Use the SIM900 GSM modules with Arduino in Chapter 55 of our Arduino Tutorials. The first chapter is **here**, the complete series is detailed **here**. Updated 14/02/2015

***Introduction***

The goal of this tutorial is to illustrate various methods of interaction between an Arduino Uno (or compatible) and the GSM cellular network using a SIM900 GSM shield, with which you can then use your **existing knowledge** to build upon those methods. Apart from setting up the shield we'll examine:

- Making a telephone call from your Arduino
- Sending an SMS text message
- Receiving text messages with the shield and displaying them on the serial monitor
- Simple controlling of the host Arduino by calling it from another telephone
- Controlling the Arduino via SMS text message

We'll be using a SIMCOM **SIM900 GSM module shield**.. There must be scores of Arduino shields or modules using the SIM900, so as you can imagine each one may be a little bit different with regards to the hardware side of things – so we're assuming you have an understanding of how hardware and software serial works as well as supply voltages and the hardware side of the Arduino world.

As for the specific shield to use, we just chose the cheapest one available at the time – which turned out to be the "**SIM900 GPRS/GSM Arduino shield**


However with a little research and work on your part, the sketches provided *should* also work with any SIM900 module/shield and Arduino – as long as you have the appropriate serial and power settings.

***Getting Started***

A little preparation goes a long way, so make sure you've covered the following points:

- Regarding your cellular provider. Do you have coverage on a GSM 850 MHz, GSM 900 MHz, DCS 1800 MHz or PCS 1900 MHz network? When we say GSM that means 2G – not 3G, 4G or LTE. Will they allow the use of non-supported devices on the network? Some carriers will block IMEI numbers that were not provided by their sales channel. Or you may have to call the provider and supply the IMEI of your GSM module to allow it on the network.

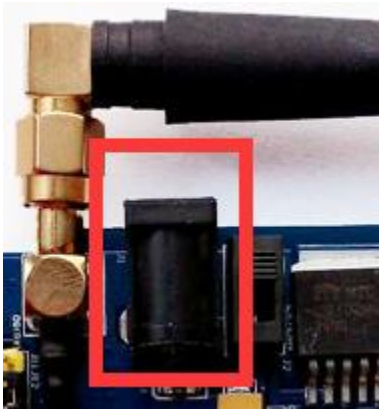
Finally, it would be wise to use either a prepaid or an account that offers unlimited SMS text messaging – you don't want any large bills if things go wrong.

- Power. Do you have adequate power for your SIM900 module? Some shields will use more current than the Arduino can supply (up to 2A), so you may need an external high-current supply. The Linksprite shield we use needs 5V up to 2A or 9V 1A or 12V 1A into the onboard DC socket. Otherwise, check with your supplier.
- Antenna. If your module/shield etc. doesn't have an antenna – get one. You do need it.
- Turn off the PIN lock on the SIM card. The easiest way to do this is to put the SIM in a handset and use the menu function.
- And as always, please don't make an auto-dialler…

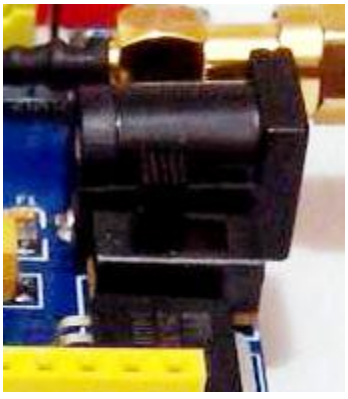
Furthermore, download the **SIM900 hardware manual** (.pdf) and the **AT command manual** (.pdf), as we'll refer to those throughout the tutorial.

***Power***

There is a DC socket on the shield, which is for a 5V power supply:

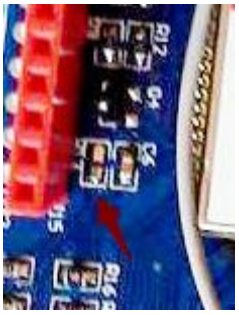


Although the data from Link sprite claims the shield will use no more than 450 mA, the SIMCOM hardware manual (page 22) for the module notes that it can draw up to 2A for short bursts. So get yourself a 5V 2A power supply and connect it via the DC socket, and also ensure the switch next to the socket is set to like the following picture



Furthermore, you can turn the GSM module on and off with the power button on the side of the shield, and it defaults to *off* during an initial power-up.

Also, if you connect the R13, you can set D9 to HIGH for one second in your sketch to turn the module on (or off if required for power-saving). Don't panic, we'll show how this is done in the sketches below.



***Software Serial***

We will use the Arduino software serial library in this tutorial, and the Link sprite shield has hard-wired the serial from the SIM900 to a set of jumpers, and uses a default speed of 19200. Make sure you your jumpers are set to the “SWserial” side, as shown below:


And thus whenever an instance of SoftwareSerial is created, we use 7,8 as shown below:

```
1 SoftwareSerial SIM900(7, 8); // RX, TX
```

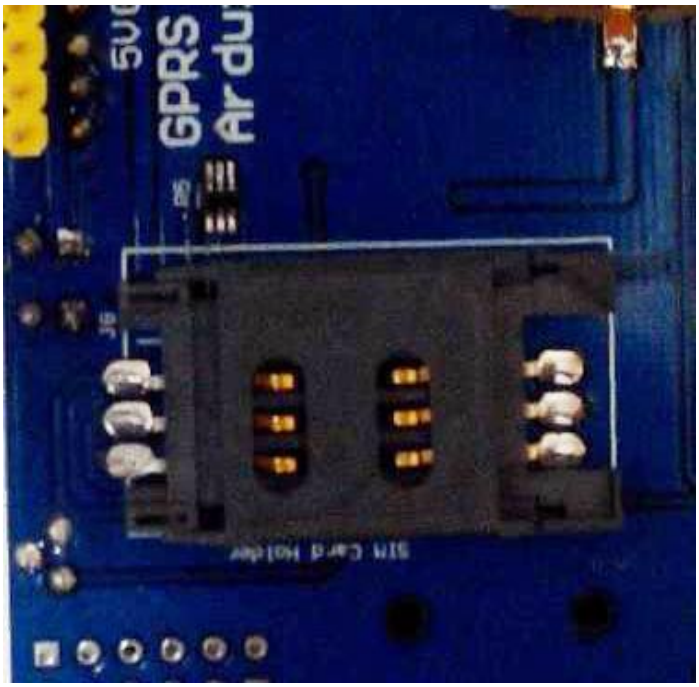
If you shield is different, you’ll need to change the TX and RX pin numbers. This also means you can’t use an Arduino Leonardo or Mega (easily). Finally – note this for later – if your shield is having problems sending data back to your Arduino you may need to edit the SoftwareSerial library – **read this** for more information.

***Wow – all those rules and warnings?***

The sections above may sound a little authoritarian, however we want your project to be a success. Now, let’s get started…

***A quick test…***

At this point we’ll check to make sure your shield and locate and connect to the cellular network. So make sure your SIM card is active with your cellular provider, the PIN lock is off, and then insert it and lock the SIM card  to the carrier on the bottom of the shield:




Then plug the shield into your Uno, attach 5V power to the DC socked *on the GSM shield*, and USB from the Uno to the PC. Press the “PWRKEY” button on the side of the shield for a second, then watch the following two LEDs:


The bright “STATUS” LED will come on, and then the “NETLIGHT” LED will blink once every 800 milliseconds- until the GSM module has found the network, at which point it will blink once every three seconds.

Nothing can happen until that magic three-second blink – so if that doesn’t appear after a minute, something is wrong. Check your shield has the appropriate power supply, the antenna is connected correctly, the SIM card is seated properly and locked in- and that your cellular account is in order. Finally, you may not have reception in that particular area, so check using a phone on the same network or move to a different location.

***Making a telephone call from your Arduino***

You can have your Arduino call a telephone number, wait a moment – and then hang up. This is an inexpensive way of alerting you of and considers the following sketch:

```
1  // Example 55.1
2
3  #include <SoftwareSerial.h>
4  SoftwareSerial SIM900(7, 8); // configure software serial port
5
6  void setup()
7  {
8    SIM900.begin(19200);
9    SIM900power();
10   delay(20000);  // give time to log on to network.
11 }
12
13 void SIM900power()
14 // software equivalent of pressing the GSM shield "power" button
15 {
16   digitalWrite(9, HIGH);
17   delay(1000);
18   digitalWrite(9, LOW);
19   delay(5000);
20 }
21
22 void callSomeone()
23 {
24   SIM900.println("ATD + +12128675309;"); // dial US (212) 8675309
25   delay(100);
26   SIM900.println();
27   delay(30000);           // wait for 30 seconds...
28   SIM900.println("ATH");    // hang up
29 }
30
31 void loop()
32 {
33   callSomeone(); // call someone
34   SIM900power();    // power off GSM shield
35   do {} while (1); // do nothing
36 }
```

The sketch first creates a software serial port, then in void setup() starts the software serial port, and also turns on the GSM shield with the function SIM900power (which simply sets D9 high for a second which is the equivalent of pressing the power button). Notice the delay function in void setup – this gives the GSM module a period of time to locate and log on to the

cellular network. You may need to increase (or be able to decrease) the delay value depending on your particular situation. If in doubt, leave it as a long period.
The process of actually making the call is in the function callSomeone(). It sends a string of text to the GSM module which consists of an AT command. These are considered the "language" for modems and thus used for various tasks. We use the ATD command to dial (AT… D for dial) a number. The number as you can see in the sketch needs to be in world-format. So that's a "+" then the country code, then the phone number with area code (without the preceding zero).
So if your number to call is Australia (02) 92679111 you would enter +61292679111. Etcetera. A carriage return is then sent to finalise the command and off it goes dialling the number.
Here's a quick video demonstration for the **non-believers**:
After thirty seconds we instruct the module to hand up with another AT command – "ATH" (AT… H for "hang up"), followed by turning off the power to the module. By separating the call feature into a function – you can now insert this into a sketch (plus the preceding setup code) to call a number when required.

***Sending an SMS text message***

This is a great way of getting data from your Arduino to almost any mobile phone in the world, at a very low cost. For reference, the maximum length of an SMS text message is 160 characters – however you can still say a lot with that size limit. First we'll demonstrate sending an arbitrary SMS. Consider the following sketch:

```
1  // Example 55.2
2
3  #include <SoftwareSerial.h>
4  SoftwareSerial SIM900(7, 8);
5
6  void setup()
7  {
8    SIM900.begin(19200);
9    SIM900power();
10   delay(20000);  // give time to log on to network.
11 }
12
13 void SIM900power()
14 // software equivalent of pressing the GSM shield "power" button
15 {
16   digitalWrite(9, HIGH);
17   delay(1000);
18   digitalWrite(9, LOW);
19   delay(5000);
20 }
```

```
21
22 void sendSMS()
23 {
24   SIM900.print("AT+CMGF=1\r");                                    // AT command to send SMS
25 message
26   delay(100);
27   SIM900.println("AT + CMGS = \"+12128675309\"");                  // recipient's mobile
28 number, in international format
29   delay(100);
30   SIM900.println("Hello, world. This is a text message from an Arduino Uno.");       // message to
31 send
32   delay(100);
33   SIM900.println((char)26);                  // End AT command with a ^Z, ASCII code 26
34   delay(100);
35   SIM900.println();
36   delay(5000);                          // give module time to send SMS
37   SIM900power();                          // turn off module
38 }
39
40 void loop()
41 {
     sendSMS();
     do {} while (1);
   }
```

The basic structure and setup functions of the sketch are the same as the previous example, however the difference here is the function sendSMS(). It used the AT command “AT+CMGF” to tell the GSM module we want to send an SMS in text form, and then “AT+CMGS” followed by the recipient’s number. Once again note the number is in international format. After sending the send SMS commands, the module needs five seconds to do this before we can switch it off.

You can also send text messages that are comprised of numerical data and so on – by compiling the required text and data into a string, and then sending that. Doing so gives you a method to send such information as sensor data or other parameters by text message.
For example, you might want to send daily temperature reports or hourly water tank levels. For our example, we’ll demonstrate how to send a couple of random numbers and some text as an SMS. You can then use this as a framework for your own requirements. Consider the following sketch:

```
1  // Example 55.3
2
```

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900(7, 8);
int x,y;
String textForSMS;

void setup()
{
  SIM900.begin(19200);
  SIM900power();
  delay(20000);  // give time to log on to network.
  randomSeed(analogRead(0));
}

void SIM900power()
// software equivalent of pressing the GSM shield "power" button
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(7000);
}

void sendSMS(String message)
{
  SIM900.print("AT+CMGF=1\r");                    // AT command to send SMS message
  delay(100);
  SIM900.println("AT + CMGS = \"+12128675309\"");  // recipient's mobile number, in
international format
  delay(100);
  SIM900.println(message);                        // message to send
  delay(100);
  SIM900.println((char)26);                       // End AT command with a ^Z, ASCII code 26
  delay(100);
  SIM900.println();
  delay(5000);                                    // give module time to send SMS
  SIM900power();                                  // turn off module
}

void loop()
{
  x = random(0,255);
  y = random(0,255);
  textForSMS = "Your random numbers are ";
  textForSMS.concat(x);
```

```
47   textForSMS = textForSMS + " and ";
48   textForSMS.concat(y);
49   textForSMS = textForSMS + ". Enjoy!";
50   sendSMS(textForSMS);
51   do {} while (1);
     }
```

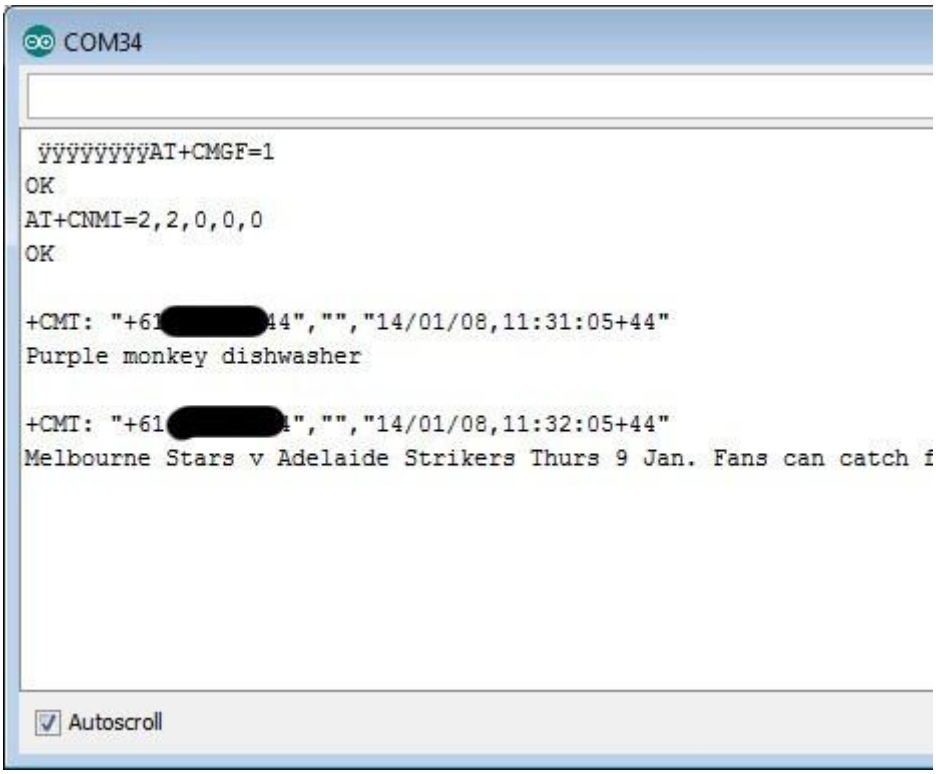
Take note of the changes to the function sendSMS(). It now has a parameter – *message*, which is a String which contains the text to send as an SMS. In void loop() the string variable *textForSMS* is constructed. First it contains some text, then the values for x and y are added with some more text. Finally the string is passed to be sent as an SMS. ***Receiving text messages and displaying them on the serial monitor***

Now let's examine receiving text messages. All we need is to send two AT commands inside *void setup()* and then repeat every character sent from the shield to the serial monitor. The first command to use is*AT+CMGF=1* which sets the SMS mode to text (as used in the previous example) and the second is *AT+CNMI=2,2,0,0* – which tells the GSM module to send the contents of any new SMS out to the serial line. To demonstrate this, set up your hardware as before, upload the following sketch:

```
1  // Example 55.4
2
3  #include <SoftwareSerial.h>
4  SoftwareSerial SIM900(7, 8);
5
6  char incoming_char=0;
7
8  void setup()
9  {
10   Serial.begin(19200); // for serial monitor
11   SIM900.begin(19200); // for GSM shield
12   SIM900power();  // turn on shield
13   delay(20000);  // give time to log on to network.
14
15   SIM900.print("AT+CMGF=1\r");  // set SMS mode to text
16   delay(100);
17   SIM900.print("AT+CNMI=2,2,0,0,0\r");
18   // blurt out contents of new SMS upon receipt to the GSM shield's serial out
19   delay(100);
20 }
21
22 void SIM900power()
23 // software equivalent of pressing the GSM shield "power" button
```

```
24 {
25   digitalWrite(9, HIGH);
26   delay(1000);
27   digitalWrite(9, LOW);
28   delay(7000);
29 }
30
31 void loop()
32 {
33   // Now we simply display any text that the GSM shield sends out on the serial monitor
34   if(SIM900.available() >0)
35   {
36     incoming_char=SIM900.read(); //Get the character from the cellular serial port.
37     Serial.print(incoming_char); //Print the incoming character to the terminal.
38   }
39 }
```

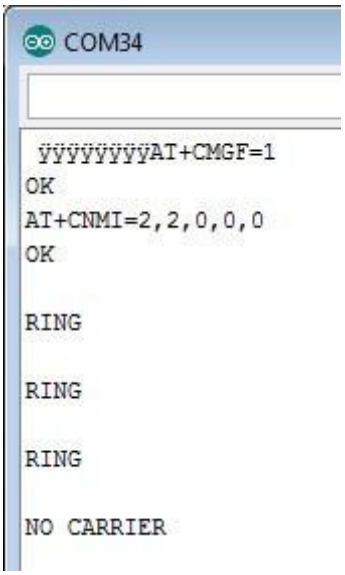
… then open the serial monitor, check it's set to 19200 bps and wait around thirty seconds. You will see some odd characters, then the "OK" responses from the GSM module. Now send a text message to the shield – the time/date stamp, sender and message will appear, for example:




To preserve my sanity the number used in the demonstrations will be blanked out.

***Simple controlling of the host Arduino by calling it from another telephone***

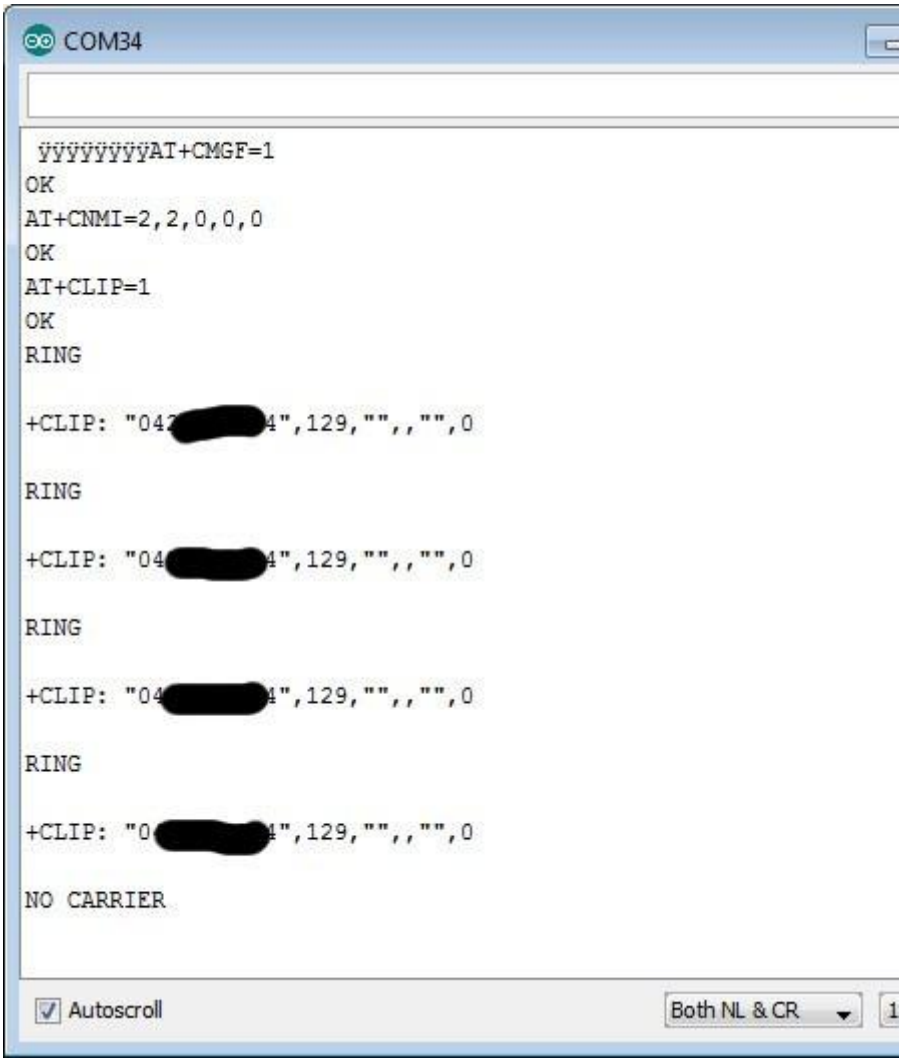
When we call our shield from another telephone, it sends the text "RING" out to serial, then "NO CARRIER" when you hang up – which can be harnessed to create a simple dial-in remote control. Use the sketch from the previous example to test this – for example:




You can also display the number calling in by using the AT command *AT+CLIP=1.* To do this, just add the following lines to *void setup()* in the previous sketch:

```
1 SIM900.print("AT+CLIP=1\r"); // turn on caller ID notification
2 delay(100);
```

Now when a call is received, the caller's number appears as well – for example:




Note that the caller ID data for incoming calls isn't in the international format as it was with SMSs. This will vary depending on your country, so check it out for yourself.

So how can we control the Arduino by calling in? By counting the number of times the shield sends "RING" to the Arduino (just as we did with the **other** GSM shield). To do this we simply count the number of times the word "RING" comes in from the shield, and then after the third ring – the Arduino will do something.

For our example we have two LEDs connected (via 560Ω resistors) to D12 and D13. When we call the shield and let it ring three times, they will alternate between on and off. Enter and upload the following sketch:

```
1  // Example 55.5
2
3  #include <SoftwareSerial.h>
4  char inchar; // Will hold the incoming character from the GSM shield
5  SoftwareSerial SIM900(7, 8);
```

```
6
7  int numring=0;
8  int comring=3;
9  int onoff=0; // 0 = off, 1 = on
10
11 void setup()
12 {
13   Serial.begin(19200);
14   // set up the digital pins to control
15   pinMode(12, OUTPUT);
16   pinMode(13, OUTPUT); // LEDs - off = red, on = green
17   digitalWrite(12, HIGH);
18   digitalWrite(13, LOW);
19
20   // wake up the GSM shield
21   SIM900power();
22   SIM900.begin(19200);
23   SIM900.print("AT+CLIP=1\r"); // turn on caller ID notification
24   delay(100);
25 }
26
27 void SIM900power()
28 // software equivalent of pressing the GSM shield "power" button
29 {
30   digitalWrite(9, HIGH);
31   delay(1000);
32   digitalWrite(9, LOW);
33   delay(7000);
34 }
35
36 void doSomething()
37 {
38   if (onoff==0)
39   {
40     onoff=1;
41     digitalWrite(12, HIGH);
42     digitalWrite(13, LOW);
43     Serial.println("D12 high D13 low");
44   }
45   else
46     if (onoff==1)
47     {
48       onoff=0;
49       digitalWrite(12, LOW);
```

```
50      digitalWrite(13, HIGH);
51      Serial.println("D12 low D13 high");
52    }
53 }
54
55 void loop()
56 {
57   if(SIM900.available() >0)
58   {
59     inchar=SIM900.read();
60     if (inchar=='R')
61     {
62       delay(10);
63       inchar=SIM900.read();
64       if (inchar=='I')
65       {
66         delay(10);
67         inchar=SIM900.read();
68         if (inchar=='N')
69         {
70           delay(10);
71           inchar=SIM900.read();
72           if (inchar=='G')
73           {
74             delay(10);
75             // So the phone (our GSM shield) has 'rung' once, i.e. if it were a real phone
76             // it would have sounded 'ring-ring' or 'blurrrrr' or whatever one cycle of your ring tone is
77             numring++;
78             Serial.println("ring!");
79             if (numring==comring)
80             {
81               numring=0; // reset ring counter
82               doSomething();
83             }
84           }
85         }
86       }
87     }
88   }
89 }
```

You can change the number of rings before action with the variable *comring*, and the action to take is in the function *void doSomething()*.

We can also modify this system so it only allows control by callers from one number – as long as caller ID works on your system (well it should… it's 2014 not 1996). So in the next example the system will only call the function *doSomething* if the call is from a certain number. The sketch works in the same manner as last time – but instead of counting the word "RING", it will compare the incoming caller's ID number against one in the sketch.
It may look a little clunky, but it works. In the following example sketch, the number is 2128675309 – so just change the digits to check in *void loop()*:

```
1  // Example 55.6
2
3  #include <SoftwareSerial.h>
4  char inchar; // Will hold the incoming character from the GSM shield
5  SoftwareSerial SIM900(7, 8);
6
7  int onoff=0; // 0 = off, 1 = on
8
9  void setup()
10 {
11   Serial.begin(19200);
12   // set up the digital pins to control
13   pinMode(12, OUTPUT);
14   pinMode(13, OUTPUT); // LEDs - off = red, on = green
15   digitalWrite(12, HIGH);
16   digitalWrite(13, LOW);
17
18   // wake up the GSM shield
19   SIM900power();
20   SIM900.begin(19200);
21   delay(20000);  // give time to log on to network.
22   SIM900.print("AT+CLIP=1\r"); // turn on caller ID notification
23   delay(100);
24 }
25
26 void SIM900power()
27 // software equivalent of pressing the GSM shield "power" button
28 {
29   digitalWrite(9, HIGH);
30   delay(1000);
31   digitalWrite(9, LOW);
32   delay(7000);
33 }
```

```
34
35  void doSomething()
36  {
37    if (onoff==0)
38    {
39      onoff=1;
40      digitalWrite(12, HIGH);
41      digitalWrite(13, LOW);
42      Serial.println("D12 high D13 low");
43    }
44    else
45      if (onoff==1)
46      {
47        onoff=0;
48        digitalWrite(12, LOW);
49        digitalWrite(13, HIGH);
50        Serial.println("D12 low D13 high");
51      }
52  }
53
54  void loop()
55  {
56    if(SIM900.available() >0)
57    {
58      inchar=SIM900.read();
59      if (inchar=='2')
60      {
61        delay(10);
62        inchar=SIM900.read();
63        if (inchar=='1')
64        {
65          delay(10);
66          inchar=SIM900.read();
67          if (inchar=='2')
68          {
69            delay(10);
70            inchar=SIM900.read();
71            if (inchar=='8')
72            {
73              delay(10);
74              inchar=SIM900.read();
75              if (inchar=='6')
76              {
77                delay(10);
```

```
78              inchar=SIM900.read();
79              if (inchar=='7')
80              {
81                delay(10);
82                inchar=SIM900.read();
83                if (inchar=='5')
84                {
85                  delay(10);
86                  inchar=SIM900.read();
87                  if (inchar=='3')
88                  {
89                    delay(10);
90                    inchar=SIM900.read();
91                    if (inchar=='0')
92                    {
93                      delay(10);
94                      inchar=SIM900.read();
95                      if (inchar=='9')
96                      {
97                        Serial.println("do sometehing");
98                        delay(10);
99                        // now the number is matched, do something
100                       doSomething();
101                       // arbitrary delay so the function isn't called again on the same phone call
102                       delay(60000);
103                     }
104                   }
105                 }
106               }
107             }
108           }
109         }
110       }
111     }
112   }
113 }
114 }
```

The large delay after calling *doSomething()* exists to stop the same action being called twice (or more) on the same inbound call. Anyhow, you should now have a grasp on interrogating the data from the shield. Which leads us to the final section…

***Controlling the Arduino via SMS text message***

As you did with the caller ID data, you can also control the Arduino via SMS fairly easily, and have more options. In our example we'll explain how to control four digital output pins via SMS. The example works in two stages. First it will wait for an SMS to be received, and then have

the contents sent to the Arduino via serial just as we did earlier with the example 55.4. The next stage is to filter out the commands in the text message as we did with example 55.6. The commands (that is, the contents of your text message to the Arduino) will be in the form #axbxcxdx

where ‘x’ will be 0 (for off) and 1 (for on) – and a, b, c and d will relate to digital pins 10, 11, 12 and 13. For example, to turn on D10, 11 and turn off D12, D13 you would compose your SMS as #a1b1c0d0. After processing the SMS we use the AT command *AT+CMGD=1,4* to delete all the SMSs from the SIM card, otherwise it will fill up and reject further commands. Moving on, here’s the example sketch:

```
1  // Example 55.7
2
3  #include <SoftwareSerial.h>
4  char inchar; // Will hold the incoming character from the GSM shield
5  SoftwareSerial SIM900(7, 8);
6
7  int led1 = 10;
8  int led2 = 11;
9  int led3 = 12;
10 int led4 = 13;
11
12 void setup()
13 {
14   Serial.begin(19200);
15   // set up the digital pins to control
16   pinMode(led1, OUTPUT);
17   pinMode(led2, OUTPUT);
18   pinMode(led3, OUTPUT);
19   pinMode(led4, OUTPUT);
20   digitalWrite(led1, LOW);
21   digitalWrite(led2, LOW);
22   digitalWrite(led3, LOW);
23   digitalWrite(led4, LOW);
24
25   // wake up the GSM shield
26   SIM900power();
27   SIM900.begin(19200);
28   delay(20000);  // give time to log on to network.
29   SIM900.print("AT+CMGF=1\r");  // set SMS mode to text
30   delay(100);
31   SIM900.print("AT+CNMI=2,2,0,0,0\r");
```

```
  // blurt out contents of new SMS upon receipt to the GSM shield's serial out
  delay(100);
  Serial.println("Ready...");
}

void SIM900power()
// software equivalent of pressing the GSM shield "power" button
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(7000);
}

void loop()
{
  //If a character comes in from the cellular module...
  if(SIM900.available() >0)
  {
    inchar=SIM900.read();
    if (inchar=='#')
    {
      delay(10);

      inchar=SIM900.read();
      if (inchar=='a')
      {
        delay(10);
        inchar=SIM900.read();
        if (inchar=='0')
        {
          digitalWrite(led1, LOW);
        }
        else if (inchar=='1')
        {
          digitalWrite(led1, HIGH);
        }
        delay(10);
        inchar=SIM900.read();
        if (inchar=='b')
        {
          inchar=SIM900.read();
          if (inchar=='0')
          {
```

```
76              digitalWrite(led2, LOW);
77            }
78            else if (inchar=='1')
79            {
80              digitalWrite(led2, HIGH);
81            }
82            delay(10);
83            inchar=SIM900.read();
84            if (inchar=='c')
85            {
86              inchar=SIM900.read();
87              if (inchar=='0')
88              {
89                digitalWrite(led3, LOW);
90              }
91              else if (inchar=='1')
92              {
93                digitalWrite(led3, HIGH);
94              }
95              delay(10);
96              inchar=SIM900.read();
97              if (inchar=='d')
98              {
99                delay(10);
100               inchar=SIM900.read();
101               if (inchar=='0')
102               {
103                 digitalWrite(led4, LOW);
104               }
105               else if (inchar=='1')
106               {
107                 digitalWrite(led4, HIGH);
108               }
109               delay(10);
110             }
111           }
112           SIM900.println("AT+CMGD=1,4"); // delete all SMS
113         }
114       }
115     }
116   }
117 }
```