bei "value" des JSON-Exports ausgelesen:

Unter "Visualisierung" bei den Eigenschaften sollte bei "Kategorie" und "Raum" jeweils eine Bezeichnung eingetragen werden, damit die spätere Darstellung in der Loxone-App entsprechend funktioniert (hier: Außenbereich, Wetter). Die nun ausgelesenen Werte des Außentemperaturfühlers können dann in der Loxone-App angezeigt und die Statistik per Graph visualisiert werden.





Hinweis:
Das Setzen von Parametern/Werten könnte analog zu obigem Beispiel mit der Funktion "virtueller Ausgang" und dem URL-Befehl `/JS` *(JSON) oder via regulärem URL-Befehl* `/S<x>=<y>` *möglich sein (s. entspr. Kapitel), wurde allerdings noch nicht gestestet.*

11.6 IP-Symcon

The BSB-LAN users „sokkederheld" and „Joachim" presented their IPS scripts plus screenshots in the Symcon forum: here and here. Thanks a lot!

11.7 MQTT, InfluxDB, Telegraf and Grafana

A well documented example by BSB-LAN user „cubase" for the usage of MQTT, InfluxDB, Telegraf and Grafana is available here. Thanks a lot!

11.8 MQTT and FHEM

The following example is written by FHEM forum member „mifh".
Thanks a lot!

The following example script uses the integrated MQTT server of FHEM and is written for a gas fired burner including the calculation of the heating power. So the last part isn't neccessary for / available within oil fired burners, heat pumps and so on.

Configuration of the MQTT server of FHEM:

```
define MQTT_TST MQTT2_SERVER 1883 global
define MQTT_2 MQTT <IP-Adresse>:1883
```

Display heater as a MQTT device in FHEM:

```
define Hzg_Therme MQTT_DEVICE
attr Hzg_Therme IODev MQTT_2
attr Hzg_Therme alias Brötje Heizung
attr Hzg_Therme group Heizung
attr Hzg_Therme room Heizung
attr Hzg_Therme subscribeReading_Kesseltemperatur Zuhause/Heizungsraum/BSB-LAN/8310
attr Hzg_Therme subscribeReading_Ruecklauftemperatur Zuhause/Heizungsraum/BSB-LAN/8314
attr Hzg_Therme subscribeReading_Geblaesedrehzahl Zuhause/Heizungsraum/BSB-LAN/8323
attr Hzg_Therme subscribeReading_Brennermodulation Zuhause/Heizungsraum/BSB-LAN/8326
attr Hzg_Therme subscribeReading_BetriebsstundenStufe1 Zuhause/Heizungsraum/BSB-LAN/8330
attr Hzg_Therme subscribeReading_StartzaehlerBrenner Zuhause/Heizungsraum/BSB-LAN/8331
attr Hzg_Therme subscribeReading_BetriebsstundenHeizbetrieb Zuhause/Heizungsraum/BSB-LAN/8338
attr Hzg_Therme subscribeReading_BetriebsstundenTWW Zuhause/Heizungsraum/BSB-LAN/8339
attr Hzg_Therme subscribeReading_Gesamt_Gasenergie_Heizen Zuhause/Heizungsraum/BSB-LAN/8378
attr Hzg_Therme subscribeReading_Gesamt_Gasenergie_TWW Zuhause/Heizungsraum/BSB-LAN/8379

attr Hzg_Therme subscribeReading_Aussentemperatur Zuhause/Heizungsraum/BSB-LAN/8700
attr Hzg_Therme subscribeReading_DrehzahlHeizkreispumpe Zuhause/Heizungsraum/BSB-LAN/8735

attr Hzg_Therme stateFormat {sprintf("Leistung: %.1f kW",ReadingsVal($name,"Leistung",0))}
attr Hzg_Therme verbose 3 Hzg_Therme

define Hzg_Therme_NF1 notify Hzg_Therme:Geblaesedrehzahl.* {setHzgLeistung()}
```

The notify uses a perl function in *99_myUtils.pm* to create the reading "Leistung":

```
sub setHzgLeistung{
    my $drehzahl=ReadingsVal("Hzg_Therme", "Geblaesedrehzahl",0);
    my $leistung;
    if ($drehzahl > 0) {
        $leistung = ($drehzahl- 1039.1)/383.1; # Heizungspezifische Parameter
        }
    else {
        $leistung = 0;
        }
    fhem("setreading Hzg_Therme Leistung $leistung");
}
```

11.9 MQTT2 and FHEM

The following example was written by FHEM forum member „FunkOdyssey".
Thanks a lot!

This example uses the integrated MQTT2 server in FHEM, after the successful configuration the readings are displayed automatically.

Configure the MQTT2 server following the command reference of FHEM:

```
defmod mqtt2Server MQTT2_SERVER 1883 global
attr mqtt2Server autocreate 1
```

As soon as you set the IP of the FHEM server in the file *BSB_lan_config.h*, the MQTT2 device with the readings appears:

```
defmod MQTT2_BSB_LAN MQTT2_DEVICE BSB_LAN
attr MQTT2_BSB_LAN IODev mqtt2Server
attr MQTT2_BSB_LAN readingList BSB_LAN:BSB/8314:.* Kesselruecklauftemperatur\
BSB_LAN:BSB/8700:.* Aussentemperatur\
BSB_LAN:BSB/8323:.* Geblaesedrehzahl\
BSB_LAN:BSB/8324:.* Brennergeblaesesollwert\
BSB_LAN:BSB/700:.* Betriebsart\
...
```

The following example was written by the member "Luposoft" of the FHEM forum. You can see the original post here.
Thanks a lot!

```
define mqtt2Server MQTT2_SERVER 1883 global
define MQTT2_BSB_LAN MQTT2_DEVICE BSB_LAN
define FileLog_MQTT2 FileLog ./log/%V-%G-MQTT2.log MQTT2_BSB_LAN
```

This publish output sends the MQTT telegram to the world undirected, without knowing if anyone is listening (a basic principle of MQTT).

`set mqtt2Server publish BSB-LAN S5890=0` → Output of the control command (some also need an I instead of the S) and our Arduino listens exactly to BSB-LAN (at least in the default config).

The corresponding entries in FileLog_MQTT2:
`2021-02-03_16:56:28 MQTT2_BSB_LAN MQTT: ACK_S5890=0` → Here BSB-LAN confirms the reception.
`2021-02-03_16:56:29 MQTT2_BSB_LAN BSB-LAN_5890: 0 - Kein` → This is the output after the query of the heater.

`set mqtt2Server publish BSB-LAN 5890` → Simple query.
`2021-02-04_13:24:15 MQTT2_BSB_LAN MQTT: ACK_5890` → Here BSB-LAN confirms the reception.
`2021-02-04_13:24:16 MQTT2_BSB_LAN BSB-LAN_5890: 1 - Zirkulationspumpe Q4` → This is the output after querying the heater.

11.10 EDOMI

Sorry, not yet translated.. :(

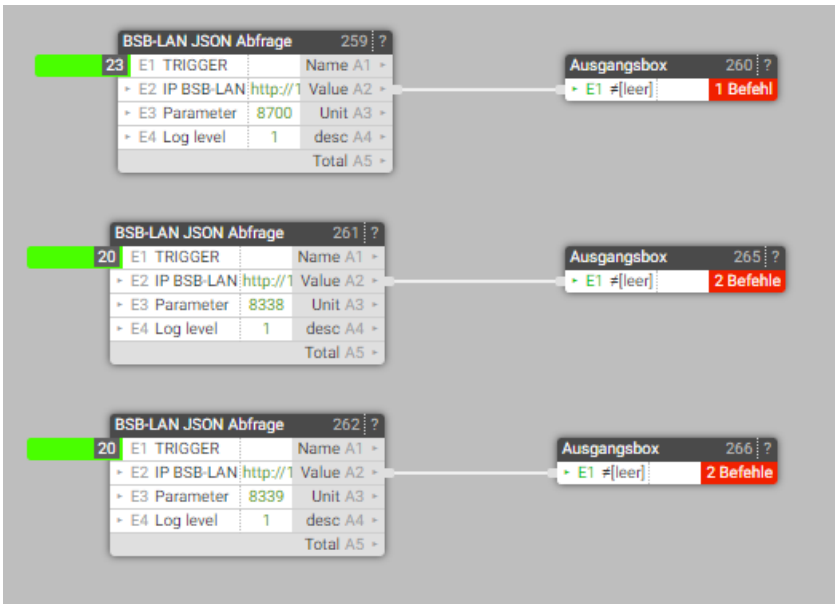
Das folgende Beispiel stammt vom BSB-LAN-User Lutz.
Vielen Dank!

Die Abfrage von Werten aus BSB-LAN erfolgt mittels des erstellten Logikbausteins 19001820.

Der Baustein greift über die JSON Schnittstelle von BSB LAN auf das Gateway zu und liefert je nach angegebenem Parameter die entsprechenden Werte.

Dabei sind folgende Eingangswerte einzutragen:
E1 = Trigger, nicht gleich Null
E2 = IP Adresse BSB-LAN Gateway
E3 = Parameter, z.B. Wert 8700 für Außentemperatur
E4 = Log Level

Als Ergebnis erhält EDOMI folgende Werte zurück:
A1 = Name des Parameters, z.B. "Aussentemperatur"
A2 = Wert des Parameters, z.B. "10,5"
A3 = Einheit des Parameters z.B. "°C"
A4 = Beschreibung von A2, wenn als Code ausgegeben.
A5 = Verkettung von A1 bis A4



Die Werte A1 bis A5 können dann über andere Bausteine weiterverarbeitet werden. In diesem Beispiel wird die Außentemperatur in ein internes Kommunikationsobjekt geschrieben, um den Inhalt z.B. in der Visu auszugeben oder die Werte für die Betriebszeit in ein Datenarchiv gespeichert, um daraus später Laufzeiten der Heizung zu ermitteln.

11.11 Home Assistant

BSB-Lan is now an official Home-Assistant integration maintained by Willem-Jan.
Documentation of the integration can be found here.
The initial support is only for basic thermostat support in home-assistant.
Thanks a lot!

The previous solution was expanded by BSB-LAN user Florian for being able to control two controllers and four heating circuits. He made his solution available in his GitHub Repo.
Thanks a lot!

BSB-LAN user Torben is using MQTT within his Home Assistant setup. The following example shows the way of how to set it up.
Tahnks a lot!

The example below shows an exemplary sensor configuration for Home Assistant. It allows to query the value of the room temperature comfort setpoint (BSB parameter 710). The example assumes, that this parameter is registered in log_parameters, so that it is actually published.

```
- platform: mqtt
  name: "BSB Room temperature Comfort setpoint"
  state_topic: "BSB-LAN/710"
  unique_id: "bsb710"
  unit_of_measurement: '°C'
  device_class: temperature
  availability_topic: "BSB-LAN/status"
  icon: "mdi:thermometer-chevron-up"
```

See also: https://www.home-assistant.io/integrations/sensor.mqtt/

11.12 SmartHomeNG

BSB-LAN-User Thomas wrote a plugin for SmartHomeNG and made it available in his GitHub Repo.
Thanks a lot!

11.13 Node-RED

BSB-LAN-User Konrad wrote a module for Node-RED which makes it easy to implement BSB-LAN.
Thanks a lot!

11.14 Data Processing Using Bash Script

BSB-LAN user Karl-Heinz has written two bash scripts that can be used under Linux to read data from the heating controller and display it graphically using gnuplot.
Thanks a lot!

Karl-Heinz's solution is interesting for those users who do not (want to) use complex home automation software to retrieve data from the heating controller under Linux and display it graphically. He kindly provides the scripts in his GitHub repo.

Further on to chapter 12
Back to TOC

12. Hardware in Conjunction with the BSB-LPB-LAN Adapter

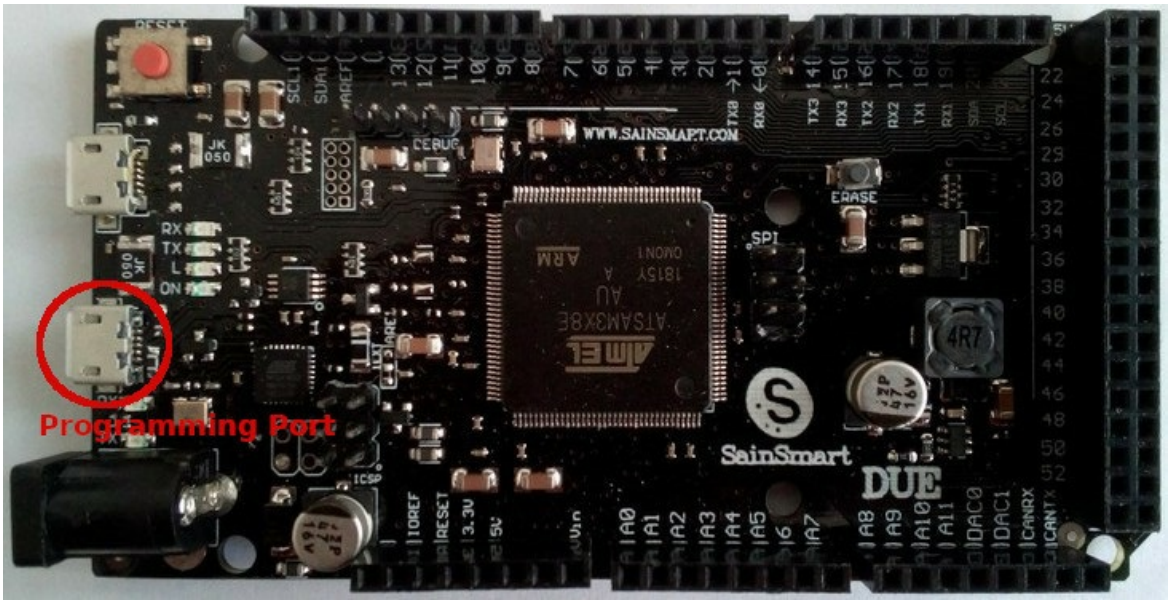
Back to TOC
Back to chapter 11

12. Hardware in Conjunction with the BSB-LPB-LAN Adapter

12.1 The Arduino Due

In general, the use of an original Arduino Due is recommended.
From experience, however, cheap replicas ("clones") of the Arduino Due can also be used, the use of these clones is usually possible without any problems. But: It should be paid attention if a modified board layout (e.g. changed pin assignments) is described in the prduct description. If this is the case and you still want to buy it, you may need to make specific adjustments in the file *BSB_lan_config.h*.

A pinout diagram of the Arduino Due is available in appendix b.



A compatible clone of the Arduino Due.

Notes:

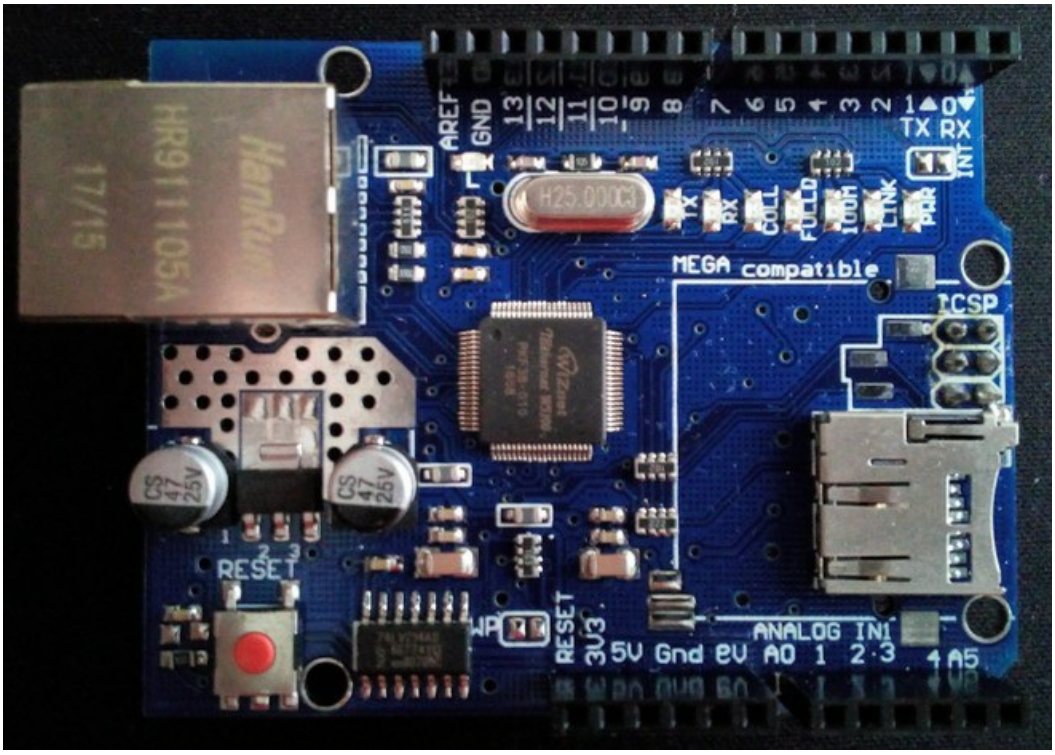
- Regarding to the tech specs of the Arduino Due, it is recommended to use an external power source (recommended: 7-12V, limits: 6-16V) at the intended connection of the Arduino (e.g. 9V/1000mA).
- If you want to power the Due via USB, please use the "Programming Port".
- It's possible to power the Due via the DC-IN and use USB connection at the programming port for connecting it to the computer at the same time.
- You can let the adapter be connected to the controller bus of the heater when flashing the Due.
- *Make sure that you are using a high-quality USB cable!* This applies to the case that you want to power the Due via USB as well as to the case that you want to connect the Due to your PC for flashing. Especially cheap and thin cables (e.g. accessories of smartphones) can cause problems with the power supply and thus the stability of the Due and/or are not always fully wired, so that a use for data transfer is not possible.
- With some Due models/clones it can happen that they do not seem to work properly after an initial start (e.g. after a power failure) and only work correctly after pressing the reset button. A possible solution for this problem could be to add a capacitor.

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

12.1.1 Due + LAN: The LAN Shield

In general, the use of an original Arduino LAN shield (v2) is recommended.
From experience, however, cheap replicas ("clones") of these LAN shields can also be used, the use of these clones is usually possible without any problems. But: It should be paid attention if a modified board layout (e.g. changed pin assignments) is described in the product description. If this is the case and you still want to buy it, you may need to make specific adjustments in the file *BSB_lan_config.h*.

There are / have been two different versions of LAN shields available on the market: one with a WIZnet W5100 chip (v1) and one with a W5500 chip (v2). The usage of a v2-shield is recommended, it's also available at the official Arduino store.

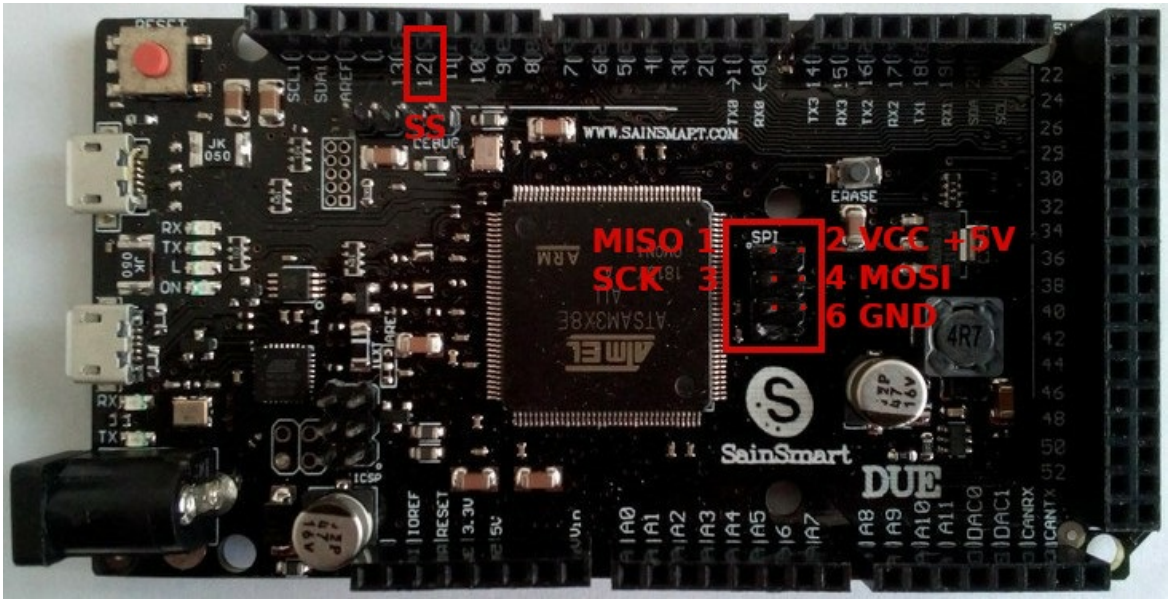


A compatible clone of a LAN shield with a W5100 chip.

Notes:
After the installation of the Arduino IDE it should be checked that the current version of the Ethernet Library (min. v2) is installed.
As a LAN cable one should preferably use a S/FTP type with a minimum length of one metre.

12.1.2 Due + WLAN: The ESP8266-WiFi-Solution

Another option for integrating the adapter setup into your WLAN is connecting an ESP8266 (NodeMCU or Wemos D1) additionally to the Arduino Due via the six-pole SPI header.
The ESP8266 is supplied with power (+5V) by the Due and basically serves instead of the LAN shield only as an interface to access the Due via the network. The ESP8266 has to be flashed with a special firmware for this purpose, you can read more about this later in this chapter. The BSB-LAN software is still installed on the Due.

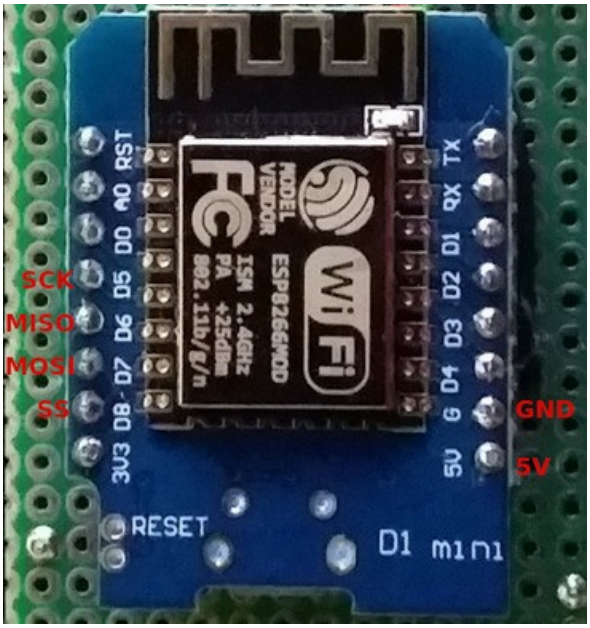


The six-pole SPI header of the Arduino Due which has to be used.

The connections have to be done as follows:

Pin DUE	Function	Pin ESP8266
SPI 1	MISO (Master Innput Slave Output)	D06
SPI 2	VCC (power supply ESP)	+5V / Vin
SPI 3	SCK (Serial Clock)	D05
SPI 4	MOSI (Master Output Slave Input)	D07
SPI 6	GND (power supply ESP)	GND
Pin 12	SS (Slave Select)	D08

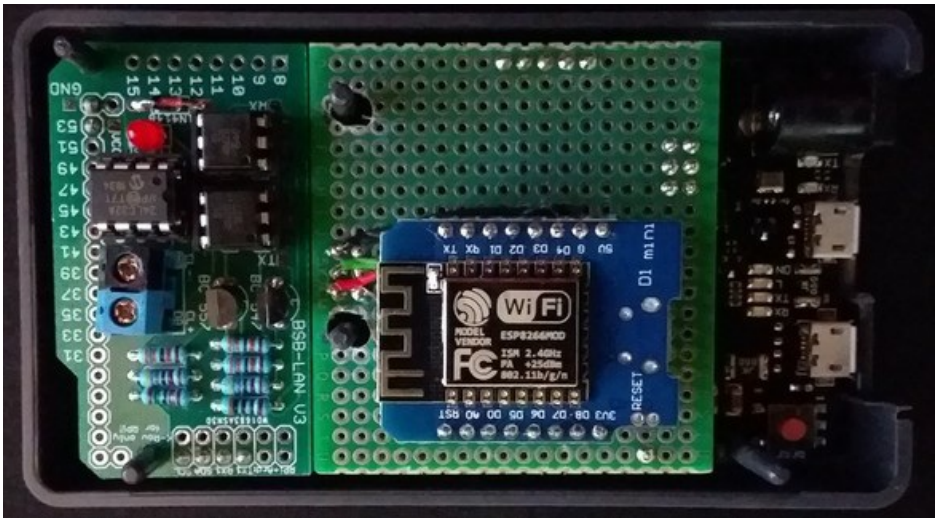
If no further component connected via SPI (e.g. LAN shield, card reader) is used, the connection of "SS" (SlaveSelect, DUE pin 12 = D08 at ESP8266) can be omitted.
In case of the use of SS the connection can also be made to another pin than pin 12, the corresponding pin must be defined accordingly in the file *BSB_lan_config.h*. In this case, however, it must be ensured that the pin to be used is not one of the protected pins and is not used elsewhere. It is therefore recommended to leave it at the default setting (pin 12).



The corresponding connectors at the Wemos D1.

It is suitable to remove the LAN shield, place an unpopulated circuit board on the Due and provide it with the appropriate connections. So the

Wemos D1 / NodeMCU can be placed stable onto the Due. Depending on the housing, the height may have to be taken into account.



Wemos D1 at an empty circuit board onto the Arduino Due.

Note:
However, this solution does not allow data to be logged to a microSD card. If this still should be possible using the WiFi connection, either a corresponding card module must be connected additionally or the ESP must be connected in parallel to the existing LAN shield. In both cases, the SS pin *must* be connected (see pin assignment/connection). If a parallel usage of LAN shield and ESP8266 is possible without problems has not been tested yet though.

Flashing the ESP8266:
The ESP8266 must be flashed with a special firmware. For the use of the Arduino IDE it must be ensured that the corresponding ESP8266 libraries have been installed before by using the board manager.
The required firmware WiFiSpiESP is already available as a zip-file in the BSB-LAN repository. The zip-file *must be unpacked in another folder than BSB_lan*! The ESP8266 has then to be flashed with the file *WiFiSPIESP.ino*.

Configuration of BSB-LAN:
To use the WiFi function, the definement `#define WIFI` must be activated in the file *BSB_lan_config.h*. Furthermore, the two variables `wifi_ssid` and `wifi_pass` must be adapted accordingly and the SSID of the WLAN and the password must be entered. These entries can also be changed afterwards via the web interface.

Notes:

- When using DHCP, the IP address assigned by the router can be read out in the Serial Monitor of the Arduino IDE when starting the DUE.
- When using the ESP WiFi solution, the host name is *not* WIZnetXYZXYZ, but usually ESP-XYZXYZ, where the digit-letter combination "XYZXYZ" after "ESP-" is composed of the last three bytes (the last six characters) of the MAC address of the ESP.
- When using the ESP WiFi solution, the MAC address of the ESP *can't* be set on your own.

12.2 The ESP32

Attention: We have tested a lot, but ALL functions etc. we have not been able to test. If you encounter any problems, incompatibilities, function restrictions or general bugs regarding the ESP32 usage, please report it (ideally in English as an issue in the repo)!

BSB-LAN is also executable on an ESP32. However, it is mandatory to make certain adjustments:

- Remove (or move) the two folders "ArduinoMDNS" and "WiFiSpi" from the BSB-LAN subfolder "src" - these must no longer be present in the "BSB-LAN" or "src" folder!
- Activate the definement `#define WIFI` in the file *BSB_LAN_config.h*!
- Enter the access data for your WLAN (SSID and password)!

Furthermore you have to pay attention to the following when using the Arduino IDE:

- In the Arduino IDE the ESP32 platform must be installed and available in the board manager. *Note: For the Joy-It board recommended in the following chapter, a user manual is available from the manufacturer. There, in addition to the board-specific pinout scheme, is also a general*

guide to installing and using ESP32 boards with the Arduino IDE!.

- Select the appropriate ESP32 board type and port in the Arduino IDE. If you use the recommended Joy-It board or an identical clone with a "WROOM32" chip, you have to select "ESP32 Dev Module" as board type in the Arduino IDE.
- Set the transfer speed/baud rate to 115200 (Attention: Per default the Arduino IDE usually sets 921600 for ESP32 boards).
- Please select the variant "Default 4MB with spiffs (1.2BM APP/1.5MB SPIFFS)" for "Partition Scheme".

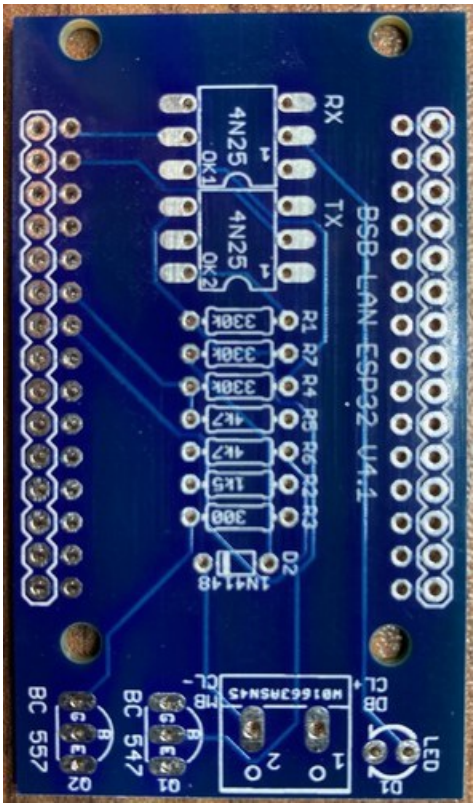
Note: If BSB-LAN cannot connect to WiFi on ESP32, it will set up its own access point "BSB-LAN" with password "BSB-LPB-PPS-LAN" for 30 minutes. After that, it will reboot and try to connect again.

Note: Even though the logging function also works with the ESP32, it is not advisable to use that function excessively due to the wear of the flash memory.

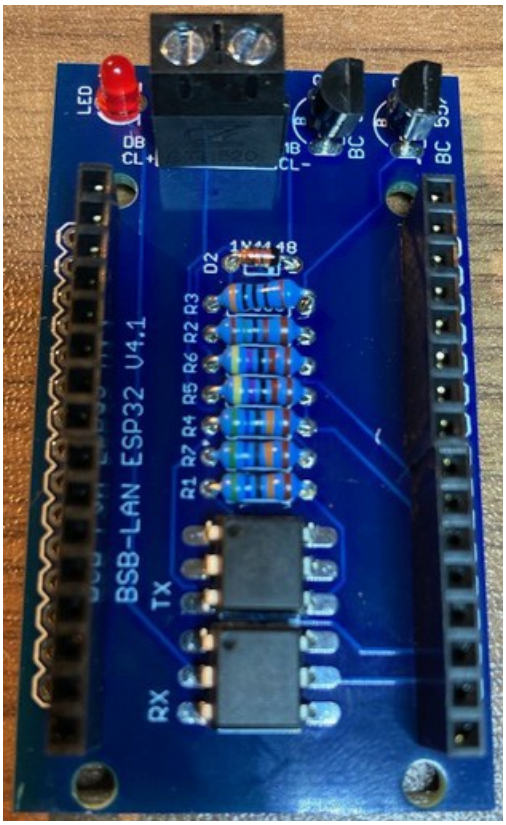
12.2.1 ESP32 With Specific "BSB-LAN ESP32"-Adapter

Attention: We have tested a lot, but ALL functions etc. we have not been able to test. If you encounter any problems, incompatibilities, function restrictions or general bugs regarding the ESP32 usage, please report it (ideally in English as an issue in the repo)!

For a specific ESP32 board variant there is a separate BSB-LAN adapter board: "BSB-LAN ESP32".



The "BSB-LAN ESP32" adapter board, unpopulated.

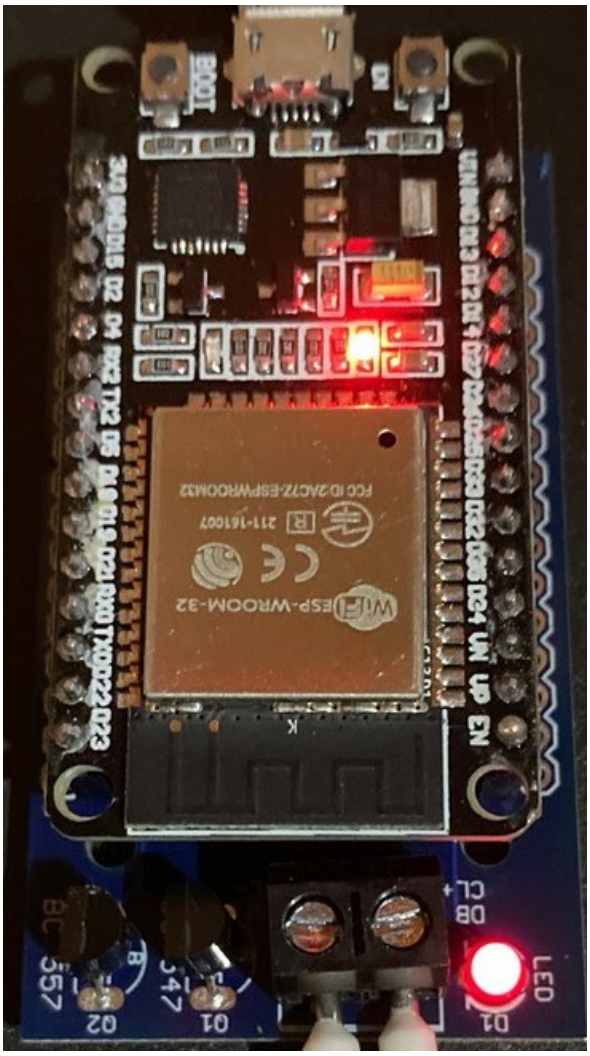


The "BSB-LAN ESP32" adapter board, assembled.

This BSB-LAN adapter board is designed for the *30 pin* ESP32 NodeMCU board from Joy-It (WROOM32 chip).
The ESP32 adapter version can also be used with an Olimex ESP32-OVB and can be plugged onto the ten pin UEXT connector of Olimex boards by adding a double row five pin socket (female pinheader, 2x5 pins, grid dimension 2.54mm) to the bottom side of the PCB.

12.2.1.1 ESP32: NodeMCU "Joy-It"

This BSB-LAN adapter board is designed for the *30 pin* ESP32 NodeMCU board from Joy-It (WROOM32 chip). A user manual is available for the board from the manufacturer. There are both the board-specific pinout scheme and a general guide to using ESP32 boards with the Arduino IDE!



The Joy-It ESP32-NodeMCU on the "BSB-LAN ESP32" adapter.

If the Joy-It board is not available and another NodeMCU-ESP32 board is used, two things must be taken care of in any case, so that the ESP32-specific BSB-LAN adapter fits:

1. The board *must* be a **30 pin** ESP32 NodeMCU! There are also 38 pin NodeMCUs - these do *not* fit!
2. The pinout scheme *must* be identical to that of the Joy-It board.

Note:
When using the Joy-It-Board or an identical clone with a "WROOM32" chip, "ESP32 Dev Module" must be selected as board type in the Arduino IDE.

12.2.2 ESP32 With Due-Compatible BSB-LAN-Adapter From V3

Attention: We have tested a lot, but ALL functions etc. we have not been able to test. If you encounter any problems, incompatibilities, function restrictions or general bugs regarding the ESP32 usage, please report it (ideally in English as an issue in the repo)!

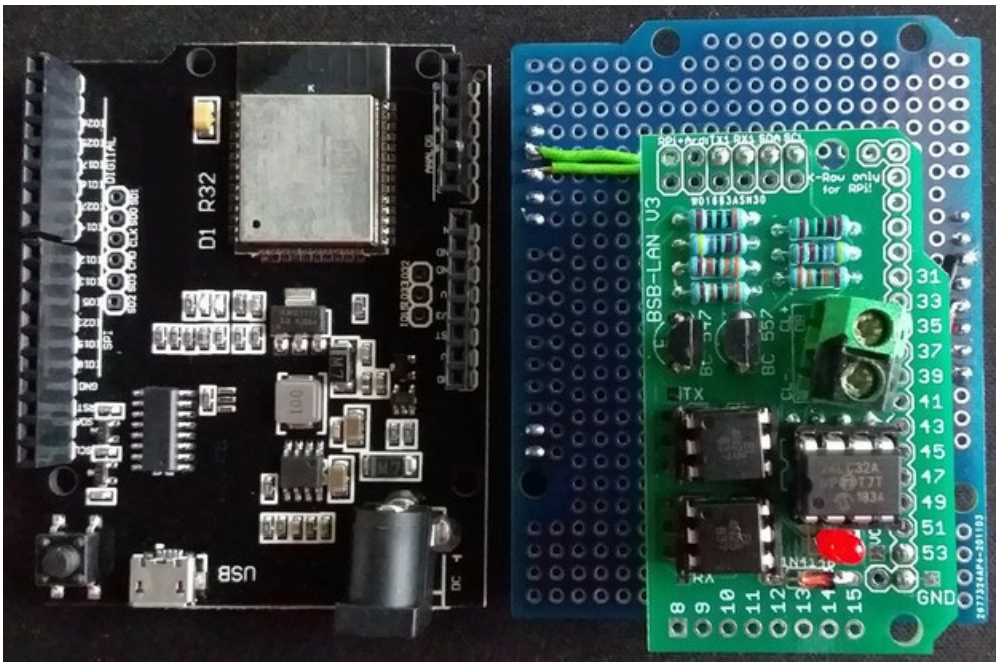
The previous Due-compatible adapter (from v3) can also be used with an ESP32. The EEPROM of the adapter is not needed/used here and is accordingly also not to be considered with the wiring.
With the choice of an ESP32 here is no compelling restriction on the Joy-It-board-compatible NodeMCU variant mentioned before, since a 'loose' wiring or the self-construction of a small adapter board for the more stable admission of the BSB-LAN adapter and the ESP32 is necessary anyway. However, care should be taken to ensure that the pin numbers/assignments given below match those of the selected ESP32.

The connections are to be made as follows:

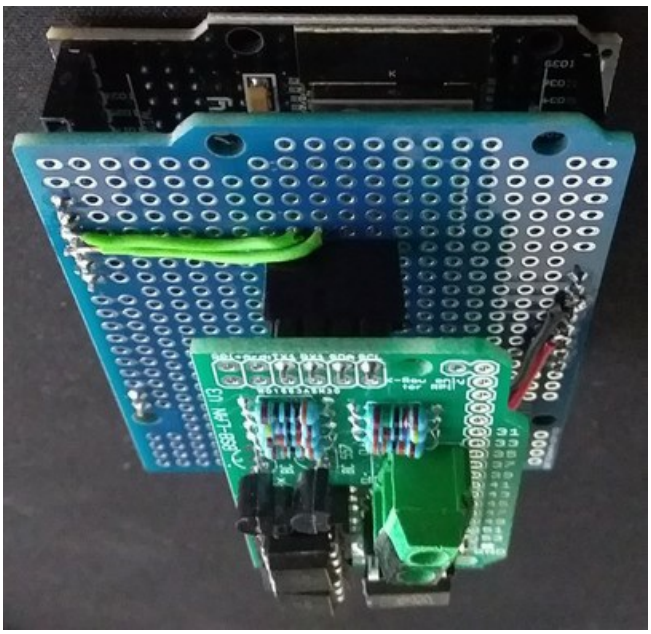
BSB-LAN adapter from v3	Function	ESP32 board
Pin 53	VCC (power supply adapter)	3,3V
GND	GND (power supply adapter)	GND

BSB-LAN adapter from v3	Function	ESP32 board
TX1	TX (send)	Pin 17 (TX2)
RX1	RX (receive)	Pin 16 (RX2)

As an example, an "ESP32 D1 R32 developer board" (WROOM32 chip) in the size of an Arduino Uno with a self-made adapter board (Uno-compatible prototyping board) for the inclusion of the BSB-LAN adapter v3 (Due version) is shown below. Of course, other variants are also possible, such as with an ESP32 NodeMCU and an appropriately adapted breadboard.



Left the "ESP32 D1 R32" board, right the corresponding plug-on board for the BSB-LAN adapter v3 (due version).



The complete Assembly.

12.2.3 ESP32 With Due-Compatible BSB-LAN-Adapter V2

Attention: We have tested a lot, but ALL functions etc. we have not been able to test. If you encounter any problems, incompatibilities, function restrictions or general bugs regarding the ESP32 usage, please report it (ideally in English as an issue in the repo)!

The BSB-LAN adapter v2 can also be operated on an ESP32. In this way it is possible to benefit from the further development and the new functions of the BSB-LAN software from v2.x without having to purchase a new adapter. To do this, some changes must be made to the adapter itself, which are described below.

Caution: The steps described below to 'convert' the adapter to 3.3V are only valid for use on an ESP32 - on a Due the adapter v2 cannot be used due to the missing EEPROM!

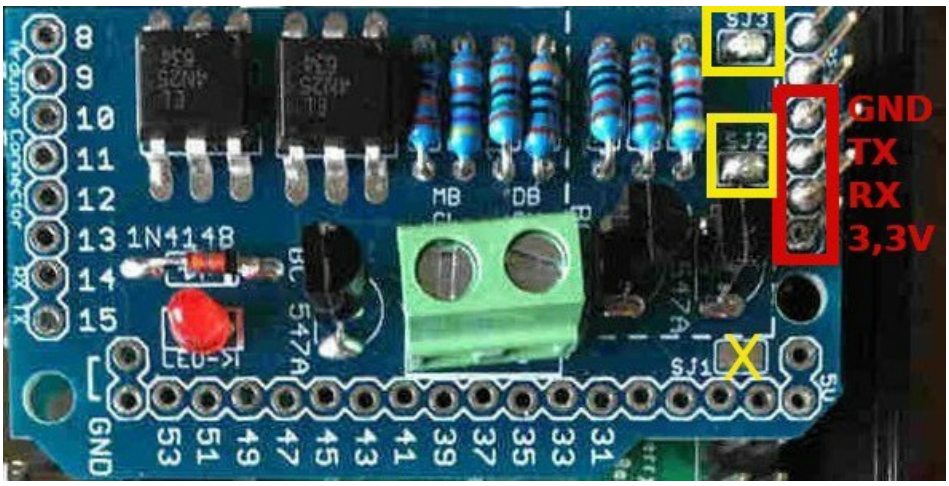
To successfully operate the adapter v2 on an ESP32, the adapter must be 'adjusted' to operate with 3.3V. This is already provided for use with a Raspberry Pi. The following steps need to be taken:

- The adapter must be *completely* assembled. If the adapter is so far only equipped for use with the Arduino Mega 2560, the following components must be retrofitted:
  - 1x resistor 4.7kΩ (→ R11)
  - 2x resistor 10kΩ (→ R12, R13)
  - 1x transistor BC557A (→ Q11)
  - 1x transistor BC547A (→ Q12)
- The solder jumpers *SJ2* and *SJ3* are to be *closed* by a solder point.
- The solder jumper *SJ1* is to be *removed*.

Now the adapter is prepared for operation on a 3.3V system.
For connection to the ESP now the "RasPi" contact row must be used and connected to the ESP32 as follows:

BSB-LAN adapter v2	Function	ESP32 board
Pin 06	GND (power supply adapter)	GND
Pin 08	TX (send)	Pin 17 (TX2)
Pin 10	RX (receive)	Pin 16 (RX2)
Pin 12	3,3V (power supply adapter)	3,3V

The following picture shows a correspondingly equipped adapter v2. The yellow "X" at SJ1 marks the *removed* solder jumper (the non-closed contact), the two yellow outlines at SJ2 and SJ3 mark the solder jumpers *to be closed*.



The adjusted adapter v2 for use with an ESP32.

It is advisable to solder additional pins for the four contacts on the adapter and build yourself a small adapter board from a perforated board and pin headers, on which the adapter and the ESP32 board can be plugged to ensure a stable setup and a secure connection.

12.3 Usage of Optional Sensors: DHT22, DS18B20, BME280

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

There is the possibility to connect additional sensors directly to certain pins of the adapter or the Arduino:

- DHT22 (temperature, humidity; parameter numbers 20100-20199)

- DS18B20 (OneWire sensor: temperature; parameter numbers 20300-20399)
- BME280 (temperature, humidity, pressure; parameter numbers 20200-20299)

The necessary libraries for the Arduino IDE are already included in the repository of the BSB-LAN software.

Usually, the sensors can be connected to GND and +3,3V of the adapter/Arduino (by usage of the necessary additional pullup-resistors!).
For the usage of these sensors, one has to activate the belonging definements in the file *BSB_lan_config.h* and has to set the specific pins which are used for DATA (also see chapter 5). Make sure you don't use any of the protected pins listed in the file *BSB_lan_config.h*!

After successful installation you can access the values of the sensors either by clicking at the button "sensors" at the top of the webinterface, by clicking at the category "One Wire, DHT & MAX! Sensors" or by using the url command with the specific number of that category.

Besides that, they are also displayed in the IPWE extension by default, which can be accessed by using the URL `<ip-address>/ipwe.cgi`. For using the IPWE extension, one has to activate the belonging definement in the file *BSB_lan_config.h* though.

If you want to log the measured values or if you want to create 24h average calculations, you can realize that by adjusting the belonging parameters in the file *BSB_lan_config.h*.
werden.

12.3.1 Notes on DHT22 Temperature/Humidity Sensors

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

DHT22 sensors are often advertised as "1 wire", but they are NOT part of the real OneWire bus system by Maxim Integrated and aren't compatible with these components.
Furthermore they are not even part of any real bus system, because the sensors don't have any specific sensor id and can't be connected to the same DATA-pin if you are using more than one sensor.

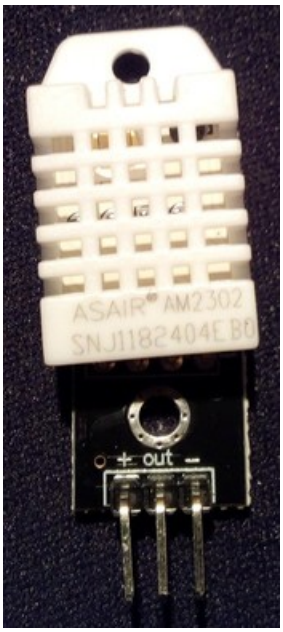
Usually these sensors have four pins, but only three of these are connected internally. Most in the time it's the third pin from the left (when viewed from the front) which isn't connected, but you should verify this before soldering.
The most common pinout is:

- Pin 1 = VCC (+)
- Pin 2 = DATA
- Pin 3 = usually not connected
- Pin 4 = GND (-)

When you connect the sensor, an additional pullup resistance has to be placed between VCC (pin 1) and DATA (pin 2) which should be in the range between 4,7kΩ to 10kΩ. In most cases a value of 10kΩ is suggested, but this should be determined individually (especially if any problems with the sensor occur).

Please note:
If more than one DHT22 sensor should be used, you have to use an own pin at the Arduino for each DATA pin of the sensor. Furthermore you have to define them in the file BSB_LAN_config.h.

Besides the 'plain' sensors there are models which are already soldered onto a little circuit board, where the three necessary pins are lead out and labeled. The following picture shows one of these types with the identical sensor AM2302.



The query of the sensors/measured values can be done either via direct parameter call (`URL/20100-20199`) or by calling the corresponding category. The following screenshot shows the web output of a connected DHT22 sensor.

20100 One Wire, DHT & MAX! Sensors - DHT22 Sensor ID #1: 2	2
20101 One Wire, DHT & MAX! Sensors - DHT22 Sensor Temperatur #1: 19.90 °C	19.90
20102 One Wire, DHT & MAX! Sensors - DHT22 Sensor Luftfeuchtigkeit #1: 41.60 %	41.60
20103 One Wire, DHT & MAX! Sensors - DHT22 Sensor Abs Luftfeuchtigkeit #1: 7.13 g/m³	7.13

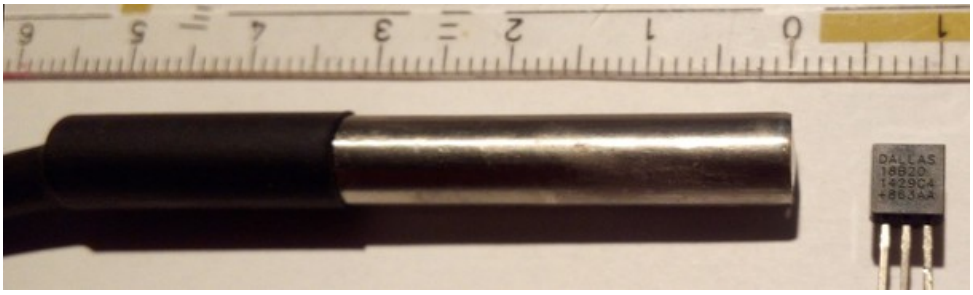
Display of the measured values of a DHT22 in the web interface (category "One Wire, DHT & MAX! Sensors").

Note:
You can find various tutorials and examples within the internet about the installation and usage of DHT22 sensors.

12.3.2 Notes on DS18B20 Temperature Sensors

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

Sensors of the type DS18B20 are 'real' 1-wire/OneWire components of Maxim Integrated (initially Dallas Semiconductor).
Each sensors has a unique internal sensor id which allows the clear identification of a certain sensor within a more complex installation of the bus system - if you wrote down the specific id for each sensor (regard the note in chapter 12.3).
Besides the regular TO-92 type they are also available as waterproof capsuled types, which already have a cable connected.



Especially for the usage within heating system installations the capsuled type is very interesting, because you can realize an individual (and waterproof!) installation easily and const-effective.

The query of the sensors/measured values can be done either via direct parameter call (`URL/20300-20399`) or by calling the corresponding category. The following screenshot shows the web output of four DS18B20 sensors connected to pin 7.

20300 One Wire, DHT & MAX! Sensors - DS18B20 Sensor ID #1: 28B0333B07000075	28B0333B07000075
20301 One Wire, DHT & MAX! Sensors - DS18B20 Sensor Temperatur #1: 21.13 °C	21.13
20302 One Wire, DHT & MAX! Sensors - DS18B20 Sensor ID #2: 28AEA63B07000058	28AEA63B07000058
20303 One Wire, DHT & MAX! Sensors - DS18B20 Sensor Temperatur #2: 20.25 °C	20.25
20304 One Wire, DHT & MAX! Sensors - DS18B20 Sensor ID #3: 28FF4158C2160493	28FF4158C2160493
20305 One Wire, DHT & MAX! Sensors - DS18B20 Sensor Temperatur #3: 19.81 °C	19.81
20306 One Wire, DHT & MAX! Sensors - DS18B20 Sensor ID #4: 28FF4B03C0160573	28FF4B03C0160573
20307 One Wire, DHT & MAX! Sensors - DS18B20 Sensor Temperatur #4: 19.87 °C	19.87

Display of the measured values of four DS18B20 in the web interface (category "One Wire, DHT & MAX! Sensors").

Note:
If you are using DS18B20 sensors, the specific sensor id of each sensor will also be listed within the output of the category sensors (and the output of the IPWE extension, if used). Especially if more than one sensor will be added to the system, these unique sensor ids are necessary to identify a specific sensor later. So if you integrate BSB-LAN and/or these sensors in your home automation software, you should consider this (e.g. use RegEx on the sensor ids).
It's adviseable to read out the sensor id (e.g. by using /K49) and label each sensor, so that you don't get confused later. For this, you can raise or lower the temperature of one sensor (e.g. hold it in your hand) and query the category sensors again after a certain time. Now you can see the changed value of one sensor and write down the specific sensor id.
Besides that, if any sensor will be exchanged or added, most of the time the displayed order (within the output of the category sensors or the IPWE extension) of the sensors will change also, because internally they are listed following the specific sensor ids. So if you only adjust the reading following the order and name the sensors like that, it can happen, that the belonging name doesn't show the correct sensor anymore. The following screenshots show this circumstance.
If any changes within the installation of the sensors occur (e.g. if you exchange, add or remove something), you have to reboot the Arduio, so that the sensors will be initially read out and added to the software.

Notes on the elecrtical installation:
Each sensor usually offers three pins: VCC, DATA and GND.
Within the capsuled types, the colors of belonging wires are often as follows:

- Red = VCC (+3,3V)
- Yellow = DATA
- Black = GND (-)

If you are using more than one sensor and/or larger cable lengths, it's advisable to add a 100nF ceramic capacitor (and maybe also an addditional 10µF tantal capacitor) for each sensor. The capacitors should be added as close as possible to the sensor and need to be connected between GND and VCC so that a brownout at the time of the query will be compensated.

Besides the (optional but advisable) usage of capacitors, you have to use a pullup resistance (only one!) at the output of the adapter/Arduino and place it between DATA and VCC (+3,3V). If you are using more than one sensor and/or larger cable lengths, you probably have to evaluate the correct dimension of the resistor, which can be smaller than the 4,7kΩ which is suggested most of the times.

Furthermore, in more complex or larger installations, it seems in individual cases that the voltage supply with the 3.3V of the Due does not always allow a problem-free operation of the sensors. Since these OneWire sensors are "open drain", they can also be operated with 5V of the Due, which seems to result in a more stable operation. However, it must then be ensured that the 5V is *never* applied to the GPIO of the Due!
For the installation this means that VCC of the sensors is connected to the 5V pin of the Due, but the PullUp resistor to be used must be placed between DATA and a 3.3V pin of the Due!

Notes:

- If you are using the mentioned capsuled and already wired types, it's usually sufficient to place the capacitors where the wires will be connected. So you don't have to cut the wires at the capsule to place the capacitor there (according to experience, at least with the types which come with a cable length of 1m or 3m it's not necessary).
- In contrary to ceramic capacitors you have to pay attention to the correct polarity if you are using additional tantal capacitors!
- It's not advisable to use the 'parsite power mode'.
- It's advisable to use a shielded cable for the connection. The shield should be connected to GND at one end of the cable.

- To minimize the risk of electrical interference, try not to lead the cable parallel to power cords. Besides that, you can also add a ferrite ring to minimize the risk of electrical interference which maybe can come from the power supply of the Arduino. Just lead the cable a few times through the ferrite ring.

If you have to use *larger* cable lengths, it's necessary to pay attention to the correct network topology. Have a look at the tutorial which was written from the manufacturer: "Guidelines for Reliable Long Line 1-Wire Networks".

Note:
You can find various tutorials and examples within the internet about the installation and usage of DS18B20 sensors.

Summary of needed parts for an installation:

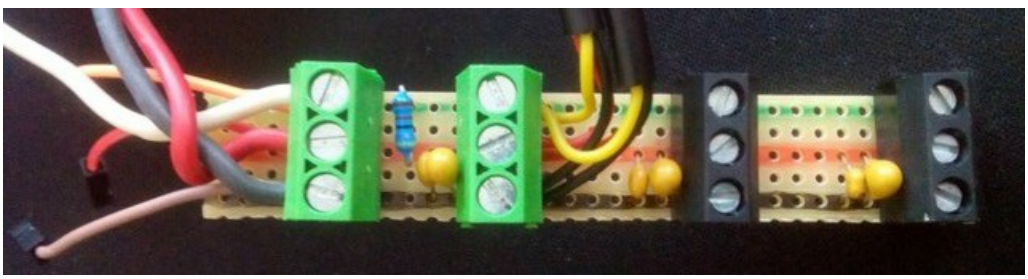
- three-wired cable (if shielded, connect the shield at one end to GND)
- one pullup resistance 4,7kΩ or maybe smaller, positioned between VCC and DATA at the adapter/Arduino
- ceramic capacitor 100nF, one for each sensor, positioned between VCC and GND close to the sensor
- optional: tantal capacitor 10µF, one for each sensor (additional to the ceramic capacitor!), positioned between VCC and GND close to the sensor (please pay attention to the correct polarity!)
- optional: screw terminals, circuit board, housing, ...

Notes for the usage within your heating system installation:

- If you want to use the capsuled types of sensors, especially within bigger installations it can be adviseable to use the version with 3m cable instead of 1m cable. They are only a little bit more expensive but offer a greater freedom of movement when you want to place the sensors.
- If you want to place the sensors at some pipes, it's adviseable to create a little bed made of thermal paste for the contact area. Fasten the sensor with a metal pipe clamp to the pipe and also fasten the cable itself with a cable tie, so that tensile forces won't work on the sensor itself and that the sensors stays in place. Of course you need to place the sensor between the pipe an the insulation and close the insulation after you are done with the installation. If there is no pipe insulation it's advisable to -at least- cover the sensor with a piece of insulation, so that it's not affected by any cold air or so.
- In general, the sensors should me mounted one or two meters away from a heat source, so that they aren't affected by that.

Please note:
Already installed sensors which belong to the heating system (e.g. sensors for a warm water tank or a heating buffer tank) are always more important than any sensor for your home automation system! The given installation of your existent heating system should never be adversely affected by any optional installed DS18B20 sensor!

Construction plan:
If you want to set up an installation with more than one sensor and the common capsuled sensors with 1m or 3m cable length, you can build a little 'distribution box'. For this, you can solder the connection wires of the sensors and the belonging capacitors in line onto a circuit board. If you use screw terminals instead of soldering the sensors straight to the board, you can easily add or exchange sensors later. At the 'beginning' of this board, you connect the cable which leads to the adapter/Arduino. The following pictures show two of these little 'distribution boxes' I made - they work perfectly.



12.3.3 Notes on BME280 Sensors

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

Sensors of the BME280 type offer three (or five) measured variables: Temperature, humidity (plus the calculated absolute humidity) and air pressure (plus the calculated altitude). They are small, usually uncomplicated to connect and provide (sufficiently) accurate measurement results. **Up to two sensors of the type BME280 can be connected to the I2C bus of the Arduino Due (also to the Mega 2560).** To use them, the corresponding definition in the file *BSB_lan_config.h* must be activated and the number of connected sensors must be defined (see chapter 5.2). *Note: In principle BME280 can also be connected to an SPI, but* ***not*** *on the Arduino of our BSB-LAN setup!*

A BME280 sensor on a typical breakout board (clone); left = front, right = back.

The following points must be observed:

- Make sure that the sensor is of type BME280 (and not e.g. a BMP280, BMP180,..).
- Make sure that you use a module that already has pull-up resistors on the breakout board (like on the picture above). If your variant does *no* have pull-up resistors installed, you have to add them when connecting it to the Arduino (approx. 10kOhm, connect between SDA and 3.3V and between SCL and 3.3V)!
- Make sure that the first sensor has the I2C address 0x76! This is usually the case with the module shown above.
- The second sensor must get the address 0x77. How to do this on the module shown above is described below.

- Make sure you connect the sensor to the 3.3V pin of the Arduino! The module shown above has a voltage regulator and level shifter built in, so in this case you *could* connect it to 5V - but to make sure that there is never 5V at SDA/SCL, you should always prefer to connect it to 3.3V.

Connection

The breakout board is usually already clearly labeled, so the connections can be clearly identified here.
Depending on the Arduino used, a different I2C connector must be used:

- The **Due** has two I2C bus connections: SDA/SCL at pins 20/21 and SDA1/SCL1. Care must be taken to use the **SDA1 & SCL1** connectors, as the BSB-LAN adapter already uses the SDA/SCL connectors. SDA1/SCL1 are located next to the "AREF" pin. They are usually covered by the LAN-Shield and are not carried out upwards to/through the LAN shield. However, they are accessible below the LAN shield directly on the Due. For an exact positioning of SDA1/SCL1 please have a look at the pinout diagram in appendix B.
- The **Mega 2560**, on the other hand, has only one I2C bus connector: SDA/SCL on pins 20/21. This is not occupied by the old adapter v2, the connector can be used for the BME280.

The wiring has to be done as follows:

BME280	DUE	Mega2560
VIN	3,3V	3,3V
GND	GND	GND
SDA	SDA1	SDA 20
SCL	SCL1	SCL 21

Addressing

Common breakout boards like the BME280 module shown above have three solder pads on the front side below the actual sensor, where usually the *left* and the middle solder pad are connected by a conducting path. This usually corresponds to the address 0x76. The following picture shows this connection circled in yellow.

Address 0x76: trace between the left and the middle solder point.

If you want to connect a second sensor in parallel, you have to cut this conducting path carefully(!) and conscientiously with a fine sharp object (e.g. cutter, scalpel). After that the *right* and the middle pin have to be connected by some solder. The following picture shows the necessary steps: the red line on the left marks the necessary 'cut' on the board, the green line on the right marks the connection to be made afterwards using solder.

Address 0x77: The red line marks the cut trace, the green line marks the new connection to be made.

Readout

The measured values of the connected BME280(s) can be read out as usual, e.g. by calling up the category "One Wire, DHT & MAX! Sensors" under "Heating Functions", by a direct click on the button "Sensors" or also by entering the specific parameter numbers. BME280 sensors can be found in the number range 'URL/20200-20299'. If a logging, a display within the IPWE extension etc. should be done, the specific parameter numbers of the desired measured values of the respective sensor have to be used.
The following screenshot shows the corresponding display of a BME280 within the category "One Wire, DHT & MAX! sensors".

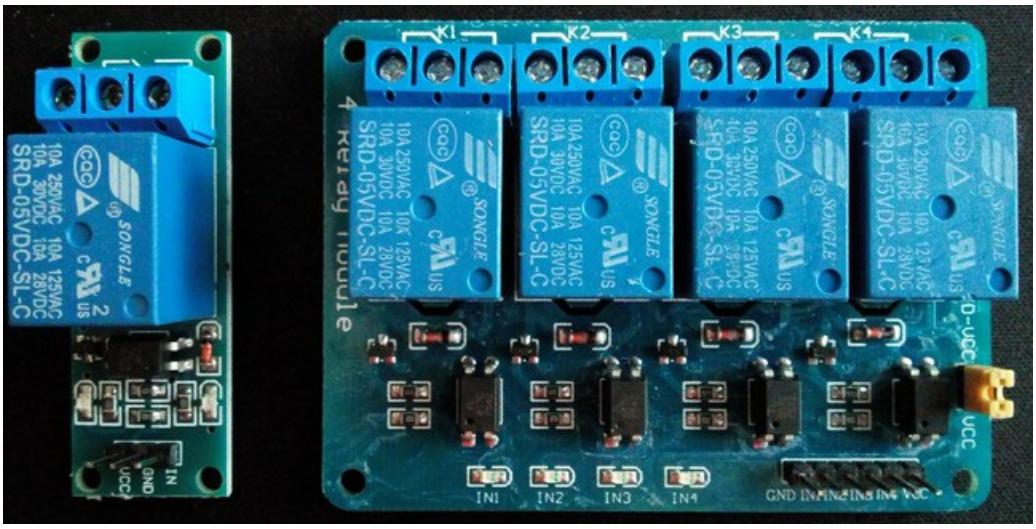
20200 One Wire, DHT & MAX! Sensors - BME280 Sensor ID #1: 76	76
20201 One Wire, DHT & MAX! Sensors - BME280 Sensor Temperatur #1: 18.27 °C	18.27
20202 One Wire, DHT & MAX! Sensors - BME280 Sensor Luftfeuchtigkeit #1: 38.98 %	38.98
20203 One Wire, DHT & MAX! Sensors - BME280 Sensor Pressure #1: 1012.54 hPa	1012.54
20204 One Wire, DHT & MAX! Sensors - BME280 Sensor Altitude #1: 3.83 m	3.83
20205 One Wire, DHT & MAX! Sensors - BME280 Sensor Abs Luftfeuchtigkeit #1: 6.07 g/m³	6.07

Display of the measured values of a BME280 in the web interface (category "One Wire, DHT & MAX! Sensors".

12.4 Relays and Relayboards

ATTENTION: The GPIOs of the Arduino Due are only 3.3v compatible!

In general it's possible and within BSB-LAN already implemented to connect and query a relay which is connected to the Arduino. By this one couldn't only change the state of a relay by sending a specific command, it's also possible to just query the state.
It is NOT possible to connect the Arduino directly with the multifunctional inputs of the controller!



A single and a 4-channel relaymodule for the usage with an Arduino.

The often cheap relaymodules available for the usage with an Arduino are often already supplied with a relay which can handle high voltage like 125V or 230V. However, due to poor quality or just an overload, different risky damage can occur. Because of that one should consider to (additionally) use common couple or solid state relays which are used by electricians. in that case one should see the specific data sheet to confirm that the electrical current of the Arduino is strong enough to trigger the swithcing process of the relay.

WATCH OUT:
Electrical installations should only be done by an electrician! High voltage like 230V or 125V can be deadly! *It's adviseable to already include an electrician at the state of planning.*

A common coupling relay. At this specific type, the corresponding pins at the Arduino have to be connected with "14" and "13".

Example:
If the controller of a solarthermic installation isn't already connected with the controller of the heating system, it's possible to query the state of the pump by installing a coupling relay parallel to the pump and connect the other 'side' of the relay with the specific pins of the Arduino. Now you can query the state of the relay and therefore the state of the pump with the Arduino.

12.5 MAX! Components

BSB-LAN is already prepared for the usage of MAX! heating system components. MAX! thermostats that shall be included into BSB-LAN, have to be entered with their serial number (printed on a small label, sometimes in the battery compartment) in the file *BSB_lan_config.h* into the array `max_device_list[]`. After starting BSB-LAN, the pairing button has to be pressed on the thermostats in order to establish a connection between BSB-LAN and the thermostats.

In *BSB_lan_custom.h* you can use the following variables for using MAX! devices:

- `custom_timer`

This variable is set to the value of millis() with each iteration of the loop() function.

- `custom_timer_compare`
  This variable can be used in conjunction with `custom_timer` to create timed executions of tasks, for example to execute a function every x milliseconds.

In addition to that, all global variables from *BSB_lan.ino* are available. In regard to MAX! functionality, these are most notably:

- `max_devices[]`
  This array contains the DeviceID of each paired MAX! device. You can use this for example to exclude specific thermostats from calculations etc.

- `max_cur_temp[]`
  This array contains the current temperature of each thermostat. However, only temperatures from wall thermostats are reliable because they transmit their temperature constantly and regularly. Other thermostats do this only when there is a change in the valve opening or upon a new time schedule.

- `max_dst_temp[]`
  This array contains the desired temperature of each thermostat.

- `max_valve[]`
  This array contains the current valve opening of a thermostat (wall thermostats only carry this value when they are paired with a heater thermostat).

The order inside of these arrays is always the same, i.e. if `max_devices[3]` is wall thermostat with ID xyz in the living room, then `max_cur_temp[3]` contains the current temperature in the living room, `max_dst_temp[3]` the desired temperature in the living room etc.

The order inside `max_devices[]` depends on how the devices have been paired with BSB-LAN and remains the same after restarts of BSB-LAN since they are stored in EEPROM until this is erased by calling `http://<IP-Adresse>/N`. However, one should not completely rely on this and rather compare the ID stored in `max_device[]` for example when planning to ignore a specific thermostat in some kind of calculations. You can obtain this ID from the second column of `http://<IP-Adresse>/X` and take note that this is not the same as the ID printed on the label.

Important note for those users who use a Max!Cube that has been flashed to CUL/CUNO (see information here):
If BSB-LAN was not running (or was busy otherwise) when the CUNO was set up, then you have to press the pairing button again on these devices, because only in that specific pairing process the ID printed on the devices label is sent together with the internally used device ID (and is also used by FHEM).

You can also use the MAX! thermostats to calculate a weighted or average current or desired temperature (see here for configuring MAX devices under FHEM and here for using the average temperature in FHEM).

FHEM forum user *„Andreas29"* has created an example on how to use MAX! thermostats with BSB-LAN without using FHEM. A detailed description can be found in this forum post here. The "Arduino room controller light" is described in chapter 12.6.2.

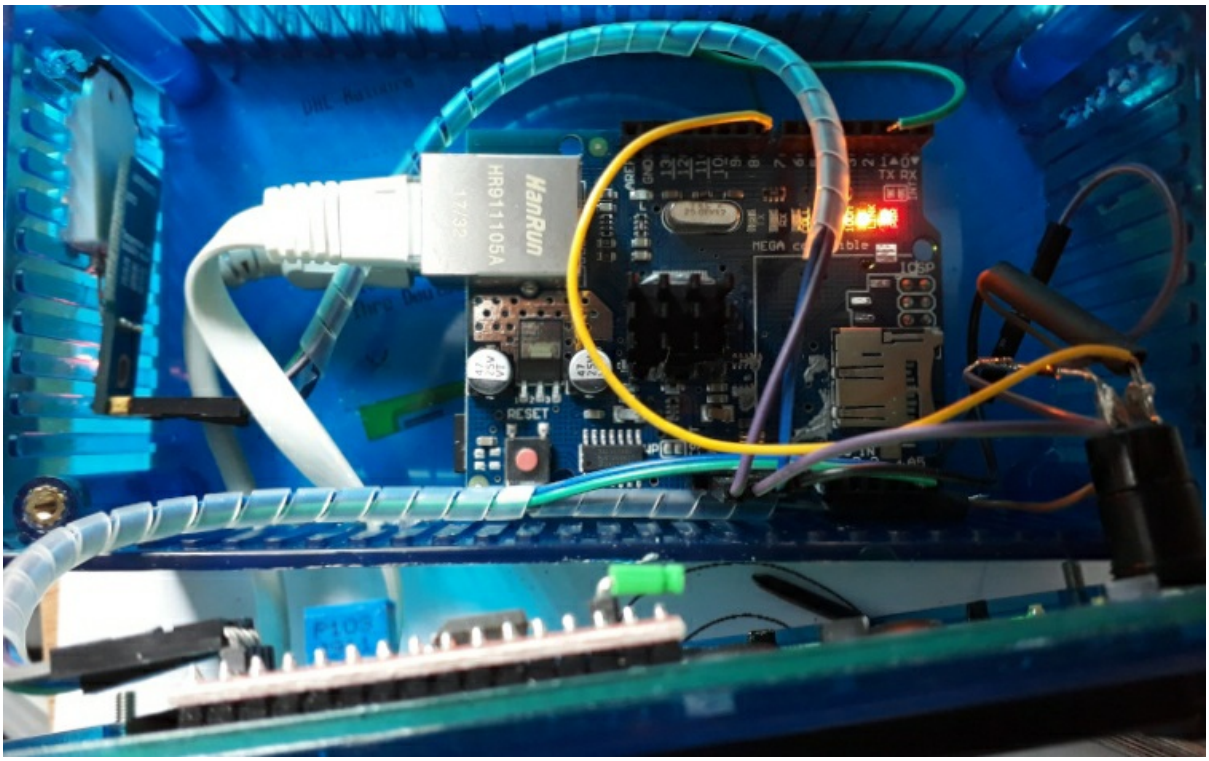
12.6 Own Hardwaresolutions

The following solutions have been developed by BSB-LAN users. They should not only be a stimulation for re-building but also an example what's possible with additional own built hardware solutions in combination with BSB-LAN.

If you also created something by your own of which you think that it could be interesting for other users, please feel free to contact me (Ulf) via email at `adapter (at) quantentunnel.de`, so that I eventually can present it here in the manual. Thanks!

12.6.1 Substitute for a Room Unit (Arduino Uno, LAN Shield, DHT22, Display, Push Button Switch)

The member *„Andreas29"* of the German FHEM forum has built a substitute for a room unit, based on an Arduino Uno. Besides the data from a DHT22 sensor, the current state of function of the heating system is displayed on a 4x20 LCD. With a little push button he imitates the function of the presence button of a common room unit.



The 'inside' of his substitute of a room unit.

The display of his own built room unit.

A more detailed description including the circuit diagram and the software is available here in the German FHEM forum.

Also, he expanded the functionality and implemented push messaging for certain error cases. The description and the software can be found here in the German FHEM forum.

12.6.2 Room Temperature Sensor (Wemos D1 mini, DHT22, Display)

The member *„Gizmo_the_great"* of the FHEM forum has built a room temperature sensor based on a Wemos D1 mini and a DHT22 sensor. The current temperatures on the heating circuits 1 and 2 are additionally displayed at an OLED display. The Wemos D1 ist running ESPeasy.

A more detailed description of his project you can find in his GitHub Repo.

12.6.3 Substitute for a Room Unit with UDP Communication (LAN Connection)

FHEM forum member *"fabulous "* has built a substitute for a room unit based on the above-mentioned variant of user "Andreas29", which communicates with the BSB LAN adapter via UDP. An Arduino Uno including LAN shield, a 20x4 LCD and a push button are used. A detailed description and the corresponding code can be found here.

12.7 LAN Options for the BSB-LPB-LAN Adapter

Even though the wired LAN connection is definitely the best option for integrating BSB-LAN into your network, it could be necessary to create an alternative way of connection, because a full-range wired connection (bus cable or LAN cable) just isn't possible.

12.7.1 Usage of a PowerLAN / dLAN

The use of powerline adapters for expanding the LAN is an option, which could be the best and most reliable solution.

However, sometimes powerline installations can cause trouble because of possible interferences they may cause. If you have separated phases within your electrical installation, it may just not work though. In that case ask an electrician about a phase coupler that he may could install.

12.7.2 WLAN: Usage of an Additional Router

Another option is to connect the Arduino via LAN with an old WLAN router (e.g. an old FritzBox) and integrate the router in your network via WLAN as a client. The speed of transmission usually is fast enough for the use of BSB-LAN. If the WLAN signal is weak, you can probably try to change the antennas and mount bigger ones.

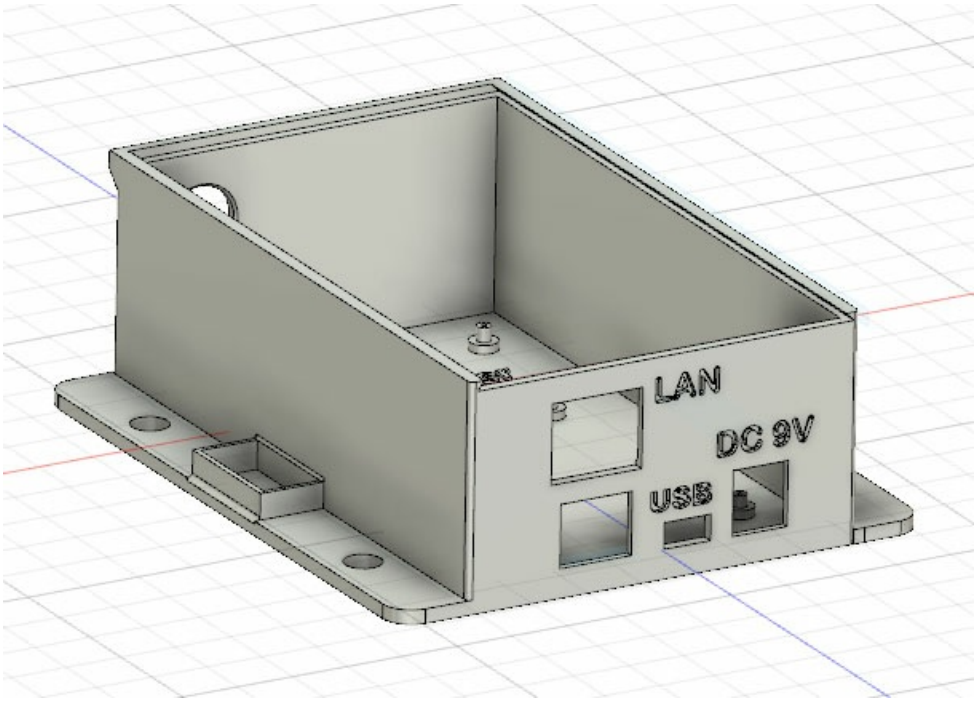
In addition to the use of a 'normal' router, there are small devices on the market that offer a RJ45 jack and a WLAN client or a WLAN client bridge mode. These devices connect to the network via WLAN (like the FritzBox solution described above). The Arduino can be connected via LAN cable to the device. These kinds of devices are often very small and can be plugged in a power outlet, so that the installation of the hardware can usually be done quite easily.

However, a stable and reliable WLAN connection should be achieved. Especially, if you are using additional smart home software to create logfiles, if you are using additional hardware like thermostats or if you want to control and influence the behaviour of your heating system.

12.8 Housing

The market offers just a small range of housings which are compatible for an Arduino Due plus additional shields. If you search for them, you probably won't find anything. In that case look out for housing which are designed for an Arduino Mega 2560, because it has the same form factor as the Due. Try to find a housing, which can accommodate the whole setup including the LAN shield though, because many housings are only designed to accommodate the plain Mega. This kind of housing has some cutouts in the top cover to plug in additional shields, but in that case the LAN shield and the adapter won't be protected at all.

Besides commercial products and creative own built solutions, a 3D printer could be used to create a great housing.
The member "EPo" of the German FHEM forum was so kind to create and offer STL datafiles for a housing.
Thanks a lot!



3D printer model of the housing for the Arduino Due, the LAN-Shield and the adapter v3.

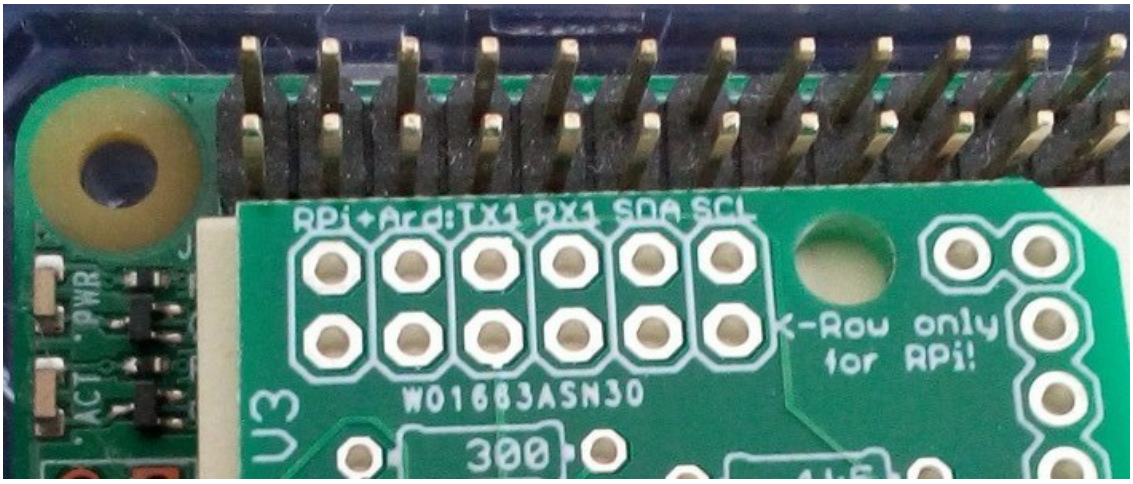
The STL data files are already included in the repository of BSB-LAN.

12.9 Raspberry Pi

The adapter v3 could also be used in conjunction with a Raspberry Pi. Therefore you have to pay attention to some points:

- **A usage of the BSB-LAN-Software is NOT possible (see notes below)!**

- You only have to use double-rowed female headers which fit the RPi pins (instead of the pin headers for the usage with an Arduino Due!).
- With the complete length of the female headers (6 pins 'long', so 12 pins in summary) the first pair of the adapter must NOT be plugged to the first pair of the RPi pins (1/2), you have to start with the second pair of the RPi pins (3/4). In other words: make sure that the pin of the adapter labeled as TX1 will fit on the RPi pin 8 (= GPIO 14, UART0_TXD), the pin RX1 in the RPi pin 10 (= GPIO 15, UART0_RXD) and so on.
  Note: This counting refers to the official RPi pinout and the naming.
  The picture below shows the plain adapter *next to* the belonging RPi pins just to visualize the displacement/alignment on the longitudinal axis.
- Before the usage of the software, the Pin 7 (GPIO 4) of the RPi must be
  a) defined as an output pin and
  b) set to "HIGH" within the OS of the RPi to achieve the power supply of the adapter.
  Therefore your have to execute two commands in the terminal (probably with a leading 'sudo'):
  `gpio -1 mode 7 output`
  `gpio -1 write 7 1`



Exemplary alignment of the adapter along the longitudinal axis of the RPi pins.

IMPORTANT NOTES:

- ***For the usage of the adapter in conjunction with an RPi you have to use a complete different software: "bsb_gateway" by J. Loehnert!***
- *For any support please contact the author of bsb_gateway!*
- *We can not and will not provide any support with regard to RPi use!*
- *From our side, the use of the adapter with the above mentioned software was only tested on an RPi 2. We are not able to judge whether it works properly with more recent RPi versions!*
- *For the usage of the adapter with an RPi at the PPS interface, the Python script PPS-monitor by D. Spinellis can be used.*

This manual only refers to BSB-LAN!

Further on to chapter 13
Back to TOC

13. Possible Error Messages and their Causes

Back to TOC
Back to chapter 12

13. Possible Error Messages and their Causes

13.1 Error Message "unknown type \<xxxxxxxx>"

This error states that there are no conversion instructions for this parameter is present to convert the raw data in a corresponding unit (time, temperature, percent, pressure, etc.).

To solve this problem, the respective telegram / command ID of the relevant parameter and the associated value should be read out and reported. Should there be multiple setting options for one parameters available, each option must also be read out, so that a clear assignment can take place.

13.2 Error Message "error 7 (parameter not supported)"

The associated Command ID is not recognized or the corresponding parameter is not supported by the controller (e.g. specific parameters related to a gas fired heater are not available at an oil fired heater).

Error messages of this type are hidden by default since v0.41 (but will still be queried within a complete query for example). If you still want them to be displayed, you have to comment out the definement `#define HIDE_UNKNOWN` in the file *BSB_lan_config.h* (so that it looks like `//#define HIDE_UNKNOWN`).

To check whether the Command ID in principle is supported by the controller but not yet released for your specific device family, please execute the URL command /Q (also see chapter 8.2.5). If any 'error 7'-messages appear with this query, please report them with the complete output of /Q.

13.3 Error Message "query failed"

This message appears when no response from the controller comes upon the request of the adapter.

Possible causes are mostly to be found on the hardware side (e. g. faulty RX and/or TX connection, wrongly installed components or even a timeout due to a switched off or not connected controller).

13.4 Error Message "ERROR: set failed! - parameter is readonly"

This message appears, when you are trying to adjust settings or when you are trying to send (e. g.) values like room temperature via BSB-LAN but didn't change the preset read-only state of BSB-LAN.

To change this setting within BSB-LAN, you have two different options:

1. You can allow BSB-LAN to be able to change all possible parameters which could be set. It's the most comfortable setting, because in this case you don't have to think about which parameters exactly you maybe want to be able to change one day. Besides that, it's easy to achieve: just open the file *BSB_lan_config.h* and change the flag in the definement `#define DEFAULT_FLAG FL_RONLY` to `#define DEFAULT_FLAG 0` (located at the bottom of the page), flash the Ardunio again and you are able to change settings of the controller via BSB-LAN.
2. You can leave the mentioned flag preset as `FL_RONLY` and make only the specific desired parameters writeable. Therefore you have to make specific changes in the file *BSB_lan_defs.h* for each parameter which you want to be able to change the setting of. Please read chapter 5 for further informations about the procedure.

13.5 Error Message "decoding error"

The error message "decoding error" means, that the parameter and the command id are known or match, but that the data packet doesn't correspond to the known decoding. The reason for this could be a different length or a different unit.

To update this for the specific type of controller / heating system, the belonging data packet, the exact value and the specific unit is needed. Please see chap. 10 for further instructions.

Further on to chapter 14
Back to TOC

14. Problems and their Possible Causes

Back to TOC
Back to chapter 13

14. Problems and their Possible Causes

14.1 The Red LED of the Adapter Isn't Lit

- Controller is switched off
- Adapter isn't connected with the controller via BSB or LPB
- Adapter is incorrectly connected to the controller (CL+/CL- or DB/MB interchanged)
- Probably hardware fault of the adapter (defect component, eroor in the construction
- Probably loose contact at the bus connector (Rx/Tx or CL+/CL-)

14.2 The Red LED Is Lit, but a Query Isn't Possible

- Probably adapter is connected wrong (usage of G+ instead of CL+)
- Probably loose contact at the bus connector (Rx/Tx or CL+/CL-)
- Probably wrong pin setting (Rx/Tx)
- Probably cold solder joints
- See subchapter „No Query of Parameters Possible"

14.3 Access to the Webinterface Isn't Possible

- Adapter doesn't have any/sufficient/unreliable power supply (→ recommended power supply via external device, 9V has been tested reliable; power supply via USB could lead to problems)
- Adapter or LAN shield isn't connected to the LAN
- IP and/or MAC address of the adapter isn't correct
- Security functions `passkey`, `TRUSTED_IP` and/or `USER_PASS_B64` activated/deactivated → URL not adjusted, access from wrong IP etc.
- Check router and/or firewall settings
- Access after power failure and/or restart of the Arduino isn't possible → press reset button at the Arduino / LAN shield
- Usage of a microSD card for logging → format as FAT32, execute URL command `/D0`, maybe try a different card and/or smaller capacity → see chapter 9.1
- (Adapter,) LAN shield and/or Arduino is faulty (→ sometimes diffuse problems occured within the usage of cheap clones, maybe try other/original units)

14.4 No Query of Parameters Possible

- See subchapter „The Red LED of the Adapter Isn't Lit"
- See subchapter „The Red LED Is Lit, but a Query Isn't Possible"
- See subchapter „Access to the Webinterface Isn't Possible"

- Rx and/or Tx assignment isn't correct, pinout and/or connection of the adapter doesn't fit to the settings in *BSB_lan_config.h*
- Wrong bus type (BSB/LPB)

14.5 Controller Isn't Recognized Correctly

- Controller is switched off
- Controller was switched on after the Arduino (automatic detection of the controller doesn't work in that case) → restart the Arduino
- Controller is not or not in the right way connected with the adapter
- Device family and variant of the controller isn't known yet → check `http://<IP-Adresse>/6225/6226` and report the output

14.6 Heating Circuit 1 Can't Be Controlled

- Adapter probably defined as room unit 2

14.7 Room Temperature Can't Be Transmitted to Heating Circuit 1

- Adapter probably defined as room unit 2
- Possible access of the adapter is readonly (`#define DEFAULT_FLAG FL_RONLY` in *BSB_lan_config.h*)

14.8 Heating Circuit 2 Can't Be Controlled

- Adapter probably defined as room unit 1

14.9 Room Temperature Can't Be Transmitted to Heating Circuit 2

- Adapter probably defined as room unit 1
- Possible access of the adapter is readonly (`#define DEFAULT_FLAG FL_RONLY` in *BSB_lan_config.h*)

14.10 Settings of the Controller Can't Be Changed via Adapter

- Possible access of the adapter is readonly (`#define DEFAULT_FLAG FL_RONLY` in *BSB_lan_config.h*)

14.11 Sometimes the Adapter Doesn't React to Queries or SET-Commands

- The Arduino doesn't have multitasking capability - wait until a query is done (e.g. especially extensive queries of many parameters, whole categories or a big logfile may take quite a long time)

14.12 'Nothing' Happens at the Query of the Logfile

- No microSD card is inserted in the slot
- Logging to microSD card was or is deactivated
- The logfile can get quite big, a query may take quite a long time
- The graphical display (`http://<IP-Adresse>/DG`) of the logfile can't occur because of JavaScript-blockers within the browser

14.13 No 24-Hour Averages Are Displayed

- The specific definement isn't activated
- No parameters for the calculation of the 24h-averages are set

14.14 'Nothing' Happens at the Query of DS18B20/DHT22 Sensors

- There are no sensors connected
- The specific definements aren't activated
- The pinout isn't set correctly
- The sensors are faulty or defect

14.15 The DS18B20 Sensors Are Showing Wrong Values

- Check power supply and whole installation (check size of the pullup-resistor, use capacitors, check wiring, use correct topology etc.)

14.16 The 'Serial Monitor' of the Arduino IDE Doesn't Provide Data

- Adapter isn't (additionally) connected via USB to your computer
- Wrong COM port or type of Arduino board is chosen
- Wring baud rate is set → set to 115200 baud
- Adapter isn't connected to the controller and/or controller is switched off → see subchapters above

Further on to chapter 15
Back to TOC

15. FAQ

Back to TOC
Back to chapter 14

15. FAQ

15.1 Can I Use the Adapter & Software with a Raspbery Pi?

Yes and no.
The adapter itself can be used in conjuction with a Raspberry Pi, if you use certain parts. Please see the following chapters for further informations: chap. 12.9 and appendix a2.2.

The BSB-LAN software can NOT be used with a RPi, it is only usable with the described Arduino. Further informations are available in chap. 12.9.

15.2 Can I Connect One Adapter to Two Controllers at the Same Time?

No, this isn't possible. If you want to connect the hardware setup (arduino, ethernet-shield, adapter) to the BSB of the controllers, you have to use one hardware setup for each controller. If the contollers are already connected via LPB though, please see the following FAQ.

15.3 Can I Connect an Adapter via LPB And Query Different Controllers?

Yes, if the existing controllers are already connected with each other via LPB. This LPB setup of the controllers already has to work without any problems, so the setup has to be done correctly (e.g. device and segment addresses have to be set right).
For querying data of each controller, the specific address has to be set within BSB-LAN. See chapter 8.1 for further informations.

15.4 Is a Multifunctional Input of the Controller Directly Switchable via Adapter?

No!

The multifunctional inputs of the controllers (e.g. H1, H2, H3 etc.) are not connectable directly to the adapter/arduino!

These inputs must be switched potential free, so you have to use a relay in conjunction with the adapter/arduino. See chapter 12.4 for further informations.

15.5 Can an Additional Relayboard Be Connected And Controlled by the Ardunio?

Yes. See chapter 12.4 for further informations.

15.6 Can I Query the State of a Connected Relay?

Yes. See the specific URL command in chapter 8.1.

15.7 Can I Be Helpful to Add Yet Unknown Parameters?

Yes! Please see chapter 10 for further instructions.

15.8 Why Do Some Parameters Appear Doubly Within a Complete Query?

When you do a complete query of all parameters via URL command (`http://<ip-address>/0-10000`) it can happen, that some parameters or program numbers are displayed doubly in the output. This happens because the same command id can occur within different parameters. Anyway, this doesn't have any negative influence the functionality in any way.

15.9 Why Aren't Certain Parameters Displayed Sometimes?

First of all, not all of the parameters which are known by BSB-LAN are available within every type of controller. Certain parameters which are controller / heating system specific just aren't available. E.g.: specific paramaters of a gas-fired heating system aren't available within an oil-fired system.

Besides that, if the controller has been powered on after the arduino already started, the automatic detection of the connected controller doesn't work. In this case just restart the arduino, so that the connected controller can be recognized by BSB-LAN.

If still certain parameters don't appear which are available via the operational unit of the heating system, please do a query of /Q (see chapter 8.2.5).
If the desired parameters still don't appear there (reported as 'error 7' parameters), please follow the instructions in chapter 10.

15.10 Why Isn't Access to Connected Sensors Possible?

If you connected DHT22 and/or DS18B20 sensors correctly to the adpater/arduino but the corresponding link in the webinterface doesn't work, you probably didn't adjust the belonging parameter in the file *BSB_lan_config.h*.
See the description in the file *BSB_lan_config.h* and the chapter 12.3.

15.11 I'm Using a W5500 LAN-Shield, What Do I Have to Do?

Make sure you are using the latest ethernet library within the Arduino IDE (min. version 2.0).

15.12 Can States Or Values Be Sent As Push-Messages?

No, not by only using BSB-LAN. For this, you have to use additional software (e.g. FHEM) to query the desired parameters and process the data.

15.13 Can (e.g.) FHEM 'Listen' to Certain Broadcasts?

Well, FHEM can 'listen' - but BSB-LAN can't send any messages by itself.

15.14 Why Sometimes Timeout Problems Occur Within FHEM?

This could be due to the length of the send / receive process. You should calculate the timeout value in FHEM in the way, that you estimate approx. two seconds for a query and one second for a setting of a parameter.

Besides that it could be helpful to put as many BSB-LAN specific readings as possible in one group for a query, so that collisions could be avoided.

15.15 Is There a Module For FHEM?

Yes and no. The member *„justme1968"* develops a module: https://forum.fhem.de/index.php/topic,84381.0.html
But: the development isn't done yet, so that an unproblematic and reliable usage can't be guaranteed at this time.

15.16 Why Aren't Any Values Displayed At Burnerstage 2 Within /K49?

If you are using a gas-fired heating system, it's most likely modulating and doesn't even have a two-staged burning system. These systems mostly are only available within oil-fired heating systems. But even within these systems not all controllers spread the differentiation between stage one and two over the bus by broadcast messages, so that BSB-LAN doesn't recognize either it's stage one or two which is active.

15.17 It Appears to Me That the Displayed Burner-Values of /K49 Aren't Correct.

This could be. The specific starts and runtimes are determined by the detection of the belonging broadcasts sent by the controller. Sometimes it can occur that a broadcast doesn't reach the other bus members, e.g. when a query is done at the same time.

15.18 What Is the Exact Difference Between /M1 and /V1?

Please see the descriptions of the monitor mode (/M) and the verbose mode (/V) in chapter 8.1.

15.19 Can I Implement My Own Code In BSB-LAN?

Yes, for this you can and should use the designated file *BSB_lan_custom.h*. Here you can add own code which will be initiated at each loop function.

15.20 Can I Integrate MAX! Thermostats?

Yes, that's possible. You have to activate and adjust the specific definement in the file *BSB_lan_config.h*.
For further informations see the corresponding chapter 12.5.

15.21 Why Isn't the Adapter Reachable After a Power Failure?

This behaviour was noticed sometimes with cheap clones of the lan shields. Just press the reset button of the arduino, after that everything should work fine again. If this happens more often at your home, you can probably add a little emergency power supply to prevent this.

15.22 Why Isn't the Adapter Reachable Sometimes (Without a Power Failure)?

This problem only occured in rare cases, there is no clear solution for this behaviour. The only solution was a reset and reboot of the arduino.

15.23 Why Do 'Query Failed' Messages Occur Sometimes?

If this occurs within a system which is usually working fine, then it could probably be due to hardware issues. Some cheap arduino clones produce unspecific problems. Therefore you should try another arduino clone or buy an original arduino.

15.24 I Don't Find a LPB or BSB Connector, Only L-BUS And R-BUS?!

In this case you have a controller which isn't compatible with BSB-LAN - please DON'T try to connect the adapter!
You can see chapter 3.3 for further informations.

15.25 Is There An Alternative Besides Using LAN?

Yes, please see chapter 12.7.

15.26 I Am Using The Outdated Setup Adapter v2 + Arduino Mega 2560 - Is There Anything I Have To Take Care Of?

Yes! Please see appendix D.

15.27 I Am Getting Error Messages from the Arduino IDE - What Can I Do?

Error messages from the Arduino IDE can be various and have different reasons, so we can't go into this in detail here. In this case you should check all settings regarding port, board type etc. If Google does not provide any further information, you can also ask in the forum.
As an example, here is a type of error message that occurs when the wrong board type is set when using an Arduino DUE:

```
BSB_lan:802:27: error: 'pgm_read_byte_far' was not declared in this scope

 uint8_t second_char = pgm_read_byte_far(enum_addr + page + 1);

                       ^~~~~~~~~~~~~~~~
```

15.28 I Have Further Questions, Who Can I Contact?

The best option is to create an account at the german FHEM forum (https://forum.fhem.de/) and ask your questions in the specific BSB-LAN thread: https://forum.fhem.de/index.php/topic,29762.0.html.

Further on to chapter 16
Back to TOC

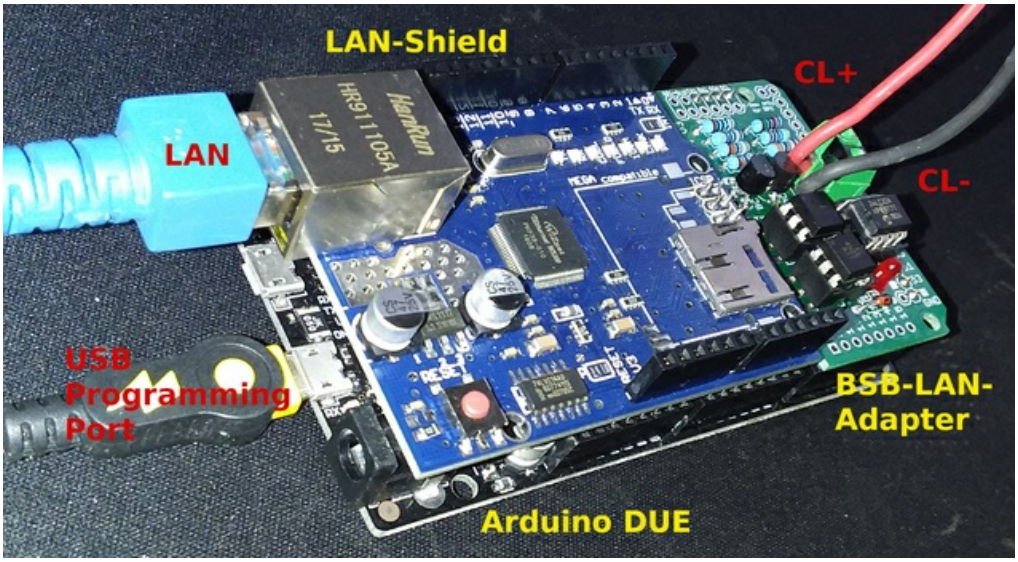
16. Quick Installation Guide

Back to TOC
Back to chapter 15

16. Quick Installation Guide

The following brief instructions do not replace the reading of the detailed manual! Please also read the respective more detailed explanations in the corresponding chapters.

1. Plug the LAN shield and the adapter onto the Arduino Due and connect the Arduino setup to your computer with a USB cable. Make sure you are using the 'Programming Port' of the Due, which is the USB port in the middle, right next to the power supply.



The complete setup (Arduino Due + LAN shield + BSB-LPB-LAN adapter v3), belonging cables included.

2. Download and install the latest version of the ArduinoIDE.
3. Download the current version of BSB-LAN.
4. Unzip the downloaded file "bsb_lan-master.zip" and enter it.
5. Enter the folder "BSB_LAN". There, rename the file "BSB_LAN_config.h.default" to "BSB_LAN_config.h".
6. Start the ArduinoIDE by double-clicking the file "BSB_LAN.ino" in the BSB_LAN folder. The ArduinoIDE should recognize the connected Arduino Due automatically together with the used COM port.
   For steps 2-6, see the more detailed description in chapter 4!
7. *Important:* Adjust the settings in the file "BSB_LAN_config.h" according to your wishes and circumstances.
   Note the chapter 5.2!
   When all settings have been adjusted, flash the Arduino with the BSB-LAN software.
   Note: You can also use the webinterface later for configuration ("settings" page).
8. After completing the flash process, remove the USB cable to de-energize the Arduino. Plug in the LAN cable and have the power adapter the Arduino ready.
9. Switch off your heating system so that the controller is no longer power supplied. Now connect the adapter of the Arduino setup to the controller. To do this, connect the controller-side connections "CL +" and "CL-" (for BSB use) or "DB" and "MB" (for LPB use) to the identically named connections of the adapter. Pay attention to the correct connection: The connected connections must be *the same*, e.g. "CL +" to "CL +" and "CL-" to "CL-"!
   Also note the detailed description in chapter 2.3!
10. Switch on the heating system / the controller.
11. Make the power supply to the Arduino setup, ideally with a specific power supply connected to the female connector socket. If you do not (yet) have a suitable power supply at hand, you can also power the Arduino setup via the USB port.

12. Start an internet browser and go to the page of the BSB-LAN web interface. It can be found at the IP address you previously set in step 6 (the default is "192.168.178.88"). When using DHCP, the IP can be read out from the start sequence of the Arduino Due by using the serial monitor of the Arduino IDE .

If everything is installed correctly, you will now have access to the controller of your heating system. If -contrary to expectations- errors or problems arise, then in addition to the chapters already mentioned, also read chapters 13, 14 and 15.

Now please execute "check for new parameters" (URL command /Q) and send us the output of the webinterface together with the name of the manufacturer and the name of your specific heating system.

Have fun with BSB-LAN wish you Frederik and Ulf! :)

Further on to chapter 17
Back to TOC

17. Further Informations and Sources

Back to TOC
Back to chapter 16

17. Further Informations and Sources

A lively exchange regarding the here presented hardware and software takes place in the following forum:
https://forum.fhem.de/index.php/topic,29762.0.html
This is also a good place for questions, exchange of experience and support, including informations about updates.

All documentation for the hardware and software presented here as well as the different software versions can be found here.
This manual is also available here as a PDF version.

The software and related documentation for the use of the here presented adapter in conjunction with a Raspberry Pi 2 is available here.

For the use of the adapter with an RPi at the PPS interface the Python script "PPS-monitor" can be used.

The initial idea of connecting some kind of adapter for gaining access via BSB/LPB and the process that led to this solution can be found here and here.

As a relatively extensive source with many parameter descriptions the "Brötje System Manual ISR Plus" is recommended to read. Besides numerous other and model-specific instructions it's like a basic 'reference' for the parameter definitions.

For informations about the different controllers it's always worthy searching for "Albatros 2".

In-depth informations such as specifications and technical requirements of the bus types can be found in the respective documents of the manufacturer. Especially regarding the LPB, two documents from "Siemens Building Technologies - Landis & Staefa Division" are recommended:

- CE1N2030D Local Process Bus LPB system basics
- CE1N2032D Local Process Bus LPB Configuration Basics

Regarding the installation and usage of DHT22 and OneWire sensors such as the DS18B20 there are numerous sources of information. Especially the official DataSheets of the manufacturers should be read. Besides the already mentioned documents in the specific chapters, there are many free tutorials, example installations and scripts available throughout the internet.

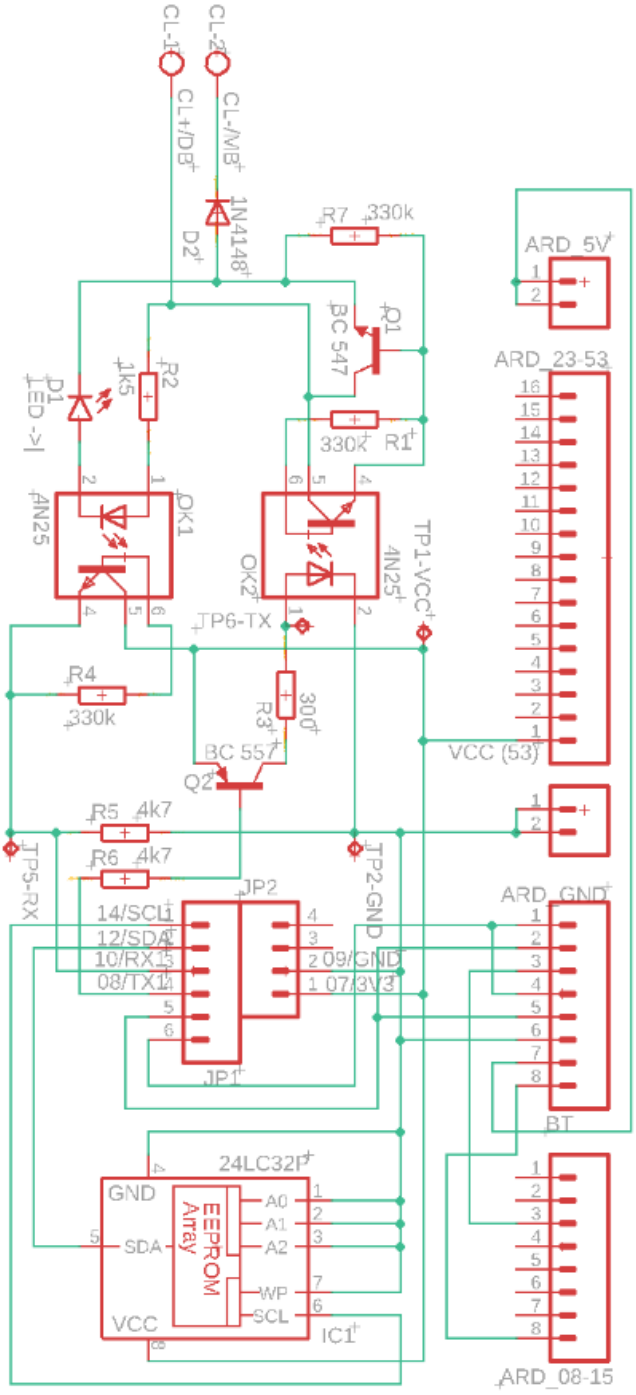
Further on to appendix A1
Back to TOC

Appendix A1: Circuit Diagram BSB-LPB-LAN Adapter v4 (Due version)

Back to TOC
Back to chapter 17

Appendix A1: Circuit Diagram BSB-LPB-LAN Adapter v4 (Due version)

Note: The circuit diagram for the ESP32 version is basically the same like the one for the Due, just the EEPROM isn't needed.



Further on to appendix A2
Back to TOC

Appendix A2: Notes on the Circuit Diagram

Back to TOC
Back to appendix A1

Appendix A2: Notes on the Circuit Diagram

Note: The circuit diagram for the ESP32 version is basically the same like the one for the Due, just the EEPROM isn't needed.

A2.1 Short Explanation of the Circuit Diagram

D1 = LED (red)
D2 = Diode
EEPROM = EEPROM
OK(x) = Optocouplers
Q(x) = Transistor
R(x) = Resistor

ARD = Arduino
RPI = Raspberry Pi

CL+/- = BSB connectors
DB/MB = LPB connectors

TXD = Digital pin: transmit
RXD = Digital pin: receive

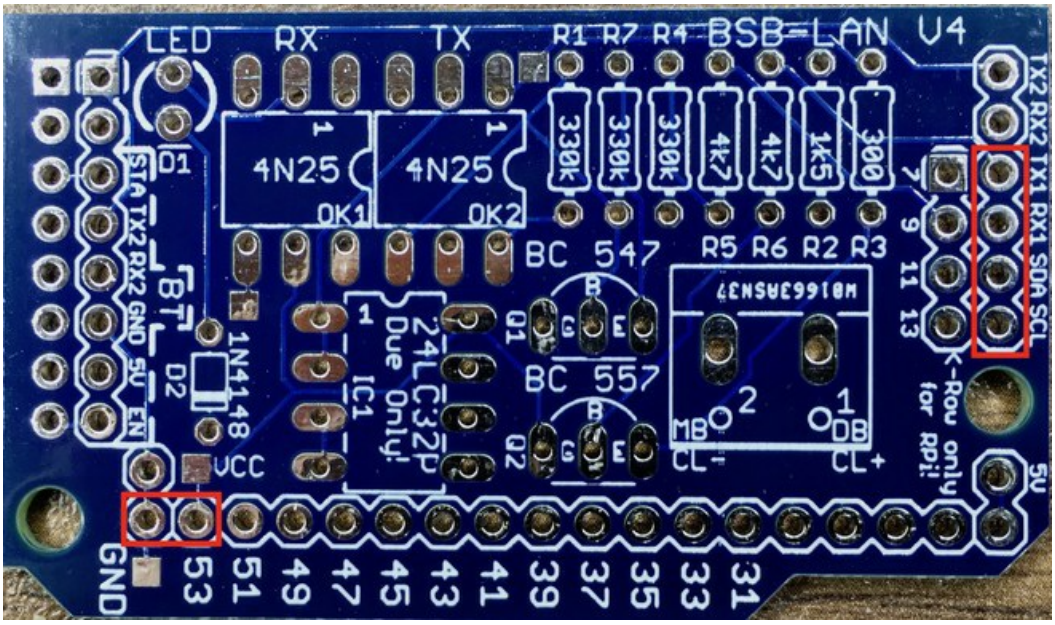
A2.2 Parts List

Amount	Component	Label	Illustration	Note
1	LED (red)	D1		operating voltage max. 2,8V, reverse voltage 5V
1	Diode 1N4148	D2		-
1	EEPROM 24LC32A-I/P	EEPROM		not needed for the ESP32 version of the PCB
2	Optocoupler 4N25	OK1, OK2		-

Amount	Component	Label	Illustration	Note
1	Transistor BC547	Q1		-
1	Transistor BC557	Q2		-
3	Resistor 330kΩ	R1, R4, R7		orange, orange, black, orange, brown
1	Resistor 1.5kΩ	R2		brown, green, black, brown, brown
1	Resistor 300Ω	R3		orange, orange, black, black, brown
2	Resistor 4.7kΩ	R5, R6		yellow, violet, black, brown, brown
1	Connector	CL+/CL		grid dimension 5,08mm

Arduino Due:
Pin header (male, grid dimension 2,54mm), optional IC sockets for optocouplers and/or EEPROM..

For the usage of the adapter v4 in conjunction with an *Arduino Due* you basically only need to assemble the pins for RX1, TX1, SDA, SCL, GND and pin 53. Other pins could be assembled due to a better stability and/or other usage.



Absolutely necessary pins for the usage in conjunction with an Arduino Due.

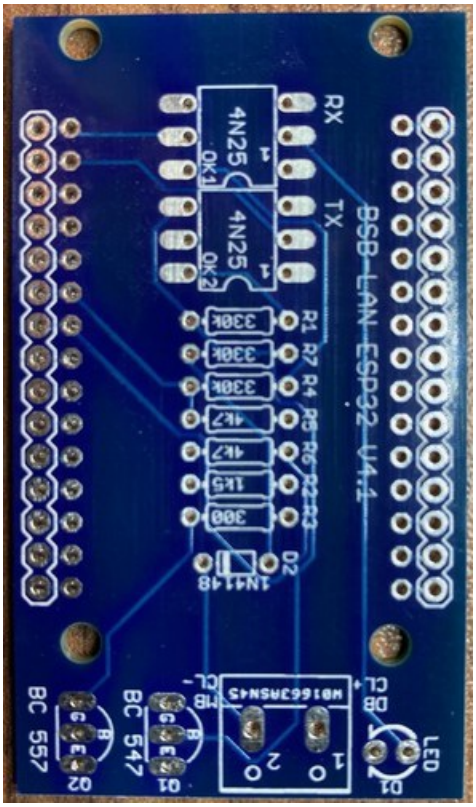
Raspberry Pi:
Female pinheader (double row, grid dimension 2,54mm), optional IC sockets for optocouplers and/or EEPROM..

For the usage of the adapter v4 in conjunction with a *Raspberry Pi* you have to put your attention on different things, which are collectively named within the chapter 12.9.

ESP32:

Female pinheader (grid dimension 2,54mm; ESP32: single row; Olimex: double row 2x5), optional IC sockets for optocouplers..

For the use of the ESP32 specific adapter v4 on the recommended *ESP32 NodeMCU from Joy-It* only the pins RX2, TX2, GND and 3.3V are needed and must be equipped with corresponding pin headers. However, for stability reasons it is recommended to equip both sides completely with one row of pin headers each.



The unpopulated ESP32 specific adapter board.

A2.3 General Notes

Important: Before soldering, study the circuit diagram attentively!

The following instructions do not replace any fundamental electronic knowledge, but maybe one or two of these hints could be helpful for electronics beginners.

Generally it is helpful to position the components, bend the 'legs' of them a little bit to hold them in position on the board and check again the positioning of each component. Start with the smallest parts first. Pay attention to the correct alignment of parts like diodes, transistors and ICs! If everything seems fine, you can turn around the board and start soldering. Be careful that you just solder where you should so that you don't produce shortcuts by accident.
A previous breadboard test setup (if you don't use the PCB) of course could be an option, though due to non-exclude problem sources (usage of a wrong plug row, possible loose contacts and so on) not necessarily recommended.

Please make sure that the components do not get too hot during soldering, since they can be damaged. Therefore it is appropriate to use corresponding IC sockets for the optocoupler ICs and the EEPROM. Once you are done with the soldering, you can just plug in the ICs. Pay attention to the correct alignment of the sockets and optocouplers/EEPROM as well as of the diodes and transistors!

Before putting the adapter into operation, it is advisable to thoroughly check the assembly again and (if possible) measure the circuit with a multimeter. Cold solder joints, accidentally bridged contacts and so on can cause inexplicable and difficult to diagnose misconduct of the adapater up to an eventual controller defect!

Good luck!

Further on to appendix B
Back to TOC

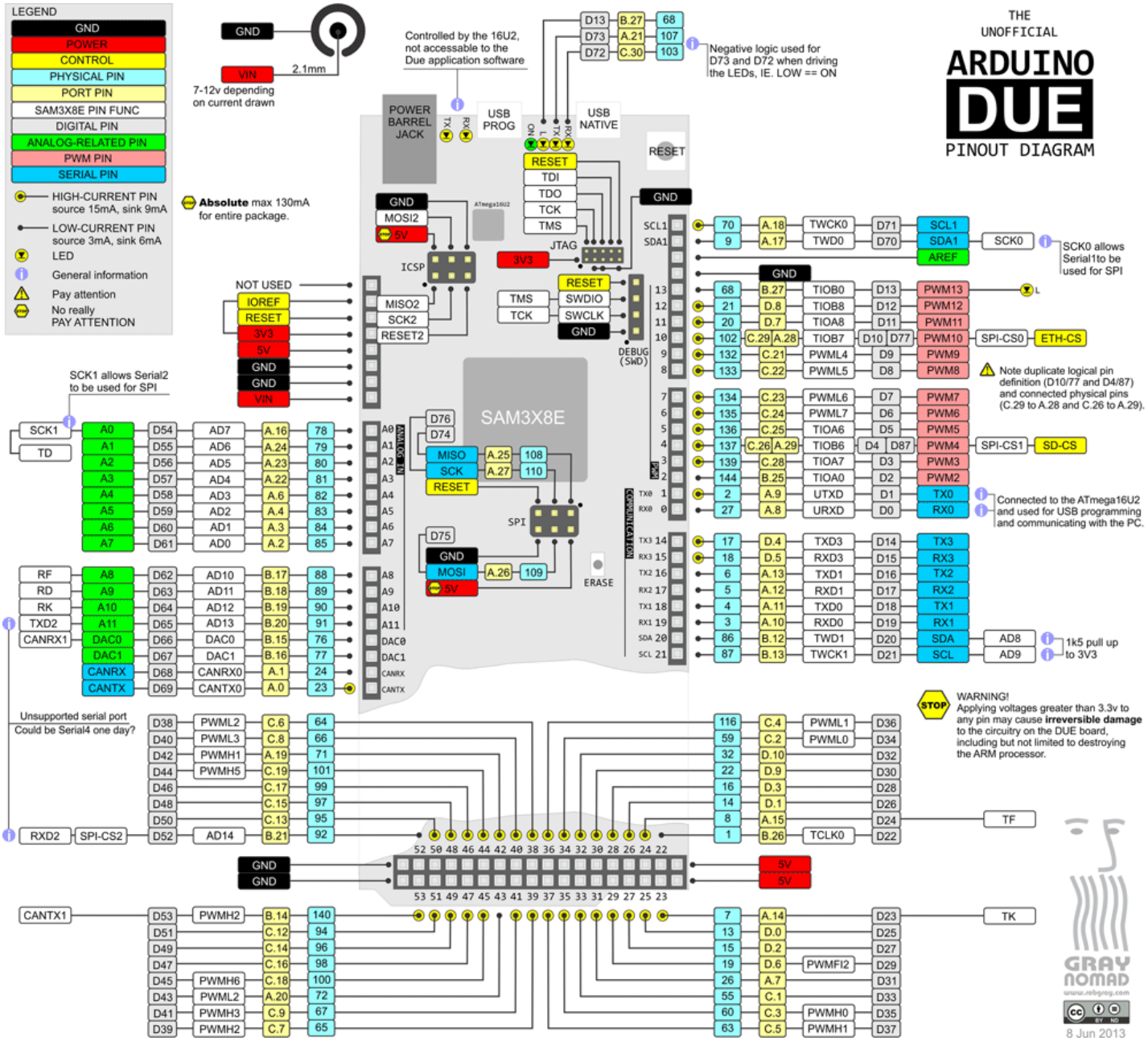
Appendix B: Arduino DUE Pinout

Back to TOC
Back to appendix A2

Appendix B: Arduino DUE Pinout

The following 'inofficial' Arduino DUE pinout diagram was made by Rob Gray.
It is also directly available as a PDF.
Many thanks for that great work!

Note: Please note that the pinout of some cheap clones may vary from the original pinout!



The 'official' pinout diagram from Arduino.cc is also available as a PDF.

Further on to appendix C
Back to TOC

Appendix C: Changelog BSB-LAN Software

Back to TOC
Back to appendix B

Appendix C: Changelog BSB-LAN Software

Version 2.0

- ATTENTION: LOTS of new functionalities, some of which break compatibility with previous versions, so be careful and read all the docs if you make the upgrade!
- Webinterface allows for configuration of most settings without the need to re-flash
- Added better WiFi option through Jiri Bilek's WiFiSpi library, using an ESP8266-based microcontroller like Wemos D1 mini or LoLin NodeMCU. Older WiFi-via-Serial approach no longer supported.
- Setting a temporary destination address for querying parameters by adding !x (where x is the destination id), e.g. /6224!10 to query the identification of the display unit
- URL command /T has been removed as all sensors can now be accessed via parameter numbers 20000 and above.
- New categories added, subsequent categories have been shifted up
- Lots of new parameters added
- URL command /JR allows for querying the standard (reset) value of a parameter in JSON format

Version 1.1

- ATTENTION: DHW Push ("Trinkwasser Push") parameter had to be moved from 1601 to 1603 because 1601 has a different "official" meaning on some heaters. Please check and change your configuration if necessary
- ATTENTION: New categories added, most category numbers (using /K) will be shifted up by a few numbers.
- /JA URL command outputs average values
- Many new parameters decoded
- New parameters for device families 25, 44, 51, 59, 68, 85, 88, 90, 96, 97, 108, 134, 162, 163, 170, 195, 209, 211
- Improved mobile display of webinterface
- Added definement "BtSerial" for diverting serial output to Serial2 where a Bluetooth adapter can be connected (5V->5V, GND->GND, RX->TX2, TX->RX2). Adapter has to be in slave mode and configured to 115200 bps, 8N1.
- Lots of added Polish translations
- New data types VT_BYTE10, VT_SPF
- Bugfix for PPS bus regarding display of heating time programs
- Bugfix for MQTT

Version 1.0

- /JI URL command outputs configuration in JSON structure
- /JC URL command gets list of possible values from user-defined list of functions. Example: /JC=505,700,701,702,711,1600,1602
- Logging telegrams (log parameter 30000) now writes to separate file (journal.txt). It can be reset with /D0 (same time with datalog.txt) command and dumped with /DJ command.
- removed WIFI configuration as it is no longer applicable for the Due
- lots of new parameters for various device families

- Code optimization and restructuring, general increase of speed
- new schemativs for board layout V3
- lots of bugfixes

Version 0.44

- Added webserver functionality via SD card and various other improvements from GitHub user dukess
- Added JSON output for MQTT
- mobile friendlier web interface
- more parameters and device families
- last version completely tested on Mega 2560. Future versions may still run on the Mega, but will only be tested on the Arduino Due.

Version 0.43

- Added support for HardwareSerial (Serial1) connection of the adapter. Use RX pin 19 in bus() definition to activate. See manual/forum for hardware details.
- Added definement DebugTelnet to divert serial output to telnet client (port 23, no password) in BSB_lan_config.h
- Added possibility to control BSB-LAN (almost?) completely via USB-serial port. Most commands supported like their URL-counterparts, i.e. //xxx to query parameter xxx or //N to restart Arduino.
- Changed default device ID from 6 (room controller "RGT1") to unused ID 66 ("LAN")
- Many new parameters, please run /Q to see any possible changes for your device family and report back to us!
- Added global variables (arrays of 20 bytes) custom_floats[] and custom_longs[] for use with BSB_lan_custom.h, for example to read sensors etc. Output of these variables is done via new URL command /U
- Added device families 23 and 29 (Grünenwald heaters)
- Added device families 49, 52, 59 (Weishaupt heaters)
- Added device fmilies 91, 92, 94, 118, 133, 136, 137, 165, 184, 188 (various controllers like QAA75 or AVS37)
- Added device family 171 (Bösch wood pellet system)
- Added device family 172 (SensoTherm BLW Split B (RVS21.826F/200))
- Added device families 186 and 164 (Olymp WHS-500)
- Added device family 195 variant 2 (Thision 19 Plus / LMS14.111B109)
- Including DHT, 1Wire and burner status parameters (>20000) to MQTT
- English is now default language
- Updated various translations
- Added STL files to print a case with a 3D printer (thanks to FHEM user EPo!)
- Moved all sensors to /T , /H is now no longer used
- New virtual parameters 702/703 for Weishaupt room controller
- New virtual parameter 10003 to set outside temperature on newer systems
- Added text descriptions for error phases (6706 ff.)
- /Q is now more comprehensive
- New data types VT_CUSTOM_ENUM and VT_CUSTOM_BYTE to extract information from non-standard telegrams (such as 702/703)
- Bugfix: DHCP (ethernet) implementation

Version 0.42

- Added localization! Now you can help translate BSB-LAN into your language! Simply copy one of the language files from the localization folder (LANG_DE.h is the most complete) and translate whatever you can. Non-translated items will be displayed in German. Attention: Language definition in BSB_lan_config.h is now #define LANG For example: #define LANG DE
- Added export to MQTT broker, use log_parameters[] in BSB_lan_config.h to define parameters and activate MQTTBrokerIP definement.
- Added support for WiFi modules such as an ESP8266 or a Wemos Mega connected to Serial3 (RX:15/TX:14) of the Arduino. The ESP8266 has to be flashed with the AT firmware from Espressif to work. Please take note that WiFi over serial is by design much slower (only 115kpbs) than "pure" TCP/IP connections.
- Added new category "34 - Konfiguration / Erweiterungsmodule". All subsequent categories move one number up!
- Lots of new parameters coming from device family 123, please run /Q to see if some parameters also work for your heater!
- Lots of new yet unknown parameters through brute force querying :) (parameter numbers 10200 and above)
- Added further PPS-Bus commands, moved parameter numbers to 15000 and above
- Default PPS mode now "listening". Use third parameter of bus definition to switch between listening and controlling, 1 stands for controlling, everything else for listening, i.e. BSB bus(68,67,1) sends data to the heater, BSB bus(68,67) only receives data from heater / room controller. You can switch between modes at run-time with URL command /P2,x where x is either 1 (for controlling) or not 1 (for listening only)
- Fixed bug that crashed PPS bus queries
- Stability improvements for PPS bus
- Improved graph legend when plotting several parameters
- Added JSON export; query with /JQ=a,b,c,d... or push queries to /JQ or push set commands to /JS
- Logging of MAX! parameters now possible with logging parameter 20007
- Added Waterstage WP device family (119)
- Added WHG Procon device family (195)
- Added unit to log file as well as average output
- Rewrote device matching in cmd_tbl to accomodate also device variant (Gerätevariante). Run /Q to see if transition has worked for your device!
- Added BSB_lan_custom_setup.h and BSB_lan_custom_global.h for you to add individual code (best used in conjunction with BSB_lan_custom.h)
- Marked all (known) OEM parameters with flag FL_OEM. OEM parameters are set by default as read-only. To make them writeable, change FL_OEM from 5 to 4 in BSB_lan_defs.h
- Increased performance for querying several parameters at once (similar to category query)
- Added config option to define subnet.
- /Q no longer needs #define DEBUG
- Bugfix ENUM memory adressing
- Bugfix in reset function (/N), clear EEPROM during reset with /NE
- Added favicon.ico
- Split of cmdtbl into cmdtbl1 and cmdtbl2 due to Arduino's(?) limit of 32kB size of struct, opening up more space for new parameters.

Version 0.41

- Interim release containing all changes from 0.42 above, except locaization, i.e. all text fragments are still part of the main code.

Version 0.40 -- 21.01.2018

- Implemented polling of MAX! heating thermostats, display with URL command /X. See BSB_lan_custom.h for an example to transmit average room temperature to heating system.
- Added new category \"22 - Energiezähler\" - please note that all subsequent categories move one up!
- New virtual parameter 1601 (manual TWW push)
- Added Fujitsu Waterstage WSYP100DG6 device family (211)
- Added CTC device family (103)
- New definement \"#define TRUSTED_IP2\" to grant access to a second local IP address
- Added optional definement \"#define GatewayIP\" in BSB_lan_config.h to enable setting router address different from x.x.x.1
- Removed parameter 10109 because it is the same as 10000
- Added function to check all known CommandIDs on your own heating system. Use /Q after enabling definement \"#define DEBUG\" in BSB_lan_config.h
- Added parameter numbers to category menu
- Updated analyze.sh
- hopefully fixing the memory issue
- Moved HTML strings to html_strings.h

Version 0.39 -- 02.01.2018

- Implementation of PPS-Bus protocol. See /K40 for the limited commands available for this bus. Use setBusType(2) to set to PPS upon boot or /P2 to switch temporarily.
- Set GPIOs to input by using /Gxx,I
- Definement \"#define CUSTOM_COMMANDS\" added. Use this in your configuration to include individual code from \"BSB_lan_custom.h\" (needs to be created by you!) which is executed at the end of each main loop. Variables \"custom_timer\" and \"custom_timer_compare\" have been added to execute code at arbitrary intervals.
- Added LogoBloc Unit L-UB 25C device family (95)
- several new parameters added
- Bugfix for logging Brennerlaufzeit Stufe 2

Version 0.38 -- 22.11.2017 ATTENTION: New BSB_lan_config.h configurations! You need to adjust your configuration when upgrading to this version!

- Webserver port is now defined in #define Port xx
- IP address is now defined in #define IPAddr 88,88,88,88 form - note the commas instead of dots!
- Special log parameters 20002 to 20006 have changed, see BSB_lan_config.h for their new meaning
- Added new virtual parameter 701 (Präsenztaste) which enters reduced temperature mode until next timed switch
- Added Brötje BOB device family (138), including many new parameters!
- Added Brötje SOB26 device family (28)
- Added Elco Aquatop 8es device family (85)
- Added Elco Thision 13 Plus device family (203)
- Added Weishaupt WTU 25-G familiy (50)
- Added output for absolute humidity (g/m3) for DHT22 sensors

- New schematics for Arduino/Raspberry board layout
- Included support for W5500 Ethernet2 shields. Activate definement ETHERNET_W5500 in BSB_lan_config.h
- Including two-stage oil furnaces BC-counters and logging - please note that logging parameters have been adjusted, see BSB_lan_config.h for new values!
- Added new options for commands /P and /S to allow specifying a different destination device during runtime
- Added new configuration definement CUSTOM_COMMANDS which includes BSB_lan_custom.h at the end of each main loop. You may use custom_timer (set to current millis()) and custom_timer_compare to execute only every x milliseconds.
- Bugfixing SD-card logging in monitor mode
- Bugfix for setting hour:time parameters via webinterface

Version 0.37 -- 08.09.2017

- LPB implementation! More than 450 parameters supported! Switch temporarily between LPB and BSB with the Px command (0=BSB, 1=LPB) or use the setBusType config option to set bus-type at boot-time. Parameter numbers are the same as for BSB.

Version 0.36 -- 23.08.2017

- bugfix: brought back VT_BIT list of options which were erroneously deleted :(, fixed/freed several memory issues

Version 0.35 -- 25.06.2017

- new category \"Sitherm Pro\"; caution: category numbers all move up by one, starting from category \"Wärmepumpe\" (from 20 to 21) onwards.
- graph display of logging data now comes with crosshair and shows detailed values as tooltip
- improved SD-card output by factor 3 (from 16 to 45 kbps), switching SD-card library from from SD.h to SdFat.h (https://github.com/greiman/SdFat) brings another 10% performance boost
- adjusted paths and directory layout of SdFat to enable compiling from sketch directory.
- new data type vt_sint for signed int data, currently only used in some Sitherm Pro parameters

Version 0.34 -- 29.05.2017

- Log data can now be displayed as graph
- Webinterface can now display and set vt_bit type parameters in human-readable form
- added KonfigRGx descriptions; caution: various sources used, no guarantee that descriptions match your individual heating system!
- vt_bit is generally read-only in the webinterface. To set, use URL command /S with decimal representation of value
- fixed a bug with vt_seconds_short5, affecting parameters 9500 and 9540.
- fixed bug regarding Fujitsu\'s device family (from 127 to 170)
- moved libraries from folder libraries to src so they can be included without copying them to the Arduino libraries folder
- modified DallasTemperature.h\'s include path for OneWire.h

Version 0.33 -- 09.05.2017

- no more heating system definements anymore due to new autodetect function based on device family (parameter 6225), or set device_id variable to parameter value directly
- two more security options: TRUSTED_IP to limit access to one IP address only, and HTTP authentication with username and password
- Average values are saved on SD-card if present and LOGGER definement is activated
- deactivate logging by setting /L0=0 - this way you can enable LOGGER definement without filling up SD card but still save average values
- new error codes for THISION

- added dump of data payload on website for commands of unknown type, greyed out unsopported parameters
- enable logging of telegrams (log parameter 30000) also in monitor mode (bsb.cpp and bsb.h updated)
- time from heating system is now retreived periodically from broadcast telegrams, further reducing bus activity
- new data type vt_bit for parameters that set individual bits. Display as binary digits, setting still using decimal representation
- new data type vt_temp_short5_us for unsigned one byte temperatures divided by 2 (so far only 887 Vorlaufsoll NormAussentemp)
- new data type vt_percent5 for unsigned one byte temperatures divided by 2 (so far only 885 Pumpe-PWM Minimum)
- new data type vt_seconds_word5 for two byte seconds divided by 2 (so far only 2232, 9500 and 9540)
- new data type vt_seconds_short4 for (signed?) one byte seconds divided by 4 (so far only 2235)
- new data type vt_seconds_short5 for (signed?) one byte seconds divided by 5 (so far only 9500, 9540)
- new data type vt_speed2 for two byte rpm (so far only 7050)
- cleaned up set() function from apparent duplicate cases
- added cases for vt_temp_word, vt_seconds_word5, vt_temp_short, vt_temp_short5, vt_seconds_short4 to set() function

Version 0.32 -- 18.04.2017

- lots of new parameters suppoerted
- newly designed webinterface allows control over heating system without any additional software or cryptic URL commands. URL commands of course are still available, so no need to change anything when using FHEM etc.
- German webinterface available with definement LANG_DE
- new URL-command /LB=x to log only broadcast messages (x=1) or all bus messages (x=0)
- new URL-command /X to reset the Arduino (need to enable RESET definement in BSB_lan_config.h)
- new logging parameters 20002 and 20003 for hot water loading times and cycles
- moved DS18B20 logging parameters from 20010-20019 to 20200-20299 and DHT22 logging parameters from 20020-20029 to 20100 to 20199
- moved average logging parameter from 20002 to 20004
- set numerous parameters to read-only because that\'s what they obviously are (K33-36)
- various bugfixes

Version 0.31 -- 10.04.2017

- increased dumping of logfile by factor 5 / as long as we still have memory left, you can increase logbuflen from 100 to 1000 to increase transfer speed from approx. 16 to 18 kB/s
- adjusted burner activity monitoring based on broadcast messages for Brötje systems
- removed definement PROGNR_5895 because so far, it has only disabled an ENUM definition.
- removed definement PROGNR_6030 because double command ID could be resolved via BROETJE / non-BROETJE definements
- renamed BROETJE_SOB to BROETJE in order to allow for fine-grained distinction between different BROETJE cases (e.g. 6800ff). This means you have to activate TWO definements when using a Brötje system now: The general BROETJE as well as BROETJE_SOB or BROETJE_BSW. Have a look at your serial log for parameters 6800 to see which command IDs fit your system and activate one of both accordingly.
- changed 16-Bit addressing of flash memory to 32-Bit to address crashes due to ever growing PROGMEM tables - now we have lots of air to breathe again for new command IDs :)
- removed trailing \0 string from several ENUMs that led to wrong ENUM listings. Please keep in mind not to end ENUMs with a trailing \0 !

Version 0.30 -- 22.03.2017

- Time library by Paul Stoffregen (https://github.com/PaulStoffregen/Time) is now required and included in the library folder.
- adds logging of raw telegram data to SD card with logging parameter
- Logging telegram data is affected by commands /V and /LU
- adds command /LU=x to log only known (x=0) or unknown (x=1) command IDs when logging telegram data
- removed define USE_BROADCAST, broadcast data is now always processed
- new internal functions GetDateTime, TranslateAddr, TranslateType

Version 0.29 -- 07.03.2017

- adds command /C to display current configuration
- adds command /L to configure logging interval and parameters
- adds option for command /A to set 24h average parameters during runtime
- adds special parameter 20002 for logging /A command (24h averages, only makes sense for long logging intervals)
- bugfixes for logging DS18B20 sensors

Version 0.28 -- 05.03.2017

- adds special parameters 20000++ for SD card logging of /B, /T and /H commands (see BSB_lan_config.h for examples)
- adds version info to BSB_LAN web interface

Version 0.27 -- 01.03.2017

- adds date field to log file (requires exact time to be sent by heating system)
- /D0 recreates datalog.txt file with table header
- added \"flags\" field to command table structure. Currently, only FL_RONLY is supported to make a parameter read-only
- added DEFAULT_FLAG in config. Defaults to NULL, i.e. all fields are read/writeable. Setting it to FL_RONLY makes all parameters read-only, e.g. for added level of security. Individual parameters can be set to NULL/FL_RONLY to make only these parameters writable/read-only.

Version 0.26 -- 27.02.2017

- added functionality for logging on micro SD card, using the slot of the w5100 Ethernet shield
- more parameters added (e.g. 8009)

Version 0.25 -- 21.02.2017

- more FUJITSU parameters added

Version 0.24 -- 14.02.2017

- updated README with added functions
- added German translations of FAQ and README, courtesy of Ulf Dieckmann

Version 0.23 -- 12.02.2017

- minor bugfix

Version 0.22 -- 07.02.2017

- more FUJITSU parameters
- (hopefully) correct implementation of VT_VOLTAGE readings

- minor bugfixes

Version 0.21 -- 06.02.2017

- added numerous parameters for Fujitsu Wärmepumpe, including new #define FUJITSU directive to activate these parameters due to different parameter numbers
- minor bugfixes

Version 0.20 -- 27.01.2017

- added more parameters for Feststoffkessel
- minor bugfixes

Version 0.19 -- 01.01.2017

- added humidity command \"H\", currently for DHT22 sensors
- added 24h average command \"A\", define parameters in BSB_lan_config.h
- removed trailing whitespace from menu strings
- fixed command id 0x053D04A2 for THISION heaters
- included Rob Tillaart\'s DHT library because there are various libraries implementing the protocol and this one is used in the code for its ability to address multiple sensors with one object.
- removed /temp URL parameter as it is a duplicate of /T
- included loop to display DHT22 sensors in IPWE
- making compiling IPWE extensions optional (#define IPWE)

Version 0.18 -- 22.12.2016

- split off configuration into bsb_lan_config.h
- split off command definitions into bsb_lan_defs.h
- changed GPIO return values from LOW/HIGH to 1/0
- reactivated and updated IPWE (define parameters in config)
- check for protected pins when accessing GPIO (define in config)
- added schematics and PCB files to new subfolder \"schematics\"

Version 0.17a -- 20.12.2016

- minor errors corrected

Version 0.17 -- 20.12.2016

- merged v0.16 with FHEM user miwi\'s changes

Version 0.16 -- 20.11.2016

- removed IPWE and EthRly interface
- added GPIO interface
- merged parameters from J.Weber
- resolved duplicate command IDs

Version 0.15a -- 25.07.2016

- collated the commands from a Python project and this project, merged the two versions, corrected obvious errors. Inserted hypothetical numerical values in ENUM definitions where Broetje manuals documented only the message texts.

- added information from traces in a Broetje installation with an ISR-SSR controller and a WOB 25C oil furnace.

Version 0.15 -- 21.04.2016

- added Solar and Pufferspeicher from Elco Logon B & Logon B MM

Version 0.14 -- 04.04.2016

- minor bugfixes for Broetje SOB
- extended broadcast handling (experimental)

Version 0.13 -- 31.03.2016

- change resistor value in receiving path from 4k7 to 1k5
- added values 0x0f and 0x10 to Enum8005
- fixed strings for Zeitprogramme
- added timeout for sending a message (1 second)
- added option T for querying one wire temperature sensors in mixed querys
- added special handling for Broetje SOB
- simplified settings

Version 0.12 -- 09.04.2015

- added ONEWIRE_SENSORS to ipwe
- fixed parameter decoding for ELCO Thision heating system

Version 0.11 -- 07.04.2015

- fixed parameter decoding for ELCO Thision heating system

Version 0.10 -- 15.03.2015

- added more parameters for ELCO Thision heating system

Version 0.9 -- 09.03.2015

- added more parameters for ELCO Thision heating system
- printTelegramm returns value string for further processing

Version 0.8 -- 05.03.2015

- added parameters for ELCO Thision heating system
- added IPWE extension
- minor bugfixes

Version 0.7 -- 06.02.2015

- added bus monitor functionality

Version 0.6 -- 02.02.2015

- renamed SoftwareSerial to BSBSoftwareSerial
- changed folder structure to enable simple build with arduino sdk

Version 0.5 -- 02.02.2015

- bugfixes

- added documentation (README)
- added passkey feature
- added R feature (query reset value)
- added E feature (list enum values)
- added setter for almost all value types
- fixed indentation
- added V feature to set verbosity for serial output
- set baudrate to 115200 for serial output
- redirecting favicon request
- added some images of the BSB adapter

Version 0.1 -- 21.01.2015 -- initial version

Further on to appendix D
Back to TOC

Appendix D: Notes For Users Of The Outdated Setup Adapter v2 + Mega 2560

Back to TOC
Back to appendix C

Appendix D: Notes For Users Of The Outdated Setup Adapter v2 + Mega 2560

For users of the outdated setup, the following refers to some questions and points that need clarification or that need to be considered. If further questions occur, please post your questions in the corresponding thread of the german FHEM forum.
Please understand, however, that we will not answer any questions that may arise, for example, about building an adapter v2 after the changeover to adapter v3.
PCBs v2 are no longer available, state of the art is the combination adapter v3/v4 + Due.

Note:
It is possible to use the adapter v2 with an ESP32 (after making some small changes). This way, one could use the current version of BSB-LAN without the need to use a new adapter. Please read chap. 12.2.3 for further informations.

- ***Do I have to switch to the new setup adapter v3/v4 + Due?***
  No, if you are satisfied with the outdated setup and the range of functions of BSB-LAN has met your requirements so far, you can of course continue to use the old setup.

- ***Is there anything to consider regarding the BSB-LAN version?***
  Yes. The last 'officially' tested and recommended version for your setup is version v0.44! Within the zip-file there you'll also find the last 'Mega2560-specific' version of the manual (PDF).

  But: It has been shown by several users that even the **v1.1** still runs without major restrictions, but due to the lack of memory of the Mega 2560 probably already no longer with all available options that BSB-LAN offers.

  Starting from **v2.x** it is then definitely necessary to deactivate some modules which BSB-LAN offers. Hints concerning this can be found in chap. 5.2 or in the comments of the file *BSB_lan_config.h*. Special attention should be paid to the last points, which offer a comfortable deactivation of individual modules (e.g. Webconfig, MQTT, IPWE etc.) at central place.

- ***What do I have to consider if I want to use the current v2.x?***
  As already mentioned you have to adjust the configuration in a way that it fits the smaller memory of the Mega 2560. Besides the already mentioned deactivation of certain modules you have further possibilities:

  1) You can reduce the size of some variables like `PASSKEY[]`, `avg_parameters[]`, `log_parameters[]`, `ipwe_parameters[]`, `max_device_list[]` for saving RAM (if you don't need as much parameters as maximum possible for example).

  2) Within the file *BSB_lan_config.h* you find a section at the end of the file where certain functions of BSB-LAN can be deactivated if you don't use them anyway. Notes regarding this you'll find in the file itself.

  3) Creating a controller specific *BSB_lan_defs.h*:
  There is a perlscript named *selected_defs.pl* and a Windows executable named *selected_defs.exe* in the repo that filters the file *BSB_lan_defs.h* for selected device families and creates a specific file for your own controller type. The saving is on average about 20 to 25 kB of flash memory, which can then be used for the (re-)activation of other functions. In case of a controller change (= other device family) the file must of course be regenerated accordingly.
  The script runs under Perl, which is installed by default on Mac and Linux computers, but can also be installed on Windows.

  Procedure for creating a controller-specific defs file:

  - Retrieve parameter 6225 "Device family" via BSB-LAN and note the value.
  - Before executing, copy the file *selected_defs.pl* or *selected_defs.exe* respectively in the same folder, where the file *BSB_lan_defs.h* is located.
  - Open a terminal, enter the corresponding folder and create the reduced file named *BSB_lan_defs_filtered.h* using the Perl script or the

Windows executable, which contains only the parameters relevant for the specific device family/families. If only one controller is connected, for example with device family 162, the command is
`./selected_defs.pl 162 > BSB_lan_defs_filtered.h` or
`selected_defs.exe 162 > BSB_lan_defs_filtered.h` respectively.
If you have e.g. two devices on the bus with the device families 162 and 90, you can extend the command by the second value:
`./selected_defs.pl 162 90 > BSB_lan_defs_filtered.h` or
`selected_defs.exe 162 90 > BSB_lan_defs_filtered.h` respectively.

- Move the original file *BSB_lan_defs.h* from the "BSB_lan" directory to a different location. Move the newly created file *BSB_lan_defs_filtered.h* to the directory "BSB_lan" (if you didn't already create the file within that directory).
- *Important: Now rename the newly created file to "BSB_lan_defs.h"!*

- ***Is there anything to consider regarding the settings of RX-/TX-pins?***
  Yes! If you still to test a newer version than v0.44 on the Mega, make sure that you use the corresponding file BSB_lan_config.h.default and adjust it accordingly:
  - With a version of BSB-LAN **before v2.x** it is absolutely necessary to adapt the line `BSB bus(19,18);`: The DUE uses (in contrast to the Mega) the HardwareSerial interface and other RX/TX pins than the Mega, which is already preset here. When used with the Mega, the line must therefore be changed to `BSB bus(68,69);`!
  - With a version of BSB-LAN **since v2.x** an automatic detection of the used pins is implemented. Therefore BSB-LAN autodetects if a Mega (= software serial) or a Due (= hardware serial) is used.
- ***Can I continue to use the Adapter v2 on a Due?***
  No! The reason for this is that neither the adapter v2 nor the Due has an EEPROM, which is necessary for BSB-LAN. So if you want to benefit from the new functions of BSB-LAN in the future, you have to get an adapter v3/v4 or make it yourself and use it with an Arduino Due.
- ***Can I 'convert' the adapter v2 to an adapter v3/v4?***
  No! The primary reason for this is (among other reasons) again the missing EEPROM of the Due.
- ***Can I continue to use the adapter v3/v4 with my previous Mega 2560?***
  No! Even if it would be possible after some changes to the adapter v3/v4, it would not offer any added value compared to the adapter v2. New functions of BSB-LAN would still not be able to be used due to the lack of memory of the Mega 2560. So if you want to use the new adapter v3/v4, then only in combination with an Arduino Due.
- ***Why is there an EEPROM on the v3/v4 board?***
  The Arduino Due has no EEPROM, but this is necessary for BSB-LAN.
- ***Can I continue to use the LAN-Shield if I change to the Due?***
  Yes, usually this is possible without any problems. A trouble-free usage of clones can't be guaranteed though.
- ***Can I continue to use my existing housing?***
  Yes, the Due has the same form factor as the Mega 2560, so the dimensions of the case should fit. However, you will probably have to adapt your case a bit and add a cutout or a large hole for the middle USB port of the Due ('Programming Port'), so that you can continue to connect the corresponding USB cable comfortably.

Back to TOC