GPRS Shield

Introduction

The GPRS Shield provides you a way to use the GSM cell phone network to receive data from a remote location. The shield allows you to achieve this via any of the three methods:

• Short Message Service

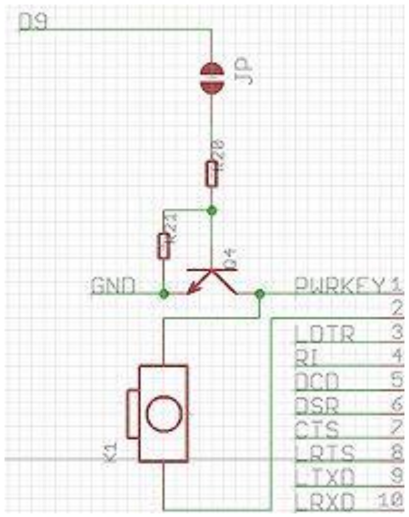
• Audio

• GPRS

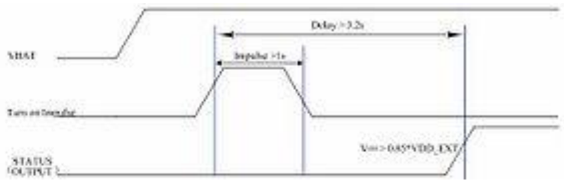
The GPRS Shield is compatible with all boards which have the same form factor (and pinout) as a standard Arduino Board. The GPRS Shield is configured and controlled via its UART using simple AT commands. Based on the SIM900 module from SIMCOM, the GPRS Shield is like a cell phone sans the Human Machine Interface. Besides the communications features, the GPRS Shield has 12 GPIOs, 2 PWMs and an ADC

What's New From Ver1.0 to Ver1.2

The "Soft-Switch" was added from GPRS Shield Ver1.0 to Ver1.2:



You can open / close the shield by pressing the switch button or in your firmware.



As the timing of turn on/off , a >1s pulse was need to trigger the truning, and a >3.2s delay was need to get the timing stable. add the following code in your firmware to turn on / off the shield without pressing the button:

```
pinMode(9, OUTPUT);
 digitalWrite(9,LOW);
 delay(1000);
 digitalWrite(9,HIGH);
 delay(2500);
 digitalWrite(9,LOW);
 delay(3500);
```

Features:

• Based on SIMCom's SIM900 Module

• Quad-Band 850 / 900/ 1800 / 1900 MHz - would work on GSM networks in all countries across the world.

• GPRS multi-slot class 10/8

• GPRS mobile station class B

• Compliant to GSM phase 2/2+

• Class 4 (2 W @ 850 / 900 MHz)

• Class 1 (1 W @ 1800 / 1900MHz)

• Control via AT commands - Standard Commands: GSM 07.07 & 07.05 | Enhanced Commands: • SIMCOM AT Commands.

• Short Message Service - so that you can send small amounts of data over the network (ASCII or raw hexadecimal).

• Embedded TCP/UDP stack - allows you to upload data to a web server.

• Speaker and Headphone jacks - so that you can send DTMF signals or play recording like an answering machine.

• SIM Card holder and GSM Antenna - present onboard.

• 12 GPIOs, 2 PWMs and an ADC (all 2.8 volt logic) - to augment your Arduino.

• Low power consumption - 1.5mA(sleep mode)

• Industrial Temperature Range - -40°C to +85 °C

Application Ideas:

• M2M (Machine 2 Machine) Applicatoions - To transfer control data using SMS or GPRS between two machines located at two different factories.

• Remote control of appliances - Send SMS while you are at your office to turn on or off your washing machine at home.

• Remote Weather station or a Wireless Sensor Network - Mate it with Seeeduino Stalker and create a sensor node capable of transferring sensor data (like from a weather station - temperature, humidity etc.) to a web server (like pachube.com).

• Interactive Voice Response System - Couple the GPRS Shield with an MP3 Decoder and DTMF Decoder (besides an Arduino) to create an Interactive Vocice Response System (IVRS).

• Vehicle Tracking System - Couple the GPRS Shield with an Arduino and GPS module and install it in your car and publish your location live on the internet. Can be used as a automotive burglar alarm.

Cautions

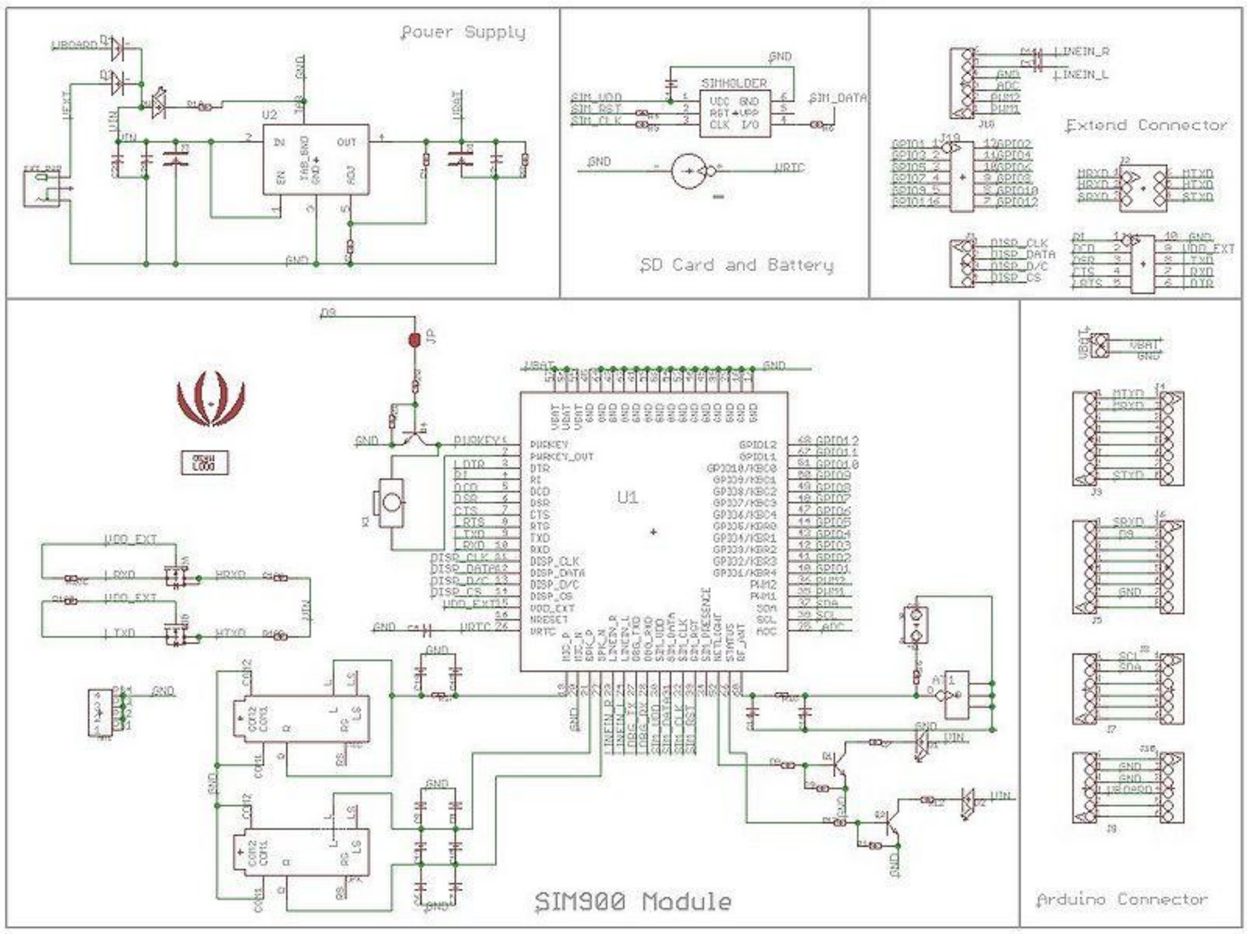
• Make sure your SIM card is unlocked.

• The product is provided as is without an insulating enclosure. Please observe ESD precautions specially in dry (low humidity) weather.

• The factory default setting for the GPRS Shield UART is 19200 bps 8-N-1. (Can be changed using AT commands).

• When using GPRS Shield with Seeeduino Stalker v2.0 please remember to dismount the OK_READ Jumper (i.e. open it). This will disconnect the Battery Charger IC's OK pin from the microcontrollers Digital Pin 7 and hence allow unhindered communication with GPRS Shield using NewSoftSerial Library.

Schematics



Usage

LED	State	Function
Status	Off	Power Off
	On	Power On
Netlight	Off	SIM900 is not working
	64ms On/800ms Off	SIM900 does not find the network
	64ms On/3000ms Off	SIM900 find the network
	64ms On/300ms Off	GPRS communication

Getting Started - Fun with AT Commands

The GPRSShield comes with all the accessories that you would need to get started with sending data over the GSM network except for of course an Arduino board and a GSM SIM Card with a data plan subscription active on it. If you want to make voice calls, you would also require a headset with microphone.

Step 1: Creating a test setup for the GPRS Shield

So you received your GPRS Shield, what would be the first thing you would want to do with it? Send out a text (SMS) to your cell phone? or call up someone (headset required)? You can do all this by talking to the GPRS Shield using AT Commands - which is a special language that it understands. AT Commands are simple textual commands sent to the GPRS modem over its serial interface (UART), so you can pretty much use any serial terminal software to communicate with it - Window's built-in HyperTerminal, Arduino IDE's Serial Monitor or Bray++ Terminal.

To experiment with AT commands, you would require a way to power up and communicate with your GPRS Shield. The best way to do this using an Arduino Duemilanove board is described below.

The same steps are applicable for Seeeduino or Seeeduino Stalker.

1. Install the SIM card in your GPRS Shield. You only require a data plan active on it if you want to use GPRS. If you are only looking to send text messages (SMSes) or make voice calls then it is not required.

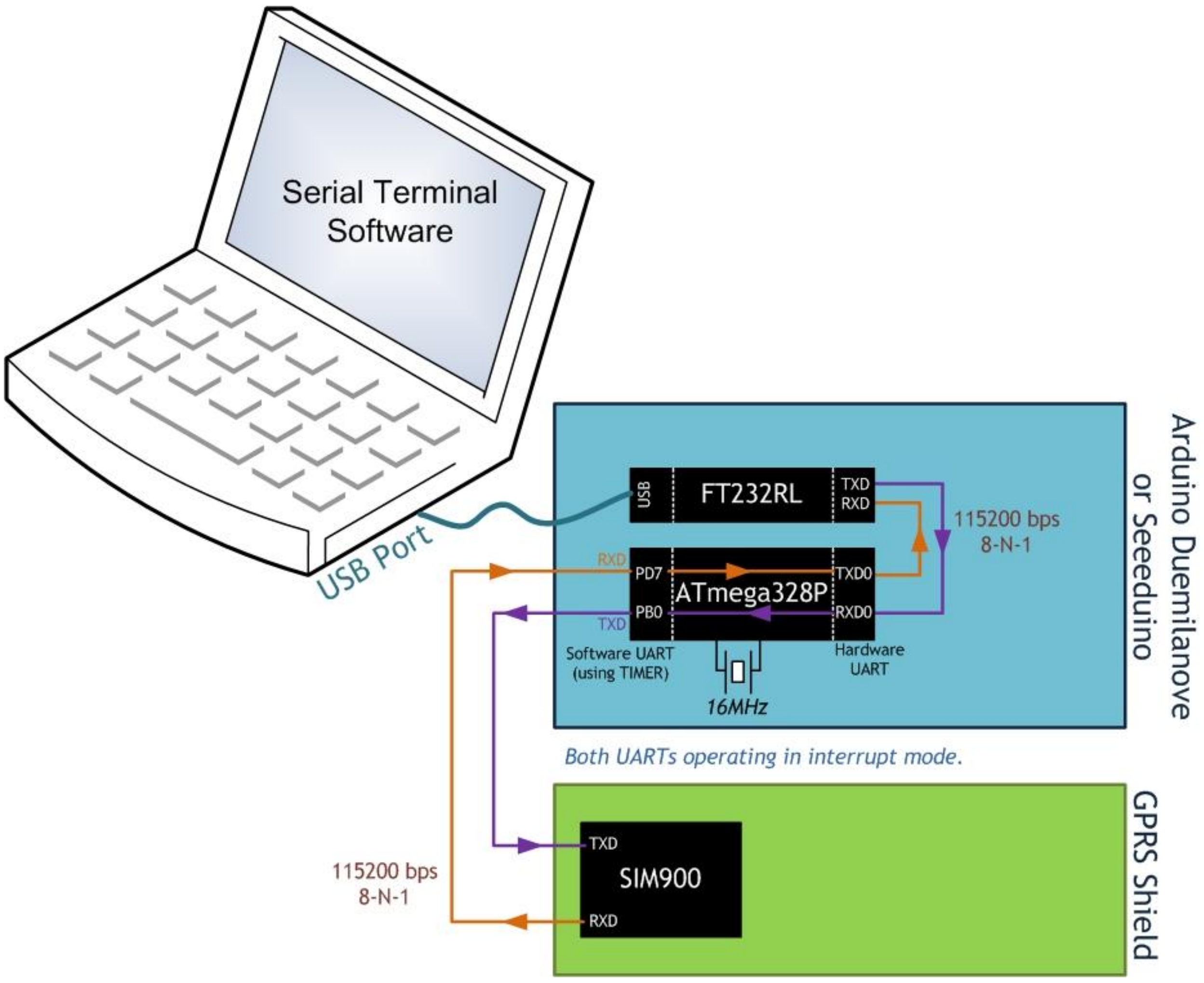
2. Connect the antenna to the GPRS Shield. Refer the Connection Notes below.

3. Take an Arduino Duemilanove (or Seeeduino) board and install the GPRS Shield over it.

4. Make sure the GPRS_TX & GPRS_RX jumpers on the GPRS Shield are mounted in SWSerial position - that is we want GPRS_TX to be connected to D7(RX) and GPRS_RX to D8(TX)

5. Connect the Arduino Duemilanove (or Seeeduino) to your computer using a USB cable.

6. The ATmega328P microcontroller on board Duemilanove has only one UART which is used for communicating with the PC. What we need is an Arduino Sketch running inside the ATmega328P that would emulate a second serial port (UART) using software on the digital pins D8 and D7 and patch through all the communication between this second software serial port and the actual hardware serial port. By doing this, all the data coming from the computer (connected to the actual hardware UART) would be relayed as is to the GPRS Shield (connected to software UART) and we would be able to issue AT commands to control the GPRS Shield. The block diagram outlining this scheme is shown below.



For developing such a program, we require installation of a new Arduino library - NewSoftSerial. This uses the on-chip TIMER of ATmega328P in interrupt mode to emulate a second serial port. The library can be obtained from here. It would be a ZIP file having a name like NewSoftSerial10c.zip. Extract the folder NewSoftSerial from within this .zip file and install it by placing it in your Arduino Installation folder which would be something like "C:\arduino-0022\libraries". Make sure to restart the Arduino IDE after the library.

7. Once the library is installed, create a new sketch with the following code and download it into your Arduino board.

```
//Serial Relay - Arduino will patch a

//serial link between the computer and the GPRS Shield

//at 19200 bps 8-N-1

//Computer is connected to Hardware UART

//GPRS Shield is connected to the Software UART


#include <NewSoftSerial.h>


NewSoftSerial mySerial(7, 8);


void setup()

{

  mySerial.begin(19200);              // the GPRS baud rate

  Serial.begin(19200);                // the GPRS baud rate

}


void loop()

{

    if(Serial.available())

    {

       mySerial.print((unsigned char)Serial.read());

     }

    else  if(mySerial.available())

    {

       Serial.print((unsigned char)mySerial.read());
    }


}
```

8. After uploading the sketch to the Arduino board, press the power button on the GPRS Shield to turn it on. Wait half a minute for the GPRS Shield to conect to the network (Led D1 will start blinking).

9. Fire up your favorite serial terminal software, choose the COM port for Arduino, set it to operate at 19200 8-N-1 and type and send "AT" (without the quotes) followed by carriage return (enter key) to the Arduino board. The GPRS Shield should respond by sending back an "OK". This would mean that you have been able to successfully setup your GPRS Shield can can now play around with various AT Commands.

Step 2: Sending a text message (SMS)

Now that our test setup is ready, let's play around with some AT Commands before moving on to programming the Arduino to do it by itself instead of doing it manually. To start off let's try sending an SMS

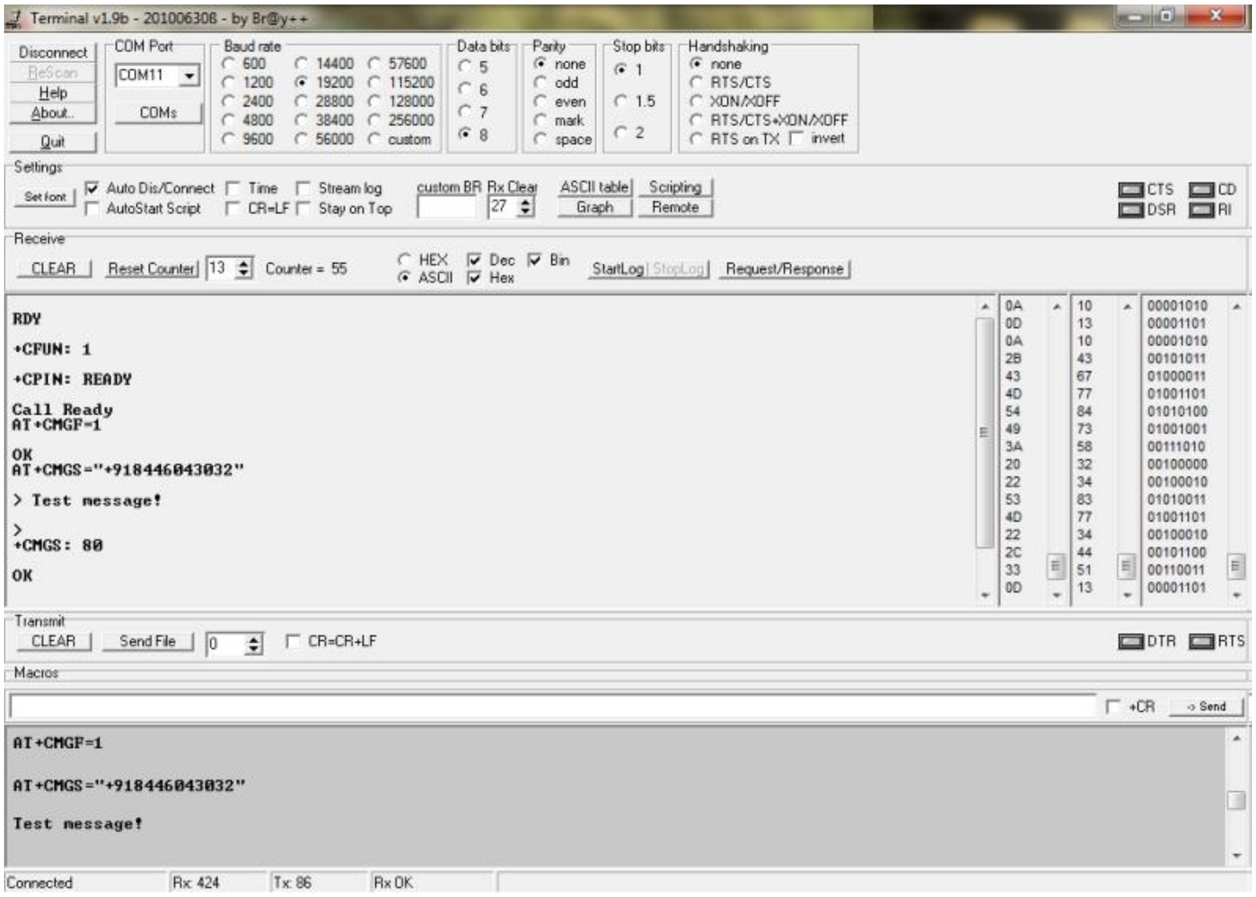
1. Create the setup as described in Step 1 above.

2. Through your serial terminal software, send AT+CMGF=1 and press the Enter key. The GPRS Shield can send SMSes in two modes: Text mode and PDU (or binary) mode. Since we want to send out a human readable message, we will select the text mode. The GPRS Shield will respond with an OK.

3. Send AT+CMGS="+918446043032" and press the Enter key (include the quotes). This will instruct the GPRS Shield to start accepting text for a new message meant for the phone number specified (replace the number with the phone number of the target phone). The GPRS Shield will send a > signaling you to

start typing the message. Please note that phone numbers specified as parameters in any AT Command must be in E.123 format.

4. Start typing your message and when you are done, press Ctrl + Z keys on your keyboard when your cursor at the end of the message. The modem will accept the message and respond with an OK. A few moments later, the message should be received on the handset whose number you had specified.



COM Port: COM11
Baud rate: 600 1200 2400 4800 9600 14400 [✓ 19200] 28800 38400 56000 57600 115200 128000 256000 custom
Data bits: 5 6 7 [✓ 8]
Parity: [✓ none] odd even mark space
Stop bits: [✓ 1] 1.5 2
Handshaking: [✓ none] RTS/CTS XON/XOFF RTS/CTS+XON/XOFF RTS on TX ☐ invert

Settings: ☑ Auto Dis/Connect ☐ AutoStart Script ☐ Time ☐ CR=LF ☐ Stream log ☐ Stay on Top custom BR Rx Clear 27

Receive: Reset Counter 13 Counter = 55 HEX [✓ ASCII] ☑ Dec ☑ Hex ☑ Bin

```
RDY

+CFUN: 1

+CPIN: READY

Call Ready
AT+CMGF=1

OK
AT+CMGS="+918446043032"

> Test message!

>
+CMGS: 80

OK
```

```
0A   10   00001010
0D   13   00001101
0A   10   00001010
2B   43   00101011
43   67   01000011
4D   77   01001101
54   84   01010100
49   73   01001001
3A   58   00111010
20   32   00100000
22   34   00100010
53   83   01010011
4D   77   01001101
22   34   00100010
2C   44   00101100
33   51   00110011
0D   13   00001101
```

Transmit: 0 ☐ CR=CR+LF

Macros

☐ +CR

```
AT+CMGF=1

AT+CMGS="+918446043032"

Test message!
```

Connected | Rx: 424 | Tx: 86 | Rx OK

NOTE: If, in spite of following the steps as specified above, you aren't able to receive the message on the target handset, then it might be that you need to set the SMS Message Center number. Send the command AT+CSCA="+919032055002" and press the Enter Key. Send this command in between the AT+CMGF and AT+CMGS commands. Replace the phone number specified in the command above with the SMS Center number of your GSM Service Provider. The message center number is specific to each service provider (for example +919032055002 is the message center number for Tata DoCoMo, Pune, India). You can get the message center number by calling up the customer care center of the GSM Service Provider and asking them for it.

Step 3: Exploring further

Now that you have gotten a taste of how the AT Commands work, you can try out some more of them before moving on to developing sketches for Arduino to use the GPRS Shield. This involves creating a sketch for sending out these same sequence of AT Commands (on your behalf) out the serial port to the GPRS Shield to perform the same task of sending and SMS, making a call or sending data over a GPRS connection. You can go through the AT Commands reference manual to figure out the sequence of commands required to do a particular task. If while developing an Arduino sketch, you find that the GPRS Shield isn't what you expected it to do, then you will need to check your AT Commands and their sequence. To do this, reload the serial relay sketch attached above in the getting started section into ATmega328P and type out the AT Commands manually and check the output. The responses sent by the GPRS Shield will help you debug the AT Command sequence.

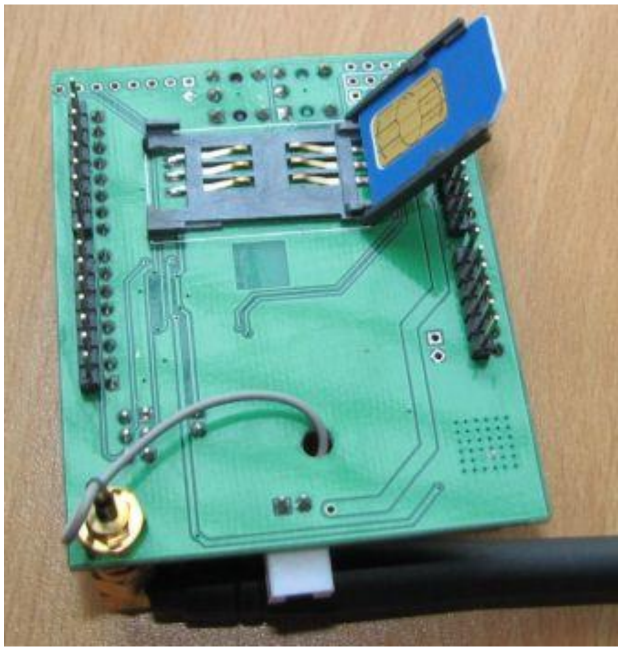
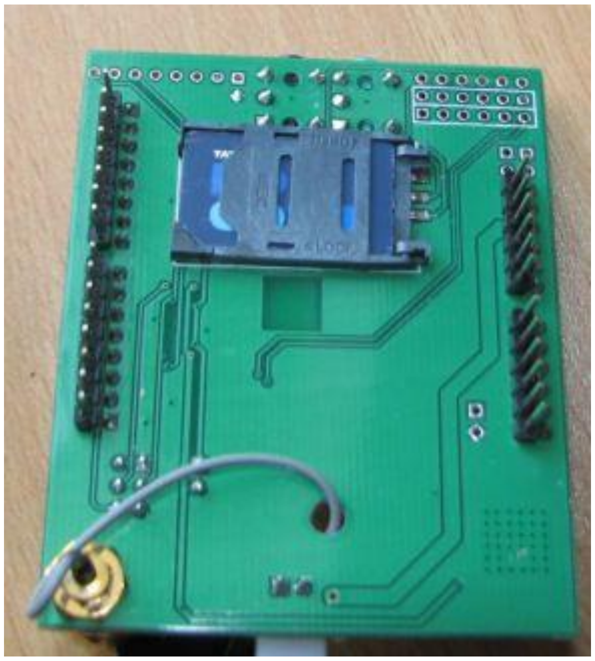
NOTE: A C program to perfom the same task as the serial relay sketch present above has also been developed and attached: File:Softuart relay atmega328p.zip. The program was developed on a Windows PC. AVRStudio4 was used as the IDE and WinAVR was used as the compiler. The ZIP file contains an AVRStudio4 Project. The C compiler (WinAVR) will generate an Intel Hex (.hex). To upload this .hex file into an Arduino board outside of Arduino IDE would require a program which is able to communicate with the Arduino boards bootloader. XLoader is such a program which runs on Windows can upload .hex files generated by various compiler into an Arduino Board.

Connection Notes

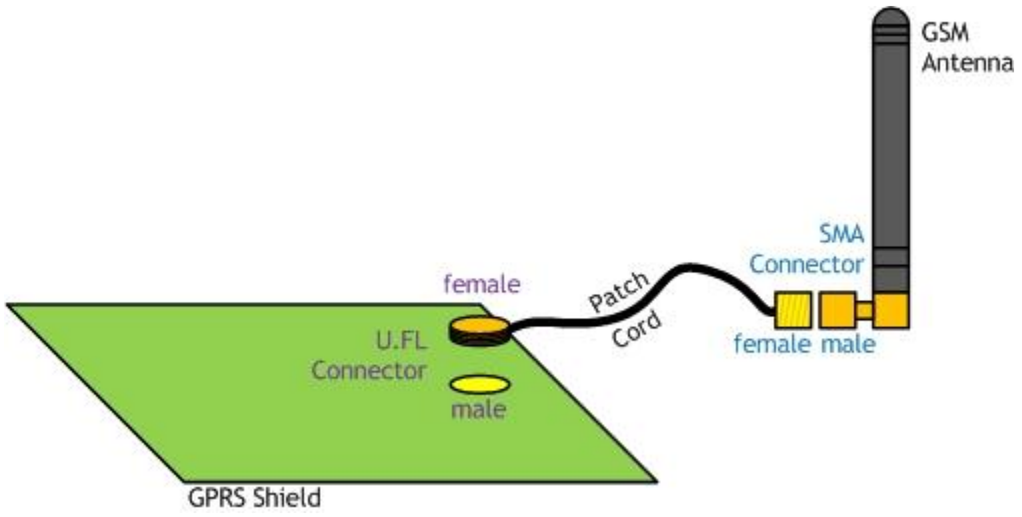
This section is a work in progress

Here are the list of connectors and pin headers present on-board the GPRS Shield:

• **SIM Card Holder** - 6 Pin Holder for SIM Cards. Both 1.8 volts and 3.0 volts SIM Cards are supported by SIM900 - the SIM card voltage type is automatically detected. Out of the 6 pins present on the SIM card holder, one is life unconnected and the other 5 are: SIM_VDD,SIM_RST,SIM_CLK,SIM_DATA,GND

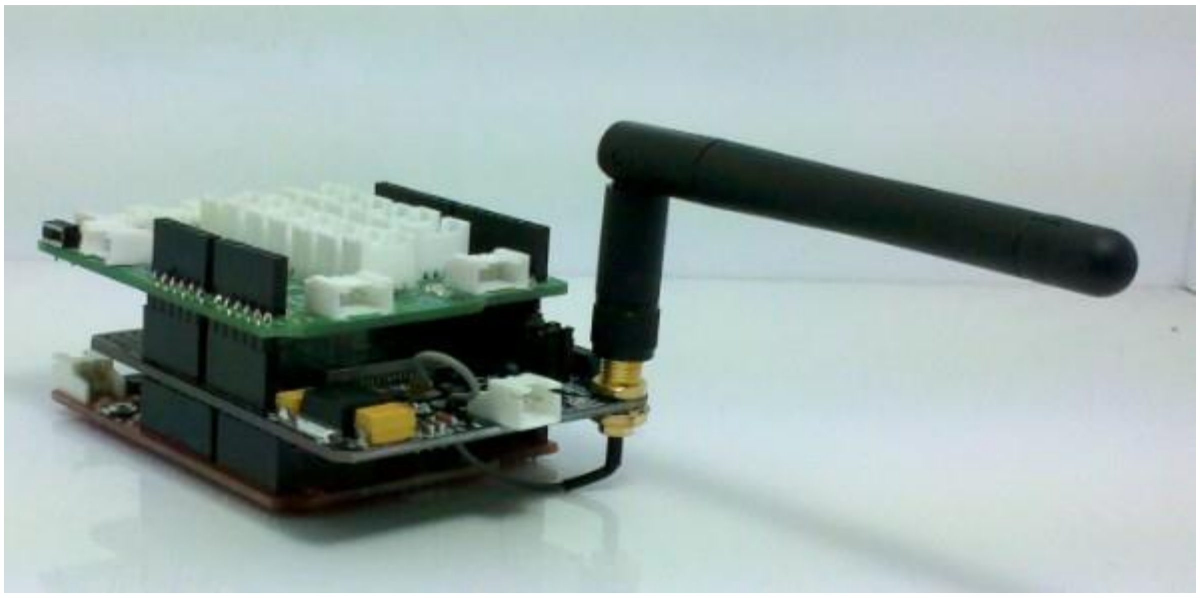



• **Antenna Connector** - A miniature coaxial RF connector is present on board the GPRS Shield to connect to a GSM Antenna. The connector present on the GPRS Shield is called a U.FL connecto]. The GSM Antenna supplied with the GPRS Shield has an SMA connector (and not an RP-SMA connector) on it. A patch cord is also supplied with the GPRS Shield to interface the antenna to the board. The connection topology is as shown in the diagram below:



• There are two holes provided on the PCB - one for mounting the antenna and the other for routing the patch cord from below the PCB to the top side so that it can connect to the U.FL connector. This would be useful if you want to mount the GPRS Shield and your Arduino board inside an enclosure.

• **Arduino headers** - The GPRS Shield, like any other well designed shield is stackable as shown in the photo below.



Source Code Examples

Here are some source code examples.

Sending SMS: using Hardware UART

The Arduino sketch for sending an SMS is presented below. It has been tested on Arduino Duemilanove but will work on any compatible variant. Please note that this sketch uses the hardware UART of ATmega328P. Please follow the following steps for running this sketch.

1. With the GPRS Shield removed, download this sketch into your Arduino. The GPRS Shield must be removed so that it doesn't interfere with the programming of Arduino which takes place over the Hardware UART (using FT232RL).

2. Disconnect the Arduino from USB port to remove power to it.

3. Set the Serial Port jumpers on the GPRS Shield in Xduino position (i.e. Arduino's RX 4. connected to GPRS_TX and TX of Arduino connected to GPRS_RX)

Connect the antenna to the GPRS Shield and insert the SIM Card.

5. Mount the GPRS Shield on Arduino.

6. Apply power to the Arduino using USB port or via external power supply.

7. Switch on the GPRS Shield by using the power switch. Wait till the Network LED (D1) starts blinking.

8. Using a pen or a plastic tweezer access the reset switch on the Arduino Board and reset the microcontroller to run the sketch from the start. Do not try resetting the Arduino by removing and applying power to it as this will turn off the GPRS Shield.

9. If nothing goes wrong, the SMS will be received on receiver's handset.

```
void setup()

{

  Serial.begin(19200);  //Default serial port setting for the GPRS modem is 19200bps 8-N-1

  Serial.print("\r");

  delay(1000);                  //Wait for a second while the modem sends an "OK"

  Serial.print("AT+CMGF=1\r");    //Because we want to send the SMS in text mode

  delay(1000);


  //Serial.print("AT+CSCA=\"+919032055002\"\r");  //Setting for the SMS Message center number,

  //delay(1000);                              //uncomment only if required and replace with

                                        //the message center number obtained from

                                        //your GSM service provider.

                                        //Note that when specifying a tring of characters

                                        // " is entered as \"


  Serial.print("AT+CMGS=\"+918446043032\"\r");    //Start accepting the text for the message

                                        //to be sent to the number specified.

                                        //Replace this number with the target mobile number.

  delay(1000);

  Serial.print("SIM900 and Arduino say Hi!\r");   //The text for the message

  delay(1000);

  Serial.print(26,BYTE);  //Equivalent to sending Ctrl+Z

}


void loop()
```

```
{
   //We just want to send the SMS only once, so there is nothing in this loop.
   //If we put the code for SMS here, it will be sent again and again and cost us a lot.
```

Making a call & Controlling GPIO and PWM pins

The sample source code below shows how one can make a call and control the GPRS Shield's General Purpose I/O pins as well as the PWM pins

```
//Source code sample showing how to make a call (use headphones and mic to talk),
//and how to control the General Purpose I/O pins and PWM

#include <NewSoftSerial.h>

NewSoftSerial mySerial(7, 8);

void setup()
{
  mySerial.begin(19200);           // the GPRS baud rate
  Serial.begin(19200);           // the GPRS baud rate
  delay(2000);
}

void loop()
{
  int count=0;

  mySerial.println("ATDxxxxxxxxx;"); // xxxxxxxxx is the number you want to dial.
```

```
delay(2000);
  while(1)
  {
    mySerial.println("AT+SPWM=2,63,100");// set PWM 2 PIN
    delay(100);
    mySerial.println("AT+SPWM=1,63,100");
    delay(100);
    mySerial.println("AT+SGPIO=0,1,1,1");// set GPIO 1 PIN to 1
    delay(100);
    mySerial.println("AT+SGPIO=0,2,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,3,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,4,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,5,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,6,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,7,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,8,1,1");
    delay(100);
    mySerial.println("AT+SGPIO=0,9,1,1");
    delay(100);
```

```
mySerial.println("AT+SGPIO=0,10,1,1");

delay(100);

mySerial.println("AT+SGPIO=0,11,1,1");

delay(100);

mySerial.println("AT+SGPIO=0,12,1,1");


delay(500);


mySerial.println("AT+SPWM=1,63,0");

delay(100);

mySerial.println("AT+SPWM=2,63,0");

delay(100);

mySerial.println("AT+SGPIO=0,1,1,0"); // set GPIO 1 PIN to 0

delay(100);

mySerial.println("AT+SGPIO=0,2,1,0");

delay(100);

mySerial.println("AT+SGPIO=0,3,1,0");

delay(100);

mySerial.println("AT+SGPIO=0,4,1,0");

delay(100);

mySerial.println("AT+SGPIO=0,5,1,0");

delay(100);

mySerial.println("AT+SGPIO=0,6,1,0");

delay(100);

mySerial.println("AT+SGPIO=0,7,1,0");
```

```
  delay(100);

  mySerial.println("AT+SGPIO=0,8,1,0");

  delay(100);

  mySerial.println("AT+SGPIO=0,9,1,0");

  delay(100);

  mySerial.println("AT+SGPIO=0,10,1,0");

  delay(100);

  mySerial.println("AT+SGPIO=0,11,1,0");

  delay(100);

  mySerial.println("AT+SGPIO=0,12,1,0");

  delay(500);


  count++;


  if(count==5)

  {

    mySerial.println("ATH"); //end the call.

    if(mySerial.available())

   {

      Serial.print((unsigned char)mySerial.read());


    }

  }

}
```

Uploading Sensor Data to Pachube.com using GPRS

We can mate a GPRS shield with a Seeeduino Stalker to create a remote telemetry system which uploads data to a server on the internet. Pachube.com is a data brokerage platform to which our Stalker+GPRS Shield combo can connect to and upload sensor data over GPRS (or EDGE). A GSM (or 3G) SIM Card with active data plan is required for this. The example below takes temperature reading from TMP102 sensor present on board the Stalker, connects to pachube.com using HTTP and uploads the data to it.

```
//Remember when using GPRS Shield with Seeeduino Stalker v2.0

//please dismount the OK_READ Jumper (i.e. open it).

//This will disconnect the Battery Charger IC's OK pin from

//the microcontrollers Digital Pin 7 and hence allow unhindered

//communication with GPRS Shield using NewSoftSerial Library.



//Replace the following items in the code:

//1. Replace the Access Point Name "TATA.DOCOMO.INTERNET"

//and the DNS server name "10.6.6.6" in the AT+CGDCONT and AT+CSTT

//commands with those of your own service provider.

//

//2. Replace the Pachube API Key with your personal ones assigned

//to your account at pachube.com

//

//3. You may choose a different name for the the data stream.

//I have choosen "TMP102". If you use a different name, you will have

//to replace this string with the new name.
```

```
//

#include <NewSoftSerial.h>
#include "tmp102.h"
//Please fetch tmp102.h and tmp102.cpp from "Stlker logger AM06 Serial.zip"
//available from Seeeduino Stalker v2.0's Wiki Page
#include <Wire.h>


float convertedtemp; // We then need to multiply our two bytes by a scaling factor, mentioned in the datasheet.
int tmp102_val; // an int is capable of storing two bytes, this is where we "chuck" the two bytes together.

NewSoftSerial GPRS_Serial(7, 8);

void setup()
{
  GPRS_Serial.begin(19200);  //GPRS Shield baud rate
  Serial.begin(19200);
  tmp102_init();

setup_start:

  Serial.println("Turn on GPRS Modem and wait for 1 minute.");
  Serial.println("and then press a key");
  Serial.println("Press c for power on configuration");

  Serial.println("press any other key for uploading");

  Serial.flush();

  while(Serial.available() == 0);

  if(Serial.read()=='c')

  {

    Serial.println("Executing AT Commands for one time power on configuration");





    GPRS_Serial.flush();





    GPRS_Serial.println("ATE0"); //Command echo off

    Serial.println("ATE0   Sent");

    if(GPRS_Serial_wait_for_bytes(4,10) == 0)

    {

      Serial.println("Timeout");

      goto setup_start;

    }

    else

    {

      Serial.print("Received:");

      while(GPRS_Serial.available()!=0)
    {
      Serial.print((unsigned char)GPRS_Serial.read());
      Serial.print("\n");
    }
  }
```

```
GPRS_Serial.println("AT+CIPMUX=0"); //We only want a single IP Connection at a time.
Serial.println("AT+CIPMUX=0   Sent");
if(GPRS_Serial_wait_for_bytes(4,10) == 0)
{
  Serial.println("Timeout");
  goto setup_start;
}
else
{
  Serial.print("Received:");
  while(GPRS_Serial.available()!=0)
  {
    Serial.print((unsigned char)GPRS_Serial.read());
    Serial.print("\n");
```

```
     }

    }

    GPRS_Serial.println("AT+CIPMODE=0"); //Selecting "Normal Mode" and NOT "Transparent Mode" as
the TCP/IP Application Mode

    Serial.println("AT+CIPMODE=0    Sent!");

    if(GPRS_Serial_wait_for_bytes(4,10) == 0)

    {

      Serial.println("Timeout");

      goto setup_start;

    }

    else

    {

      Serial.print("Received:");

      while(GPRS_Serial.available()!=0)

      {

        Serial.print((unsigned char)GPRS_Serial.read());

        Serial.print("\n");

      }

    }
```

```
   GPRS_Serial.println("AT+CGDCONT=1,\"IP\",\"TATA.DOCOMO.INTERNET\",\"10.6.6.6\",0,0");
//Defining the Packet Data

//Protocol Context - i.e. the Protocol Type, Access Point Name and IP Address

   Serial.println("AT+CGDCONT=1,\"IP\",\"TATA.DOCOMO.INTERNET\",\"10.6.6.6\",0,0   Sent!");

   if(GPRS_Serial_wait_for_bytes(4,10) == 0)

   {

     Serial.println("Timeout");

     goto setup_start;

   }

   else

   {

     Serial.print("Received:");

     while(GPRS_Serial.available()!=0)

     {

       Serial.print((unsigned char)GPRS_Serial.read());

       Serial.print("\n");

     }

   }
   GPRS_Serial.println("AT+CSTT=\"TATA.DOCOMO.INTERNET\""); //Start Task and set Access Point
Name (and username and password if any)

   Serial.println("AT+CSTT=\"TATA.DOCOMO.INTERNET\"   Sent!");

   if(GPRS_Serial_wait_for_bytes(4,10) == 0)

   {

     Serial.println("Timeout");

     goto setup_start;

   }

   else

   {

     Serial.print("Received:");

     while(GPRS_Serial.available()!=0)

     {

       Serial.print((unsigned char)GPRS_Serial.read());

       Serial.print("\n");

     }

   }
```

```
    GPRS_Serial.println("AT+CIPSHUT"); //Close any GPRS Connection if open
    Serial.println("AT+CIPSHUT  Sent!");
    if(GPRS_Serial_wait_for_bytes(7,10) == 0)
    {
      Serial.println("Timeout");
      goto setup_start;
    }
    else
    {
      Serial.print("Received:");
      while(GPRS_Serial.available()!=0)
      {
        Serial.print((unsigned char)GPRS_Serial.read());
        Serial.print("\n");
      }
    }
  }
}

void loop()
{
loop_start:

  Serial.println("Press a key to read temperature and upload it");
  Serial.flush();
  while(Serial.available() == 0);

  Serial.read();



  getTemp102();

  Serial.print("TMP102 Temperature = ");

  Serial.println(convertedtemp);


  GPRS_Serial.println("AT+CIPSTART=\"TCP\",\"api.pachube.com\",\"80\""); //Open a connection to
Pachube.com

  Serial.println("AT+CIPSTART=\"TCP\",\"api.pachube.com\",\"80\"  Sent!");

  if(GPRS_Serial_wait_for_bytes(12,255) == 0)

  {

    Serial.println("Timeout");

    goto loop_start;

  }

  else

  {

    Serial.print("Received:");

    while(GPRS_Serial.available()!=0)

    {

      Serial.print((unsigned char)GPRS_Serial.read());

      Serial.print("\n");

    }

  }
```

```
 GPRS_Serial.flush();

 GPRS_Serial.println("AT+CIPSEND"); //Start data through TCP connection

 Serial.println("AT+CIPSEND  Sent!");

 if(GPRS_Serial_wait_for_bytes(1,100) == 0)

 {

   Serial.println("Timeout");

   goto loop_start;

 }

 else

 {

   Serial.print("Received:");

   while(GPRS_Serial.available()!=0)

   {

     Serial.print((unsigned char)GPRS_Serial.read());

     Serial.print("\n");

   }

 }



 GPRS_Serial.flush();


 //Emulate HTTP and use PUT command to upload temperature datapoint using Comma Seperate Value
Method

 GPRS_Serial.print("PUT /v2/feeds/24300.csv HTTP/1.1\r\n");

 Serial.println("PUT /v2/feeds/24300.csv HTTP/1.1  Sent!");

 delay(300);
```

```
  GPRS_Serial.print("Host: api.pachube.com\r\n");

  Serial.println("Host: api.pachube.com  Sent!");

  delay(300);



  GPRS_Serial.print("X-PachubeApiKey:
abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz01\r\n"); //REPLACE THIS KEY
WITH YOUR OWN PACHUBE API KEY

  Serial.println("X-PachubeApiKey:
abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz01  Sent!"); //REPLACE THIS KEY
WITH YOUR OWN PACHUBE API KEY

  delay(300);



  GPRS_Serial.print("Content-Length: 12\r\n");

  Serial.print("Content-Length: 12  Sent!");

  delay(300);



  GPRS_Serial.print("Connection: close\r\n\r\n");

  Serial.print("Connection: close  Sent!");

  delay(300);

  GPRS_Serial.print("TMP102,"); //You may replace the stream name "TMP102" to any other string that
you have choosen.

  delay(300);

  GPRS_Serial.print(convertedtemp);

  delay(300);

  GPRS_Serial.print("\r\n");

  delay(300);
```

```
GPRS_Serial.print("\r\n");

delay(300);

GPRS_Serial.print(0x1A,BYTE);

delay(300); //Send End Of Line Character to send all the data and close connection

if(GPRS_Serial_wait_for_bytes(20,255) == 0)

{

  Serial.println("Timeout");

  goto loop_start;

}

else

{

  Serial.print("Received:");

  while(GPRS_Serial.available()!=0)

  {

    Serial.print((unsigned char)GPRS_Serial.read());

    Serial.print("\n");

  }

}





GPRS_Serial.flush();

GPRS_Serial.println("AT+CIPSHUT"); //Close the GPRS Connection

Serial.println("AT+CIPSHUT  Sent!");
```

```
 if(GPRS_Serial_wait_for_bytes(4,100) == 0)
 {
   Serial.println("Timeout");
   goto loop_start;
 }
 else
 {
   Serial.print("Received:");
   while(GPRS_Serial.available()!=0)
   {
     Serial.print((unsigned char)GPRS_Serial.read());
     Serial.print("\n");
   }
 }
}
```

```
char GPRS_Serial_wait_for_bytes(char no_of_bytes, int timeout)
```

```
{
  while(GPRS_Serial.available() < no_of_bytes)
  {
    delay(200);
    timeout-=1;
    if(timeout == 0)
    {
      return 0;
    }
  }
  return 1;

}
```

Issue Tracker

Issue Tracker is the place you can publish any suggestions for improvement or any bugs you think you might have found during use. Please write down what you have to say, your suggestions will help us improve our products.

1. **Header pinout matching UartSBee** (Issue Type: Improvement)

Right now to evaluate the AT Commands on GPRS Shield, one needs to install it on an Arduino compatible board which is running a sketch to relay characters to and fro between the computer and the GPRS Shield. This method is a bit unreliable and some characters are lost sometimes in the process and the AT Command needs to be sent again. If the header on top left corner of GPRS Shield is made to match up with that on UartSBee, then it will present to the user a nice reliable way to learn and evaluate the AT Commands without the use of an Arduino compatible board. Refer to the photos below:

2. **Arduino board must be able to control the power to GPRS Shield (Issue Type: Necessary Addition)**

Right now the GPRS Shield needs to be turned on manually using a switch. In future version, a transistor may be used in parallel with this switch which can be controlled by the Arduino board below to turn the GPRS Shield on. Also the "status" signal which is currently routed to an LED may also be routed to one of the Arduino headers so that the microcontroller on the Arduino compatible board below can sense if the GPRS Shield has turn on or not.

Additional Ideas

The Additional Ideas is the place to write your project ideas about this product, or other usages you've found. Or you can write them on Projects page.