



Bern University  
of Applied Sciences

Kernmodul Embedded Systems (EMBSY)

# **Embedded Webserver**

## Hardware und Software Manuel

für das Beaglebone Black Webhouse

Autor: Tim Wachter (wht4)

Dozent: Roger Weber (wbr1)

Datum: 27. September 2013

Version: 1.3

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Allgemeines . . . . .	1
1.2. Systemübersicht . . . . .	1
<b>2. Hardware Embedded Webserver</b>	<b>3</b>
2.1. Allgemeines . . . . .	3
2.2. Anschlüsse . . . . .	4
2.2.1. TV . . . . .	4
2.2.2. DLampe . . . . .	4
2.2.3. SLampe . . . . .	5
2.2.4. Heizung . . . . .	5
2.2.5. Alarm . . . . .	6
2.3. Belegung Extension Header Beaglebone Black . . . . .	6
<b>3. Software Embedded Webserver</b>	<b>7</b>
3.1. Allgemeines . . . . .	7
3.2. Ansteuerung Komponenten . . . . .	8
3.2.1. Initialisierung . . . . .	8
3.2.2. TV . . . . .	8
3.2.3. DLampe . . . . .	9
3.2.4. SLampe . . . . .	10
3.2.5. Heizung und Temperatur . . . . .	11
3.2.6. Alarm . . . . .	13
<b>Literaturverzeichnis</b>	<b>16</b>
<b>A. Anhang Hardware</b>	<b>A 1</b>
<b>B. Kopieren von Daten zum Webhouse</b>	<b>B 1</b>
B.1. Allgemeines . . . . .	B 1
B.2. Kopieren von Daten zum Webhouse mit FileZilla . . . . .	B 1
<b>C. Eclipse Remote System</b>	<b>C 1</b>
C.1. Allgemeines . . . . .	C 1
C.2. Erstellen einer Verbindung zum Webhouse . . . . .	C 1
<b>D. Debugging eines Projektes auf dem Webhouse</b>	<b>D 1</b>
D.1. Allgemeines . . . . .	D 1
D.2. Erstellen einer Debug-Konfiguration . . . . .	D 1

<b>E. JSON und Javascript</b>	<b>E 1</b>
E.1. Allgemeines . . . . .	E 1
E.2. JSON Syntax . . . . .	E 1
E.2.1. Allgemeines . . . . .	E 1
E.2.2. JSON Key-Value Paare . . . . .	E 1
E.2.3. JSON Values . . . . .	E 2
E.2.4. JSON Objekte . . . . .	E 2
E.3. JSON und JavaScript . . . . .	E 3
E.3.1. Allgemeines . . . . .	E 3
E.3.2. JSON-Objekt erstellen . . . . .	E 3
E.3.3. JSON Value verändern . . . . .	E 4
E.3.4. JSON-Objekt senden und empfangen . . . . .	E 5
E.4. JSON und C . . . . .	E 6
E.4.1. Allgemeines . . . . .	E 6
E.4.2. JSON API . . . . .	E 6



# 1. Einleitung

## 1.1. Allgemeines

Im Rahmen des Kernmoduls Embedded Systems (EMBSY) vertiefen die Studierenden das Erlernte mit einer Projektarbeit. Mit Hilfe eines Embedded Webservers sollen Ein- und Ausgänge innerhalb des Modells "Webhouse" angesteuert werden. Als zentrale Einheit des Webhouse ist ein Beaglebone Black<sup>1</sup> mit einem zusätzlichen Aufsteckprint für die Ansteuerung der verschiedenen Komponenten zuständig.

## 1.2. Systemübersicht

Das System umfasst die in Abbildung 1.1 dargestellten Komponenten. Nachfolgende Auflistung erläutert kurz die einzelnen Komponenten.

- **WebSocketServer:** Der WebSocketServer läuft auf dem Beaglebone Black Webhouse. Von aussen kann über die IP-Adresse des Webhouses (diese kann normalerweise am Rahmen des gewünschten Webhouses abgelesen werden) auf den WebSocketServer verbunden werden. Die Anfragen werden auf Port 80 beantwortet. Alle Ressourcen (html, css, js, usw.) welche für eine Verbindung benötigt werden, müssen im Pfad `/opt/webhuesli/www/` des Webhouses abgelegt werden.
- **Steuerung:** Die Applikation "Steuerung" wird durch die Studierenden implementiert. Sie übernimmt die Ansteuerung der einzelnen Komponenten. Befehle werden über eine Socketverbindung (localhost:5000) empfangen und versendet.
- **Komponenten:** Die Steuerung kann verschiedene Ein- und Ausgänge (Komponenten) ansteuern:
  - *Alarm:* Innerhalb des Webhouse befindet sich ein passiver Infrarot Sensor (PIR). Dieser wird als Alarm genutzt.
  - *DLampe:* Die Deckenlampe (DLampe) kann gedimmt werden.
  - *SLampe:* Eine Ständerlampe (SLampe) kann durch die Steuerung abgeblendet werden.
  - *Heizung:* Durch dimmen der Heizung kann auf eine gewünschte Temperatur geregelt werden.
  - *TempIst:* Die aktuelle Temperatur (TempIst) kann ausgelesen werden.
  - *TV:* Ein Fernseher (TV) kann ein- und ausgeschaltet werden.

---

<sup>1</sup><http://beagleboard.org/Products/BeagleBone%20Black>

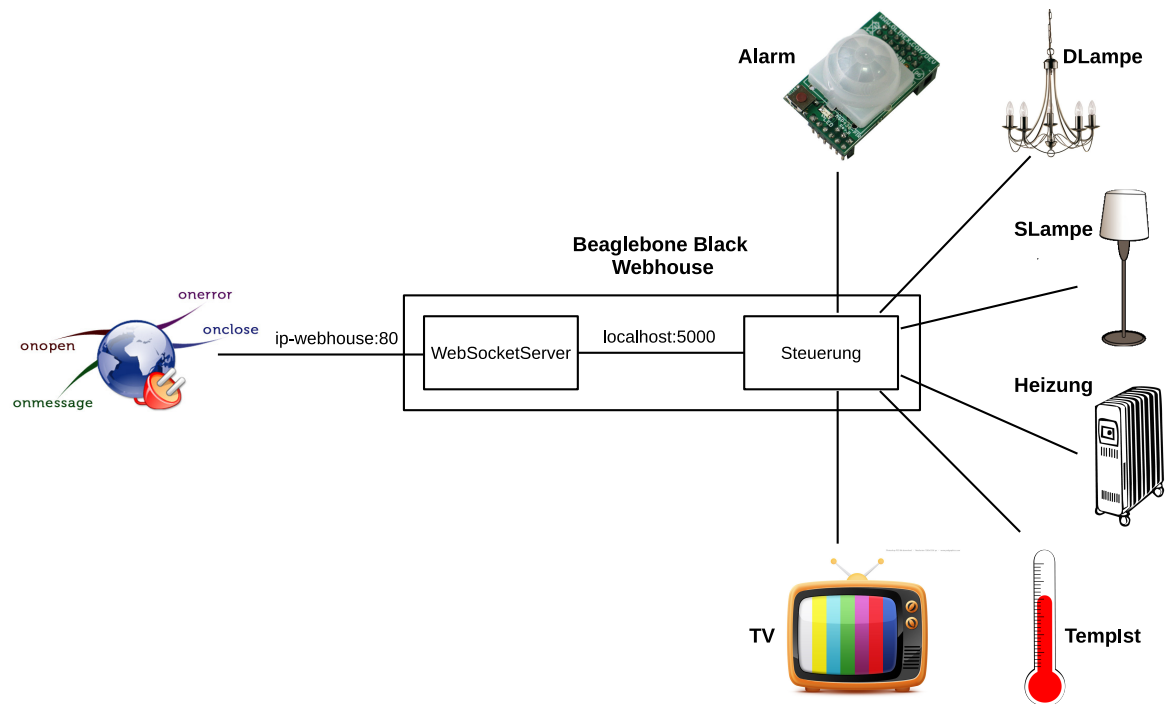


Abbildung 1.1.: Systemübersicht

## 2. Hardware Embedded Webserver

### 2.1. Allgemeines

Abbildung 2.1 zeigt den Print, welcher auf das Beaglebone Black aufgesteckt werden muss, um die Ein- und Ausgänge des Webhouses zu steuern. Die Anschlüsse der verschiedenen Komponenten werden im Abschnitt 2.2 erläutert. Die Speisung des Beaglebone Black Webhouses erfolgt mit 12V DC und ist zusätzlich abgesichert.

Der Temperatursensor LM75 (siehe Abbildung 2.1) ist unter dem Heizwiderstand montiert, welcher parallel zum Heizungsausgang des Webhouse geschaltet ist. Dadurch kann die Heizung des Webhouse simuliert werden.

Zu Testzwecken ist die serielle Konsole (dev/ttyS0) über einen USB to Serial Converter auf dem Print verfügbar.

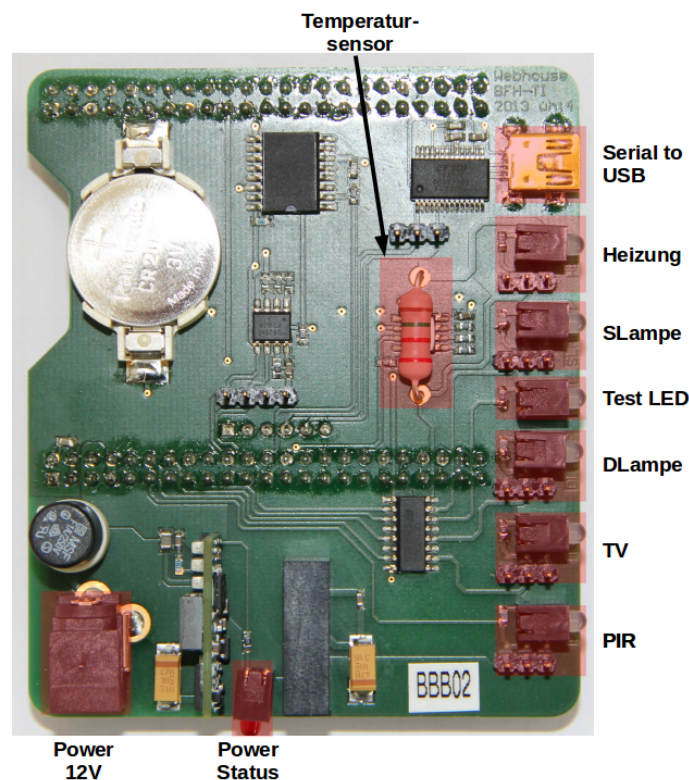
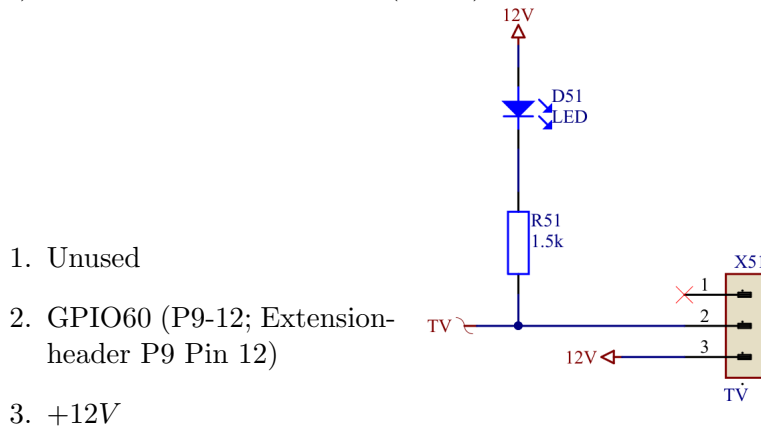


Abbildung 2.1.: Übersicht Hardware

## 2.2. Anschlüsse

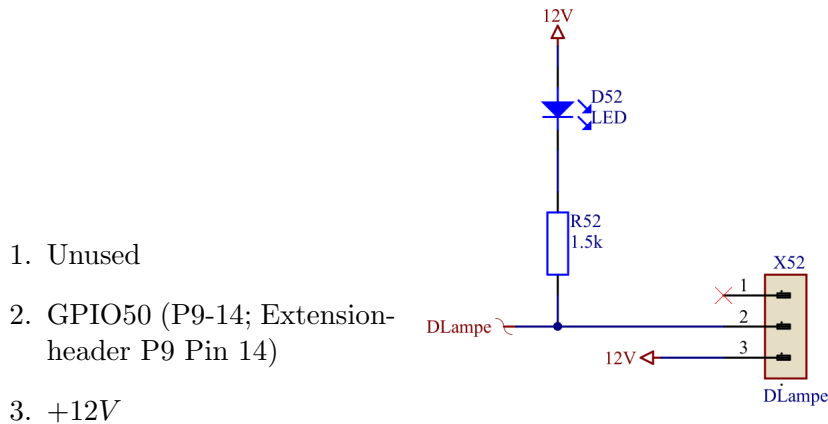
### 2.2.1. TV

Der Fernseher wird mit einer 12V Glühlampe betrieben, welche über den Anschlussstecker (X51) mit dem GPIO60 (P9-12) des Beaglebone Black Webhouse verbunden ist.



### 2.2.2. DLampe

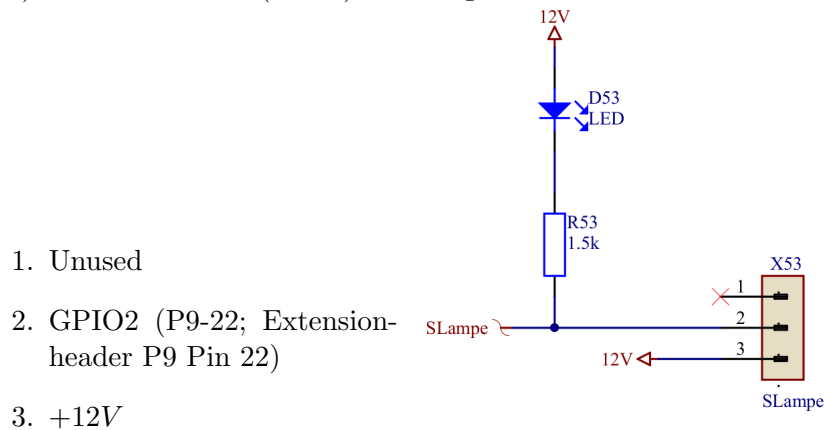
Die Deckenlampe wird mit einer 12V Glühlampe betrieben, welche über den Anschlussstecker (X52) mit dem GPIO50 (P9-14) des Beaglebone Black Webhouse verbunden ist.





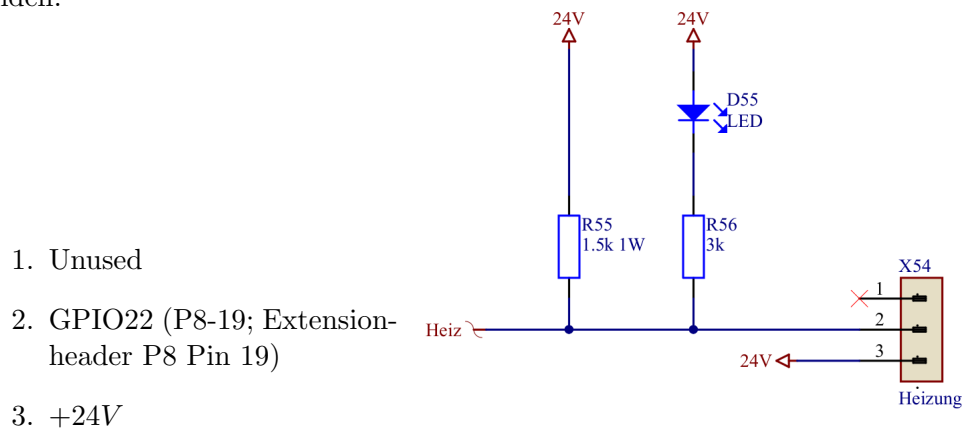
### 2.2.3. SLampe

Die Ständerlampe wird mit einer 12V Glühlampe betrieben, welche über den Anschlussstecker (X53) mit dem GPIO2 (P9-22) des Beaglebone Black Webhouse verbunden ist.



### 2.2.4. Heizung

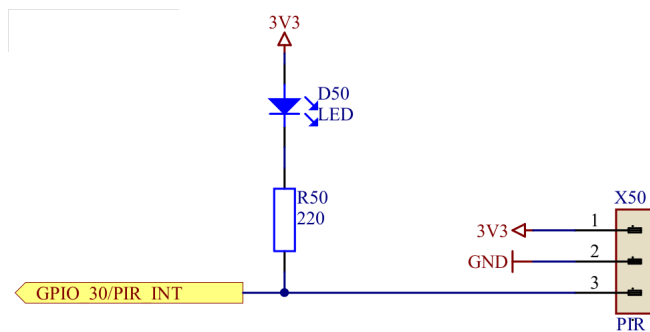
Die Heizung besteht aus einem Heizwiderstand, welcher über dem Temperatursensor (LM75) angebracht ist und zwei LED-Radiatoren. Die Heizung wird mit 24V betrieben und ist über den Anschlussstecker (X54) mit dem GPIO22 (P8-19) des Beaglebone Black Webhouse verbunden.



### 2.2.5. Alarm

Der Alarm wird durch ein Passiv Infrarot Sensormodul (PIR) von Hygrotec Messtechnik, welches auf Wärmestrahlung reagiert, realisiert. Beim Erkennen einer Wärmestrahlung wechselt der Pegel am Schaltausgang des Sensors von 0V auf 3.3V. Der Schaltausgang des PIR wird via Anschlussstecker (X50) mit dem GPIO30 (P9-11) des Beaglebone Black Webhouse verbunden.

1. +3.3V
2. GND
3. GPIO30 (P9-11; Extension-header P9 Pin 11)



## 2.3. Belegung Extension Header Beaglebone Black

Das Beaglebone Black verfügt über zwei Extension Header (P8 und P9). Der Print zur Ansteuerung der Ein- und Ausgänge (siehe Abschnitt 2.1) wird auf diesen Headern aufgesteckt. Tabelle 2.1 gibt Aufschluss über die verwendeten Pins der beiden Header und deren Funktion.

Header	Pin	GPIO	Mode	Funktion Webhouse
P8	19	GPIO22	EHRPWM2A	Ansteuerung Heizung
P9	11	GPIO30	Digital Input	PIR
P9	12	GPIO60	Digital Output	TV
P9	13	GPIO31	Digital Input	Temperatur Sensor Interrupt
P9	14	GPIO50	EHRPWM1A	Ansteuerung DLampe
P9	15	GPIO48	Digital Output	Test LED
P9	19	GPIO13	I2C2_SCL	I2C2_SCL
P9	20	GPIO12	I2C2_SDA	I2C2_SDA
P9	22	GPIO2	EHRPWM0A	Ansteuerung SLampe
P9	23	GPIO49	Digital Input	RTC Interrupt

Tabelle 2.1.: Belegung Extension Header Beaglebone Black

## 3. Software Embedded Webserver

### 3.1. Allgemeines

Die Applikation "Steuerung" wird durch die Studierenden implementiert. Sie übernimmt die Ansteuerung der einzelnen Komponenten. Die einzelnen Komponenten können mit Hilfe einer API angesteuert werden (siehe 3.2). Alle verfügbaren Funktionen sind in Abbildung 3.1 dargestellt.

Befehle für die Ansteuerung werden über eine Socketverbindung (localhost:5000) empfangen und versendet. Das verwendete Datenformat kann von den Studierenden selbst bestimmt werden. Als Option stehen Funktionen zur Verfügung, um die Daten mit Hilfe von JSON auszutauschen (siehe Anhang E).

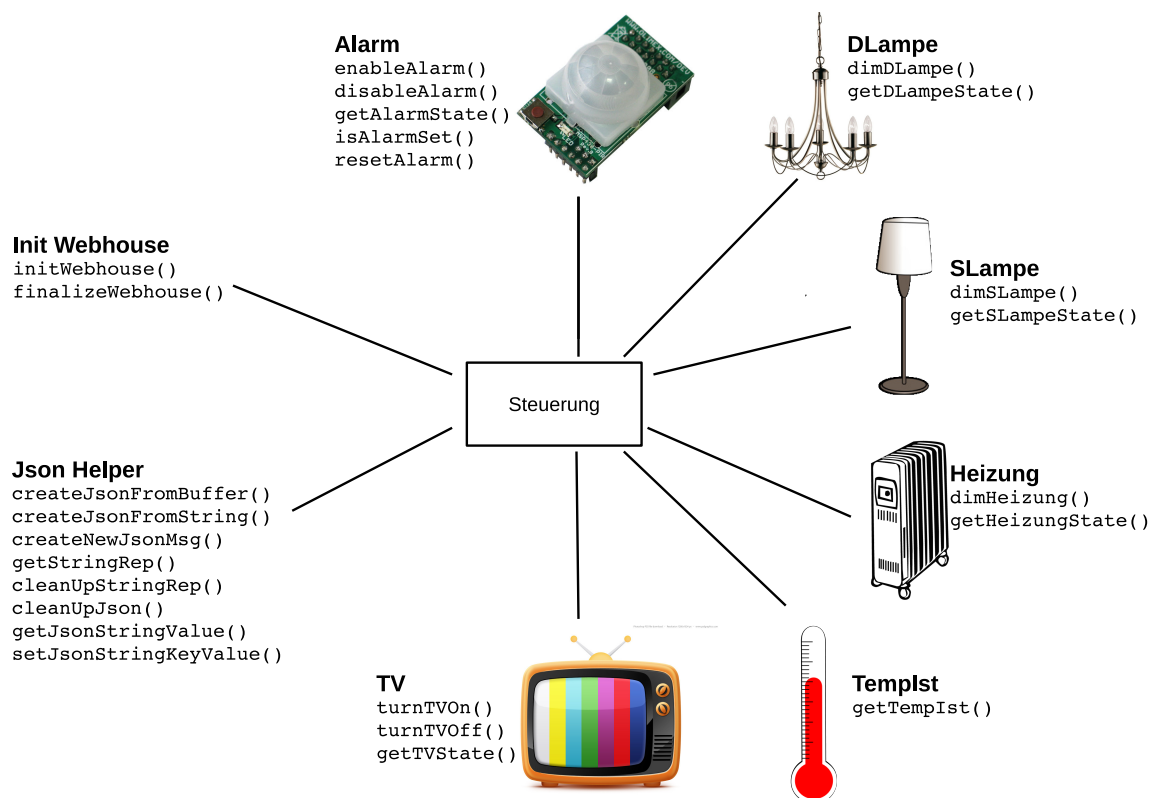


Abbildung 3.1.: Übersicht Hardware

## 3.2. Ansteuerung Komponenten

### 3.2.1. Initialisierung

#### initWebhouse

```
#include "Webhouse.h"
BBBError initWebhouse(void);
```

#### Summary

Initialisiert alle Komponenten innerhalb des Webhouse. Diese Funktion muss aufgerufen werden bevor eine Komponente angesteuert werden kann.

#### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.
BBB_ERR_PARAM	Parameter Fehler.

#### finalizeWebhouse

```
#include "Webhouse.h"
BBBError finalizeWebhouse(void);
```

#### Summary

Gibt alle Ressourcen frei welche bei der Initialisierung des Webhouses erzeugt wurden und bringt alle Komponenten in einen definierten Zustand.

#### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.
BBB_ERR_PARAM	Parameter Fehler.

### 3.2.2. TV

#### turnTVOn

```
#include "Webhouse.h"
BBBError turnTVOn(void);
```

#### Summary

Einschalten des Fernsehers.

#### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.
BBB_ERR_PARAM	Parameter Fehler.

### turnTVOff

```
#include "Webhouse.h"
BBBError turnTVOff(void);
```

#### Summary

Ausschalten des Fernsehers.

#### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.
BBB_ERR_PARAM	Parameter Fehler.

### getTVState

```
#include "Webhouse.h"
int32_t getTVState(void);
```

#### Summary

Gibt den aktuellen Zustand des Fernsehers zurück.

#### Return Values

0	Der Fernseher ist ausgeschaltet.
1	Der Fernseher ist eingeschaltet.

### 3.2.3. DLampe

#### dimDLampe

```
#include "Webhouse.h"
BBBError dimDLampe(uint8_t u8Duty);
```

#### Summary

Dimmt die Deckenlampe.

#### Parameters

u8Duty	Ein Wert im Bereich [0,100], wobei bei 0 die Lampe ausgeschaltet und bei 100 voll eingeschaltet ist.
--------	--

#### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.

#### getDLampeState

```
#include "Webhouse.h"
int32_t getDLampeState(void);
```

##### Summary

Gibt den aktuellen Zustand der Deckenlampe zurück.

##### Return Values

Ein Wert im Bereich [0,100], wobei bei 0 die Lampe ausgeschaltet und bei 100 voll eingeschaltet ist.

#### 3.2.4. SLampe

##### dimSLampe

```
#include "Webhouse.h"
BBBError dimSLampe(uint8_t u8Duty);
```

##### Summary

Dimmt die Ständerlampe.

##### Parameters

u8Duty	Ein Wert im Bereich [0,100], wobei bei 0 die Lampe ausgeschaltet und bei 100 voll eingeschaltet ist.
--------	--

##### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.

#### getSLampeState

```
#include "Webhouse.h"
int32_t getSLampeState(void);
```

##### Summary

Gibt den aktuellen Zustand der Deckenlampe zurück.

##### Return Values

Ein Wert im Bereich [0,100], wobei bei 0 die Lampe ausgeschaltet und bei 100 voll eingeschaltet ist.

### 3.2.5. Heizung und Temperatur

#### dimHeizung

```
#include "Webhouse.h"
BBBError dimHeizung( uint8_t u8Duty );
```

##### Summary

Dimmt die Heizung.

##### Parameters

u8Duty	Ein Wert im Bereich [0,100], wobei bei 0 die Heizung ausgeschaltet und bei 100 voll eingeschaltet ist.
--------	--

##### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_FILE_OPEN	Fehler beim Öffnen eines Files.

#### getHeizungState

```
#include "Webhouse.h"
int32_t getHeizungState( void );
```

##### Summary

Gibt den aktuellen Zustand der Heizung zurück.

##### Return Values

Ein Wert im Bereich [0,100], wobei bei 0 die Heizung ausgeschaltet und bei 100 voll eingeschaltet ist.

#### getTemplst

```
#include "Webhouse.h"
int32_t getTempIst( void );
```

##### Summary

Gibt die aktuelle Temperatur zurück.

##### Return Values

Aktuell gemessene Temperatur in Grad Celcius.

#### Example

Listing 3.1 zeigt ein Beispiel wie die Heizung innerhalb des Webhouses angesteuert werden kann. Dabei wird die aktuelle Temperatur ausgelesen und die Heizung eingeschaltet. Es wird solange geheizt bis die Solltemperatur erreicht wurde.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

#include "Webhouse.h"

int main(int argc, char **argv) {

    int32_t s32TempIst = 0;
    int32_t s32TempSoll = 22;

    // Initializes all used hardware within the webhouse
    initWebhouse();

    // Read the current temperature
    s32TempIst = getTempIst();
    printf("\nCurrent temperature is: %d", s32TempIst);

    // Dim the heater (100 -> full heating)
    dimHeizung(100);

    while(s32TempSoll > s32TempIst) {

        sleep(1); // sleep 1sec
        s32TempIst = getTempIst();
        printf("\nCurrent temperature is: %d", s32TempIst);
    }

    // Dim the heater (0 -> turn heating off)
    dimHeizung(0);

    finalizeWebhouse();
    return EXIT_SUCCESS;
}
```

Listing 3.1: Beispiel zur Steuerung der Heizung



### 3.2.6. Alarm

#### enableAlarm

```
#include "Webhouse.h"
BBBError enableAlarm(void)
```

##### Summary

Aktivieren des Alarms. Sobald der Alarm aktiviert ist, wird der PIR Sensor innerhalb des Webhouses gepollt. Wird eine Bewegung detektiert, so wird der Alarm ausgelöst. Ob ein Alarm ausgelöst wurde, kann mit der Funktion `isAlarmSet()` ausgelesen werden. Der Alarm kann mit `resetAlarm()` zurückgesetzt werden.

##### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_THREAD_CREATE	Der Thread in welchem der Zustand des PIR's gepollt wird konnte nicht erzeugt werden.
BBB_THREAD_RUNNING	Der Alarm ist bereits aktiviert.

#### disableAlarm

```
#include "Webhouse.h"
BBBError disableAlarm(void);
```

##### Summary

Deaktivieren des Alarms.

##### Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_THREAD_RUNNING	Der Alarm ist bereits deaktiviert.

#### getAlarmState

```
#include "Webhouse.h"
int32_t getAlarmState(void);
```

##### Summary

Gibt zurück ob der Alarm ein- oder ausgeschaltet ist.

##### Return Values

0	Der Alarm ist deaktiviert.
1	Der Alarm ist aktiviert.

#### **isAlarmSet**

```
#include "Webhouse.h"  
boolE isAlarmSet(void)
```

#### **Summary**

Wird eine Bewegung detektiert, so wird der Alarm ausgelöst. Ob ein Alarm ausgelöst wurde, kann mit dieser Funktion ausgelesen werden.

#### **Return Values**

0	Es wurde kein Alarm detektiert.
1	Ein Alarm wurde ausgelöst.

#### **resetAlarm**

```
#include "Webhouse.h"  
void resetAlarm(void)
```

#### **Summary**

Zurücksetzen des Alarms. Anschliessend kann wieder ein neuer Alarm detektiert werden.

**Example**

Listing 3.2 zeigt ein Beispiel wie der Alarm innerhalb des Webhouses angesteuert werden kann. Nach dem Aktivieren des Alarms, wird innerhalb eines Loops auf das Detektieren eines Alarms gewartet. Nachdem der Alarm drei mal detektiert wurde, wird er ausgeschaltet und das Programm wird beendet.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>

#include "Webhouse.h"

int main(int argc, char **argv) {

    int32_t i = 0;

    // Initializes all used hardware within the webhouse
    initWebhouse();

    printf("Turn alarm on, wait on pir event!\n");
    enableAlarm();
    for(i = 0; i < 3; i++) {

        while(isAlarmSet() == 0) {
            usleep(100000); // sleep 100ms
        }
        printf("PIR event detected. Event count %d", i);
        resetAlarm();
    }
    disableAlarm();

    finalizeWebhouse();
    return EXIT_SUCCESS;
}
```

Listing 3.2: Beispiel zur Steuerung des Alarms

# Literaturverzeichnis

- [1] Homepage Wikipedia  
[\*http://de.wikipedia.org/wiki/JavaScript\\_Object\\_Notation\*](http://de.wikipedia.org/wiki/JavaScript_Object_Notation)  
Stand August 2013
  
- [2] Homepage json.org  
[\*http://www.json.org/\*](http://www.json.org/)  
Stand August 2013

# Tabellenverzeichnis

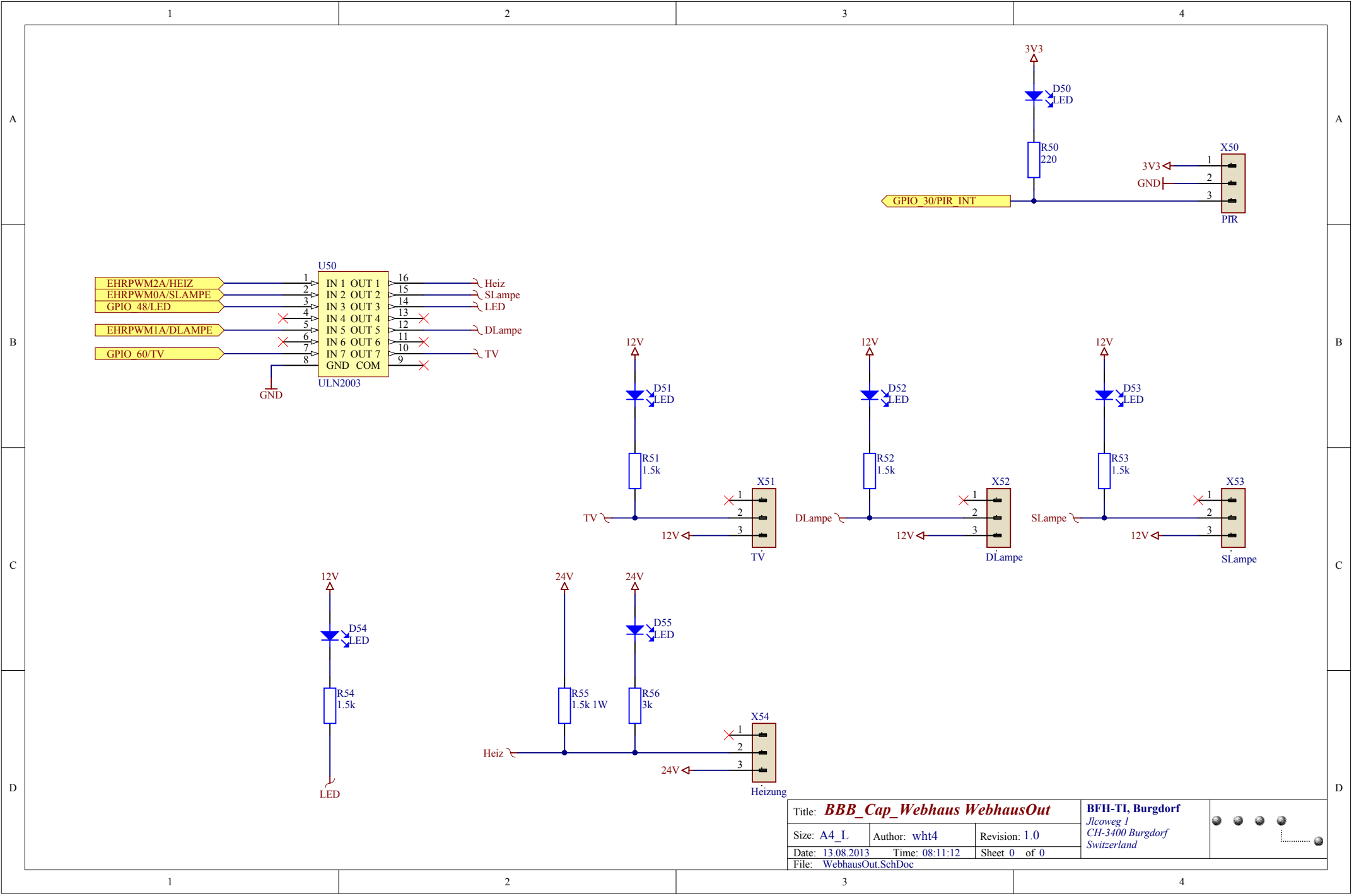
2.1. Belegung Extension Header Beaglebone Black . . . . .	6
---	---

# Abbildungsverzeichnis

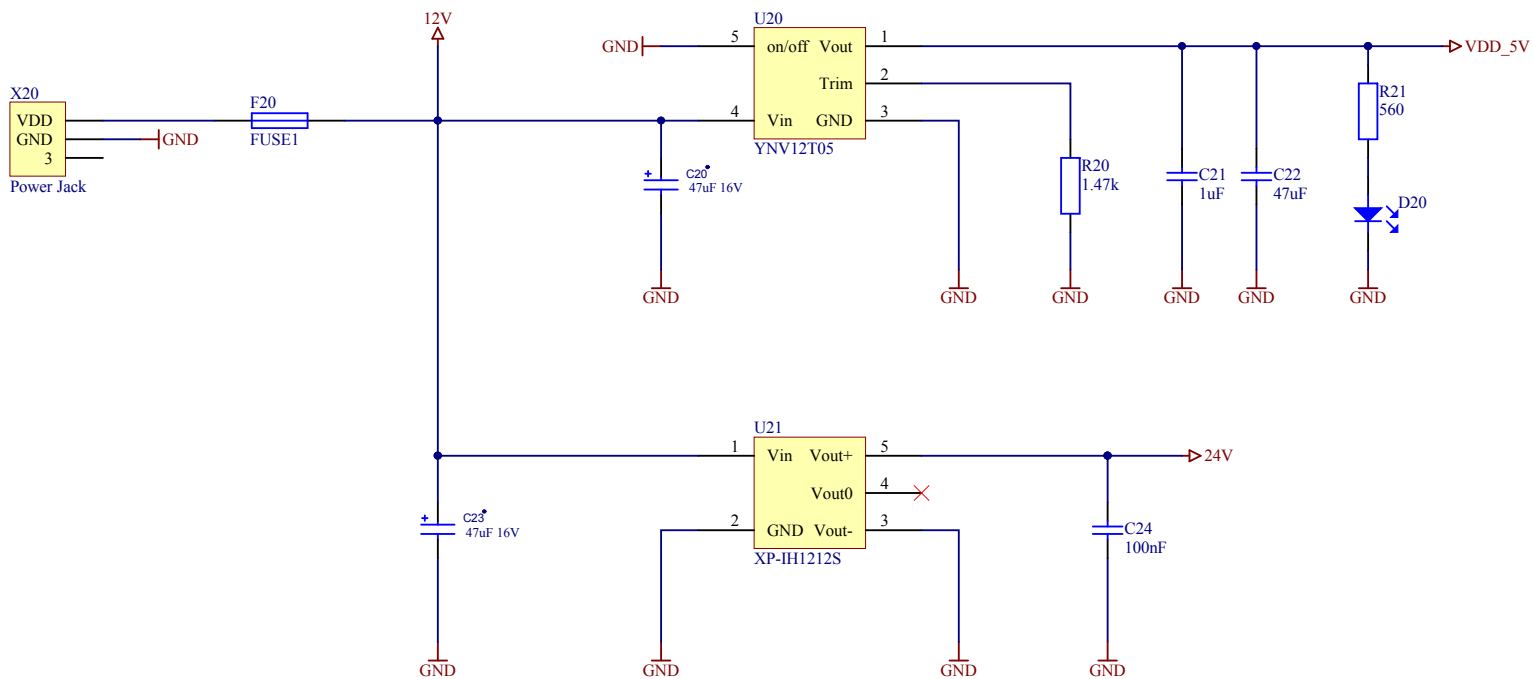
1.1. Systemübersicht . . . . .	2
2.1. Übersicht Hardware . . . . .	3
3.1. Übersicht Hardware . . . . .	7
E.1. JSON Values [2] . . . . .	E 2
E.2. JSON Objekte [2] . . . . .	E 2


**Anhang A.**

**Anhang Hardware**







Title: <b>BBB_Cap_Webhaus Power</b>			<b>BFH-TI, Burgdorf</b> <i>Jlcoweg 1</i> <i>CH-3400 Burgdorf</i> <i>Switzerland</i>	
Size: <b>A4_L</b>	Author: <b>wht4</b>	Revision: <b>1.0</b>		
Date: <b>13.08.2013</b>	Time: <b>08:11:13</b>	Sheet <b>0</b> of <b>0</b>		
File: <b>Power.SchDoc</b>				





1

2

3

4

A

A

B

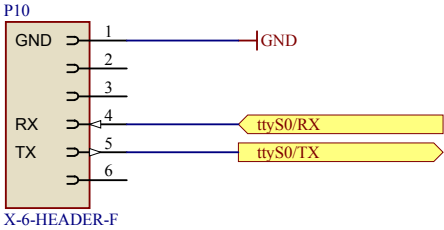
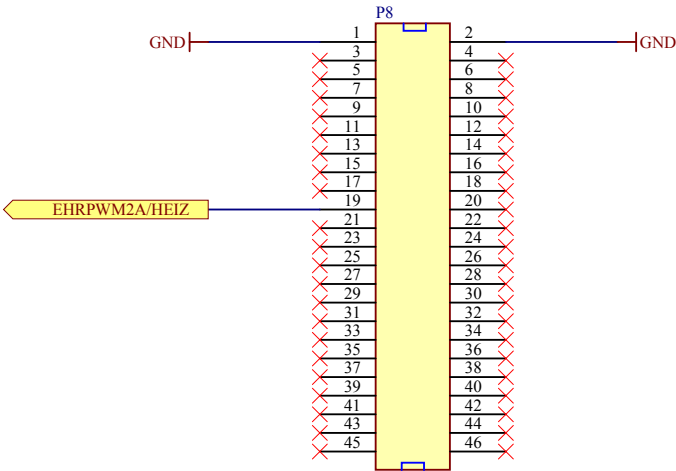
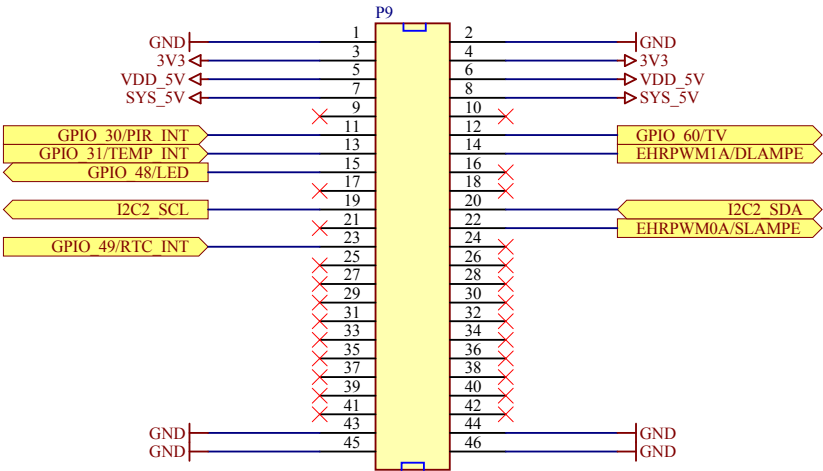
B

C

C

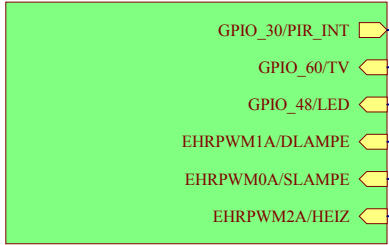
D

D

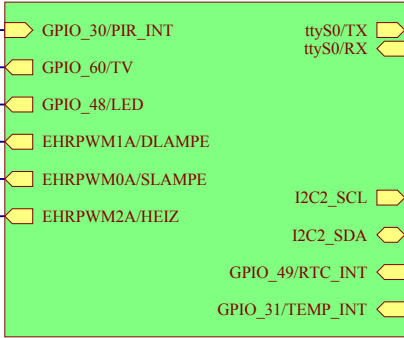


Title: <b>BBB_Cap_Webhaus Connector</b>			BFH-TI, Burgdorf	
Size: <b>A4_L</b>	Author: <b>wht4</b>	Revision: <b>1.0</b>	Jlcweg 1	
Date: <b>13.08.2013</b>	Time: <b>08:11:13</b>	Sheet <b>0</b> of <b>0</b>	CH-3400 Burgdorf	
File: <b>Connector.SchDoc</b>			Switzerland	

U\_WebhausOut  
WebhausOut.SchDoc



U\_Connector  
Connector.SchDoc



U\_FTDI  
FTDI.SchDoc




U\_I2C  
I2C.SchDoc

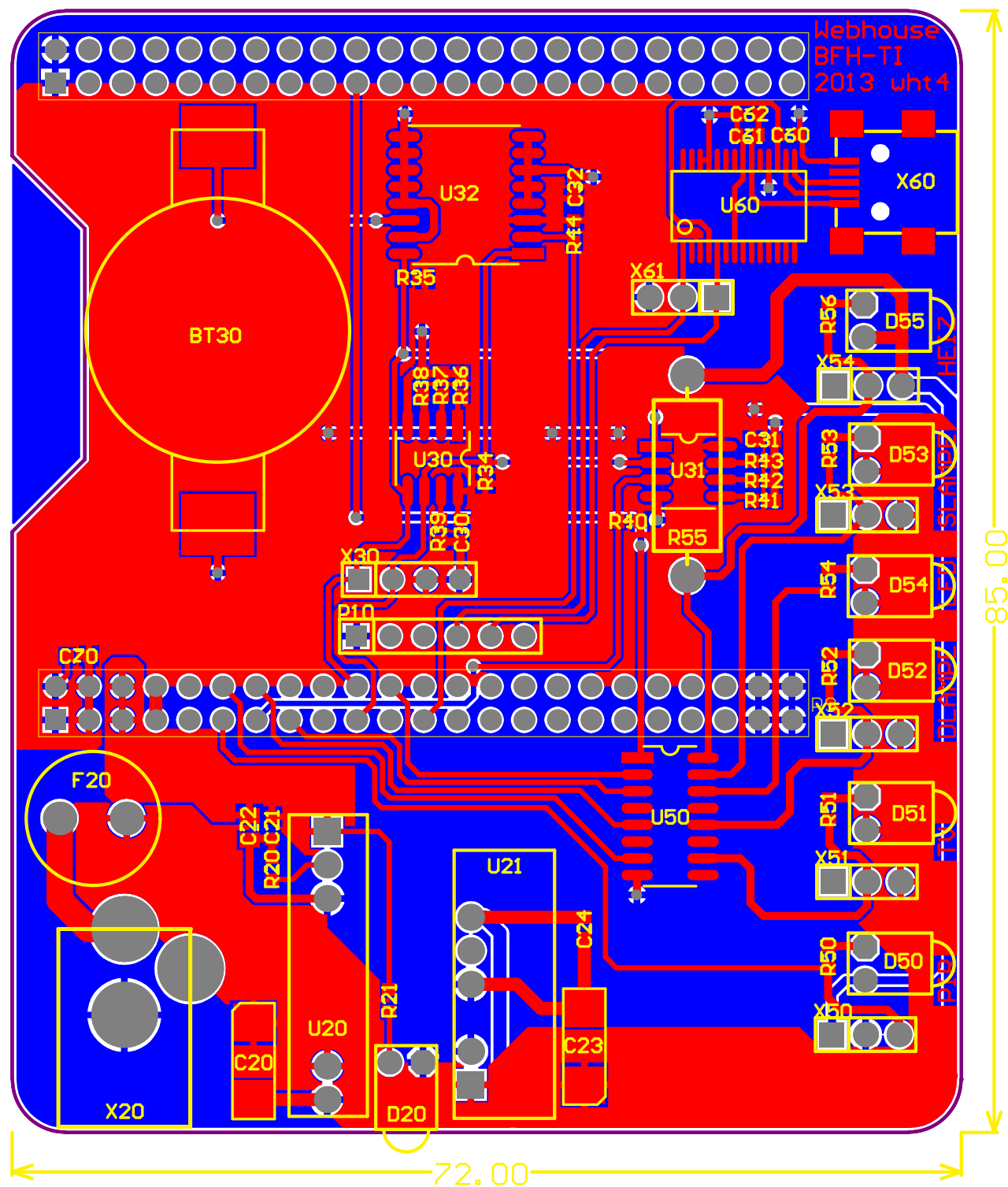


U\_Power  
Power.SchDoc



Title: <b>BBB_Cap_Webhaus</b>			<b>BFH-TI, Burgdorf</b> <i>Jlcoweg 1</i> <i>CH-3400 Burgdorf</i> <i>Switzerland</i>	
Size: <b>A4_L</b>	Author: <b>wht4</b>	Revision: <b>1.0</b>		
Date: <b>13.08.2013</b>	Time: <b>08:11:13</b>	Sheet <b>1</b> of <b>0</b>		
File: <b>BBB_Cap_Webhaus.SchDoc</b>				





Designator	Quantity	Comment	Description	Distributor	OrderCode	Preis pro Stk.
BT30	1	CR2032	Batteriehalter SMD für CR2032 Batterie	Farnell	121-6354	SFr. 1.10
BT30	1	CR2032	Battery CR2032 Lithium 3V 220mAh	Distrelec	97 08 76	SFr. 1.32
C24, C30, C31, C32, C60, C61	6	100nF	Capacitor 0603 100nF 50V			
C21	1	1uF	Capacitor 0603 1uF			
C70	1	10uF	Capacitor 0603 10uF			
C62	1	4.7uF	Capacitor 0603 4.7uF			
C22	1	47uF	Capacitor 0805 47uF	Farnell	157-2633	SFr. 1.25
C20, C23	2	47uF	Tantal Capacitor 47uF 16V SMD-C	Farnell	145-7486	SFr. 0.83
D20	1	LED	LED 3mm 50Grad Rot	Farnell	100-3383	SFr. 0.80
D50, D51, D52, D53, D54, D55	6	LED	LED 3mm 50Grad Grün	Farnell	100-3385	SFr. 0.89
F20	1	Sicherungshalter	Halter für Kleinstsicherungen	Distrelec	27 32 50	SFr. 1.76
F20	1	Sicherung	Kleinssicherung radial, 1A	Distrelec	27 07 26	SFr. 1.32
P8, P9	2	Header 23x2	Header Male 23x2, Raster 2.54mm	Distrelec	12 25 17	SFr. 3.07
P10	1	Header 6x1	Header Female 6x1, Raster 2.54mm	Distrelec	12 05 92	SFr. 1.64
R21	1	560	Resistor 0603 560			
R34, R35, R40, R44	4	10k	Resistor 0603 10k			
R36, R37, R38, R39, R41, R42, R43	7	0	Resistor 0603 0	Farnell	146-9739	SFr. 0.03
R50	1	220	Resistor 0603 220			
R20, R51. R52, R53, R54	5	1.5k	Resistor 0603 1.5k 1%			
R55	1	1.5k	Resistor Axial 1.5k 2W	Distrelec	71 21 81	SFr. 0.76
R56	1	3k	Resistor 0603 3k			
U20	1	DCDC Converter	YNV12T05-G DCDC Converter 12V 5A	Digikey	179-2376-ND	SFr. 10.94
U21	1	DCDC Converter	IH1212S DCDC Converter 2W +/-12V	Farnell	872-7988	SFr. 11.80
U30	1	EEPROM	CAT24C256W EEPROM 256kb I2C	Digikey	CAT24C256W I-GT30SCT-ND	SFr. 0.73
U31	1	LM75	LM75 I2C Temperatur Sensor	Digikey	LM75AIMX/N OPBCT-ND	SFr. 1.62
U32	1	DS1339	DS1339 I2C Serial RTC Internal Quarz	Digikey	DS1339C-33 #-ND	SFr. 6.57
U50	1	ULN2003	7 Fach Darlington Transistor Array	Digikey	497-2345-1-ND	SFr. 0.45
U60	1	FT232R	USB to serial UART interface	Farnell	114-6032	SFr. 6.65
X20	1	Power Jack	Power Jack RAPC722 5A	Distrelec	11 51 01	SFr. 3.51
X30	1	Header 4x1	Header Male 4x1, Raster 2.54mm	Distrelec	12 23 65	SFr. 1.21
X50, X51, X52, X53, X54, X61	6	Header 3x1	Header Male 3x1; Raster 2.54mm	Distrelec	12 23 51	SFr. 0.58
X60	1	USB Mini	Buchse USB Mini, PCB	Farnell	169-6539	SFr. 3.15

## Anhang B.

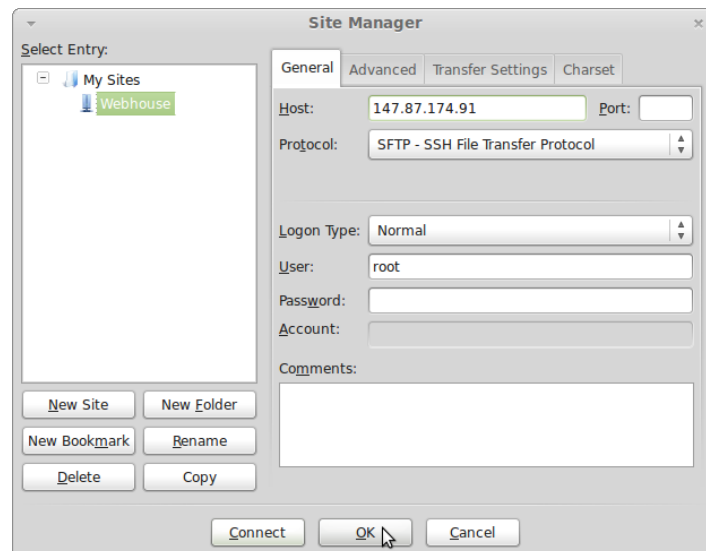
# Kopieren von Daten zum Webhouse

### B.1. Allgemeines

Von aussen kann über die IP-Adresse des Webhouses (diese kann normalerweise am Rahmen des gewünschten Webhouses abgelesen werden) auf den WebsocketServer verbunden werden. Die Anfragen werden auf Port 80 beantwortet. Alle Ressourcen (html, css, js, usw.) welche für eine Verbindung benötigt werden, müssen im Pfad `/opt/webhuesli/www/` abgelegt werden. Die einfachste Variante um Daten zum Webhouse zu kopieren ist über SFTP (SSH File Transfer Protocol). Im folgenden Abschnitt [B.2](#) wird aufgezeigt wie Daten mit Hilfe von FileZilla<sup>1</sup> auf das Webhouse kopiert werden können.

### B.2. Kopieren von Daten zum Webhouse mit FileZilla

1. **Erstellen einer Verbindung:** In FileZilla, klicke File → Site Manager... Innerhalb von Host geben Sie die IP Adresse des Webhouses ein (in unserem Beispiel 147.87.174.91). Als Protokoll wählen Sie SFTP und als User "root". Bevor Sie verbinden, können Sie der Verbindung noch einen Namen geben (Webhouse im Beispiel).



2. **Daten kopieren:** Wählen Sie auf der linken Seite die gewünschten Daten aus, welche auf das Webhouse kopiert werden sollen (im Beispiel das File index.html). Auf der

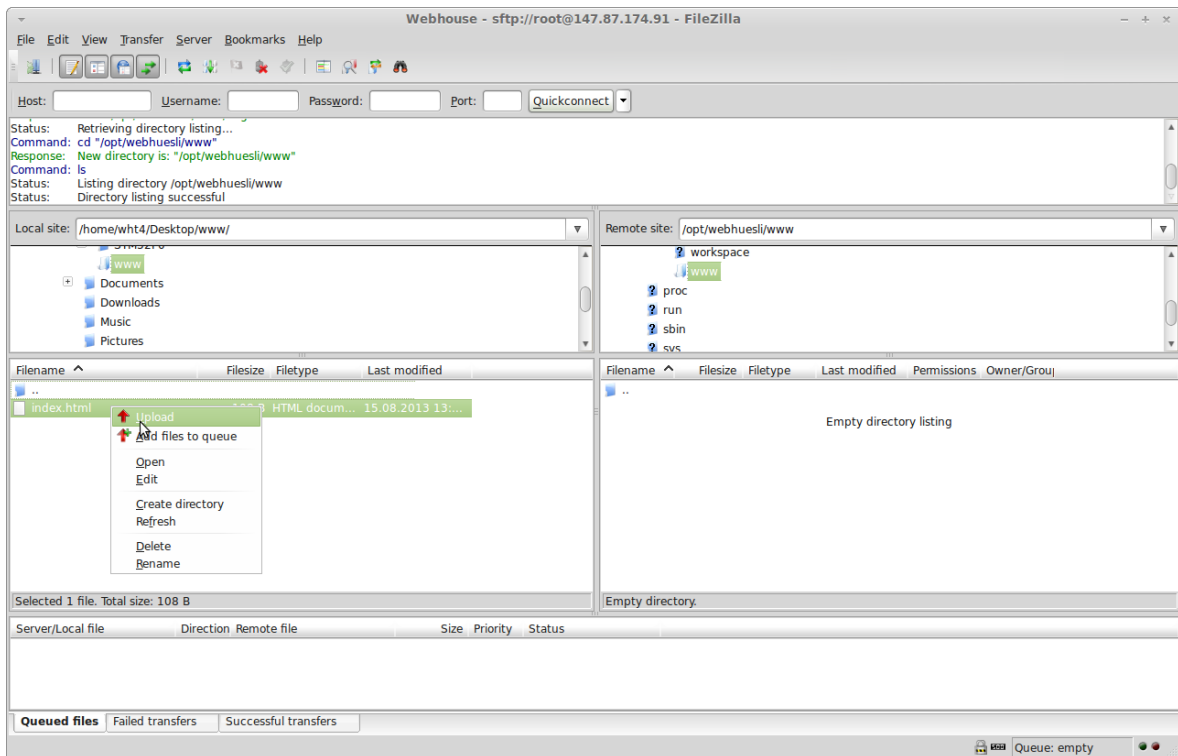
---

<sup>1</sup><https://filezilla-project.org/>



## Anhang B. Kopieren von Daten zum Webhouse

rechten Seite muss der Upload-Pfad spezifiziert werden (`/opt/webhuesli/www`). Anschließend kann der Upload durch einen Rechtsklick und Auswahl der Option Upload auf das gewünschte File gestartet werden.



# Anhang C.

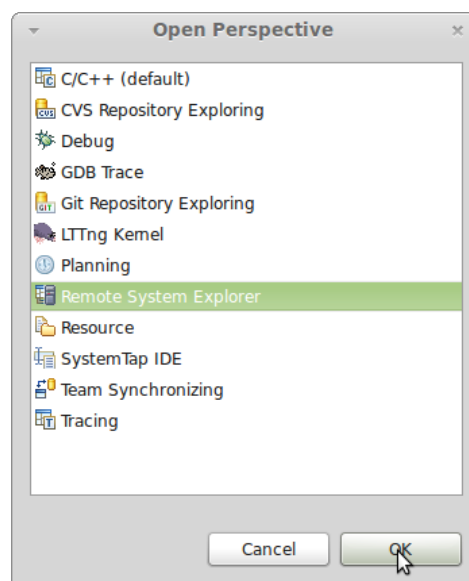
## Eclipse Remote System

### C.1. Allgemeines

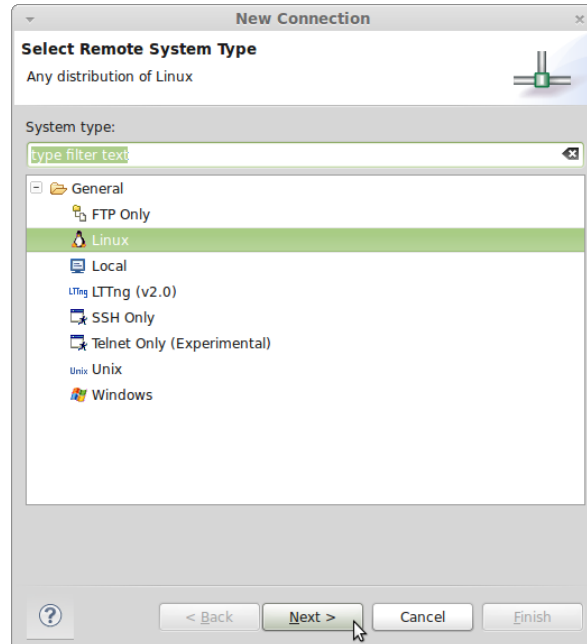
Aus Eclipse kann mit Hilfe des Remote System Plugins auf das Webhouse verbunden werden. Dies ist nötig um geschriebene Applikationen zu debuggen oder um aus Eclipse ein SSH-Terminal zum Webhouse zu öffnen. Abschnitt C.2 stellt die notwendigen Schritte für die Konfiguration einer solchen Verbindung dar. Die Konfiguration muss nur einmal vorgenommen werden und ist anschliessend im Workspace abgespeichert.

### C.2. Erstellen einer Verbindung zum Webhouse

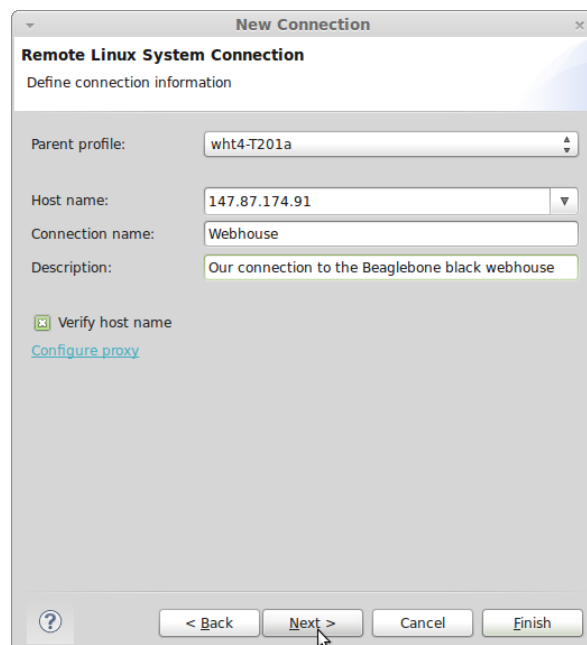
1. In Eclipse, wähle unter File → Open Perspective → Other den Remote System Explorer.



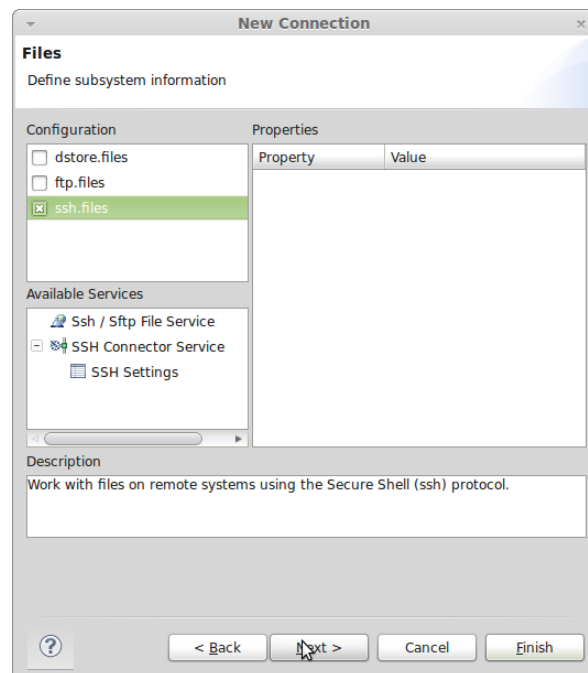
2. Mit Hilfe eines Rechtsklicks im Remote System Explorer können Sie die Option New → Connection auswählen. Im erscheinenden Dialog wählen Sie Linux.



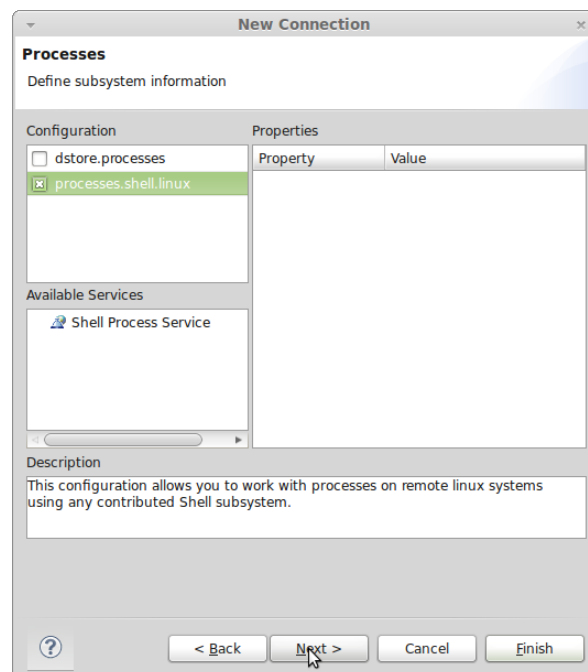
3. Geben Sie im Feld Host Name die IP-Adresse des Webhouses an (in unserem Beispiel 147.87.174.91). Geben Sie der Connection einen eindeutigen beschreibenden Namen.



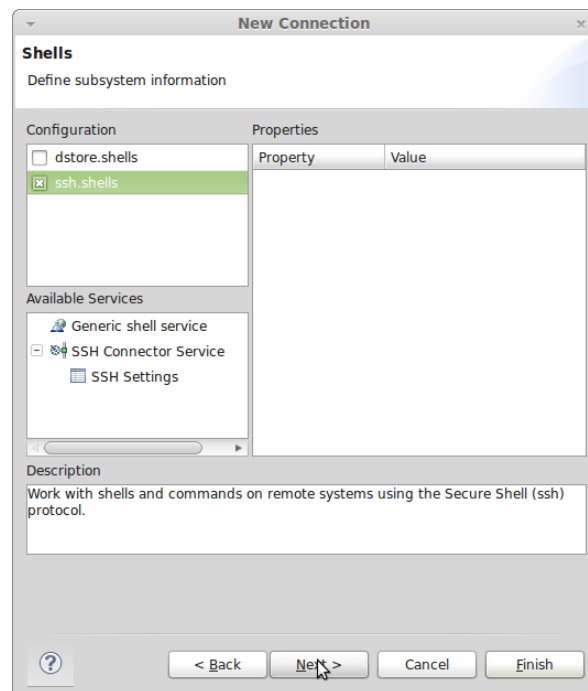
4. Unter Konfigurationen wählen Sie ssh.files



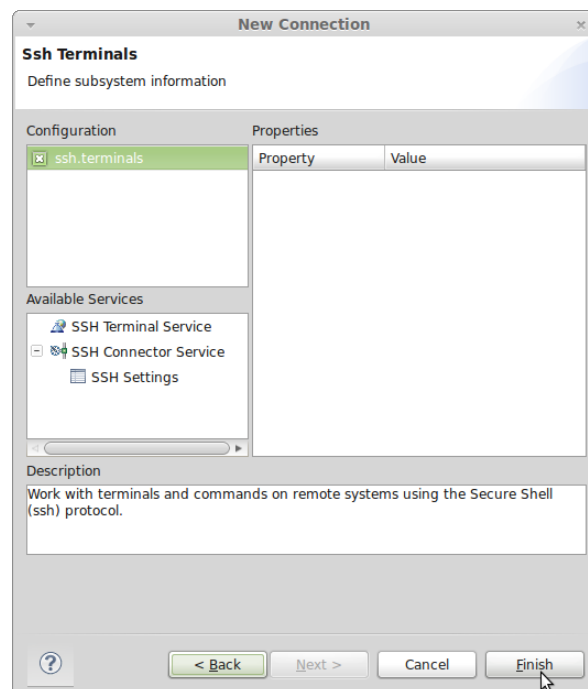
5. Unter Konfigurationen wählen Sie processes.shell.Linux



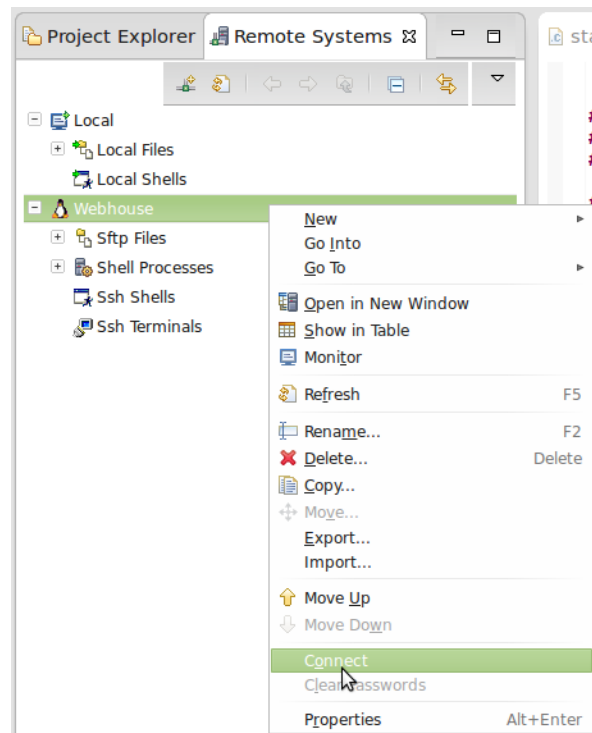
6. Unter Konfigurationen wählen Sie ssh.shells



7. Unter Konfigurationen wählen Sie ssh.terminals. Anschliessend können Sie die Konfiguration mit Finish beenden



8. Ein Verbindung zum Webhouse kann nun über den Remote System Explorer und die eben erstellte Connection hergestellt werden. Durch Rechtsklick auf die erstellte Connection kann die Option Connect ausgewählt werden.



## Anhang D.

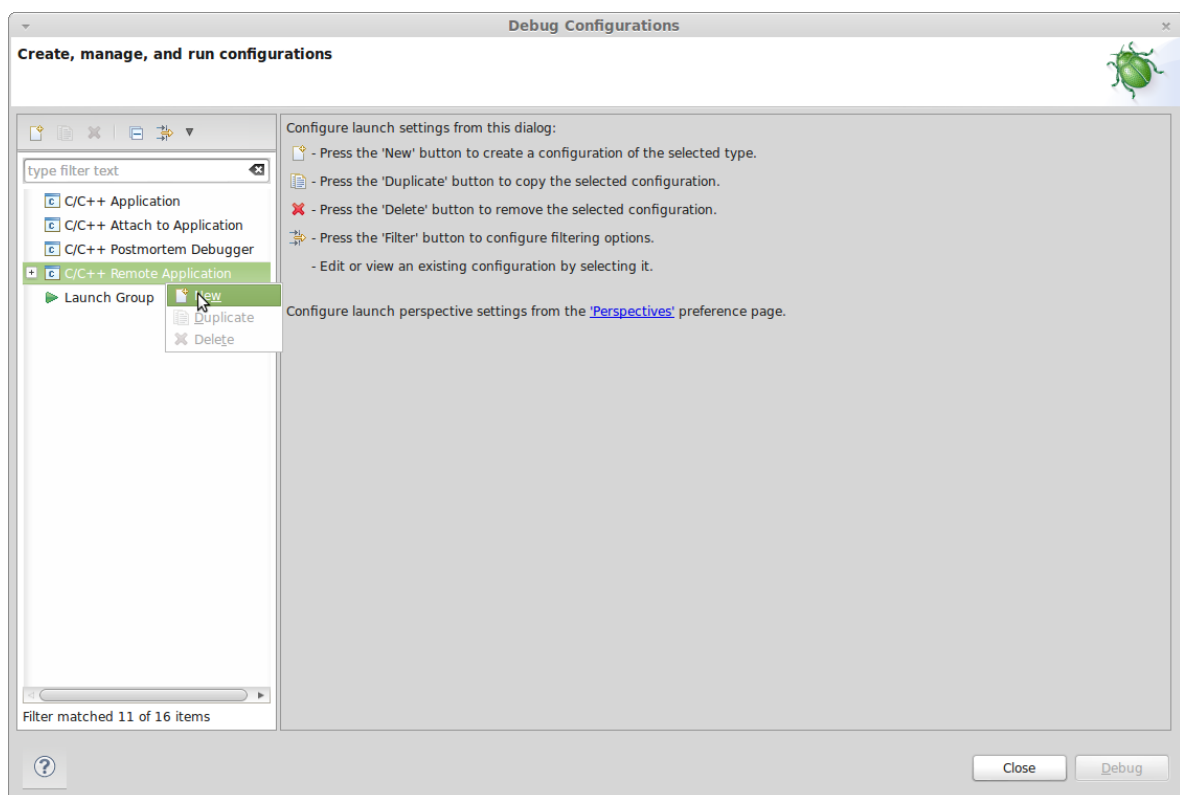
# Debugging eines Projektes auf dem Webhouse

### D.1. Allgemeines

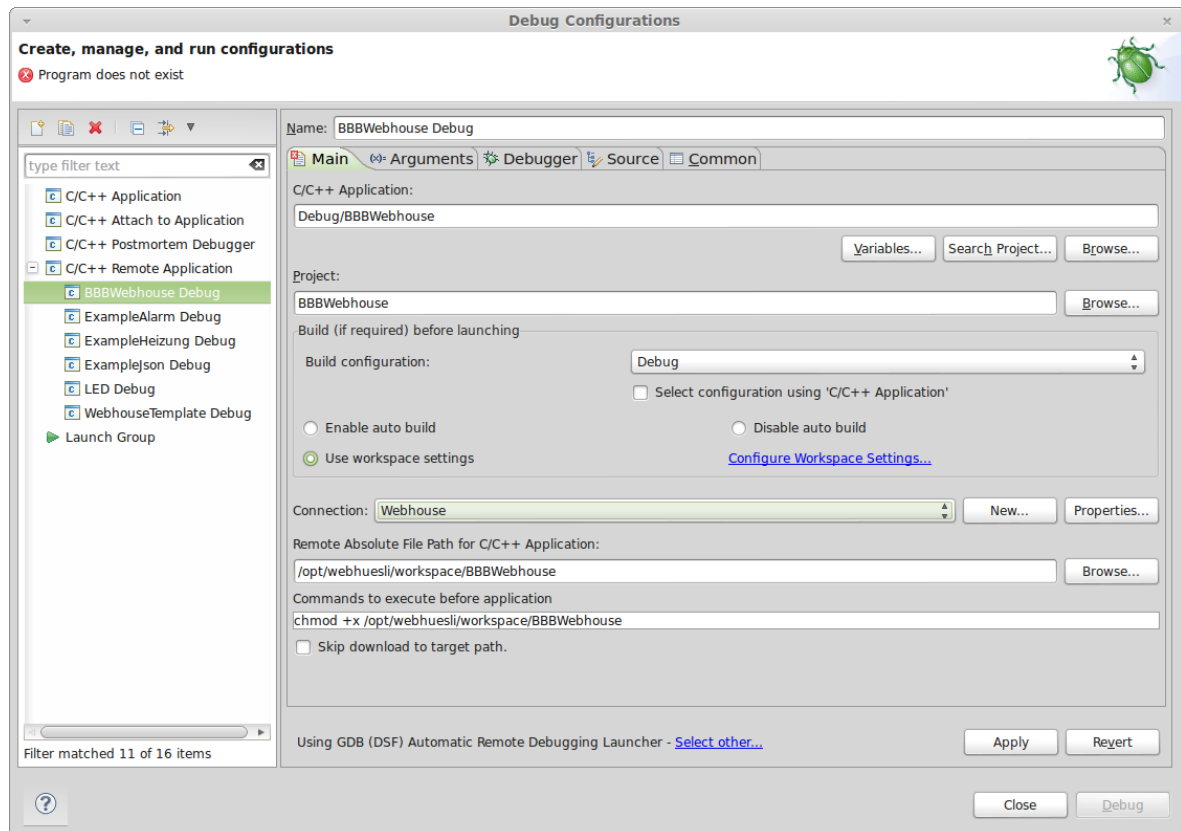
Wie im Anhang C gesehen, kann mit dem Remote System Plugin auf das Webhouse verbunden werden. Nachdem eine Verbindung hergestellt wurde, kann ein Projekt auf dem Remote System gedebugt werden. Hierzu muss eine Debug-Konfiguration erstellt werden. Der folgende Abschnitt zeigt die notwendigen Schritte für die Erstellung einer Debug-Konfiguration.

### D.2. Erstellen einer Debug-Konfiguration

1. In Eclipse, wähle unter Run → Debug Configurations...  
Anschliessend kann mit einem Rechtsklick auf "C/C++ Remote Application" eine neue Konfiguration erstellt werden.

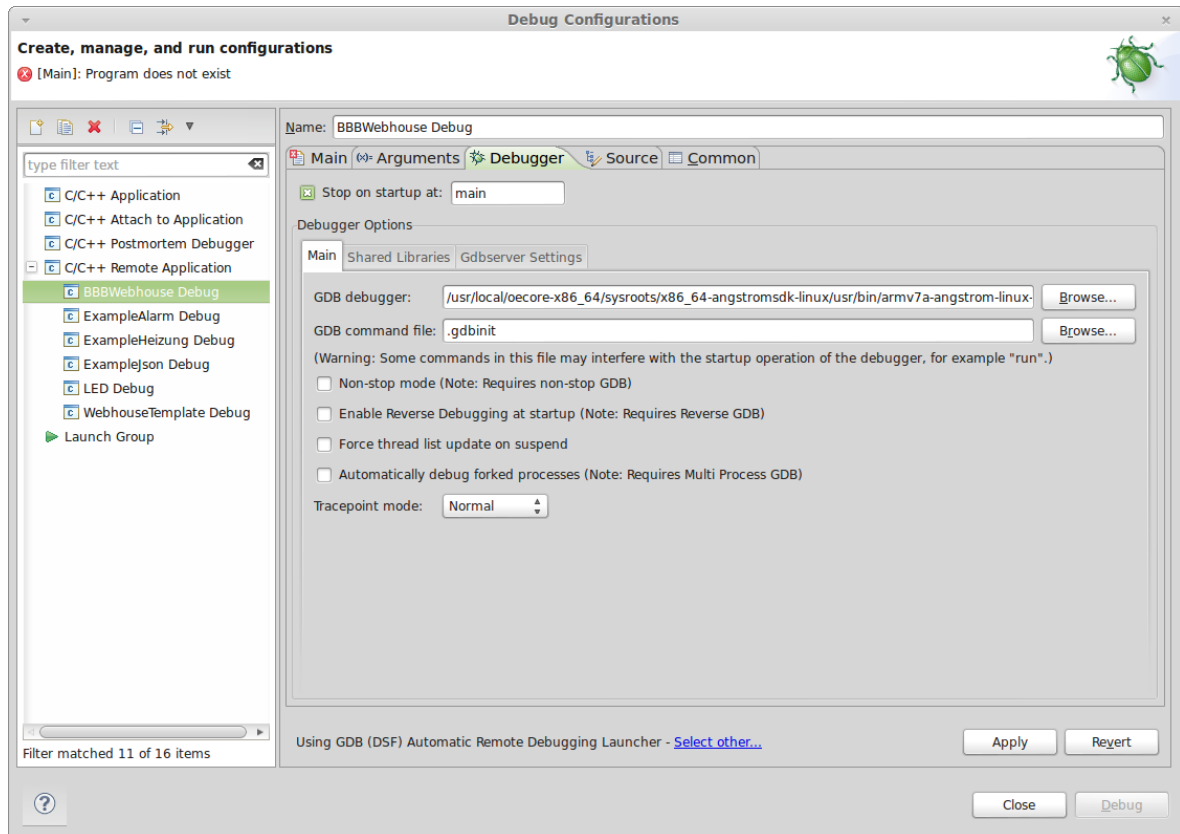


2. Im Tab "Main" kann nun das Projekt und die ausführbare Applikation ausgewählt werden. Zusätzlich kann die gewünschte Verbindung (in unserem Beispiel wird die Verbindung "Webhouse" verwendet) gewählt werden und der Pfad wohin die Applikation kopiert wird (in unserem Beispiel wird die Applikation nach `/opt/webhuesli/workspace/BBBWebhouse` kopiert). Damit die Applikation ausführbar ist, werden mit `chmod +x ...` der Applikation die entsprechenden Dateirechte hinzugefügt.





3. Im Tab "Debugger" muss noch der Cross-Debugger gewählt werden. Dieser befindet sich unter:  
/usr/local/oe-core-x86\_64/sysroots/x86\_64-angstromsdk-linux/usr/bin/armv7a-angstrom-linux-gnueabi/arm-angstrom-linux-gnueabi-gdb



4. Übernehmen Sie die Einstellungen mit "Apply" und starten Sie anschliessend den Debugger mit "Debug".

# Anhang E.

## JSON und Javascript

### E.1. Allgemeines

Die JavaScript Object Notation<sup>1</sup>, kurz JSON, ist ein kompaktes Datenformat in für Mensch und Maschine einfach lesbarer Textform zum Zweck des Datenaustauschs zwischen Anwendungen. Parser für JSON existieren in praktisch allen verbreiteten Sprachen. In Verbindung mit JavaScript wird JSON für Ajax oder WebSockets zur Übertragung von Daten zwischen Client und Server verwendet [1].

- JSON steht für **J**ava**S**cript **O**bject **N**otation
- JSON ist ein kompaktes textbasiertes Datenformat.
- JSON ist unabhängig von der Programmiersprache. JSON benutzt JavaScript-Syntax um die Datenobjekte zu beschreiben.
- JSON ist "selbst beschreibend" (lesbar für den Menschen) und einfach zu verstehen.

### E.2. JSON Syntax

#### E.2.1. Allgemeines

- Daten werden in Key-Value Paaren angeordnet.
- Daten werden durch Kommas separiert.
- Geschweifte Klammern beinhalten ein Objekt.
- Eckige Klammern beinhalten ein Array.

#### E.2.2. JSON Key-Value Paare

Daten werden in Key-Value Paaren angeordnet. Ein Key-Value Paar umfasst einen Key in Anführungszeichen gefolgt von einem Doppelpunkt und dem Value. Ein Beispiel eines JSON-Objekts ist im Listing E.1 dargestellt. Das Objekt besteht aus zwei Key-Value Paaren ("cmd": "set" sowie "dlampe": "58"). Beide Values sind String Values (siehe Abschnitt E.2.3).

---

<sup>1</sup>[www.json.org](http://www.json.org)

```
{  
  "cmd" : "set",  
  "dlampe" : "58"  
}
```

Listing E.1: Beispiel eines JSON-Objekts

### E.2.3. JSON Values

Folgende Values sind möglich (siehe auch Abbildung E.1):

- Ein String in Anführungszeichen
- Eine Zahl (Integer oder Floating point)
- Ein Objekt umfasst mit geschweiften Klammern
- Ein Array umfasst mit eckigen Klammern
- Ein Boolean (`true` oder `false`)
- Der `null` Value

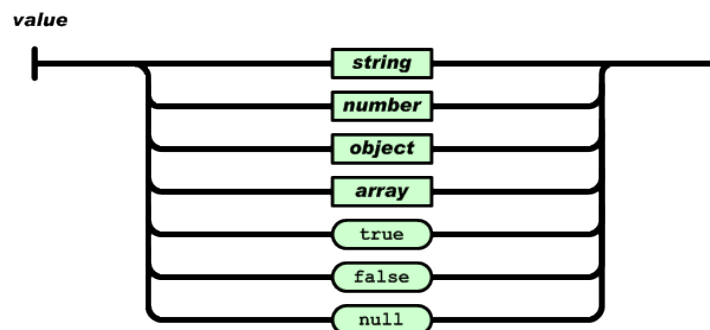


Abbildung E.1.: JSON Values [2]

### E.2.4. JSON Objekte

JSON Objekte werden durch geschweifte Klammern umfasst. Ein Objekt kann mehrere Key-Value Paare beinhalten. Abbildung E.2 zeigt den grundlegenden Aufbau von JSON-Objekten.

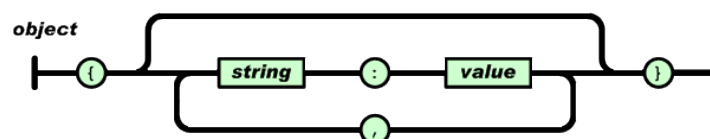


Abbildung E.2.: JSON Objekte [2]

## E.3. JSON und JavaScript

### E.3.1. Allgemeines

Mit einfachen Beispielen soll dieser Abschnitt einen Einstieg in die Benutzung von JSON mit JavaScript geben. Alle Beispiele sind eigenständig und sollten in einem Browser geöffnet werden können.

### E.3.2. JSON-Objekt erstellen

Im Beispiel von Listing E.2 wird ein JSON-Objekt erstellt mit den beiden Key-Value Paaren "tempist":"23" und "tempsoll":"30". Die beiden Values werden anschliessend im entsprechenden Textfeld eingefügt

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Create JSON-Object</title>
    <script type="text/javascript">

      var data = {"tempist":"23","tempsoll":"30"};

      window.onload=function(){
        document.getElementById('txt_tempist').value = data.tempist;
        document.getElementById('txt_tempsoll').value = data.tempsoll;
      }
    </script>
  </head>
  <body>
    <div class="input">
      <div class="label">Temperature Ist:</div>
      <input type="text" id="txt_tempist" readonly="readonly" />
    </div>
    <div class="input">
      <div class="label">Temperature Soll:</div>
      <input type="text" id="txt_tempsoll" />
    </div>
  </body>
</html>
```

Listing E.2: Beispiel JSON-Objekt erstellen

### E.3.3. JSON Value verändern

Listing E.3 ergänzt das Beispiel von Abschnitt E.3.2. Es wird zusätzlich eine Funktion hinzugefügt, welche bei Änderungen im Textfeld "Temperature Soll" aufgerufen wird. Die Änderung wird im JSON-Objekt übernommen und der neue Wert wird in einer Dialogbox dargestellt.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Edit JSON-Value</title>
    <script type="text/javascript">

      var data = {"tempist": "23", "tempsoll": "30"};

      window.onload=function() {
        document.getElementById('txt_tempist').value = data.tempist;
        document.getElementById('txt_tempsoll').value = data.tempsoll;
      }

      function changeTempSoll() {
        data.tempsoll = document.getElementById('txt_tempsoll').value;
        alert( "New Soll Temperature: " + data.tempsoll );
      }
    </script>
  </head>
  <body>
    <div class="input">
      <div class="label">Temperature Ist:</div>
      <input type="text" id="txt_tempist" readonly="readonly" />
    </div>
    <div class="input">
      <div class="label">Temperature Soll:</div>
      <input type="text" id="txt_tempsoll" onchange="changeTempSoll()" />
    </div>
  </body>
</html>
```

Listing E.3: Beispiel JSON-Objekt erstellen

### E.3.4. JSON-Objekt senden und empfangen

Listing E.4 simuliert das Senden und Empfangen eines JSON-Objekts. Das zu sendende JSON-Objekt wird in einem Textfeld dargestellt und kann verändert werden. Um ein JSON-Objekt in eine Textarea einfügen zu können, muss es mit Hilfe von `JSON.stringify()` in ein String umgewandelt werden. Wird eine Änderung im Textfeld vorgenommen, wird die Funktion `send()` aufgerufen. Der String kann unverändert weitergeleitet werden (im Beispiel an die Funktion `receive()`). Dort wird durch Aufruf von `JSON.parse()` der empfangene String wieder in ein JSON-Objekt umgewandelt. In einem Loop werden zum Schluss alle verfügbaren Key-Value Paare in einer Dialogbox dargestellt.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Send/receive JSON-Value</title>
    <script type="text/javascript">

      window.onload=function () {
        var TestData = {"tempist":"23","tempsoll":"30"};
        document.getElementById('txt_send').value = JSON.stringify(TestData);
      }

      function send() {

        var jsonTxt = document.getElementById('txt_send').value;
        receive(jsonTxt);
      }

      function receive(data) {

        var jsonObject = JSON.parse(data);
        for(var key in jsonObject) {
          alert("Key: " + key + " and Value: " + jsonObject[key]);
        }
      }

    </script>
  </head>
  <body>
    <div class="input">
      <div class="label">Zu sendender JSON-String:</div>
      <input type="text" id="txt_send" onchange="send()" />
    </div>
  </body>
</html>
```

Listing E.4: Beispiel JSON-Objekt senden und empfangen

## E.4. JSON und C

### E.4.1. Allgemeines

Damit die ankommenden und ausgehenden Daten von der Steuerung verarbeitet werden können, stehen die im Abschnitt [E.4.2](#) beschriebenen Funktionen zur Verfügung. Es stehen nur Funktionen zur Verfügung um String-Values einzufügen oder auszulesen.

### E.4.2. JSON API

#### createJsonFromBuffer

```
#include "Json.h"
json_t * createJsonFromBuffer(char * pcMsg,
                              uint32_t u32Length);
```

#### Summary

Erzeugt ein neues JSON-Objekt aus einem Charakter-Buffer. Das zurückgegebene JSON-Objekt muss am Ende mit `cleanupJson()` freigegeben werden.

#### Parameters

<code>pcMsg</code>	Der Charakter-Buffer in welchem sich die zu dekodierende JSON-Nachricht befindet.
<code>u32Length</code>	Länge der JSON-Nachricht innerhalb des Buffers.

#### Return Values

Gibt das erzeugte JSON-Objekt zurück oder `NULL` falls ein Fehler aufgetreten ist.

#### createJsonFromString

```
#include "Json.h"
json_t * createJsonFromString(char *pcMsg)
```

#### Summary

Erzeugt ein neues JSON-Objekt aus einem String (Folge von Charakteren welche mit einem `'\0'`-Charakter abgeschlossen sind). Das zurückgegebene JSON-Objekt muss am Ende mit `cleanupJson()` freigegeben werden.

#### Parameters

<code>pcMsg</code>	Der String in welchem sich die zu dekodierende JSON-Nachricht befindet.
--------------------	---

#### Return Values

Gibt das erzeugte JSON-Objekt zurück oder `NULL` falls ein Fehler aufgetreten ist.

### createNewJsonMsg

```
#include "Json.h"
json_t * createNewJsonMsg(void)
```

#### Summary

Erzeugt eine neues leeres JSON-Objekt. Die String-Repräsentation des neu erzeugten Objektes hat die Form: "{}". Das zurückgegebene JSON-Objekt muss am Ende mit `cleanUpJson()` freigegeben werden.

#### Return Values

Gibt das erzeugte JSON-Objekt zurück oder `NULL` falls ein Fehler aufgetreten ist.

### cleanUpJson

```
#include "Json.h"
BBBError cleanUpJson(json_t *pJson);
```

#### Summary

Zurückgeben aller Ressourcen welche durch das JSON-Objekt belegt werden.

#### Parameters

<code>pJson</code>	JSON-Objekt welches freigegeben werden soll.
--------------------	--

#### Return Values

<code>BBB_SUCCESS</code>	Funktion erfolgreich ausgeführt.
<code>BBB_ERR_PARAM</code>	Parameter Fehler.

### getStringRep

```
#include "Json.h"
char * getStringRep(json_t * pJson);
```

#### Summary

Gibt die String-Repräsentation eines JSON-Objektes zurück. Die durch den String belegten Ressourcen, müssen mit `cleanUpStringRep()` freigegeben werden.

#### Parameters

<code>pJson</code>	JSON-Objekt welches enkodiert wird.
--------------------	-------------------------------------

#### Return Values

Die String-Repräsentation des JSON-Objektes oder `NULL` falls ein Fehler aufgetreten ist



## cleanUpStringRep

```
#include "Json.h"
void cleanUpStringRep(char * pcJsonMsg);
```

### Summary

Gibt die Ressourcen zurück welche durch den Aufruf der Funktion `getStringRep()` erzeugt wurden.

### Parameters

<code>pcJsonMsg</code>	Die Ressourcen welche durch <code>getStringRep()</code> erzeugt wurden und freizugeben sind.
------------------------	--

## getJSONStringValue

```
#include "Json.h"
char * getJSONStringValue(json_t * pJson ,
                           char * pcKey);
```

### Summary

Gibt den zum Key dazugehörigen String-Value aus dem JSON-Objekt zurück. Der zurückgegebene Value ist ein String und darf nicht verändert werden.

### Parameters

<code>pJson</code>	JSON-Objekt
<code>pcKey</code>	Key-String

### Return Values

Der zum Key dazugehörige String-Value oder `NULL` falls dieser nicht verfügbar ist.

## setJSONStringKeyValue

```
#include "Json.h"
BBBError setJSONStringKeyValue(json_t * pJson ,
                               char * pcKey ,
                               char * pcValue);
```

### Summary

Setzt innerhalb eines JSON-Objektes ein neues Key-Value Paar. Falls der Key innerhalb des JSON-Objektes bereits vorhanden ist, wird der alte Value mit dem Neuen überschrieben.

### Parameters

<code>pJson</code>	JSON-Objekt
<code>pcKey</code>	Key-String
<code>pcValue</code>	Value-String

## Return Values

BBB_SUCCESS	Funktion erfolgreich ausgeführt.
BBB_ERR_PARAM	Parameter Fehler.
BBB_JSON_PACK	Key-Value Paar konnte nicht eingefügt werden.

## Example

Listing E.5 zeigt ein Beispiel für die Benutzung der JSON-API. Im ersten Teil wird aus einem String ein JSON-Objekt erzeugt und der Value zum Key "Hello" wird ausgegeben. Der zweite Teil erzeugt ein leeres JSON-Objekt und fügt diesem das Key-Value Paar "Hello": "World" hinzu. Anschliessend wird die String-Repräsentation des JSON-Objektes ausgegeben.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#include "Json.h"

int main(int argc, char **argv) {

    char * pcString = "{ \" Hello \": \" World \"}";
    char * pcValue = NULL;
    char * pcMsg = NULL;
    json_t * jsonMsg = NULL;

    // 1.)
    // Create a Json object out of a string
    jsonMsg = createJsonFromString(pcString);
    // jsonMsg = createJsonFromBuffer(pcString, 17); // would achieve the same!
    if(jsonMsg != NULL) {

        // Get the value which corresponds to the key Hello
        pcValue = getJsonStringValue(jsonMsg, "Hello");
        if(pcValue != NULL)
            printf("%s\n", pcValue);

        // Don't forget to free Json object!
        cleanUpJson(jsonMsg);
    }

    // 2.)
    // Create an empty Json object
    jsonMsg = createNewJsonMsg();
    if(jsonMsg != NULL) {

        // Add a key-value pair
        setJsonStringKeyValue(jsonMsg, "Hello", "World");

        // Get the string representation of the Json object
        pcMsg = getStringRep(jsonMsg);
        if(pcMsg != NULL) {
            printf("%s\n", pcMsg);

            // Don't forget to free string resource!
            cleanUpStringRep(pcMsg);
        }
        // Don't forget to free Json object!
        cleanUpJson(jsonMsg);
    }
    return EXIT_SUCCESS;
}
```

Listing E.5: Beispiel zur Benutzung der JSON-API