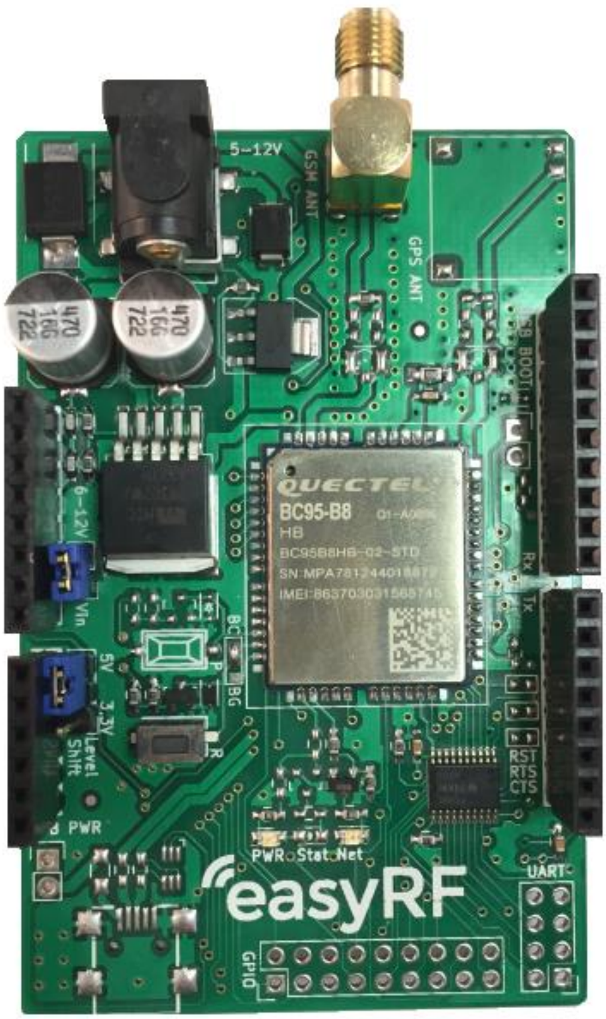
Datasheet

ERF3001

RoHS
COMPLIANT
2002/95/EC

ERF3001, NB-IoT Arduino shield



Document information

Title	**ERF3001**
Subtitle	**NB-IoT Arduino shield**
Document type	**Datasheet**
Document number	**1**
Revision and date	**R1 31-01-2019**

Product status

In production

This document applies to the following products:

Name	Type number	Application version	Product status
ERF3001_v1	**ERF3001**	**1.0**	**In production**
ERF3101_v1	**ERF3101**	**1.0**	**In production**

ERF3001, NB-IoT Arduino shield

Contents

1 Functional description

1.1 Product overview

The ERF3001 Arduino shield with Quectel BC95 module is a high-performance NB-IoT module with low power consumption. The shield can be programmed through a serial connection on the Arduino UNO board. The shield uses Embedded Internet Service protocols that extend the applicability of the module to a wide range of IoT applications, such as smart metering, smart city, security and remote sensing, agricultural and many more.

The ERF3001 combines the power of the Quectel BC95 module with an easy to use Arduino platform and allows for rapid testing and development.

The easyRF Arduino shield is also available with the Quectel GSM/GPRS module M95 and the LTE Cat. M1/Cat.NB1/EGPRS module BG96.

The ERF3001 is also available in combination with an Arduino. for more information please see: Ordering information

1.2 Product features

General features		
Frequency band	BC95-B8	900 MHz
	BC95-B5	850 MHz
	BC95-B20	800 MHz
Supply voltage range		5.0V~12.0V
Operation temperature		-40°C~+85°C
PCB dimensions		83.5*55.3 mm (± 0.1)
Weight		30 g
AT command		3GPP REL-13 enhanced AT commands
Programming		Through Arduino Serial communication to Quectel BC95

Specifications	
Data transmission	100bps< bit rate< 100kbps (TBC)
Protocol stack	IPV4/IPV6 , UDP/CoAP
Protocols	Point-to-point MO and MT Text/PDU Mode

Interfaces	
USIM	1x
UART	2x
ADC	1x
RESET	1x
STATUS	-
NETLIGHT	Network Status Indication

Electrical characteristics (BC95 module)	
Output power	23dBm
Sensitivity	-129dBm
Power consumption	Sleep: 5 µA Idle: 6 mA

2 Interfaces

2.1 Power

Module supply input

The ERF3001 shield must be supplied through the VCC pins by a DC power supply with nominal voltage of 5V to 12V (by shield barreljack) or 6V to 12V (by Arduino barreljack). The DC power adapter must be capable of supplying a minimum of 2A to the shield. Voltage must be stable during module operation, taking into account that the current drawn from VCC pins may vary significantly based on the power consumption profile of the NB-IoT system.

⚠ The module can also be powered from the Arduino(via USB cable) however, the module could require more power than the Arduino can supply. In this process the Arduino can be damaged. Because of this, this power option is not recommended.

Digital I/O interfaces supply output (V_INT)

The ERF3001 shield provides an internally generated supply rail output (V_EXT) operating at 3.0V. This can be used in place of an external discrete regulator to supply external digital interfaces (max. 20mA). The voltage level present at the V_INT pin depends on the module operating mode:

- in the deep-sleep mode the voltage level is kept “Low” (i.e. 0V)

- in active and connected mode the voltage level is maintained “High” (i.e. 3.0V)

2.2 Antennas

The ERF3001 uses 1 antenna , a NB-IoT antenna capable of covering 800MHz to 900MHz. The antenna needs to be connected to the SMA connector.

Take care when connecting the shield to the Arduino. the SMA connector pins can’t touch the USB connector on the Arduino!

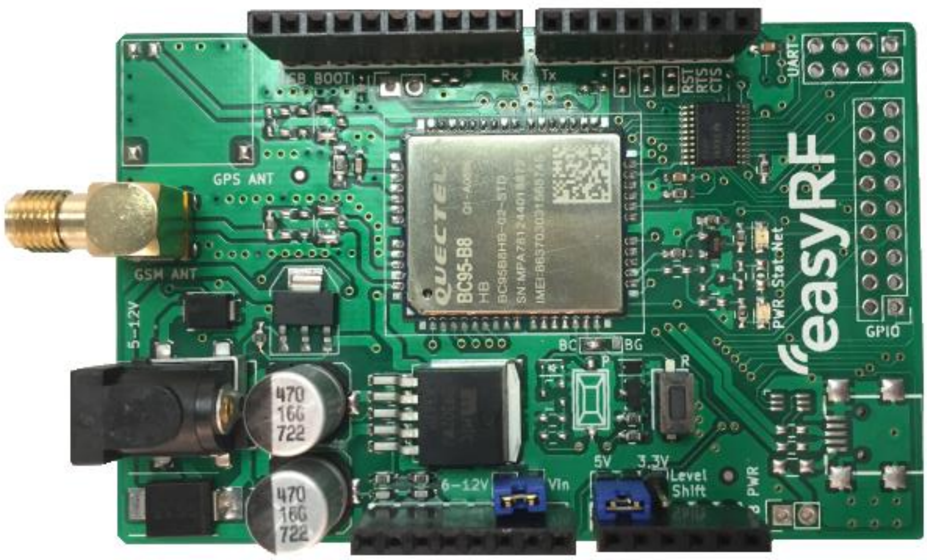
2.2.1 GSM_ANT (ERF4041 GSM antenna)

The ERF4041 antenna is a black GSM antenna with a SMA male connector. The antenna can be used in the frequency band of 800~900 / 1640 / 2200~2230 / 3410MHz. The antenna can be used for applications, GSM, NB-IoT, M2M and more.

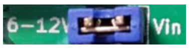
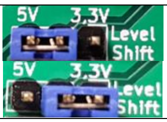
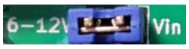
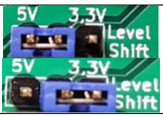
For more information see ERF4041 datasheet

2.3 System function

The shield has 2 important power jumpers, the PWR jumper and the level shift jumper. The barrel jack on the shield only powers the BC95 (not the Arduino). The recommended power option is using the shield in combination with an Arduino with the DC power adapter connected to the Arduino.



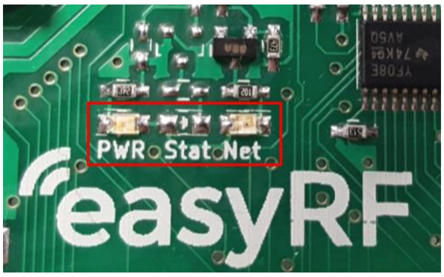
There are 2 ways to power the shield:

Power option	Hardware	Connector	Jumper	Jumper position	Description
1	Shield + Arduino	Via Arduino DC-Jack	Vin		Jumper Vin closed, Jumper Level Shift on 3.3V or 5V (depending on voltage of Arduino)
			Level Shift	 	
2	Shield + Arduino	Via shield DC-Jack	Vin		Jumper Vin closed, Jumper Level Shift on 3.3V or 5V (depending on voltage of Arduino)
			Level Shift	 	

2.4 Status LED's

The shield also has 2 LED's to give an indication of the status of the shield.

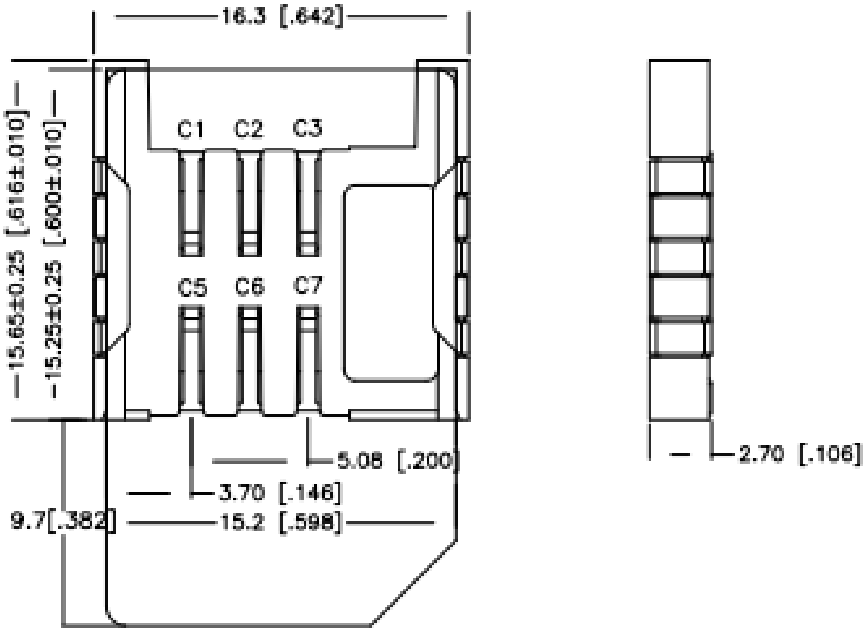
Led	Color	Status	Description
PWR	Green	Always on	Shield is powered
		Always off	Shield is not powered
Stat	-	-	-
Net	Yellow	Always on	Module synchronized with network
		Always off	Module not working or not connected to network

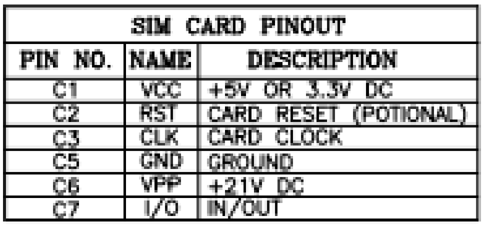


2.5 SIM interface

To register and connect to a network a SIM card needs to be inserted and activated.
When an activated SIM is inserted, the correct APN must be set using AT commands*.
The Attend model number of the SIM card socket is 115C-AC00-R.

*For more information see: Quectel_BC95_AT_Commands_Manual_V1.5.pdf





SIM CARD PINOUT

PIN NO.	NAME	DESCRIPTION
C1	VCC	+5V OR 3.3V DC
C2	RST	CARD RESET (POTIONAL)
C3	CLK	CARD CLOCK
C5	GND	GROUND
C6	VPP	+21V DC
C7	I/O	IN/OUT

2.6 UART interface

The ERF3001 shield provides 1 UART port: the main port. The BC95 module is designed as a DCE (Data Communication Equipment), following the traditional DCE-DTE (Data Terminal Equipment) connection.

The main port:

- TXD: send data to RXD of DTE
- RXD: receive data from TXD of DTE
- RI : ring indicator (when an SMS is received or data transmitted, the module will output signals to inform DTE)
- The baud rate of the main port is 9600 bps

2.7 Programming the shield

To program the device an USB cable needs to be connected to the Arduino with shield attached, and a DC power adapter needs to be connected to either the Arduino or the shield. Also a NB-IoT SIM card needs to be inserted in the SIM socket on the shield and the antenna needs to be connected.

The AT-Manuals for the bc95 module can be downloaded here:
https://www.quectel.com/ProductDownload/BC95.html

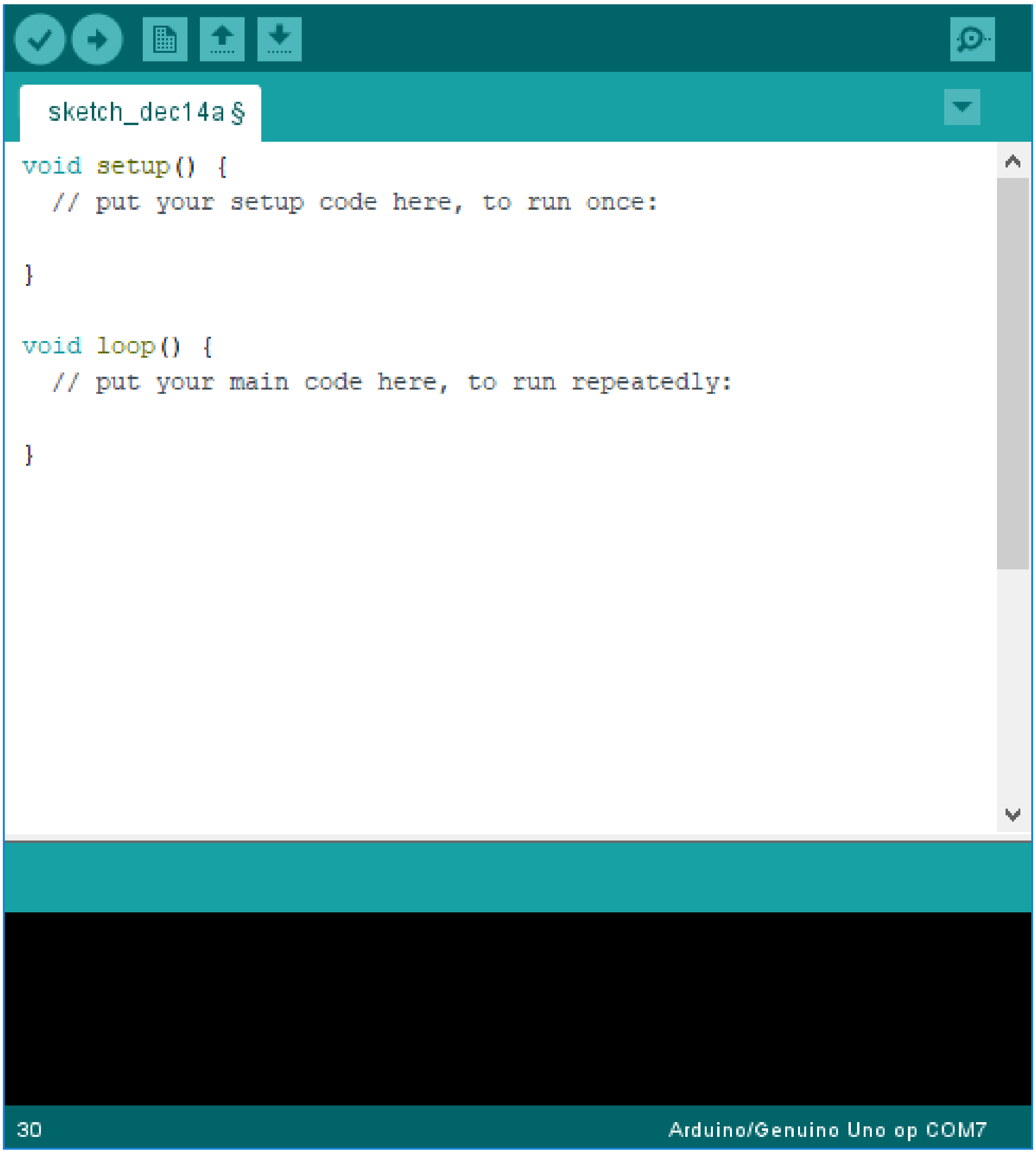
Once this is done, download the Arduino IDE software on your PC. This can be found here: https://www.arduino.cc/en/Main/Software. To communicate with the shield a standard Arduino library called “SoftwareSerial” is needed. This is already installed with the Arduino IDE.

Information can be found here: https://www.arduino.cc/en/Reference/SoftwareSerial.

Before starting to program the shield, please check with your local provider if IMEI registration of the Quectel hardware is necessary. If you don’t register the IMEI number with you provider and still connect to their network you may be put on a blacklist, and access to the network will be blocked. The IMEI number can be found on the metal casing of the Quectel module.

Once Arduino IDE has finished installing open, the software.

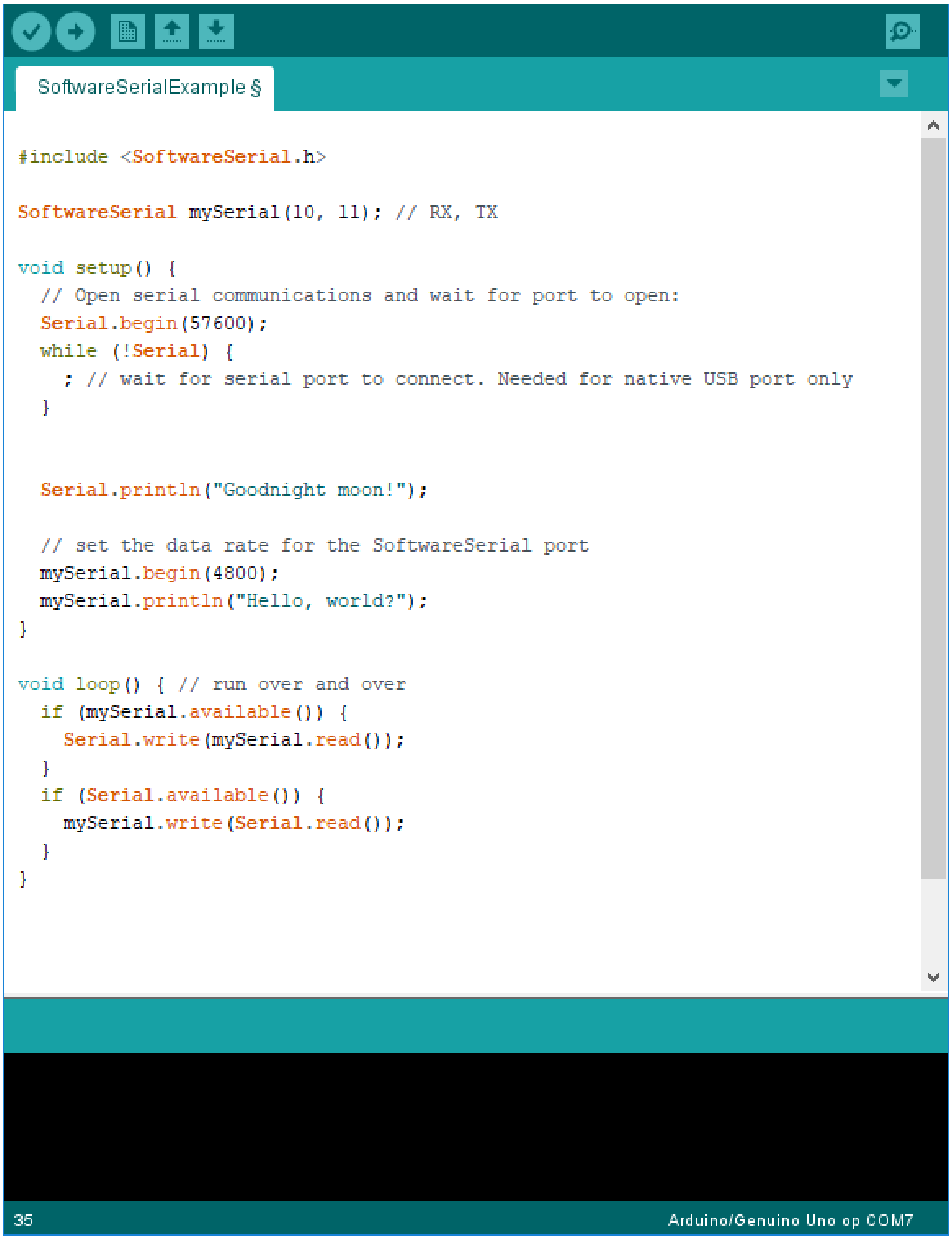
The following image will appear:



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

From here go to : File->examples and under Examples for Arduino/Genuino Uno select SoftwareSerialExample.

The following file opens:

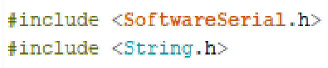


```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.println("Goodnight moon!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop() { // run over and over
  if (mySerial.available()) {
    Serial.write(mySerial.read());
  }
  if (Serial.available()) {
    mySerial.write(Serial.read());
  }
}
```

A few alterations in this code are needed to setup a communication with the ERF3001 shield.
Starting with adding an additional library called “String.h”
This is done by adding the following line of text at the top of the file:

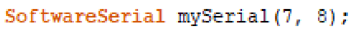
```
#include <String.h>
```

So the top of the file now looks like this:



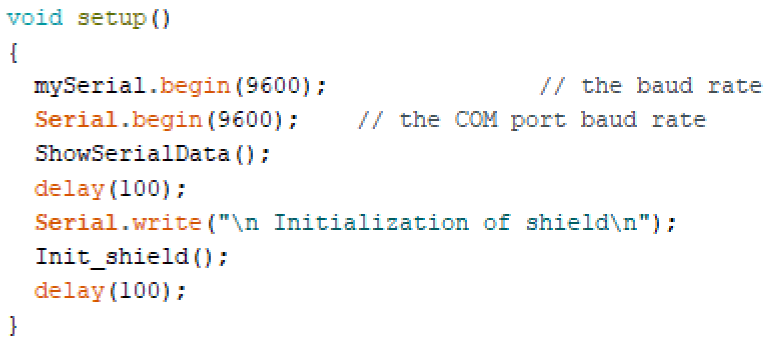
```
#include <SoftwareSerial.h>
#include <String.h>
```

Also, the correct RX and TX pin must be set. This is done by changing pins 10 and 11 to 7 and 8.

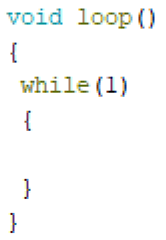


```
SoftwareSerial mySerial(7, 8);
```

Now change the void setup() and the void loop() to the following code :



```
void setup()
{
  mySerial.begin(9600);              // the baud rate
  Serial.begin(9600);    // the COM port baud rate
  ShowSerialData();
  delay(100);
  Serial.write("\n Initialization of shield\n");
  Init_shield();
  delay(100);
}
```



```
void loop()
{
 while(1)
 {

 }
}
```

Now that the setup() and loop() are set we need to add some functions to send AT commands to the shield.
The functions are: void ShowSerialData() and void Init_shield(void).

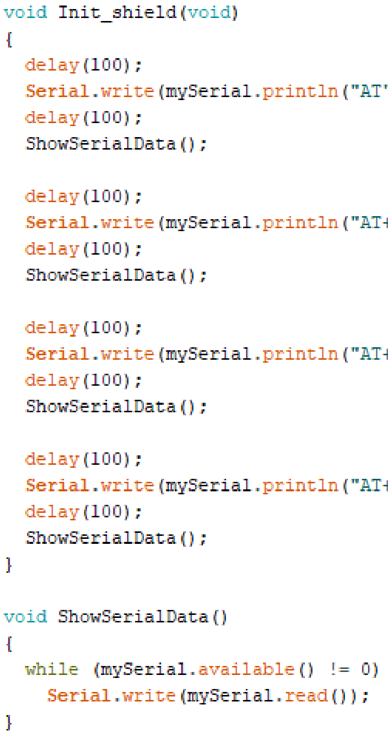
void Init_shield(void) is a function that consists of 4 AT commands:

Command	Description
AT	This returns OK if the command is received correct
AT+CGMI	This returns the manufacturer ID
AT+CGMM	This returns the manufacturer model
AT+CGMR	This returns the manufacturer revision

If all commands return the correct information, the connection between the Arduino and the shield is correct.

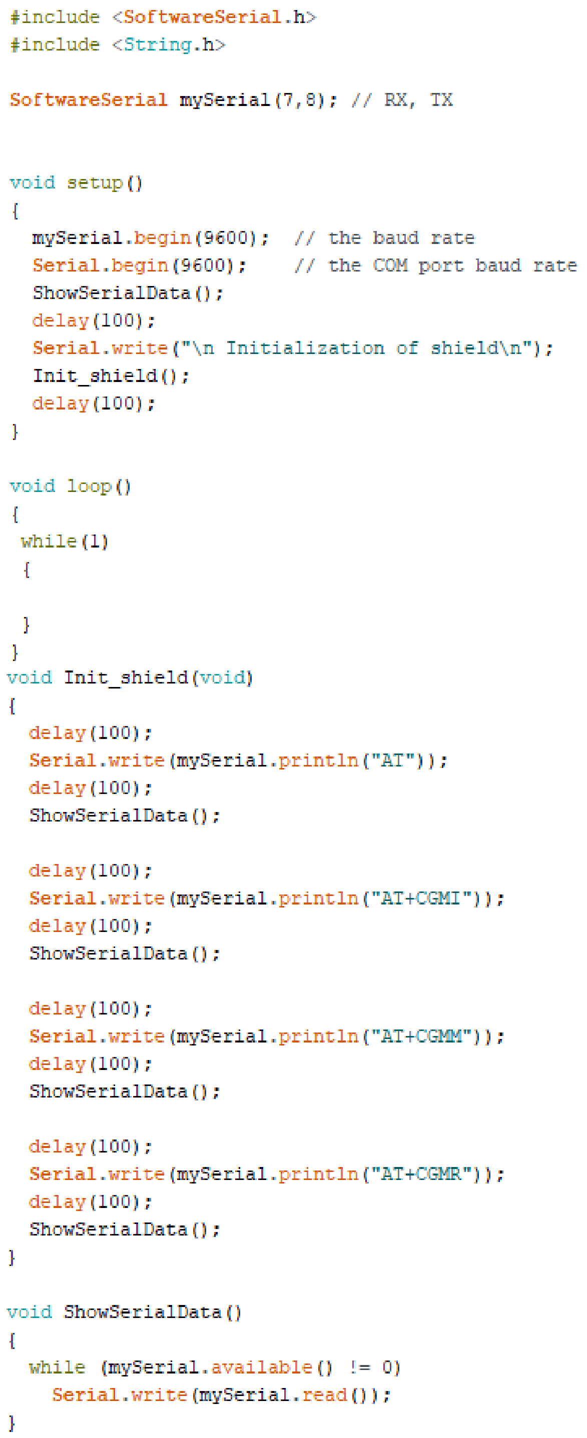
void ShowSerialData() is a function that processes the response of the shield and sends the data to the serial COM port.

These functions need to be added outside of the void setup() and void loop().



```
void Init_shield(void)
{
  delay(100);
  Serial.write(mySerial.println("AT"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMI"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMM"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMR"));
  delay(100);
  ShowSerialData();
}

void ShowSerialData()
{
  while (mySerial.available() != 0)
    Serial.write(mySerial.read());
}
```

If all steps are followed correctly, the end code looks like this:



```
#include <SoftwareSerial.h>
#include <String.h>

SoftwareSerial mySerial(7,8); // RX, TX


void setup()
{
  mySerial.begin(9600);  // the baud rate
  Serial.begin(9600);    // the COM port baud rate
  ShowSerialData();
  delay(100);
  Serial.write("\n Initialization of shield\n");
  Init_shield();
  delay(100);
}

void loop()
{
 while(1)
 {

 }
}
void Init_shield(void)
{
  delay(100);
  Serial.write(mySerial.println("AT"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMI"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMM"));
  delay(100);
  ShowSerialData();

  delay(100);
  Serial.write(mySerial.println("AT+CGMR"));
  delay(100);
  ShowSerialData();
}

void ShowSerialData()
{
  while (mySerial.available() != 0)
    Serial.write(mySerial.read());
}
```

Now that the code is done, select the right COM port and board. This can be done by selecting: Tools->Board, and Tools->Port. The board should be set on : Arduino/ Genuino Uno.

The COM port should be the port the Arduino is connected to. This can be found in the device management on your PC. Under ports look for the port USB-SERIAL CH340 is connected to. This port should be selected in the Arduino software.

The code can be verified and uploaded. This is done by clicking: to ✓ verify the code. Wait for the verification to complete.
If any errors occur during verification please check the code.

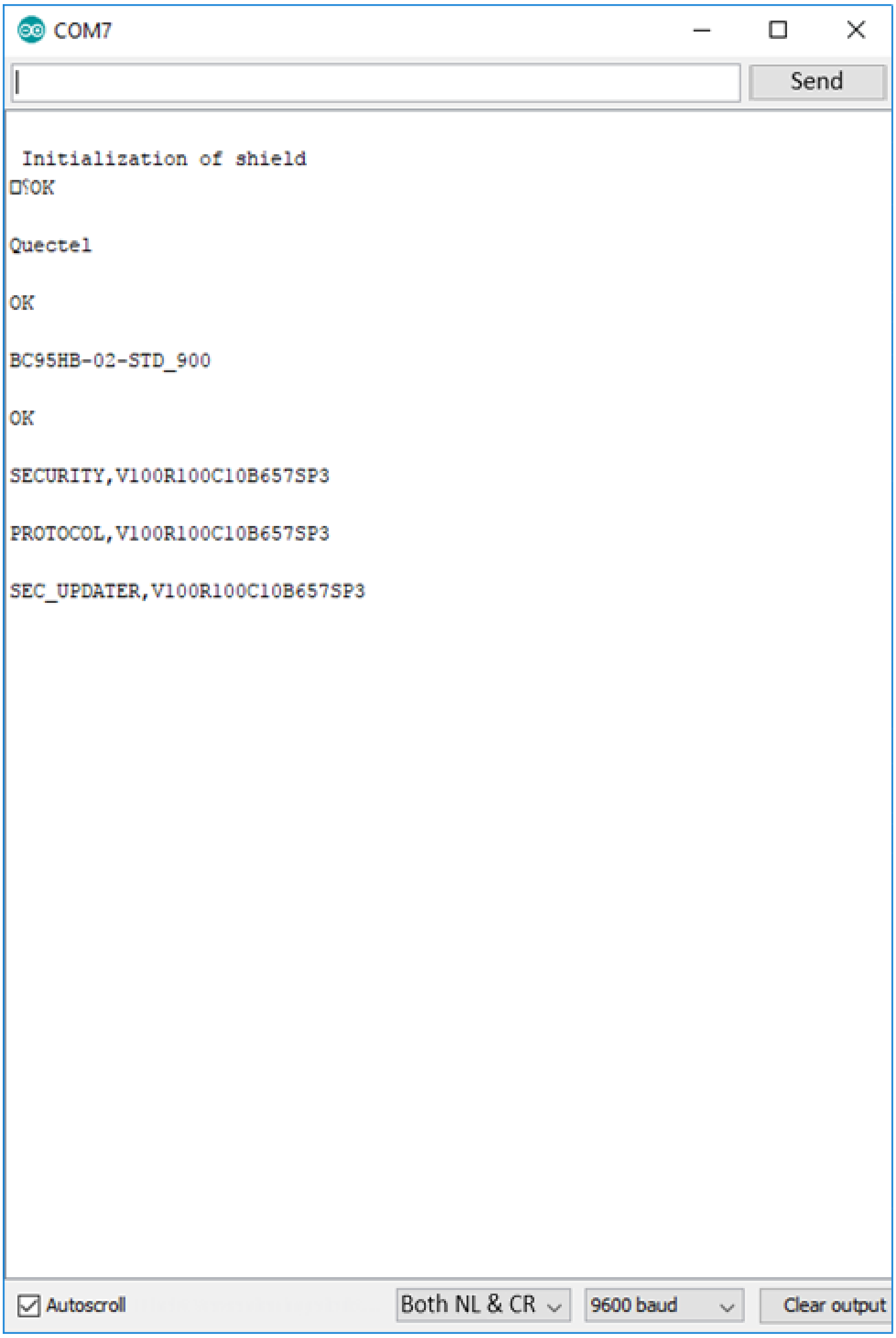
If there are no errors click on: → to start uploading the code to the Arduino board.

Uploading the software may take a while. again no errors should show up during uploading. If there are any check the connection with the Arduino, and try reuploading.

When the code is done uploading open the serial monitor by clicking : 🔍

Select 9600 baud In the serial monitor.

On the serial port the following image should show:



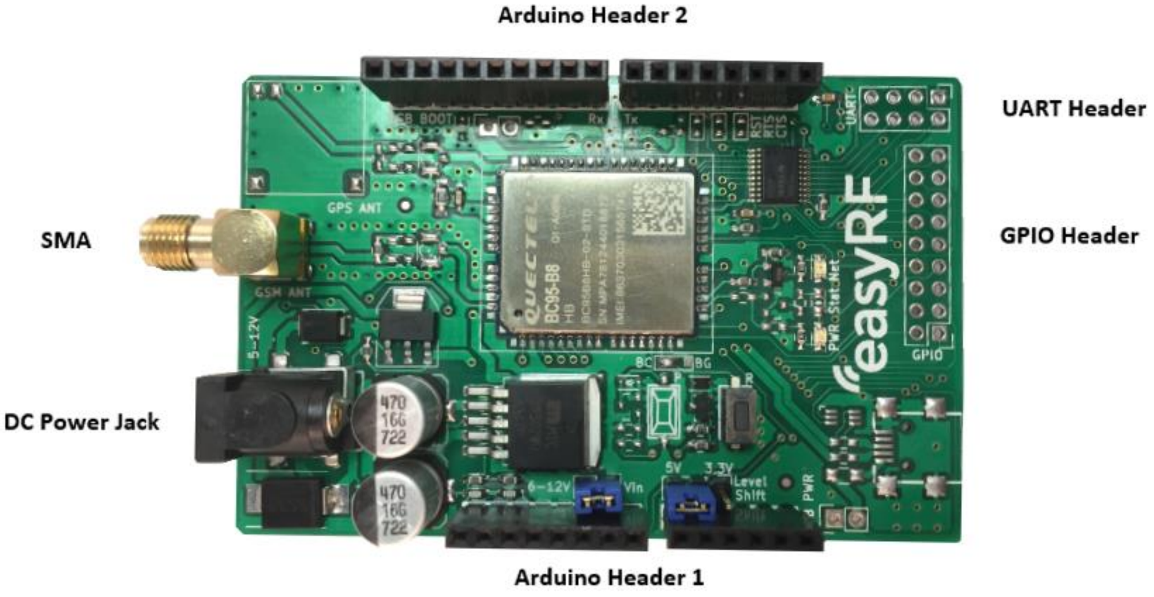
```
 Initialization of shield
□!OK

Quectel

OK

BC95HB-02-STD_900

OK

SECURITY,V100R100C10B657SP3

PROTOCOL,V100R100C10B657SP3

SEC_UPDATER,V100R100C10B657SP3
```

2.8 GPIO

The ERF3001 shield has multiple GPIO pins. The GPIO header* has all the GPIO pins from the BC95 module. The Arduino headers 1 and 2* have the normal Arduino GPIO pins, if the module is used in combination with an Arduino. This allows the possibility to add extra functionality to the shield and integration in your application.

*the GPIO and Arduino header pinout can be found in chapter 3 Pin Assignment

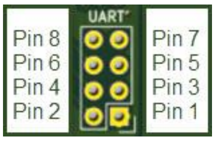
3 Pin assignment



I/O Parameter definition

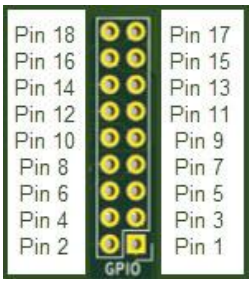
Type	Description
IO	Bidirectional
DI	Digital input
DO	Digital output
PI	Power input
PO	Power output
AI	Analog input
AO	Analog output
OD	Open drain

3.1 UART header



Header	Pin No.	Name	Power Domain	I/O	Description
UART	1	GND	GND	-	Ground
UART	2	RI	3.0V	DO	Ring indicator
UART	3	Reserved	-	-	Reserved (keep unconnected)
UART	4	Reserved	-	-	Reserved (keep unconnected)
UART	5	Reserved	-	-	Reserved (keep unconnected)
UART	6	Reserved	-	-	Reserved (keep unconnected)
UART	7	TXD	3.0V	DO	Transmit data
UART	8	RXD	3.0V	DI	Receive data

3.2 GPIO header



Header	Pin No.	Name	Power Domain	I/O	Description
GPIO	1	Reserved	-	-	Reserved (keep unconnected)
GPIO	2	Reserved	-	-	Reserved (keep unconnected)
GPIO	3	Reserved	-	-	Reserved (keep unconnected)
GPIO	4	Reserved	-	-	Reserved (keep unconnected)
GPIO	5	Reserved	-	-	Reserved (keep unconnected)
GPIO	6	NC	-	-	-
GPIO	7	NC	-	-	-
GPIO	8	NC	-	-	-
GPIO	9	Reserved	-	-	Reserved (keep unconnected)
GPIO	10	Reserved	-	-	Reserved (keep unconnected)
GPIO	11	Reserved	-	-	Reserved (keep unconnected)
GPIO	12	Reserved	-	-	Reserved (keep unconnected)
GPIO	13	Reserved	-	-	Reserved (keep unconnected)
GPIO	14	Reserved	-	-	Reserved (keep unconnected)
GPIO	15	Reserved	-	-	Reserved (keep unconnected)
GPIO	16	ADC	Input voltage range: 0V to VBAT	AI	General purpose analog to digital converter
GPIO	17	Reserved	-	-	Reserved (keep unconnected)
GPIO	18	Reserved	-	-	Reserved (keep unconnected)

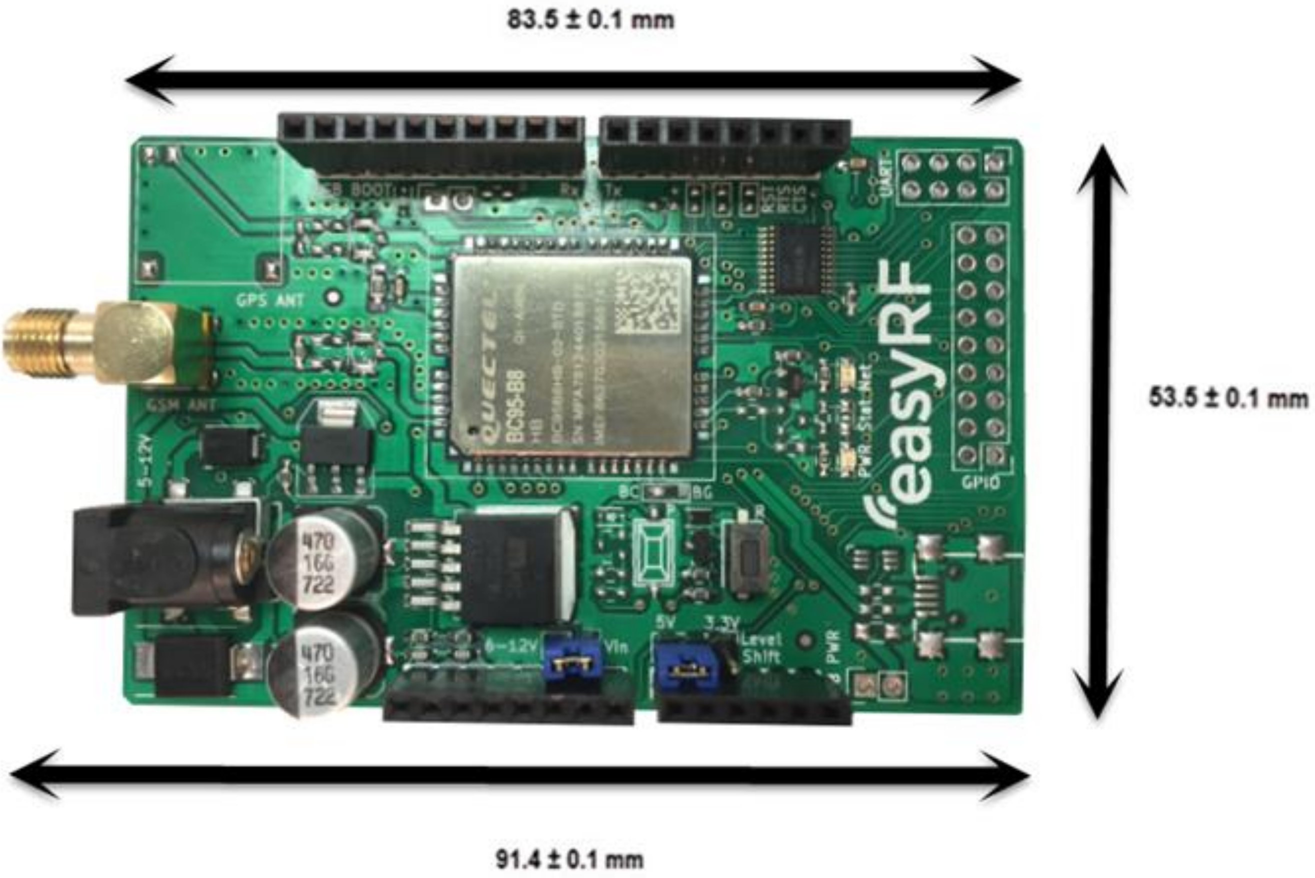
3.3 Arduino header 1 & 2

Header	Pin No.	Name	Power Domain	I/O	Description
Arduino header 1	1	N.C.	-	-	Not connected
Arduino header 1	2	IOREF	5 V	PI	
Arduino header 1	3	RESET	5V		Reset
Arduino header 1	4	3.3V	3.3V	PO	3.3V pin on Arduino
Arduino header 1	5	5V	5V	PO	5V pin on Arduino
Arduino header 1	6	GND	GND	-	Ground
Arduino header 1	7	GND	GND	-	Ground
Arduino header 1	8	VIN	5V	PI	Input voltage
Arduino header 1	9	AD0	5V	AI	Analog input
Arduino header 1	10	AD1	5V	AI	Analog input
Arduino header 1	11	AD2	5V	AI	Analog input
Arduino header 1	12	AD3	5V	AI	Analog input
Arduino header 1	13	AD4/SDA	5V	AI/IO	Analog input / I2C interface data
Arduino header 1	14	AD5/SCL	5V	AI/AO	Analog input / I2C interface clock
Arduino header 2	15	RXD/IO0	5V	DI/IO	TXD
Arduino header 2	16	TXD/IO1	5V	DO/IO	RXD
Arduino header 2	17	INT0/IO2	5V	DI/IO	External interrupt / Digital I/O
Arduino header 2	18	INT1/IO3	5V	DI/IO	External interrupt / Digital I/O
Arduino header 2	19	T0/IO4	5V	DO/IO	Timer 0 / Digital I/O
Arduino header 2	20	T1/PWM/IO5	5V	DO/DO/IO	Timer 1 / PWM / Digital I/O
Arduino header 2	21	AIN0/PWM/IO6	5V	AI/DO/IO	Analog input / PWM/ Digital I/O
Arduino header 2	22	AIN1/IO7	5V	AI/IO	Analog input / Digital I/O
Arduino header 2	23	ICP/IO8	5V	DI/IO	TC1 input compare match / Digital I/O
Arduino header 2	24	OC1/PWM/IO9	5V	DO/DO/IO	TC1 output compare match / PWM / Digital I/O
Arduino header 2	25	SS/PWM/IO10	5V	DO/DO/IO	SPI slave select / PWM / Digital I/O
Arduino header 2	26	MOSI/PWM/IO11	5V	DO/DO/IO	SPI master out slave in / PWM / Digital I/O
Arduino header 2	27	MISO/IO12	5V	DI/IO	SPI master in slave out / Digital I/O
Arduino header 2	28	SCK/IO13	5V	DO/IO	SPI clock / Digital I/O
Arduino header 2	29	GND	GND	-	Ground
Arduino header 2	30	AREF	5V	PI	Analog reference for the A/D converter
Arduino header 2	31	AD4/SDA	5V	AI/IO	Analog input / I2C interface data
Arduino header 2	32	AD5/SCL	5V	AI/AO	Analog input / I2C interface clock

4 Electrical specification

A power supply with 5V ~ 12V DC output with a minimum of 2A is needed to power the shield.
12V is the absolute max, do not exceed this.
The power must be supplied to the DC jack located on the shield or to the Arduino DC jack.
Please see chapter: 2.3 for the power options.

5 Mechanical specifications



Hardware	Height
Shield + Arduino	27.5 ±0.1 mm
Shield wit header pins	23.0 ±0.1 mm
Shield without header pins	12.0 ± 0.1 mm

6 Product handling

Handle with ESD safety care.

Related documents

Document
ERF4041 Datasheet.pdf
Quectel_BC95_AT_Commands_Manual_V1.5.pdf

About easyRF

easyRF is supplier and manufacturer of wireless communication solutions with an easy-to-use approach, targeting different applications and markets. The products are standard off-the-shelf products, but customization of the products is possible.

easyRF is successful in the a wide range of markets, such as: industrial, agriculture, security, building automation.

Ordering information

Please check www.easyRF.eu for distributors in your area or contact us at info@easyRF.eu for more information.

The shield is available as the following package:

Package	ERF3001	ERF3101	ERF3000	ERF3100
Includes	- BG96 Arduino shield - GSM antenna - Power adapter	- BG96 Arduino shield - Arduino UNO - GSM antenna - USB cable - Power adapter	- BC95 Arduino shield - GSM antenna	- BC95 Arduino shield - Arduino UNO - GSM antenna

Technical support

For all product related questions please contact us via info@easyRF.eu